Game Playing

Here, we have
↳ Well defined rules
↳ Easy to evaluate

⇒ Game theory → Rational choice in a multi agent scenario.

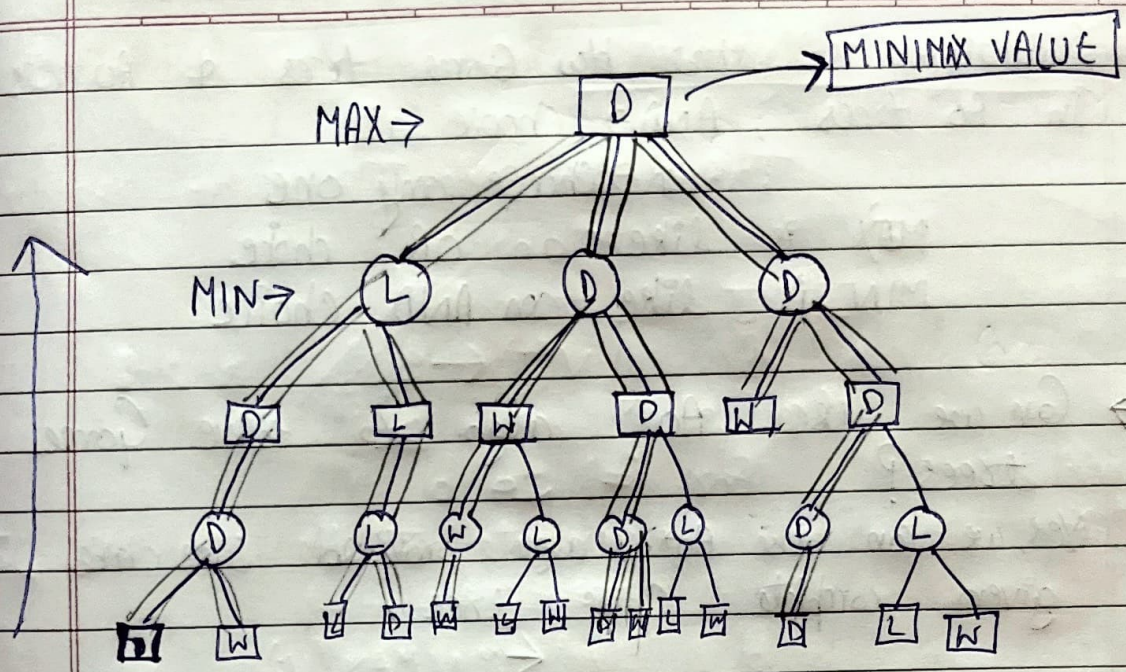
We are interested in Board Games:-
↳ Two player
↳ Zero Sum (Win + loose = 0)
↳ Alternate moves
↳ Deterministic (There is no element of chance)
↳ Complete Information
↳ means a player knows what moves are available to him & other players.

eg → Win = +1
Lose = -1
Draw = 0
∴ Zero Sum
(+1 - 1 = 0)

⇒ Such Games are represented by Game Trees
↳ Layered tree.
One layer → Player 1
2nd layer → Player 2
3rd layer → Player 1
⋮

leaves → Finished Games
labels:
(+1) WIN (-1) LOSE (0) Draw [From [MAX Perspective

Page No.
Date



⇒ First MAX makes move then MIN makes a move then again MAX .....

⇒ MINIMAX Value → Outcome of perfect play.
each player makes perfect choices.

eg→ MINIMAX value of Tic-Tac-Toe game is Draw.

Strategy (for MAX) ⇒ A Subtree
↳ one choice for MAX
↳ all choices for MIN

This strategy gives us set of leaves (nodes)

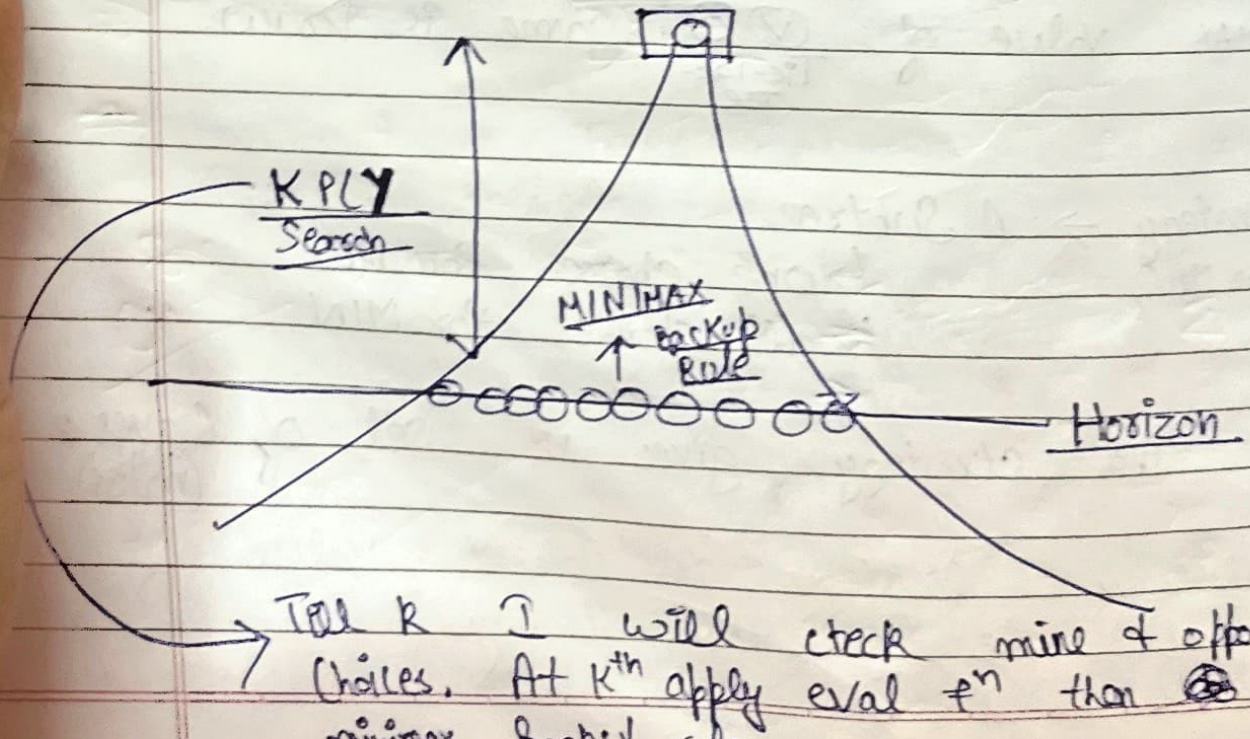
⇒ Value of a strategy is lowest value of leaf node for that strategy.

⇒ Similarity is there b/w Game trees & Ao trees
In Ao trees, AND ⇒ Choose all
OR ⇒ Choose any one
MAX is like an OR choice
MIN is like an AND choice

↳ Can we use Ao* algo. to solve Game trees?
Yes, we can use Ao* algo provided we are given complete Game tree

↳ But, having a complete game tree is not possible always.
Chess has a very long Game tree because there are many-many possibilities in it.
Hence Ao* is not feasible.

↳ Instead in complex games we do limited lookahead. (Search ~~in~~ partial tree)



→ Tell K I will check mine & opponents choices. At K^th apply eval f^n then apply minimax BackUp rule

⇒ Based on lookAhead we decide the best move. We apply evaluation f^n.

↓

Range $[-1000, 1000]$

We will apply eval fn on each of node present on the horizon,
then we can apply same MiniMax backup rule to evaluate value of game.

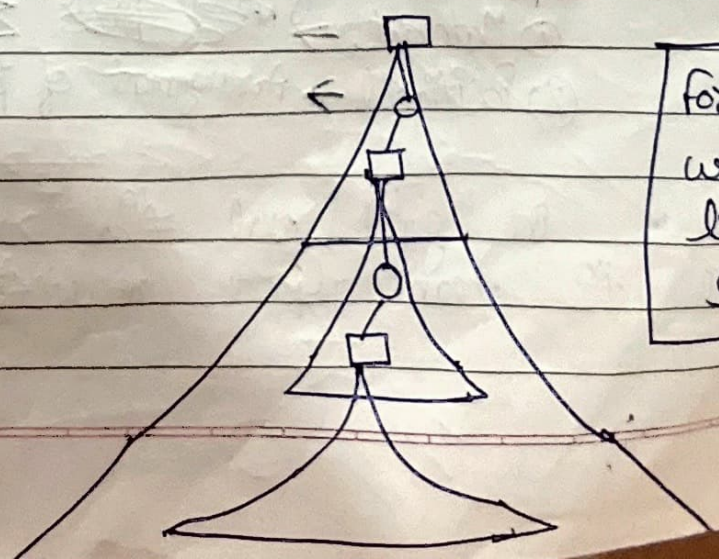
The performance will depend on how good the eval fn is.

↳ We want perfect eval fn.
But not possible ∵ eval fn are like h $f(n)$, they are just an estimation.

Algorithm :-

loop till game ends

- Do K-PLY Search
- Make a move
- Get opponent move



for every move we are doing a limited K-PLY Search

⇒ MINIMAX Algorithm

Takes a node J as an argument

(J)

If J is terminal (means J is a horizon node)

V(J) ← e(J)

else

For i=1 to b

b ⇒ branching factor

generate J_i the ith child of J

val ← MINIMAX (V_j)

If i=1

V(J) ← val

Else if J is max

⇒ MAX TURN

V(J) = Max (V(J), val)

Else

⇒ MIN TURN

V(J) ← Min (V(J), val)

Return V(J)

⇒ This is doing **DBDFS** Search

How to write Evaluation f^n for a Board position

For CHESS ↳ Two Components of eval fn

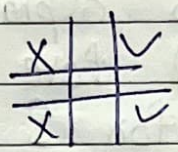
① Material → Σ(value of pieces)

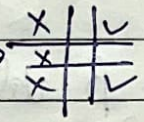
② Positional → Arrangement of pieces.

- Secret is in good eval f^n.
  Good eval f^n means Good results, less search.

⇒ Do we really have to look to partial tree & then make a move?

⇒ Say in Game of Tic-Tac-Toe current position is :-



We do K-Ply Search & if we conside this children →  then no need to look at other children. ∴ this is a winning position

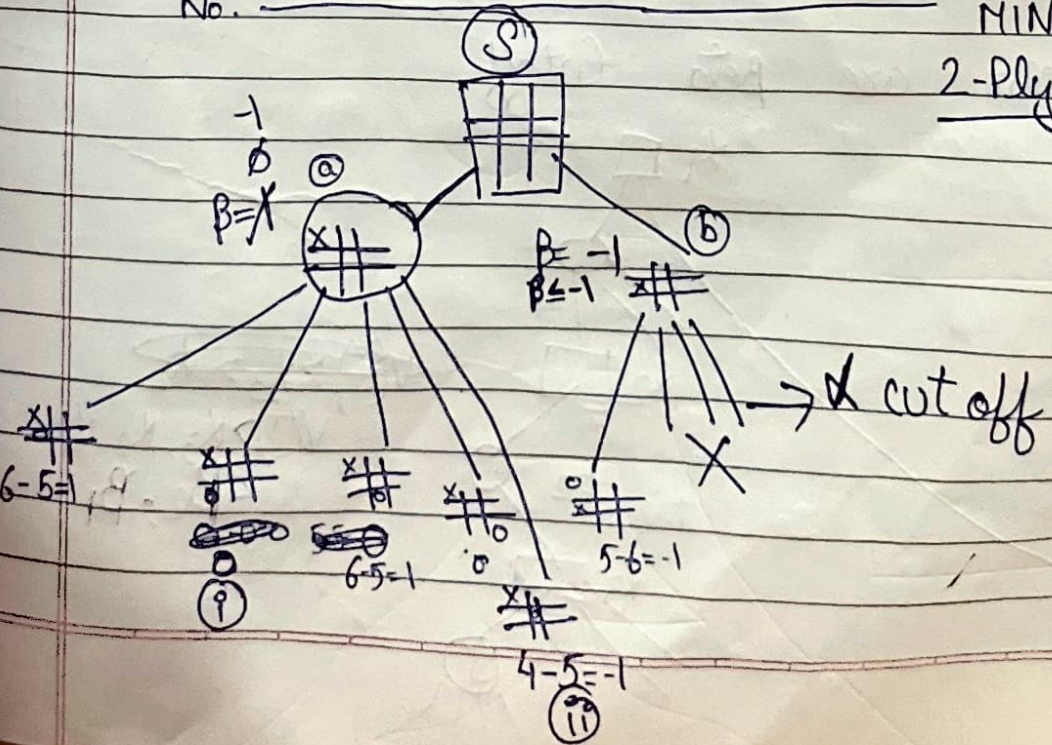
Alpha-Beta Algorithm

Max → Alpha nodes → Stores alpha values → Lower Bound
Min → Beta nodes → Stores Beta value → Upper Bound

⇒ Example of Tic-Tac-Toe Game

e(I) = No. of rows, Col^n, diag^n available to MAX — No. of MIN

2-Ply Search



U.B for ⓐ becomes 1 ∵ min → Beta value → U.B
then 0 after (i)
then -1 after (ii)

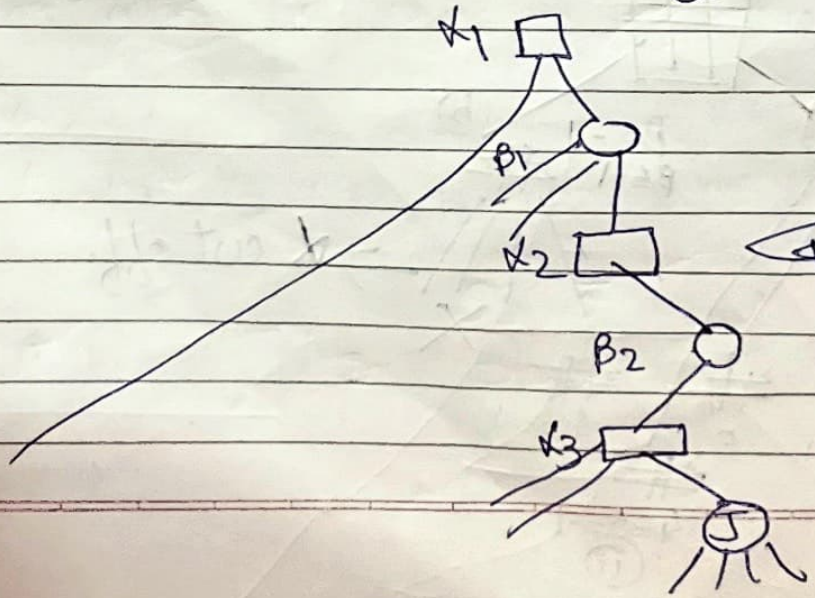
Beta are Supplieres to alpha nodes
Now, our (S) will have value ≥ -1
∴ α for S ≥ -1

Now, (b) β value = -1
& β is U.B
∴ value ≤ -1 possible

∵ In S we want α ≥ -1
∴ PRUNE (b)
We don't go to its other child

α cutoff appears in β nodes which are child of that α node.
happens when β has value lower than what α already has.

Similar are Beta Cutoff



V(J) > α3, α2, α1
< β2, β1

We will evaluate node J only if

$$\boxed{\alpha} < V(J) < \boxed{\beta}$$

where $\alpha = \max(\alpha_1, \alpha_2 \cdots)$ [all α ancestors of a node]

& $\beta = \min(\beta_1, \beta_2 \cdots)$ [all β ancestors of a node]

↳ α & β are bounds b/w which algo. should search otherwise PRUNE that subtree.

Algorithm →

Parameters

Initially ⇒ α = -large & β = +large

Alphabeta (J, α, β)

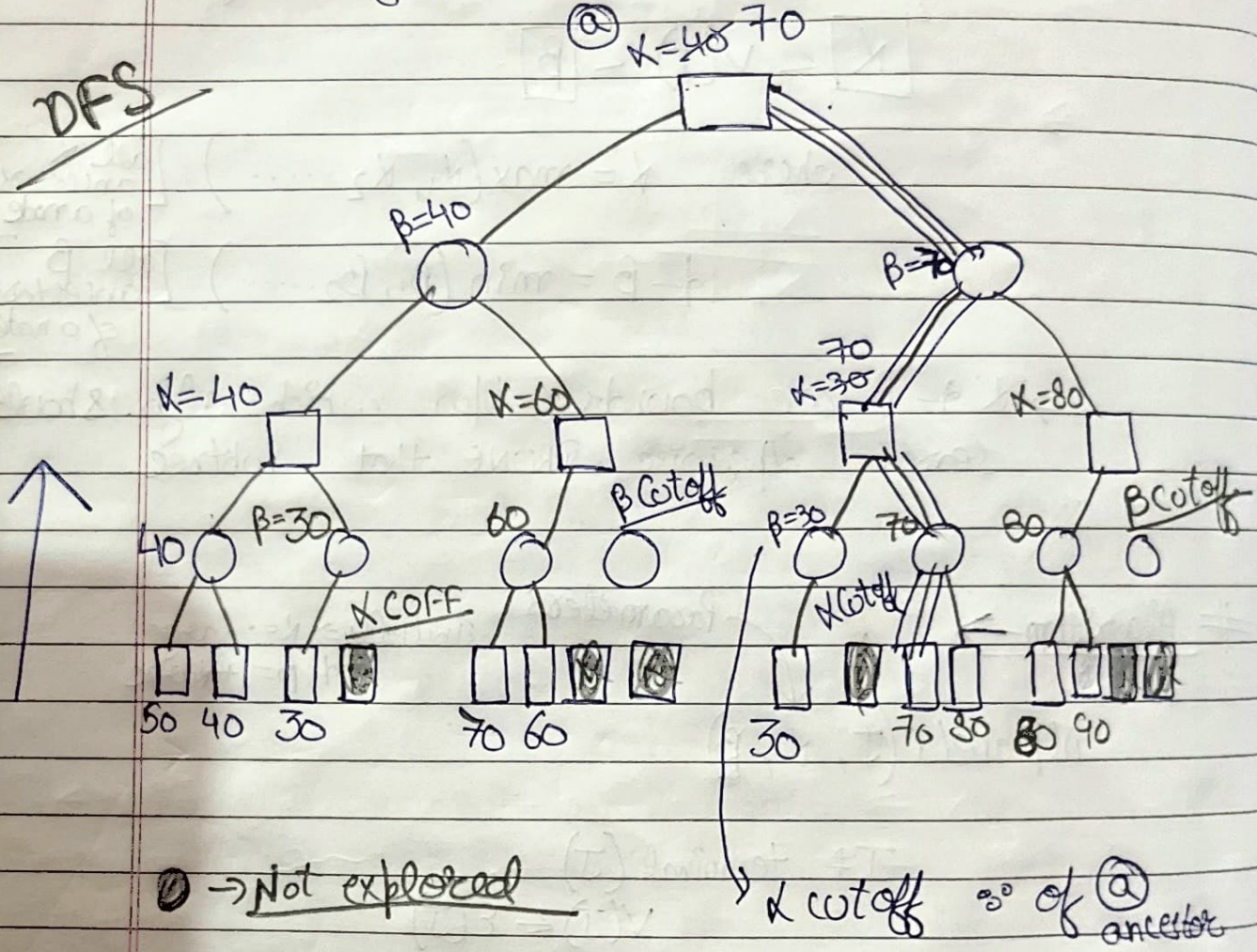
If terminal (J)
    V(J) ← E(J)
else
    for i ← 1 to b
        if J is MAX Node
            $\alpha \leftarrow \max(\alpha, \text{Alphabeta}(J_i, \alpha, \beta))$
            If $\alpha \geq \beta$
                RETURN β
            If i = b
                RETURN α
        ELSE J is MIN
            $\beta \leftarrow \min(\beta, \text{Alphabeta}(J_i, \alpha, \beta))$
            If $\alpha > \beta$
                RETURN α
            If i == b
                RETURN β

RETURN V(J)

β Cutoff

α Cutoff

⇒ Example of B.S.T ⇒ 4 Ply deep



● → Not explored

α cutoff ∵ of @ ancestor

If Best Value in LHS then No of Cutoff is more

⇒ This algo is blind, has no dirn.

We want an algo that moves towards what it thinks is a good soln. (heuristics)

SSS*

⇒ SSS* searches in the space of strategies.