Reading 2

Video on Constraint Satisfaction

- Constraint satisfaction are used in AI when you need to make a bunch of decisions in parallel and you can model those decisions using variables.

- A set of variables : $X_1, X_2, X_3, \ldots, X_n$
- Each variable X_i has a non-empty domain of possible values, e.g. $X_1 = \{1,2,3,4\}$
- A set of constraints, eg. $X_1 \neq 3$ (unary), $X_3 > X_4$ (binary)
- Solution : assign a value to each variable such that none of the constraints are broken.
- Simplest Algorithm : backtracking

Example Problem

$A = \{1,2,3\}$ $\quad A > B$

$B = \{1,2,3\}$ $\quad B \neq C$

$C = \{1,2,3\}$ $\quad A \neq C$

1. $A = 1$ fits all constraints so far
2. $B = 1$ breaks the constraint

Now Backtrack and change last decision made

$A = 1$

$B = 2$ breaks the constraint

Backtrack Again

$A = 1$

$B = 3$ breaks the constraint → Backtrack on A now since entire domain of B already tried.

$A = 2$

$B = 1$

$C = 1$ breaks the constraint

Backtrack last decision

$A = 2$

$B = 1$

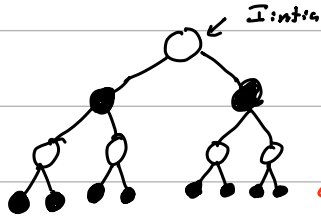
$C = 2$ breaks constraint → Backtrack

$A = 2$
$B = 1$
$C = 3$ solution

Video: Alpha Beta Pruning

- Alpha: best already explored option along path to the root for maximizer
- Beta: best already explored option along path to the root for minimizer

Video: Algorithms explained – minimax and alpha-beta pruning

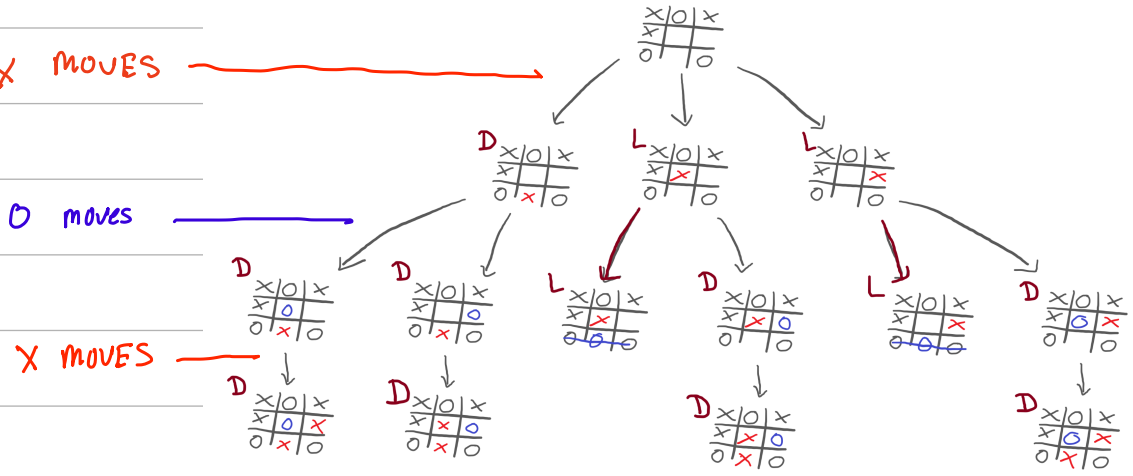


← Static evaluation on these game states. (evaluating how good the position is for one side without making any more moves)

Blog Post: Minimax algorithm and alpha-beta pruning

- Alpha-Beta Pruning is an optimization over the minimax algorithm.
- Tic Tac Toe Example!
  - I want to play what is best for me; I want to maximize my score, because I want to win.
  - You want to play what is worst for me; you want to minimize my score, because you want me to lose.

Game Tree:

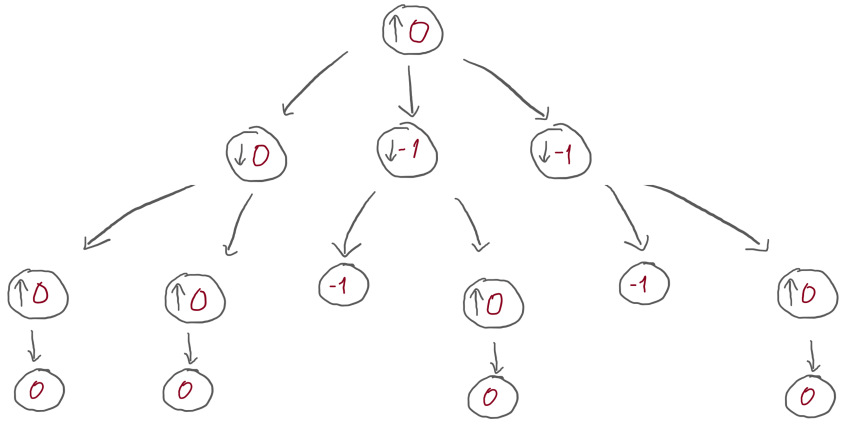


- Now we can see from the top that if X goes:

  left: the game will draw

  any of the 2 right branches: X will lose

Tree Structure Abstraction



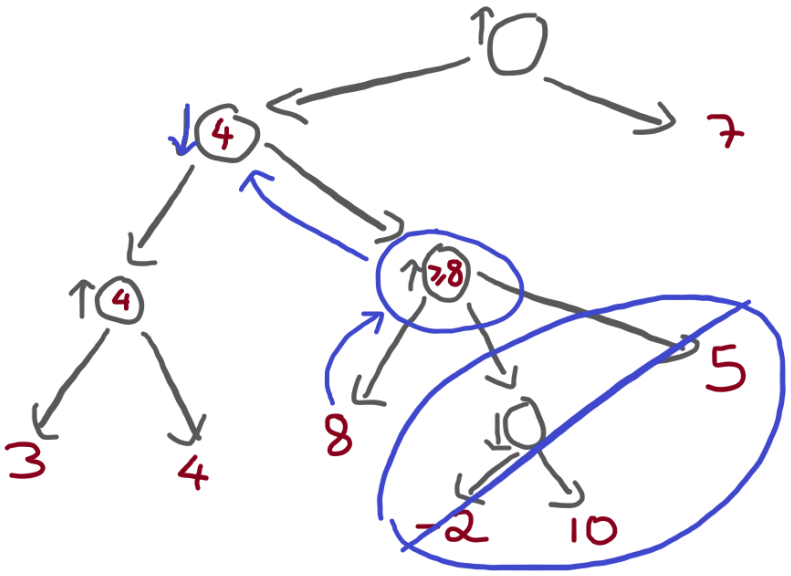
- up arrows indicate the maximizer makes a choice at that node
  - ↳ i.e. I pick the move best for me
- down arrows indicate the minimizer makes the move at that node
  - ↳ i.e. you make the move worst for me
- 0 indicates draw
- -1 indicates loss (for me)

• Alpha-beta Pruning Rationale

- In games like chess, where each player can make many choices at a node, and the game lasts much longer than 2 or 3 moves, the game tree can get very wide and deep.
  - ↳ This means it takes a lot of resources to traverse the whole tree.

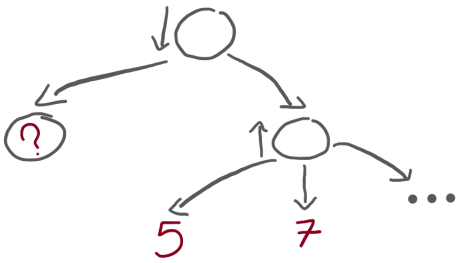
- α-β pruning optimizes by not traversing irrelevant portions of the tree.

- Essence of $\alpha - \beta$ pruning diagram:



- Exercise!

  Given this tree:



- Question! what values can [?] have so that:
  - the algorithm keeps going after it finds the 5; and
  - the algorithm stops after it finds the 7.

- Answer $5 < [?] \leq 7$

- If a minimizing node n has a value of v or lower, and the maximizing node above knows a path with a value greater than or equal to v, then we can stop.

  ↳ Therefore when we are inside a minimising node, we need to know what's the value of the highest node that the maximising node above has seen. This is called <u>alpha (α)</u>.

  ↳ If at any point $v \leq a$ we can stop traversing from that branch on

  ↳ In contrast we use <u>beta (β)</u> to keep track of the lowest option that a mininiser node has found so far.

  ↳ When a maximiser node finds its value above β, we can stop traversing any subsequent branches.

- We can prune a branch when $\alpha \geq \beta$.

- When implementing, default values are:
  - $\alpha = -\infty$
  - $\beta = +\infty$

- When implementing:
  - when we are the maximiser player, α keeps updating to get higher values (that come from val). Then we check if val $\geq$ beta
  - When we are the minimizing player, β keeps getting updated to get lower values (that come from val). Then we check if val $\leq$ alpha.