

Adversarial Search & Games

PAGE NO.

DATE

Game:-

A game can be defined as a type of search problem with the following elements.

i) S_0 :- The initial state, which defines how the game is set up at the start.

ii) $PLAYER(S)$:-

Defines which player has the move in a state.

iii) $ACTIONS(S)$:-

Returns the set of legal moves in a state.

iv) $RESULT(S, a)$:-

The transitional model, which defines the result of move.

v) $TERMINAL-TEST(S)$:-

A terminal test is true when game is over.

and it is false when game is not over.

vi) $UTILITY(S, p)$:-

- A utility function is also called as objective function or payoff function.

- $s \rightarrow$ terminal state.

$p \rightarrow$ player.

- The utility function defines the ^{final} numeric value for a game that ends in terminal state s for a player p .

for example, In Chess the outcome is a win loss or draw with numeric value $+1$ for win, 0 for loss, & $\frac{1}{2}$ for draw.

* Zero-sum Game :-

It is defined as a game where the total payoff to all players is the same for every instance of the game.

e.g. chess is a zero sum game because it has payoff either

$0 + 1$ - first player lost & 2nd won.

$1 + 0$ - first player won & 2nd lost

or $\frac{1}{2} + \frac{1}{2}$ - Match has been drawn.

* Game Tree :-

A initial state, ACTIONS function & RESULT function for game defines a game tree for that game.

- In a game tree nodes are the game states & edges are the moves.

e.g. Let's consider the example of tic-tac-toe.

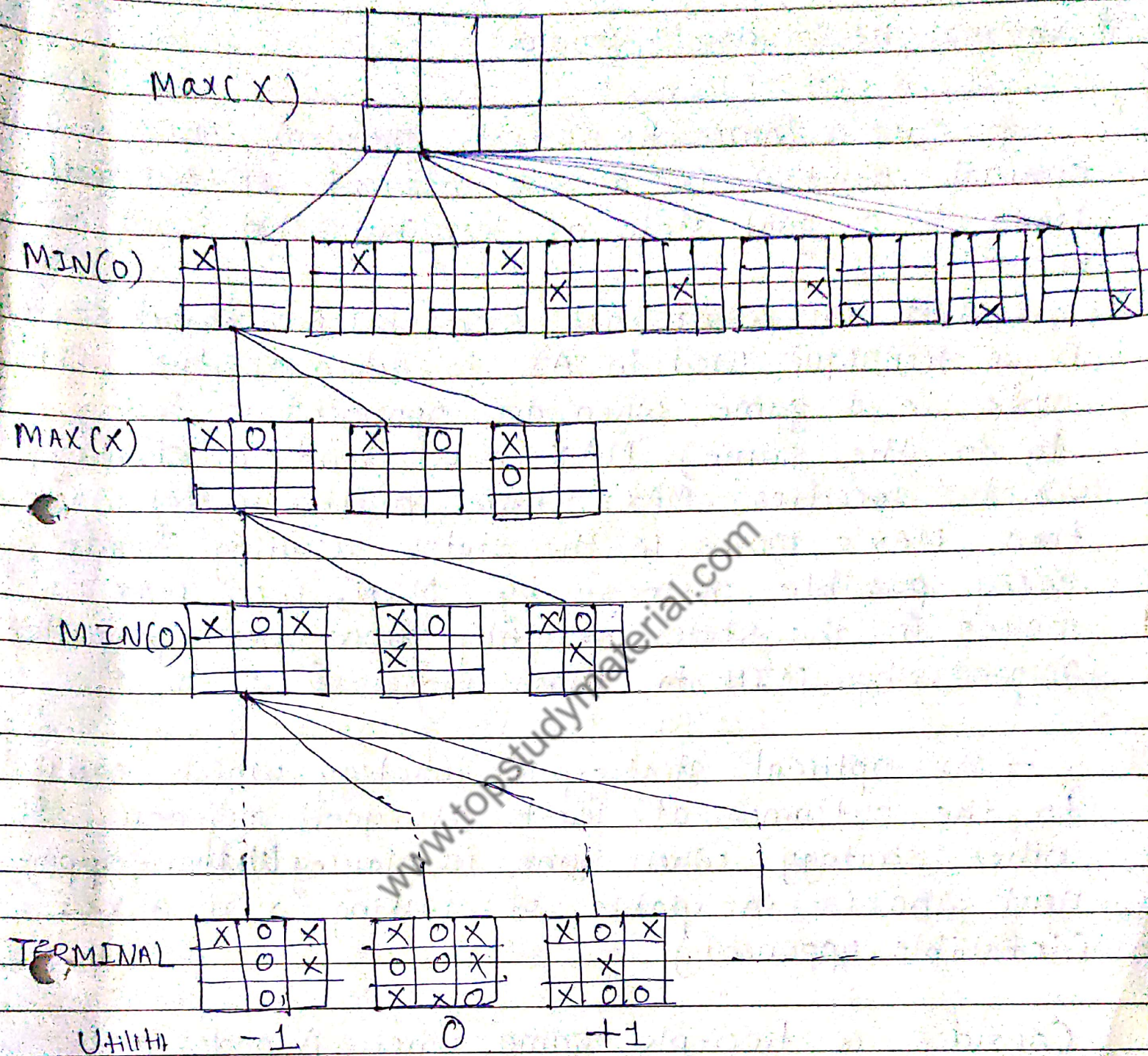
where we place noughts & crosses in square box. We have two players MAX & MIN. MAX will place X & MIN will place O in square box.

- As given in diagram, from the initial state MAX has nine possible moves.

- The game play alternates between MAX's placing an X and MIN's placing an O, until we reach leaf nodes corresponding to terminal states such that one player has three in a row or all the squares are filled.

- The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX.

- High values are assumed to be good for MAX & bad for MIN. (which is how players have given names)



* A Partial Game Tree for Tic-tac-toe *

- For tic tac toe, game tree is relatively small.
- fewer than $9! = 362880$ terminal nodes will be there. But for chess there are over 10^{40} nodes.

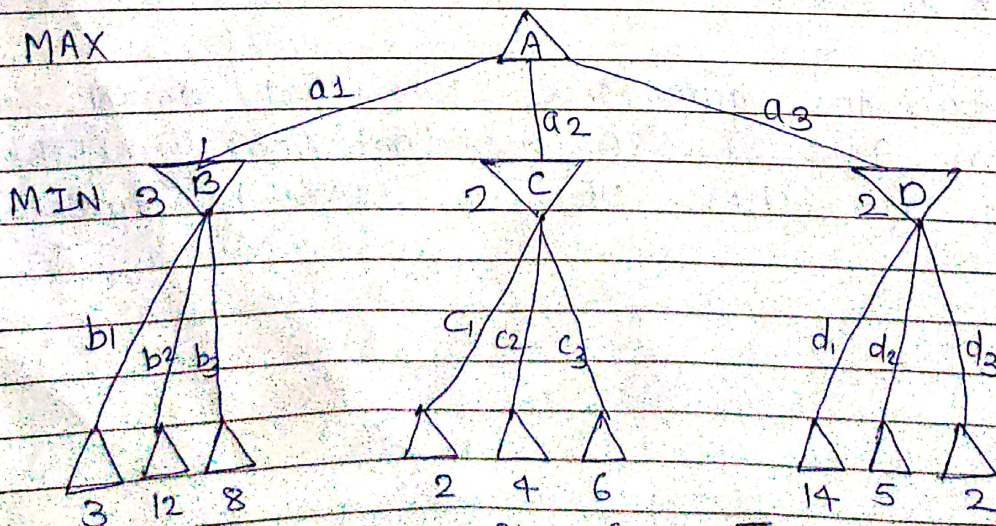
* Optimal Decisions in Games :-

- In a normal search problem, the optimal solution is a sequence of actions that leads to a goal state or terminal that is a Win.

- But in adversarial search (Adversarial search is a technique used in AI to determine the best move in a game when an opponent is trying to do the same.) MAX must find a strategy which specifies MAX's move in the initial step, then MAX's moves in the states resulting from every possible response by MIN, then MAX's moves in the states resulting from every possible response by MIN to those moves & so on.

- So, optimal strategy is strategy which leads to the outcomes at least as good as any other strategy when one is playing ~~with~~ an opponent who is incapable of making a mistake. (Infallible opponent.)

Consider a two-ply game, where Δ nodes are MAX & ∇ nodes are MIN.



* Two-PLY Game Tree *

- In a diagram, the possible moves for MAX at the root node are labeled a_1, a_2 & a_3 .
 - The possible replies to a_1 for MIN are b_1, b_2, b_3 and so on.
- In Game parlance,

*PLY-

- PLY is a single turn taken by one player in a game.

* Full move -

A full move consists of one move from each player.

* A tree one move deep:-

It includes just one move from each player (so two half moves or two plies).

For example: in chess.

If white moves first, that's one Ply.

If Black responds, that's another ply.

Together, they make one full move.

- So in given game tree, the optimal strategy can be determined from the minimax value of each node, denoted as $\text{MINIMAX}(n)$.

- The minimax value of a node is the utility (for MAX) of being in the corresponding state assuming that both players play optimally from there to end of the game.

- The minimax value of the terminal state is just its utility.

- The minimax value of a terminal state is just its utility.

- MAX prefers to move to a state of maximum value whereas MIN prefers a state of minimum value.

So,

MINIMAX(S) =

$$\begin{cases} \text{Utility}(s) & \text{if terminal-test}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

- The terminal node on the bottom level get three utility values from the game's UTILITY function.

- The first min node, labeled as B has three successor states with values 3, 12, and 8. so its minimax value is 3. So other two MIN nodes have minimax value 2.

- The root node is a MAX node, its successor states have minimax values 3, 2 and 2. so it has minimax value is 3.

- The MINIMAX decision at the root is action a1 because it is the optimal choice for MAX as it leads to the state where the state has highest minimax value.

* The minimax algorithm:-

- The minimax algorithm is a decision-making algorithm used in AI particularly in Game Theory & adversarial search problems.

- It is widely used in two player turn based games like chess, tic-tac-toe, to make optimal moves by minimizing the possible loss in a worst case scenario.

* Key components of Minimax Algorithm:-

1. Two players:-

i) Maximizer:

Tries to maximize the score.

ii) Minimizer:-

Tries to minimize the score.

2. Game Tree Representation:-

- The possible moves are represented as a tree where,

- The root node represents the current game state.

- The branches represent possible moves.

- The leaf node contains the final game score (winning, losing & draw).

3. Recursive Evaluation:-

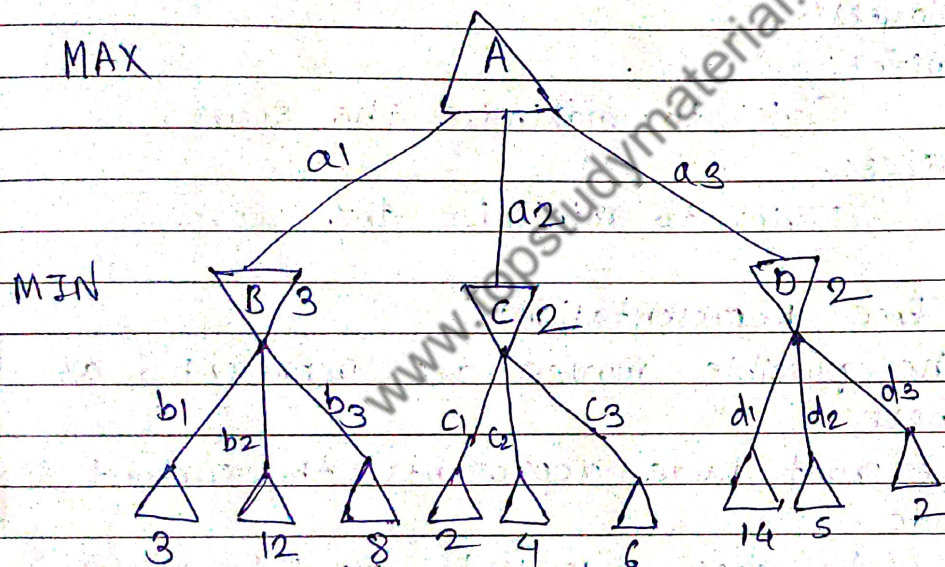
- The algorithm explores all possible game states up to a certain depth.

- Each state is assigned a heuristic value.

- The AI chooses the move that leads to the best possible outcome based on these values.

Minimax algorithm Steps:-

1. Generate the game tree up to specific depth.
2. Evaluate terminal states using a heuristic function.
3. Backtrack from the terminal states:
 - i) If it's the Maximizer's turn, Choose the maximum value among child nodes.
 - ii) If it's the minimizer's turn, choose the minimum value among child nodes.
4. Choose the best move at the root level.



* A two ply game tree *

- As shown in diagram,
- The algorithm first recurse down to the three bottom left nodes & uses the utility function on them to discover their values 3, 12 & 8 respectively.
- Then it takes minimum of these values i.e. 3. and it returns it as the

backup value of node B.

- A similar process gives backed up values of 2 for C & 2 for D.
- Finally, we take the maximum of 3, 2 & 2 to get the backup value of 3 for the root node.

* Key points:-

The minimax algo performs a complete depth first exploration of a game tree.

If the maximum depth of the tree is m and there are b legal moves at each point, then the time complexity of the minimax algorithm is $O(b^m)$.

- The space complexity is $O(bm)$ for an algorithm that generates all actions at once.

Note: For real games, the time cost is totally impractical.

* Heuristic Alpha-Beta Tree Search:-

- The heuristic alpha-beta tree search is an optimized version of the minimax algorithm used in adversarial search problems like chess, tic-tac-toe & other two player games.
- It improves efficiency by pruning unnecessary branches & using heuristic evaluation to estimate the best move when searching to a limited depth.

* Need of Alpha-Beta Pruning:-

- Minimax algorithm evaluates all possible moves, which is very slow for large game trees.
- Alpha-Beta Pruning skips branches that won't affect the final decision making search faster.
- Heuristic evaluation helps when the entire game tree cannot be explored due to complexity.

* Key Concepts:-

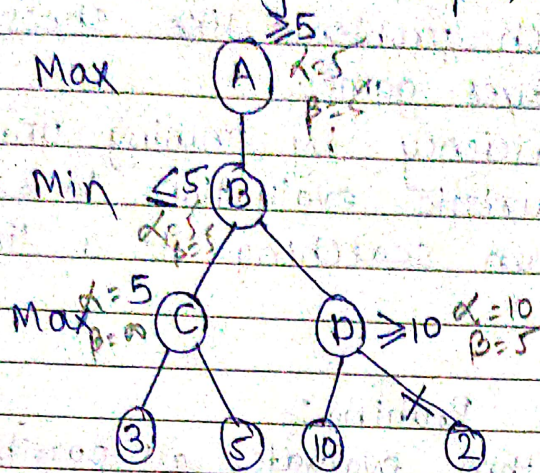
1. Alpha (α) - The best guaranteed score for the Maximizer found so far.
2. Beta (β) - The best guaranteed score for the Minimizer found so far.
3. Pruning - If the algorithm finds that a branch cannot improve the final decision it stops evaluating further moves in that branch.
4. Heuristic Evaluation Functions:-

When the search cannot reach terminal state; a heuristic function is used to estimate the value of a position.

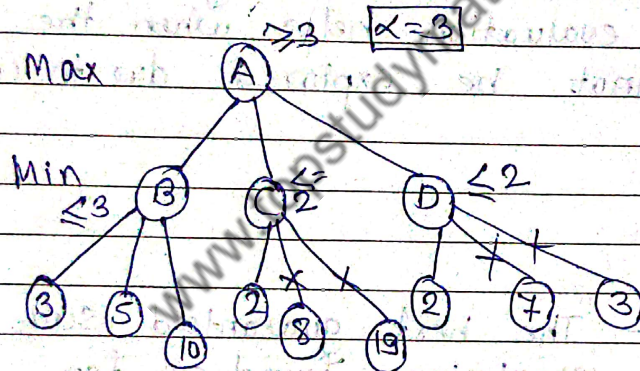
Examples:-

Considers following example,

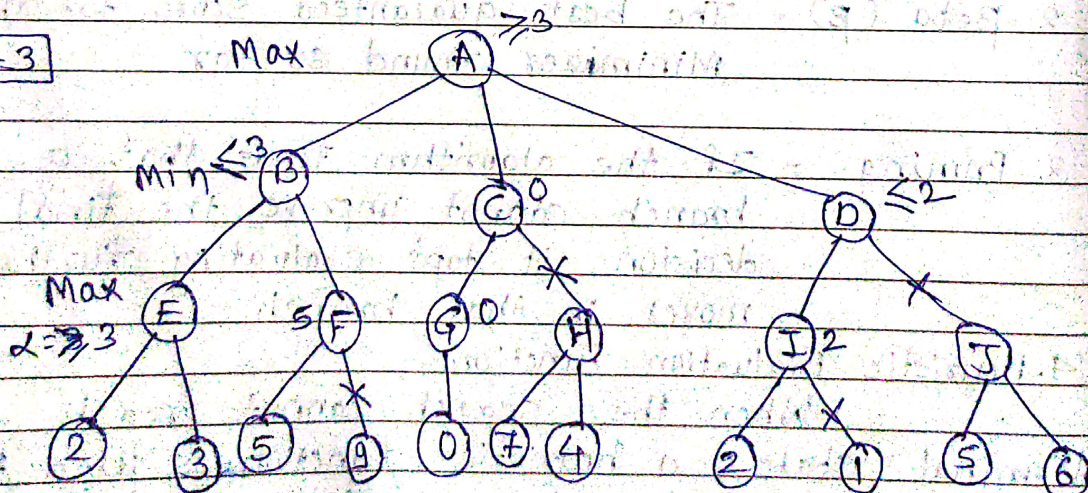
Ex-1



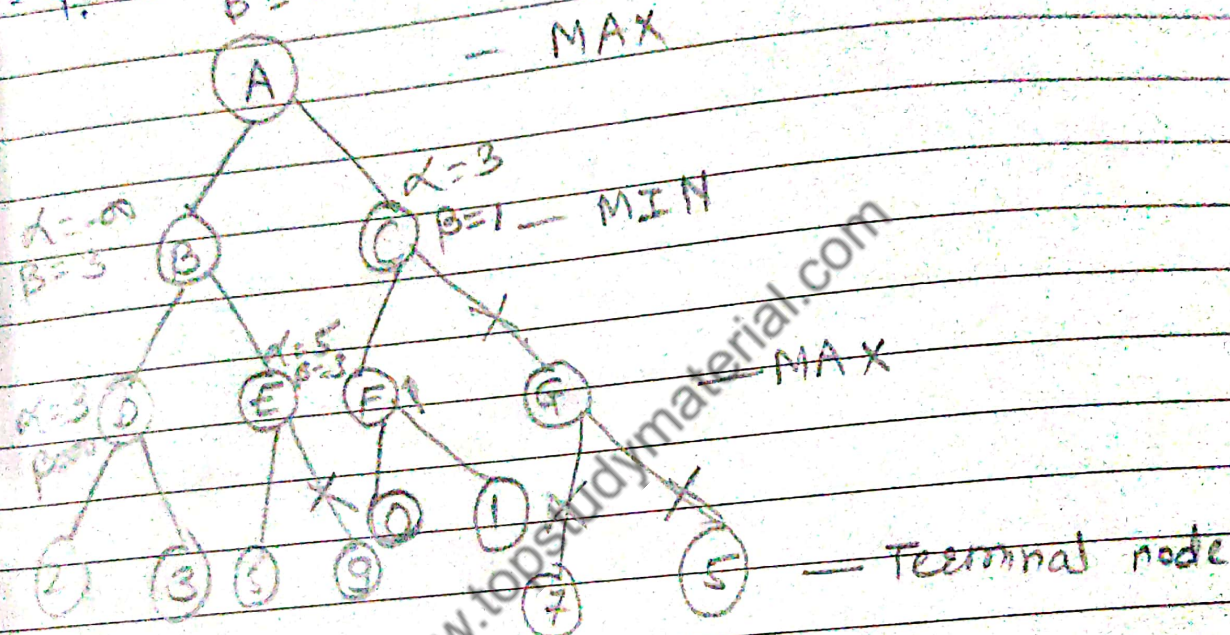
Ex-2



Ex-3



Ex-4. $\alpha = -\infty, 3$
 $\beta = \infty$



At E $\alpha \geq \beta = 5 \geq 3$ Prune
At C $\alpha \geq \beta = 3 \geq 1$ Prune

* Monte Carlo Tree Search:-

- Monte Carlo Tree Search (MCTS) is an AI algorithm used for decision-making in games like Chess, tic-tac-toe.

- It helps the AI simulate possible future moves & pick the best one based on trial & error.

Steps:-

MCTS consists of four main steps, iterated multiple times to improve decision making.

1. Selection:-

Start from the root node & traverse the tree using a policy (like UCB1 - Upper confidence bound) to select a promising node.

2. Expansion-

If the selected node is not a terminal node, expand it by adding one or more child nodes.

3. Simulation (Rollout)-

Simulate a random play (or use a heuristic policy) from the newly added node until reaching a terminal state.

4. Backpropagation-

Propagate the results of the simulation back up the tree, updating the statistics of visited node.

* Advantages of MCTS:-

i) Works well with large, complex decision spaces.
ii) No need to construct a full game tree in advance.

iii) Can be stopped at any time & still provide a valid move.

iv) Improves decision making over time with more iterations.

UCB1 :- Upper confidence bound 1

UCB1 formula used in MCTS to help the computer decide which move (node) to try next.

It balances two things:

1. Exploitation -

- Try moves that gave good results in past.

2. Exploration -

- Try new or less visited moves that might be better.

$$UCB1 = \frac{w_i}{n_i} + c \sqrt{\frac{\ln N}{n_i}}$$

w_i → Number of wins from this move.

n_i → Number of times this move was tried.

N → Total times the parent node was visited.

c → Constant to control exploration (usually around $\sqrt{2}$).

$\ln$ → Natural logarithm.

Example :- Imagine you're trying to find best pizza shop.

Win Rate :- You have tried Pizza A 10 times, & it was good 8 times. → $\frac{8}{10} = 0.8$ (first part of formula)

Exploration Bonus :- Pizza B you only tried once, and it was OK. Maybe it's better than A, but you don't know yet. The second part gives boost to B because it has not been tried much.

That means: We can say that Choose what worked before (first part of formula), and Give fair chance to new or rarely tried options (second part of formula). It helps MCTS make smart decisions, balancing known good choices.

* Stochastic Games:

The stochastic games are a type of game in which outcomes depend on both the actions of the players and some element of randomness.

In these games, players make decisions at various stages, and the state of the game evolves probabilistically over time.

* Features:-

1. State Space:-

The game is characterized by a set of states that represent different possible configurations of the game at any given time.

2. Actions:-

Players choose actions in each state.

The choices they make can affect both the next state & their own payoff.

3. Transitions:-

When players make decisions, the game moves from one state to another, but the transition is probabilistic, meaning that the next state is non deterministic.

4. Payoff function:-

Each player has a payoff function that gives the reward based on the current state & the action taken.

The goal is to maximize their expected payoff over time.

5. Discounting:-

The discounting refers to the idea that reward or payoffs received in the future are considered less

Valuable than those received in present.

* Types of Stochastic Games:-

1. Markov Decision Process.
2. Partially Observable Stochastic Games.
3. Dynamic Games.
4. Stochastic Differential Games.

* Applications:-

- 1) Economics.
- 2) Reinforcement learning.
- 3) Finance.
- 4) Biology.

* Constraint Satisfaction Problems-

A constraint satisfaction problem consists of three components, X , D and C .
where,

X is a set of variables $\{X_1, \dots, X_n\}$.

D is a set of domains $\{D_1, \dots, D_n\}$

one for each variable,

C is a set of constraints that specify allowable combinations of values.

- A domain D_i consists of a set of allowable values $\{v_1, \dots, v_k\}$ for variable X_i .

e.g. A boolean variable would have the domain $\{\text{true}, \text{false}\}$.

- Different variables can have different domains of different sizes.

- Each constraint C_j consists of pair $\langle \text{scope}, \text{rel} \rangle$

↳ scope is a tuple of variables that participate in the constraint.

↳ rel is a relation that defines the values that those variables can take on.

- The relation can be represented as an explicit set of all tuples of values that satisfy the constraint, or as a function that can compute whether a tuple is a member of the relation.

For example, variable X_1 and X_2 both have domain $\{1, 2, 3\}$ then the constraint is X_1 must be greater than X_2 .

So the representation of above problem in CSP is as,

$\langle (X_1, X_2), \{(3, 1), (3, 2), (2, 1)\} \rangle$

or

$\langle (X_1, X_2), X_1 > X_2 \rangle$.

- CSPs deals with assignments of values to variables. (domain consists of set of allowable values) like $\{v_1, v_2, \dots, v_k\}$.
i.e. $\{x_i = v_i, x_j = v_j\}$.

i) Consistent Assignment :-

An assignment that does not violate any constraint is called a consistent or legal Assignment.

ii) Complete Assignment :-

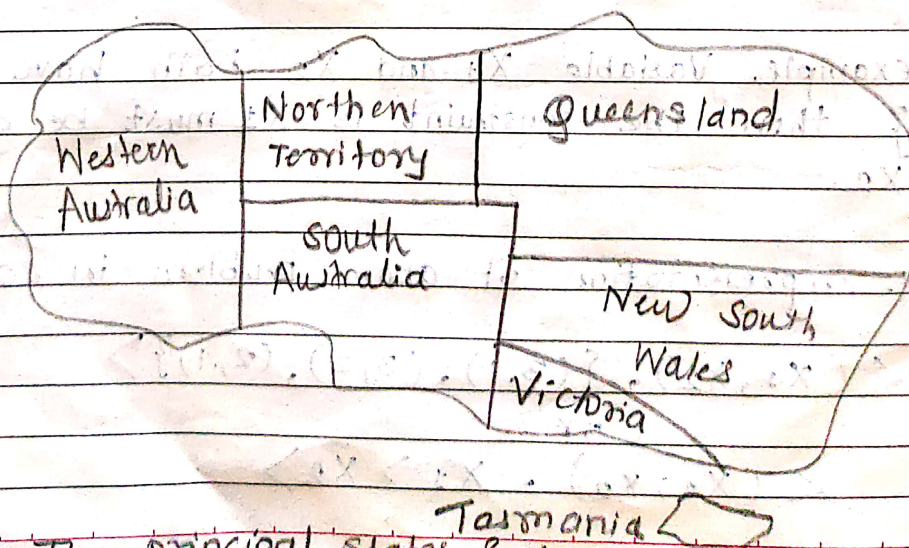
If every variable is assigned a value and a solution to a CSP is a consistent then it is called as complete assignment.

iii) Partial Assignment :-

If some variables are unassigned & partial solution is consistent then that assignment is called as partial assignment.

* Solving CSP is an NP-complete problem in general. *

* Example :- Map coloring :-



* The principal states & territories of Australia. *

* Constraint Satisfaction Problems-

A constraint satisfaction problem consists of three components, X , D and C .
where,

X is a set of variables $\{X_1, \dots, X_n\}$.

D is a set of domains $\{D_1, \dots, D_n\}$

one for each variable.

C is a set of constraints that specify allowable combinations of values.

- A domain D_i consists of a set of allowable values $\{v_1, \dots, v_k\}$ for variable X_i .

e.g. A boolean variable would have the domain $\{\text{true}, \text{false}\}$.

- Different variables can have different domains of different sizes.

- Each constraint C_j consists of pair $\langle \text{scope}, \text{rel} \rangle$

↳ scope is a tuple of variables that participate in the constraint.

↳ rel is a relation that defines the values that those variables can take on.

- The relation can be represented as an explicit set of all tuples of values that satisfy the constraint, or as a function that can compute whether a tuple is a member of the relation.

For example, Variable X_1 and X_2 both have domain $\{1, 2, 3\}$ then the constraint is X_1 must be greater than X_2 .

So the representation of above problem in CSP is as,

$\langle (X_1, X_2), \{(3, 1), (3, 2), (2, 1)\} \rangle$

or

$\langle (X_1, X_2), X_1 > X_2 \rangle$.

- CSPs deals with assignments of values to variables. (domain consists of set of allowable values) like $\{v_1, v_2, \dots, v_k\}$.
i.e. $\{x_i = v_i, x_j = v_j\}$.

i) Consistent Assignment :-

An assignment that does not violate any constraint is called a consistent or legal Assignment.

ii) Complete Assignment :-

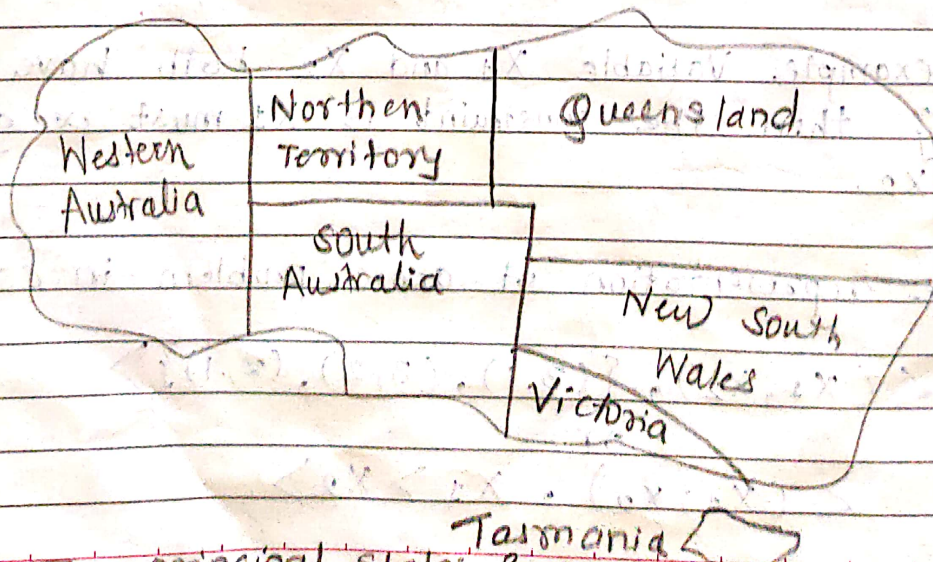
If every variable is assigned a value and a solution to a CSP is a consistent then it is called as complete assignment.

iii) Partial Assignment :-

If some variables are unassigned & partial solution is consistent then that assignment is called as partial assignment.

* Solving CSP is an NP-complete problem in general. *

* Example :- Map coloring:



* The principal states & territories of Australia. *

- To color the graph is a constraint satisfaction problem (CSP).

- The goal is to assign colors to each region so that no neighbouring regions have the same color.

Problem Statement:-

We have given a map of Australia showing each of its states & territories. We have given the task of coloring each region either red, green or blue in such a way that no two neighboring regions have the same color.

Formulate above problem in constraint satisfaction Problem.
Solution:-

Step 1:-

Formulate this problem in CSP.

i) Define Variables:-

$$X = \{ WA, NT, Q, NSW, V, SA, T \}.$$

ii) Domain for every variable

$$D_i = \{ \text{red, green, blue} \}.$$

iii) Constraint:-

The constraint requires neighboring region to have distinct colors.

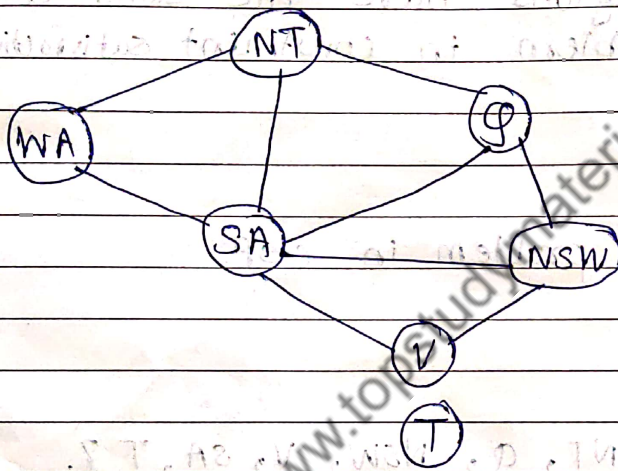
Since there are nine regions or places there are nine constraints.

$$C = \{ SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V \}$$

Here, $SA \neq WA$ is a shorthand for $\langle (SA, WA), SA \neq WA \rangle$, where $SA \neq WA$ can be fully enumerated in turn as

$\{ (red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green) \}$.

We can draw the constraint graph for above problem.



* Constraint graph for Map coloring problem.

- The nodes of a graph correspond to variables of the problem, and edge connects any two variables that participate (or represent) in a constraint.

What is the Need to formulate a problem as CSP?

→ i) The CSP provides a natural representation for a wide variety of problems.

ii) It is often easy to formulate a problem as a CSP.

iii) CSP solvers are fast & efficient.

iv) The CSP solvers ability to rapidly prune the

Search space gives a significant advantage.

For example, once we have chosen $\{SA = \text{blue}\}$ in Australia problem, we can conclude that none of the five neighboring variables can take on the value blue.

* Types of constraints in CSP :-

A constraint satisfaction problem (CSP) is a problem where variables must be assigned values that satisfy certain constraints.

The constraint defines permissible combinations of variable values.

CSP consists of:

- 1) Variable (X) - A set of variables $X_1, X_2, \dots, X_n$.
- 2) Domains (D) - A set of possible values for each variable.
- 3) Constraints (C) - Rules that define valid assignment

Types of constraints

- 1) Unary constraint.
- 2) Binary
- 3) Higher-order constraint / Global constraint.

1) Unary constraint :-

Constraints that involve a single variable.

They restrict the domain of that variable.
Or - unary constraints are the constraints which restricts the value of single variable.

- ① For example, in map coloring problem it would be the case that south Australians won't tolerate green color.

So unary constraint will be,

$$\langle (SA), SA \neq \text{green} \rangle$$

② e.g. Temperature Restrictions.

Variable: T (Temperature in Celsius.)

Domain: $D_T = \{0, 10, 20, 30, 40, 50\}$

Constraint: $T > 15^\circ\text{C}$.

Valid values: $D_T = \{20, 30, 40, 50\}$.

- Unary constraints help reduce the search space early by eliminating invalid values.

2) Binary constraints:

- Constraints that involve two variables & specify allowed relationships between them.

Example 1: Different colors in Graph coloring.

Variables: Pune & Mumbai (nodes in a graph)

Domain: $D = \{\text{Red, Green, Blue}\}$

Constraint: Pune $\neq$ Mumbai (adjacent nodes must have different colors)

If Pune = Red, then Mumbai cannot be red.

c Example 2: Scheduling Conflict.

Variables: A (Meeting 1), B (Meeting 2)

Domain: Time slots (9AM, 10AM, 11AM)

Constraint: $A \neq B$ (Meetings cannot be at the same time)

If $A = 9\text{AM}$, then B must be either 10AM or 11AM

3) Global constraint (or Higher order constraints)

- The constraints that involve three or more variables. OR A constraint involving an arbitrary number of variables is called as global constraints.

- One of the most common global constraints is Alldiff which says that all of the variables involved in the constraint must have different values.

Example 1: Alldiff constraint in Sudoku.

Variables: $X_1, X_2, X_3, \dots, X_9$ (values in Sudoku row)

Constraint: All values must be unique.

$$X_1 \neq X_2 \neq X_3 \neq \dots \neq X_9$$

If $X_1 = 5$, then no other X_i can be 5.

- In Sudoku problems, all variables in a row, column or 3×3 box must satisfy an Alldiff constraint.

Example 2: N-Queens problem.

Variables: $Q_1, Q_2, \dots, Q_N$ (Positions of N queens on an $(N \times N)$ chessboard)

Constraints:

i) No two queen in the same row $\rightarrow Q_i \neq Q_j$

ii) No two queens in the same column.

iii) No two queens in the same diagonal.

* Constraint Propagation:-

- A Constraint satisfaction problem has choices.

- It can generate successors by choosing a new variable assignment, or it can do a specific type of inference is called constraint propagation.

- It uses the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable & so on.

* Local Consistency:-

The local consistency in CSP refers to conditions applied to subsets of variables & constraints to ensure that partial assignments can be consistently extended.

By enforcing local consistency, the search space can be reduced, making the problem easier to solve.

Types of local consistency:-

- i) Node consistency:-
- ii) Arc consistency.
- iii) Path consistency.

① Node consistency:-

A variable is node consistent if all values in its domain satisfy the variable's unary constraint.
E.g. If a variable X has a domain $\{1, 2, 3, 4\}$ & a unary constraint $X \leq 3$ enforcing node consistency reduces the domain to $\{1, 2, 3\}$.

Consider another example

Suppose, in map coloring problem, the south australian

dislike green, the variable SA starts with domain $\{red, green, blue\}$ & we can make it node consistent by eliminating green.

And domain for SA = $\{red, blue\}$.

- A Graph is node-consistent if every variable in the graph is node-consistent.

② Arc Consistency:-

A variable is arc consistent with another variable if, for every value in its domain, there exists a compatible value in other variable's domain that satisfies the binary constraint between them.

The X_i is arc consistent with respect to another variable X_j , if for every value in the current domain D_i there is some value in domain D_j that satisfies the binary constraint on the arc (X_i, X_j) .

A graph is arc consistent if every variable is arc consistent with every other variable.

For example,

consider the constraint $Y = X^2$, Variables = $\{X, Y\}$.
Domain of the X & Y variables is the set of decimal digits.

To make X arc-consistent with respect to Y, we reduce X's domain to $\{0, 1, 2, 3\}$.

If we also make with respect to X, then Y's domain becomes $\{0, 1, 4, 9\}$.

And the whole CSP is arc consistent.

$\langle (X, Y), \{(0, 0), (1, 1), (2, 4), (3, 9)\} \rangle$

Consider another example,

Consider variables are X, Y

Variables = $\{X, Y\}$

Domain = $\{1, 2, 3\}$

Constraint = $X < Y$

Now, ~~$X < Y$~~

If $X = 3$, there is no value in Y 's domain that satisfies $X < Y$.

Enforcing arc consistency would remove 3 from X 's domain.

- The most popular algorithm for enforcing arc consistency is called AC-3.

* Path Consistency:-

- Suppose we are to color the map of Australia with just two colors, red & blue.

Arc consistency does nothing because every constraint can be satisfied individually with red at one end & blue at the other.

But there is no solution to the problem, because Western Australia, Northern Territory & South Australia all touch each other, we need at least three colors for them alone.

Path consistency in constraint satisfaction problems (CSP) is a way to make sure that when you have three variables, the values assigned to any two of them should be consistent with possible values of the third one.

Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triplets of variables.

A two-variable set $\{X_i, X_j\}$ is path consistent with respect to a third variable X_m if, for every assignment $\{X_i=a, X_j=b\}$ consistent with the constraints (if any) on $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraint on $\{X_i, X_m\}$ and $\{X_m, X_j\}$.

The name refers to the overall consistency of the path from X_i to X_j with X_m in the middle.

For example,

Let's say we have three variables.
A, B, C.

And there are constraints like

A & B are connected.

B & C are connected.

A & C are connected.

The path consistency checks that if $A=a$ & $B=b$ then there must be a value c for C such that

$$(A=a, C=c) \text{ and } (B=b, C=c).$$

- If two values work together, they should also work with a third value to keep the system consistent.

* Path consistency in coloring Australia map with two colors. *

- We will make the set $\{WA, SA\}$ path consistent with respect to NT.

- We start by enumerating the consistent assignments to the set.

- In this case there are only two assignments $\{WA = \text{red}, SA = \text{blue}\}$ & $\{WA = \text{blue}, SA = \text{red}\}$

- With above assignments NT can not be red or blue, because it would conflict with either WA or SA.

- Because there is no valid choice for NT. We eliminate both assignments and there will be no valid assignments for $\{WA, SA\}$

Therefore There can be no solution to this problem.

* Global Constraints:-

- In global constraint random number of variables (but not necessarily all variables) involved.

- Example of global constraint is Alldiff constraint.

- The Alldiff constraint says that all the variables involved must have distinct values.

- e.g. Cryptarithmic problem & sudoku puzzle.

- Inconsistency detection in Alldiff constraints work as follows:

if m variables are involved in the constraint if they have n possible distinct values altogether & $m > n$, then the constraint cannot be satisfied.

Algorithm to detect inconsistency:-

Step 1:- Remove any variable in the constraint that has a singleton domain.

Step 2:- Delete that variable's value from the domains of the remaining variables.

Step 3:- Repeat as long as there are singleton variables.

Step 4:- If at any point an empty domain is produced or there are more variables than domain values left, then an inconsistency has been detected.

* Global Constraints:-

- In global constraint random number of variables (but not necessarily all variables) involved.

- Example of global constraint is Alldiff constraint.

- The Alldiff constraint says that all the variables involved must have distinct values.

- e.g. Cryptarithmic problem & sudoku puzzle.

- Inconsistency detection in Alldiff constraints work as follows:

if m variable are involved in the constraint & if they have n possible distinct values altogether & $m > n$, then the constraint cannot be satisfied.

Algorithm to detect inconsistency:

Step 1:- Remove any variable in the constraint that has a singleton domain.

Step 2:- Delete that variable's value from the domains of the remaining variables.

Step 3:- Repeat as long as there are singleton variables.

Step 4:- If at any point an empty domain is produced or there are more variables than domain values left, then an inconsistency has been detected.

* Sudoku:-

- The Sudoku is constraint satisfaction problem.
- A sudoku board consists of 81 squares. Some of which are initially filled with digits from 1 to 9.
- The puzzle is to fill in all the remaining squares such that no digit appears twice in any row, column or 3×3 box.
- A row, column, or box is called a unit.

A sudoku in CSP is as follows:

i) Variables:- A sudoku puzzle has 81 variables one for each square.

- We use the variable names A1 through A9 for the top row (left to right) down to I1 through I9 for bottom row.

ii) Domain:-

The empty squares have the domain $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

The prefilled squares have a domain consisting of single value.

iii) Constraints:-

There are 27 different AllDiff constraints one for each unit (row, column, & box of 9 squares).

1) Rows:

AllDiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)

AllDiff(B1, B2, B3, B4, B5, B6, B7, B8, B9)

2) 6

2) Columns: AllDiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)

AllDiff(A2, B2, C2, D2, E2, F2, G2, H2, I2)

3) 3x3 box:-

Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)

Alldiff(A4, A5, A6, B4, B5, B6, C4, C5, C6)

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a) Initial state.

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b) Final solution

a.) A Sudoku puzzle and (b) its solution.

- We are assuming that Alldiff constraints have been expanded into binary constraints. ($A1 \neq A2$)

- Hence we can apply AC-3 algorithm directly.

- Consider variable E6 from diagram (a) initial state the empty square between 2 & 8 in the middle box.

- From constraints in the box, we can remove $\{1, 2, 7, 8\}$ from its domain.

D	1		2
E			
F	7		8

- From column constraints, we can eliminate $\{5, 6, 2, 8, 9\}$ and 3 (although 2 & 8 already removed)

Domain of E6 = $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

= SD after eliminating values considering box constraints & column constraints, E6 domain will be $= \{4\}$.

So, the answer of E6 is 4.

- Now consider I6 variable the square in the bottom middle box surrounded by 1, 3 and 3.

- Applying ac consistency in it's column.

We eliminate 5, 6, 2, 4, 8, 9, 3.

↑ The answer we have found $E6 = 4$.

- We eliminate 1 by ac consistency with I5 (row constraints)

Domain for I6 = $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

I6 = $\{7\}$

So, the answer of I6 is 7.

- Now, there are 8 known values in column 6.

So the ac consistency can infer that A6 must be 1.

Hence, $A6 = 1$.

* Backtracking Search for CSPs:-

- Sometimes we can finish the constraint propagation process & still have variables with multiple values.

- In that case we have to search for a solution.

- The backtracking search procedure ~~type~~ of backtracking algorithm repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to extend each one into a solution via a recursive call.

If the call succeeds, the solution is returned, and if it fails, the assignment is restored to a previous state and we try the next value.

If no value works then we return failure.