Chapter 4: Adversarial search and Constraint Satisfaction problems in AI

Adversarial Search

- ❖ Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which is often expressed in the form of a sequence of actions.
- But, there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed a multi-agent **environment**, in which each agent is an opponent of another agent and playing against each other. Each agent needs to consider the action of another agent and the effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**.
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Types of Games in AI:

	Deterministic	**Chance Moves**
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly

Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other's moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agents do not have all information about the game and are not aware of what's going on, such types of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and have a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called stochastic games.
  Example: Backgammon, Monopoly, Poker, etc.

Note: In this topic, we will discuss deterministic games, fully observable environment, zero-sum, and where each agent acts alternatively.

Zero-Sum Game

- Zero-sum games are adversarial search which involves pure competition.
- In Zero-sum games each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent.
- One player of the game tries to maximize one single value, while another player tries to minimize it.
- Each move by one player in the game is called ply.
- Chess and tic-tac-toe are examples of a Zero-sum game.

Zero-sum game: Embedded thinking

The Zero-sum game involved embedded thinking in which one agent or player is trying to figure out:

- What to do.
- How to decide the move
- Needs to think about his opponent as well
- The opponent also thinks what to do

Each of the players is trying to find out the response of his opponent to their actions. This requires embedded thinking or backward reasoning to solve the game problems in AI.

Formalization of the problem:

A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false in any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called the payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0.

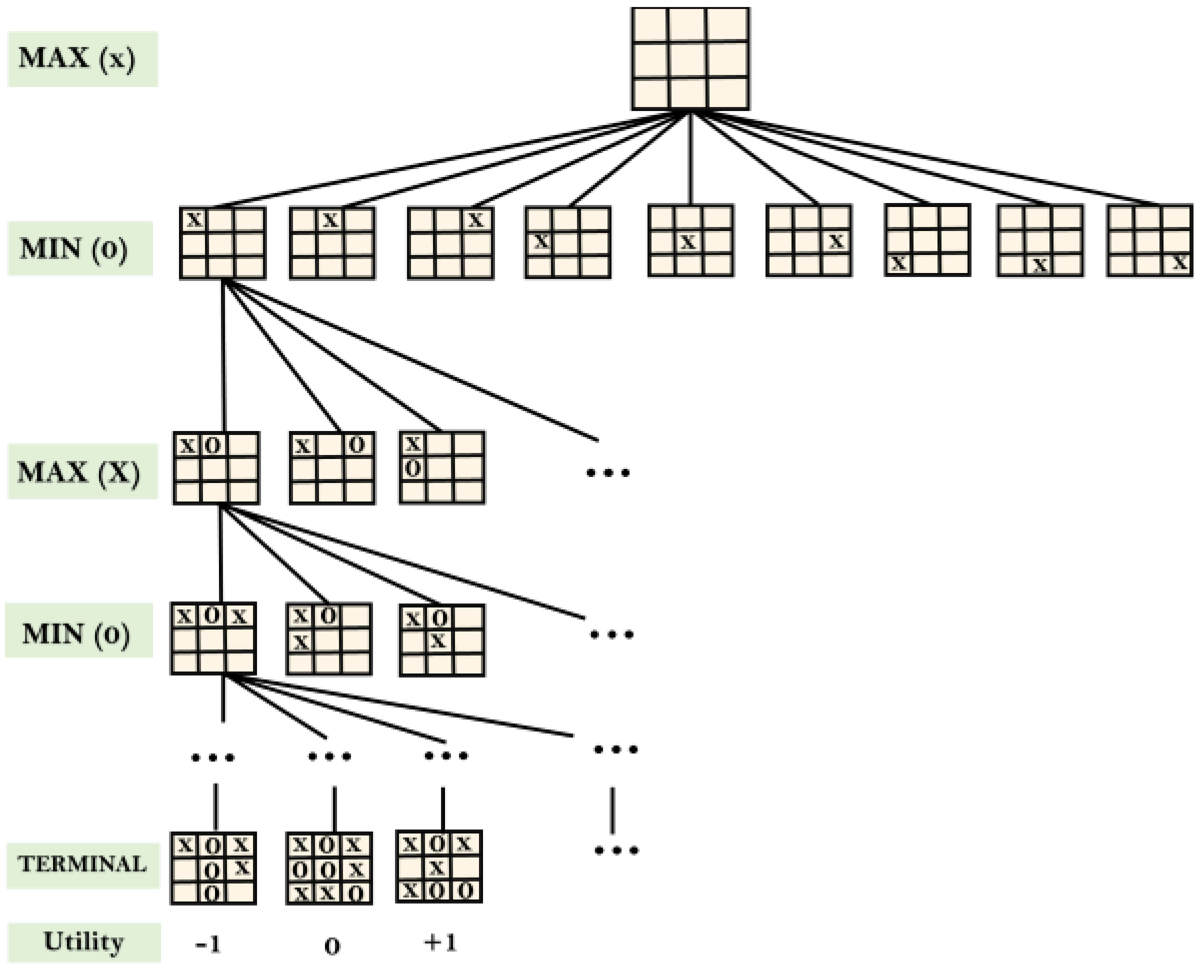
Game tree:

A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players. Game tree involves initial state, actions function, and result Function.

Example: Tic-Tac-Toe game tree:

The following figure shows part of the game-tree for tic-tac-toe games. Following are some key points of the game:

- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree
- MIN minimizes the result.



Example Explanation:

- From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both players play alternatively until we reach a leaf node where one player has three in a row or all squares are filled.

- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.
- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.
- So in the game tree, we have a layer of Max, a layer of MIN, and each layer is called as **Ply**. Max places x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.
- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

Hence adversarial Search for the minimax procedure works as follows:

- It aims to find the optimal strategy for MAX to win the game.
- It follows the approach of Depth-first search.
- In the game tree, optimal leaf nodes could appear at any depth of the tree.
- Propagate the minimax values up to the tree until the terminal node is discovered.

In a given game tree, the optimal strategy can be determined from the minimax value of each node, which can be written as MINIMAX(n). MAX prefer to move to a state of maximum value and MIN prefer to move to a state of minimum value then:

For a state S MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s) & \text{If TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{If PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{If PLAYER}(s) = \text{MIN.} \end{cases}$$

Min Max problem

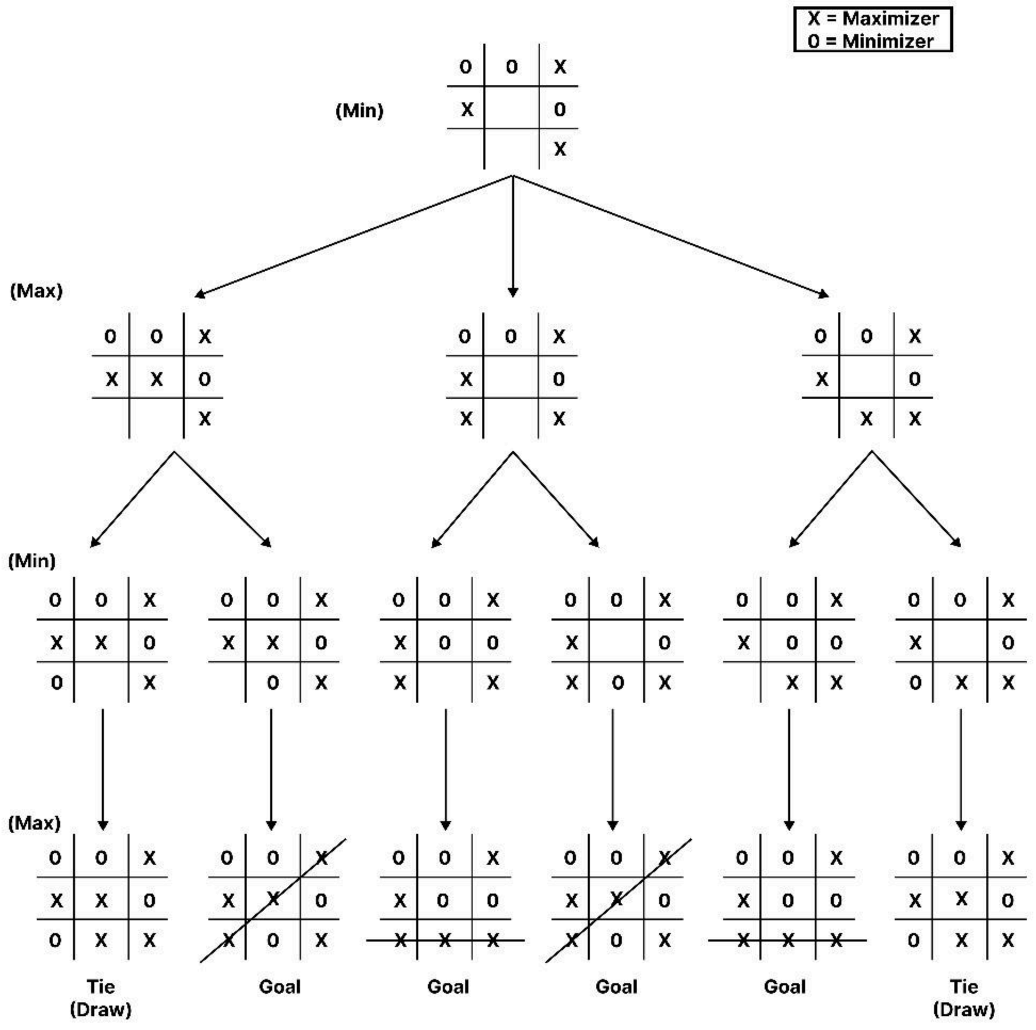


Fig: Game Tree For Tic Tac Toe

What is the problem of the minimax algorithm and how to overcome it using alpha-beta pruning?

- The Minimax algorithm becomes slow for complex games due to its exponential time complexity.
- As the game tree grows larger with more possible moves and countermoves, the time required for Minimax to make decisions increases drastically.

- This slowness makes Minimax impractical for games like chess or Go, where the branching factor is high and the game tree is deep.

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.

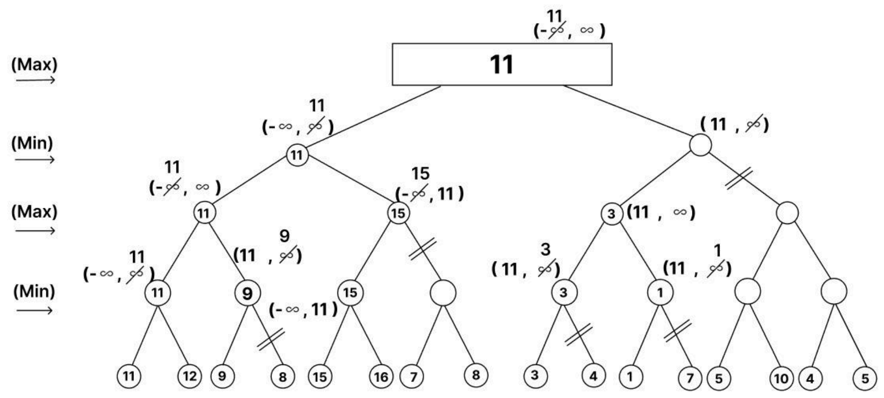
- As we have seen in the minimax search algorithm, the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameters Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called the Alpha-Beta Algorithm.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prunes the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
  a. Alpha: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is $-\infty$.
  b. Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$.
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making the algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

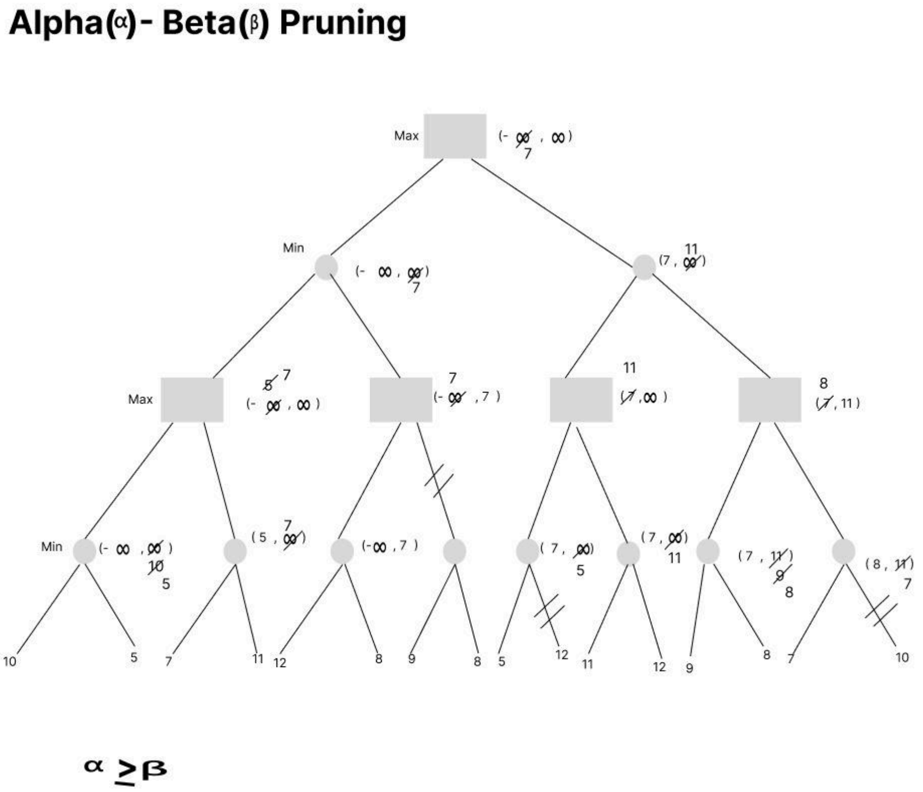
The main condition which required for alpha-beta pruning is:

1. $\alpha >= \beta$

Alpha-beta pruning



Alpha cut-off= 1
Beta cut-off= 4



Alpha cut-off= 1
Beta cut-off=2

Constraint Satisfaction Problems in Artificial Intelligence

- We have encountered a wide variety of methods, including adversarial search and instant search, to address various issues. Every method for issue has a single purpose in mind: to locate a remedy that will enable that achievement of the objective. However there were no restrictions just on bots' capability to resolve issues as well as arrive at responses in adversarial search and local search, respectively.
- This section examines the constraint optimization methodology, another form or real concern method. By its name, constraints fulfillment implies that such an issue must be solved while adhering to a set of restrictions or guidelines.
- Whenever a problem is actually variables that comply with stringent conditions of principles, it is said to have been addressed using the solving multi - objective method. Wow what a method results in a study sought to achieve of the intricacy and organization of both the issues.

Three factors affect restriction compliance, particularly regarding:

- It refers to a group of parameters, or X.
- D: The variables are contained within a collection of several domains. Every variable has a distinct scope.
- C: It is a set of restrictions that the collection of parameters must abide by.

In constraint satisfaction, domains are the areas wherein parameters were located after the restrictions that are particular to the task. Those three components make up a constraint satisfaction technique in its entirety. The pair "scope, rel" makes up the number of something like the requirement. The scope is a tuple of variables that contribute to the restriction, as well as rel is indeed a relationship that contains a list of possible solutions for the parameters should assume in order to meet the restrictions of something like the issue.

Issues with Contains A certain amount Solved

For a constraint satisfaction problem (CSP), the following conditions must be met:

- States area
- fundamental idea while behind remedy.

The definition of a state in phase space involves giving values to any or all of the parameters, like as

X1 = v1, X2 = v2, etc.

There are 3 methods to economically beneficial to something like a parameter:

1. Consistent or Legal Assignment: A task is referred to as consistent or legal if it complies with all laws and regulations.
2. Complete Assignment: An assignment in which each variable has a number associated with it and that the CSP solution is continuous. One such task is referred to as a completed task.
3. A partial assignment is one that just gives some of the variables values. Projects of this nature are referred to as incomplete assignments.

Domain Categories within CSP

The parameters utilize one of the two types of domains listed below:

- Discrete Domain: This limitless area allows for the existence of a single state with numerous variables. For instance, every parameter may receive an endless number of beginning states.
- It is a finite domain with continuous phases that really can describe just one area for just one particular variable. Another name for it is constant area.

Types of Constraints in CSP

Basically, there are three different categories of limitations in regard towards the parameters:

- Unary restrictions are the easiest kind of restrictions because they only limit the value of one variable.
- Binary resource limits: These restrictions connect two parameters. A value between x1 and x3 can be found in a variable named x2.
- Global Resource limits: This kind of restriction includes an unrestricted amount of variables.

The main kinds of restrictions are resolved using certain kinds of resolution methodologies:

- In linear programming, when every parameter carrying an integer value only occurs in a linear equation, linear constraints are frequently utilized.
- Non-linear Constraints: With non-linear programming, when each variable (an integer value) exists in a non-linear form, several types of restrictions are utilized.

> Note: The preferences restriction is a unique restriction that operates in the actual world.

Think of a Sudoku puzzle where some of the squares have initial fills of certain integers.

You must complete the empty squares with numbers between 1 and 9, making sure that no rows, columns, or blocks contain a recurring integer of any kind. This solving multi - objective issue is pretty elementary. A problem must be solved while taking certain limitations into consideration.

The integer range (1-9) that really can occupy the other spaces is referred to as a domain, while the empty spaces themselves were referred to as variables. The values of the variables are drawn first from the realm. Constraints are the rules that determine how a variable will select the scope.

CryptoArithmetic problems and solutions

Following mentioned points are meant to be kept in mind while solving cryptarithmetic questions:

- **Understand the Problem:**
    - Carefully read the cryptarithmetic puzzle to understand the given equations and constraints.
    - Identify the letters that represent unique digits in the equation.
- **List the Letters:**
    - List all the distinct letters involved in the puzzle.
    - Assign a placeholder for each letter (e.g., A, B, C, … Z) to represent digits.
- **Formulate the Equation:**
    - Convert the given words into a mathematical equation.

- Ensure that the equation follows standard mathematical rules (addition, subtraction, multiplication, or division).

- **Identify Constraints:**
  - Identify any constraints or relationships given in the puzzle. For example, if a letter cannot be zero (0), note it down.
- **Start with the Easiest Part:**
  - Look for parts of the equation that are straightforward to solve. This may involve finding single-digit additions or subtractions.
- **Trial and Error:**
  - Begin assigning possible digits to the letters, starting with the constraints and parts of the equation with the fewest possibilities.
  - Use a systematic approach, making sure to keep track of your assignments.
- **Check for Consistency:**
  - After making an assignment, check if it maintains consistency within the equation and adheres to the constraints.
  - If a digit becomes inconsistent with other assignments, backtrack and try a different digit.
- **Iterate and Refine:**
  - Continue the process of assigning digits, checking for consistency, and refining your assignments.
  - Use logical deductions and process of elimination to narrow down possibilities.

Some Examples of crypto-arithemetic problems in an AI

1)

$$\begin{array}{rccc}
 & & T & O \\
+ & & G & O \\
\hline
 & O & U & T
\end{array}$$

⇒

Letters	Digits
T	2
G	8
O	1
U	0

$$\begin{array}{rccc}
 & \boxed{1} & 2 & 1 \\
+ & & 8 & 1 \\
\hline
 & 1 & 0 & 2
\end{array}$$

---

2)

$$\begin{array}{rcccccc}
 & & L & O & G & I & C \\
+ & & L & O & G & I & C \\
\hline
 & P & R & O & L & O & G
\end{array}$$

⇒

Letters	Digits
L	9
O	0
G	4
I	5
C	2
P	1
R	8

$$\begin{array}{rcccccc}
 & \boxed{1} & & & \boxed{1} & & \\
 & & 9 & 0 & 4 & 5 & 2 \\
+ & & 9 & 0 & 4 & 5 & 2 \\
\hline
 & 1 & 8 & 0 & 9 & 0 & 4
\end{array}$$

---

3)

$$\begin{array}{rccccc}
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
 & M & O & N & E & Y
\end{array}$$

⇒

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

$$\begin{array}{rccccc}
 & \boxed{1} & & \boxed{1} & \boxed{1} & \\
 & & 9 & 5 & 6 & 7 \\
+ & & 1 & 0 & 8 & 5 \\
\hline
 & 1 & 0 & 6 & 5 & 2
\end{array}$$