Artificial Intelligence

Adversarial Search

Andrea Torsello

Games Vs. Planning Problems

In many problems you’re are pitted against an opponent

- certain operators are beyond your control: you cannot control your opponent’s moves

You cannot search the entire space for an optimal path

- your opponent may make a move which makes any path you find obsolete

You need a strategy that leads to a winning position **regardless** of how your opponent plays

Search strategies must take into account the conflicting goals of the agents

“Unpredictable” opponent: ⇒ solution is a strategy

- Agents goals are in conflict: adversarial search (game)
- Specify a move for every possible opponent reply

Time limits: unlikely to find optimal move, must approximate

Why Study Games in AI?

- problems are formalized
- real world knowledge (common sense knowledge) is not too important
- rules are fixed
- adversary modeling is of general importance (e.g., in economic situations, in military operations, ...)
  - opponent introduces uncertainty
  - programs must deal with the contingency problem
- complexity of games??
  - number of nodes in a search tree (e.g., 1040 legal positions in chess)
  - specification is simple (no missing information, well-defined problem)

Types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information		bridge, poker, scrabble, nuclear war

We restrict our analysis to a very specific set of games:

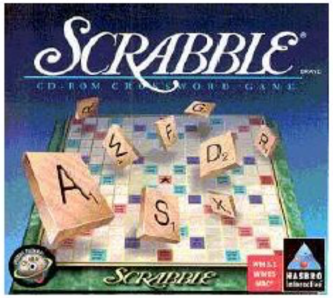
2-player zero-sum discrete finite deterministic games of perfect information

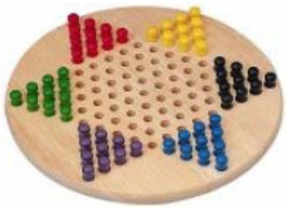
2-player zero-sum discrete finite deterministic games of perfect information

What does it means?

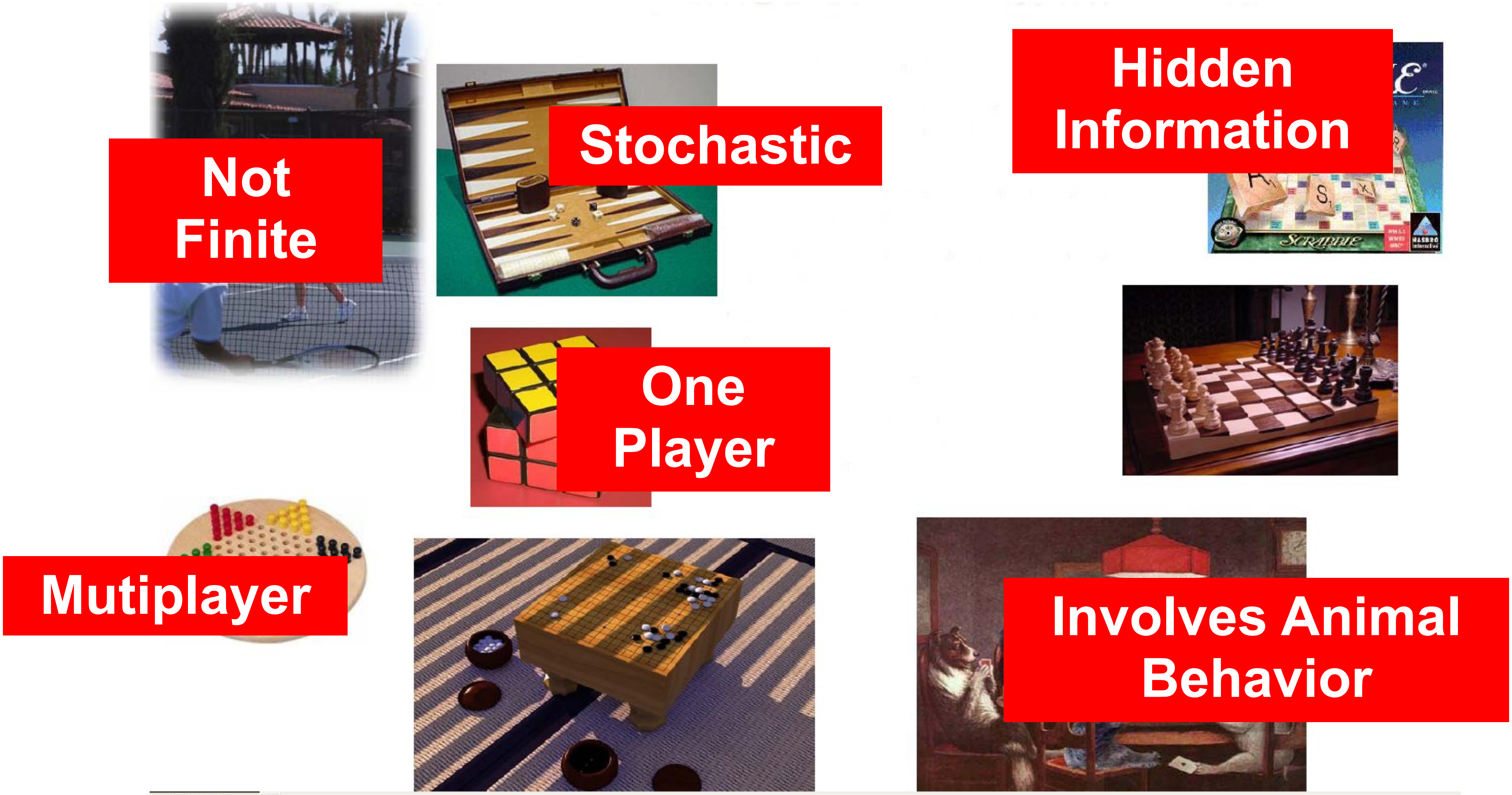
- Two player: :-)
- Zero-sum: In any outcome of any game, Player A's gains equal player B's losses.
- Discrete: All game states and decisions are discrete values.
- Finite: Only a finite number of states and decisions.
- Deterministic: No chance (no die rolls).
- Perfect information: Both players can see the state, and each decision is made sequentially (no simultaneous moves).

Types of games





Types of games



Game Tree Search

Initial state: initial board position and player

Operators: one for each legal move

Goal states: winning board positions

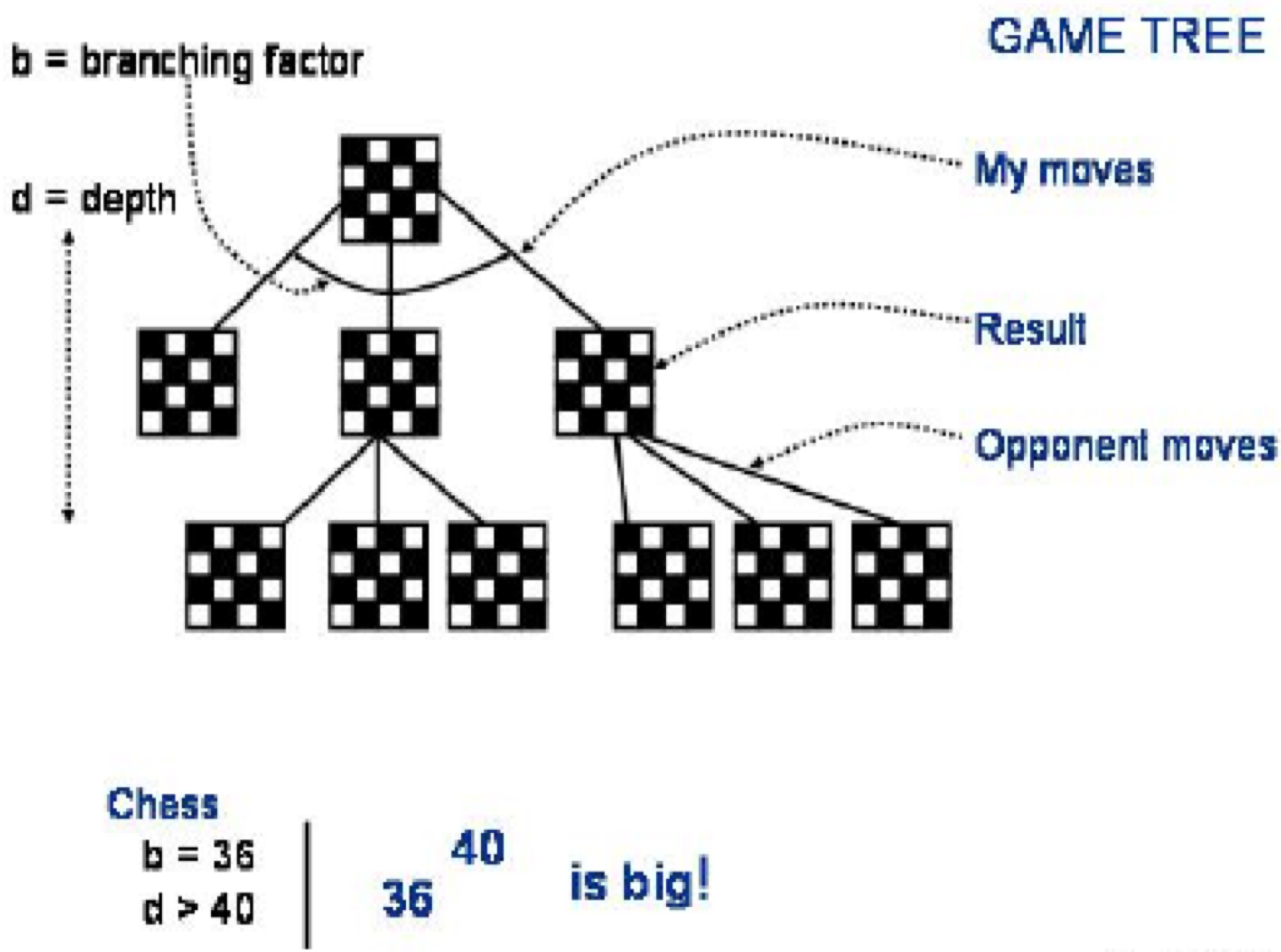
Scoring Function: assigns numeric value to states

Game tree: encodes all possible games

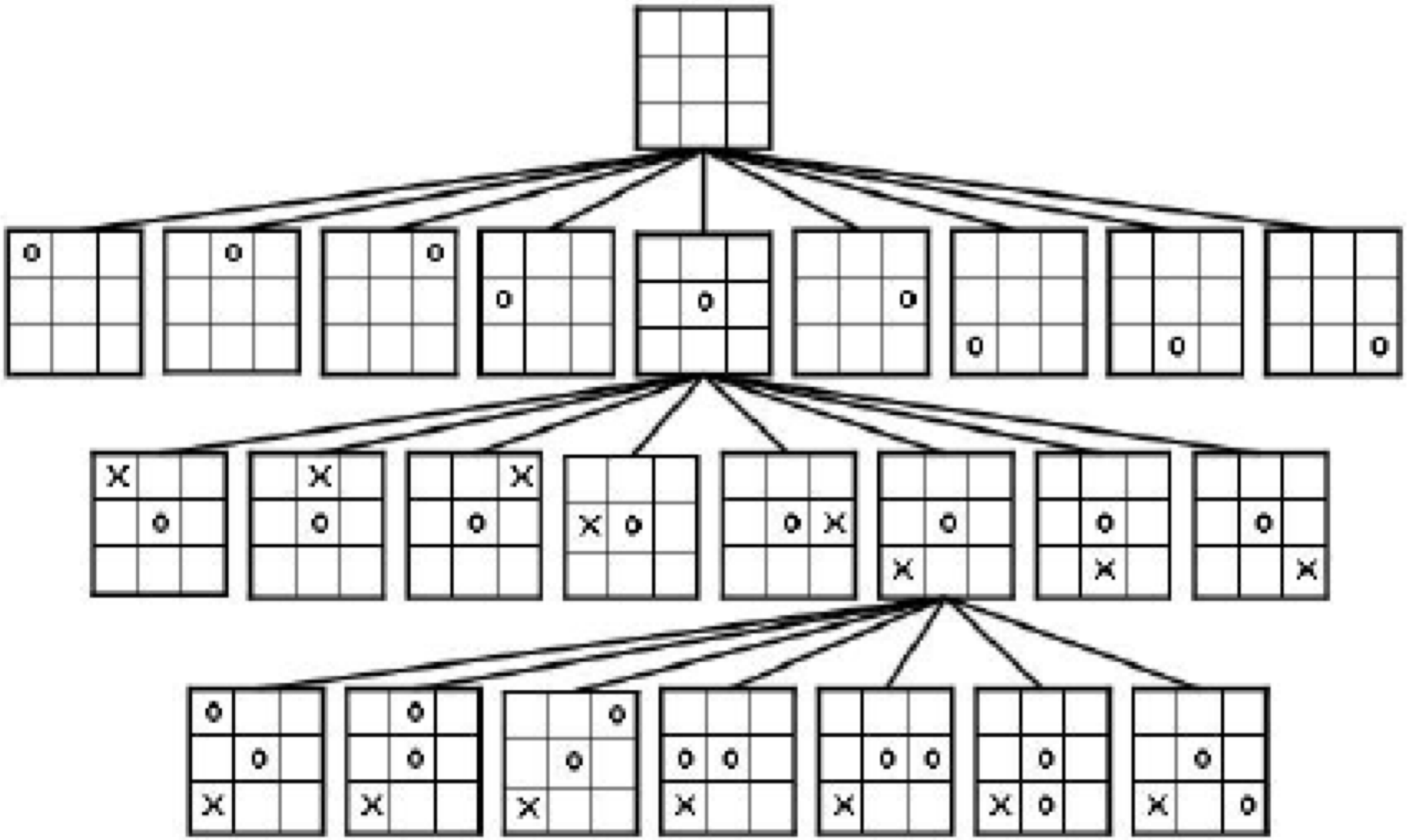
We are not looking for a path, only the next move to make

Our best move depends on what the other player does

Move generation



(Partial) Game Tree for Tic-Tac-Toe



Minimax Criterion

Assume game tree of uniform depth (to simplify matters)

- Generate entire game tree
- Apply utility function to each terminal state
- To determine utility of nodes at any level, if Min's turn to play it will choose child with minimum utility, otherwise Max will choose child with maximum utility
- Continue backing up values from leaf to root, one level at a time

Maximizes utility under assumption that opponent will play perfectly to minimize it (assuming also opponent has same evaluation function)

Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

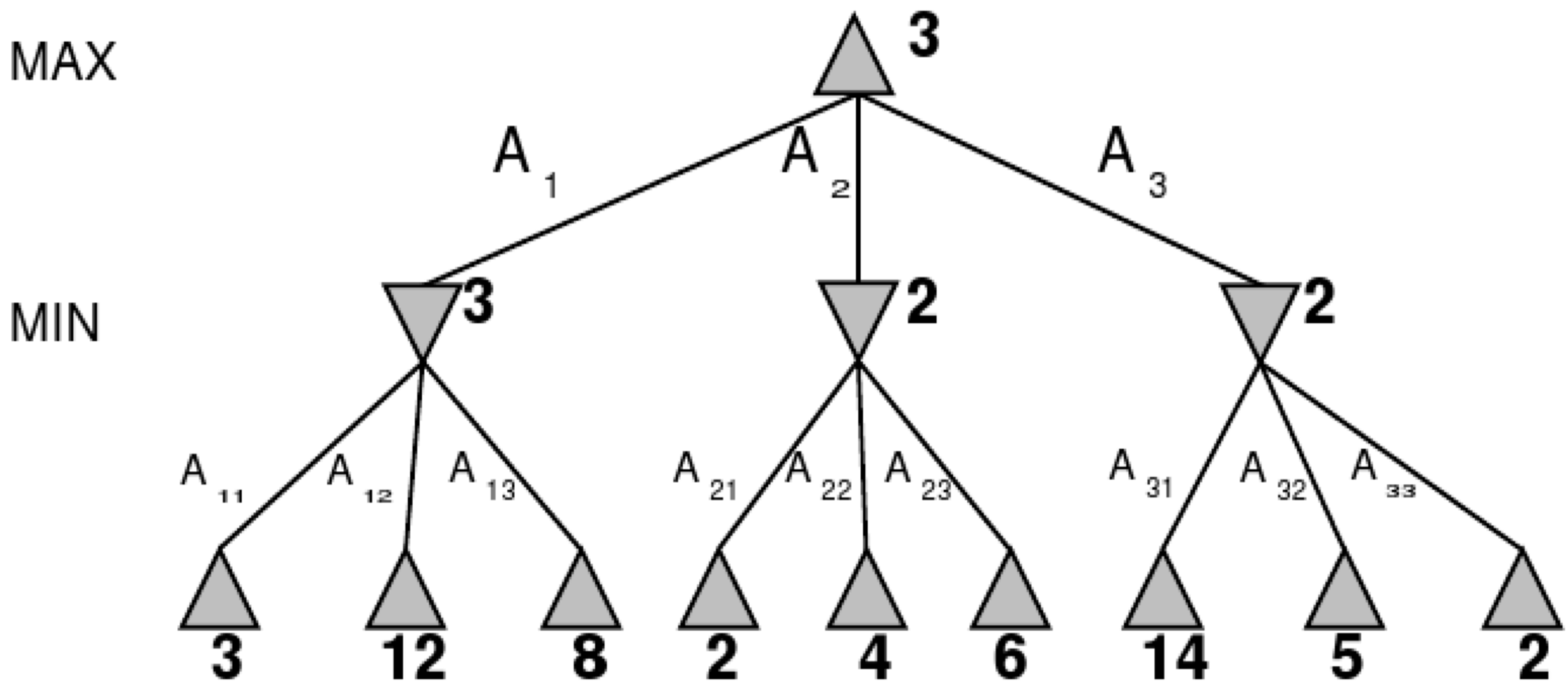
Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value

- best achievable payoff against best play

E.g., 2-ply game:



Properties of Minimax

Complete: Yes, if tree is finite (chess has specific rules for this)

Optimal: Yes, against an optimal opponent.

- Otherwise??

Time complexity: $O(b^m)$

Space complexity: $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
$\Rightarrow$ exact solution completely infeasible

Resource Limits

Suppose we have 100 seconds, explore 104 nodes/second
⇒ 10^6 nodes per move

Standard approach:

- **cutoff test**
  e.g., depth limit
- **evaluation function**
  = estimated desirability of position and explore only (hopeful) nodes with certain values

MINIMAX-CUTOFF is identical to MINIMAX except

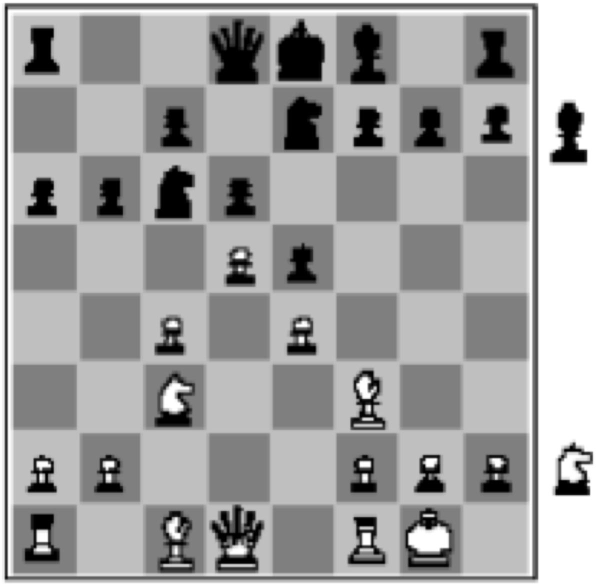
- T ERMINAL-TEST is replaced by C UTOFF-TEST
- U TILITY is replaced by EVAL

Search depth in chess:

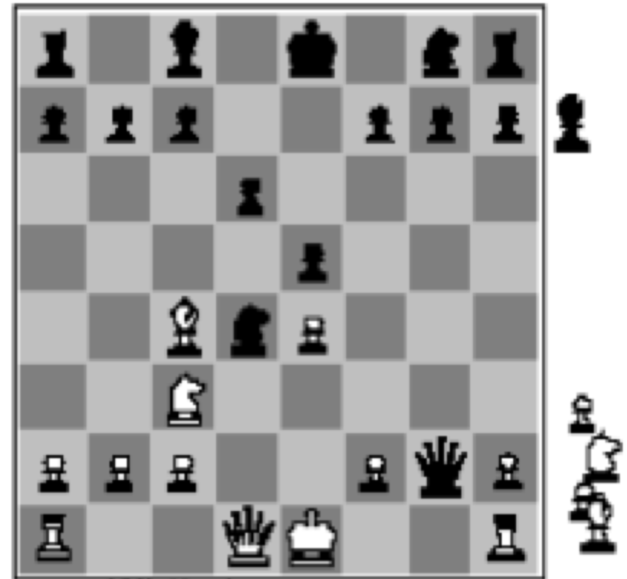
4-ply ≈ human novice

8-ply ≈ typical PC, human master

Evaluation functions



Black to move
White slightly better



White to move
Black winning

For chess, typically linear weighted sum of features

Eval(s) = w1 f1(s) + w2 f2(s) + … + wn fn(s)

e.g., weight of figures on the board:

- w1 = 9 with
- f1(s) = (number of white queens) – (number of black queens), etc.

Other features which could be taken into account: number of threats, good structure of pawns, measure of safety of the king.

α-β Pruning

The problem with minimax algorithm search is that the number of game states it has to examine is exponential in the number of moves:

α-β proposes to compute the correct minimax algorithm decision without looking at every node in the game tree.

PRUNING!

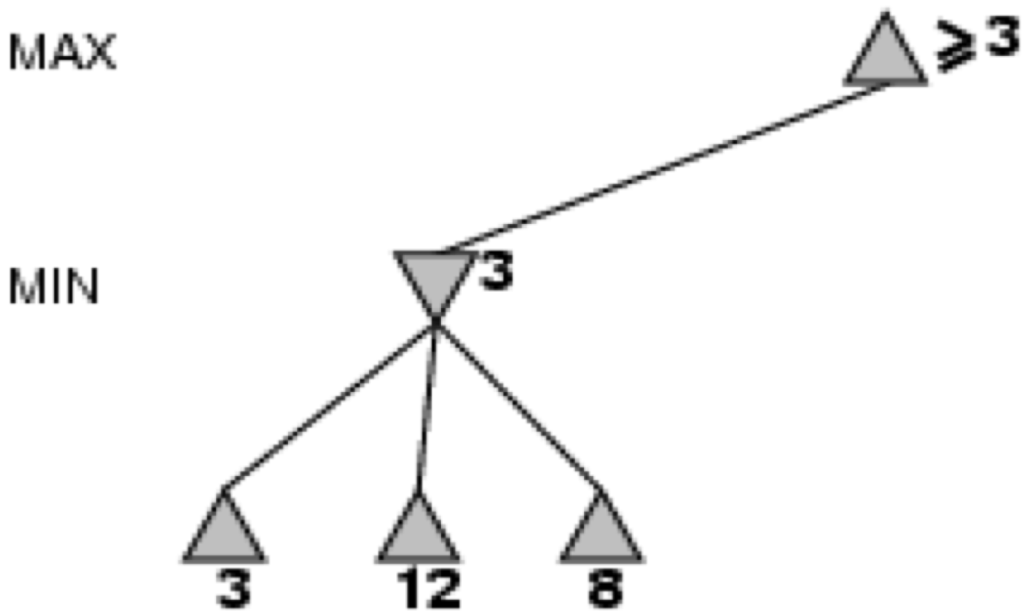
α-β Pruning

The problem with minimax algorithm search is that the number of game states it has to examine is exponential in the number of moves:

α-β proposes to compute the correct minimax algorithm decision without looking at every node in the game tree.

PRUNING!

EXAMPLE



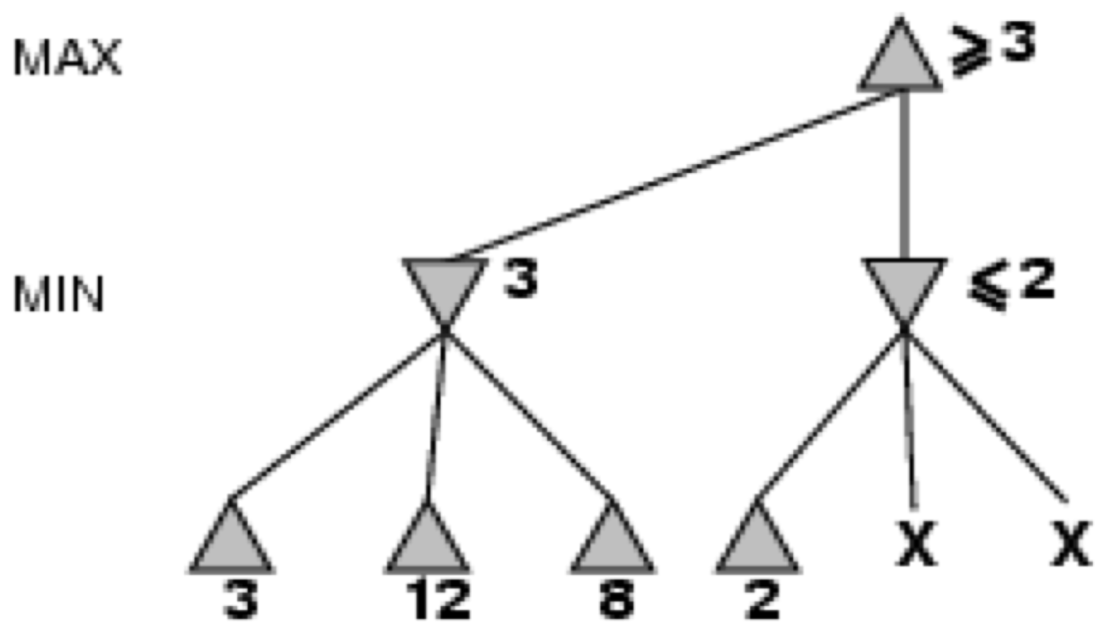
α-β Pruning

The problem with minimax algorithm search is that the number of game states it has to examine is exponential in the number of moves:

α-β proposes to compute the correct minimax algorithm decision without looking at every node in the game tree.

PRUNING!

EXAMPLE



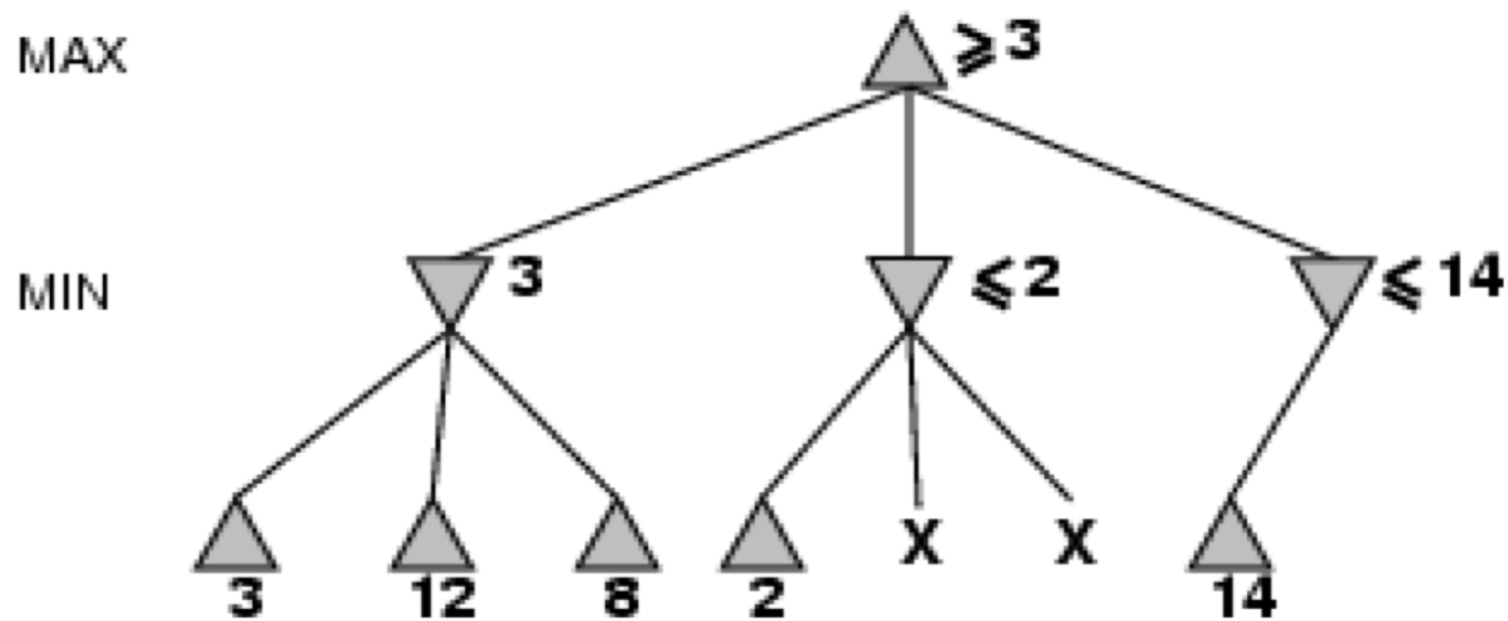
α-β Pruning

The problem with minimax algorithm search is that the number of game states it has to examine is exponential in the number of moves:

α-β proposes to compute the correct minimax algorithm decision without looking at every node in the game tree.

PRUNING!

EXAMPLE



We see: possibility to prune depends on the order of processing the successors!

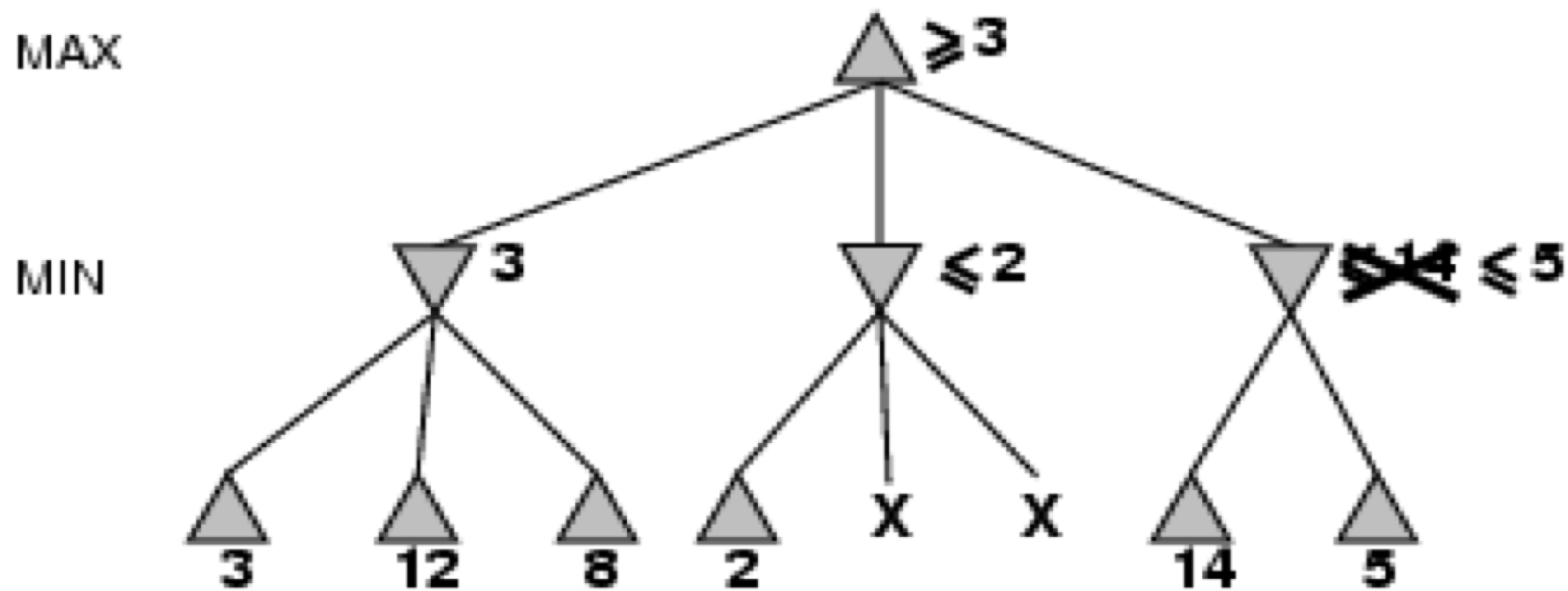
α-β **Pruning**

The problem with minimax algorithm search is that the number of game states it has to examine is exponential in the number of moves:

α-β proposes to compute the correct minimax algorithm decision without looking at every node in the game tree.

PRUNING!

EXAMPLE



We see: possibility to prune depends on the order of processing the successors!

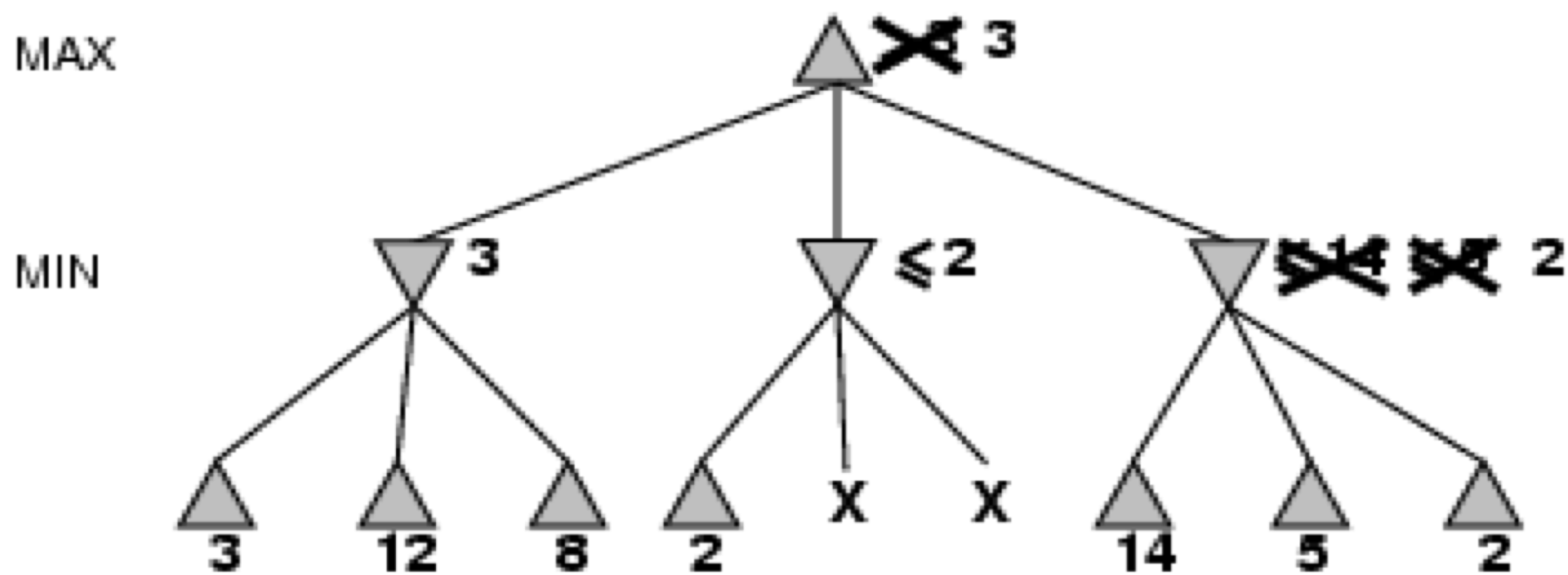
α-β **Pruning**

The problem with minimax algorithm search is that the number of game states it has to examine is exponential in the number of moves:

α-β proposes to compute the correct minimax algorithm decision without looking at every node in the game tree.

PRUNING!

EXAMPLE



We see: possibility to prune depends on the order of processing the successors!

Properties of α-β Pruning

Pruning does not affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity = $O(b^{m/2})$

- doubles possible depth of search doable in the same time

A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)

α-β Algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

α-β Algorithm

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

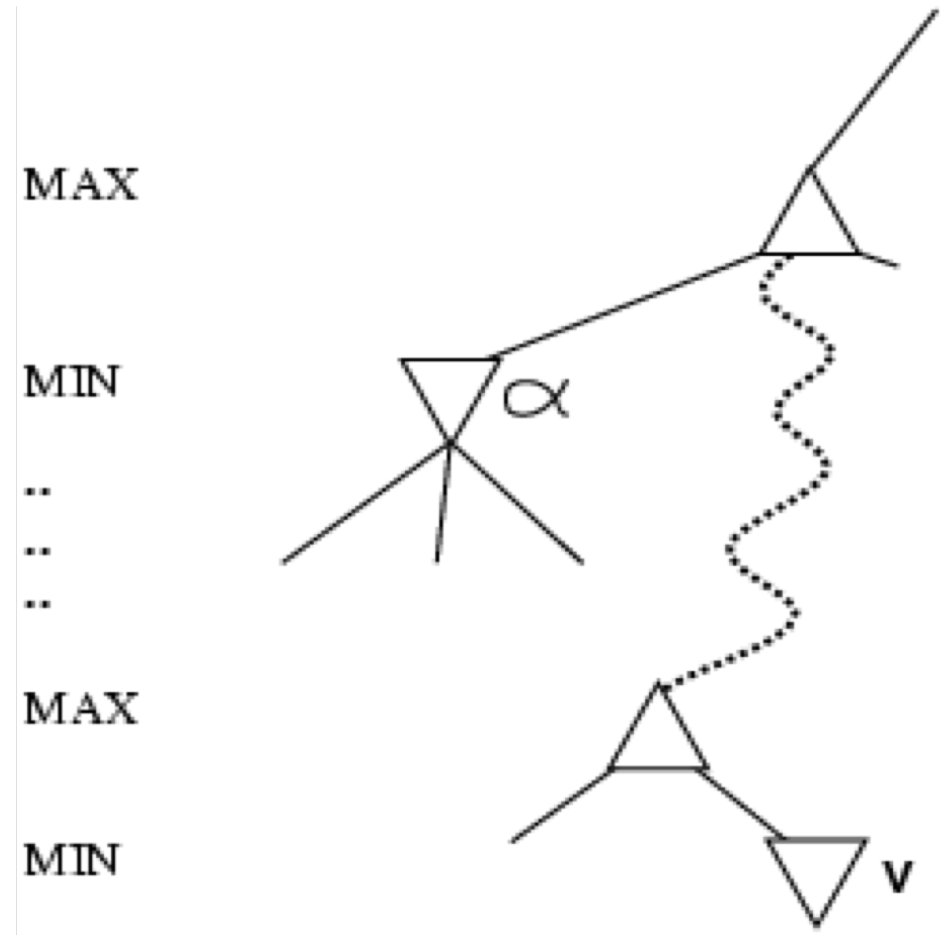
Why is it called of α-β?

α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for max

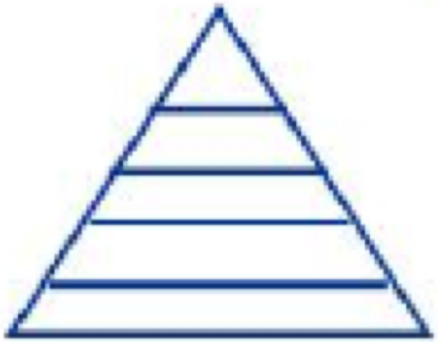
If v is worse than α, max will avoid it

- prune that branch

Define β similarly for min



Practical Matters



Variable branching

Iterative deepening

- order best move from last search first
- use previous backed up value to initialize $[\alpha, \beta]$
- keep track of repeated positions (transposition tables)

Horizon effect

- quiescence
- Pushing the inevitable over search horizon

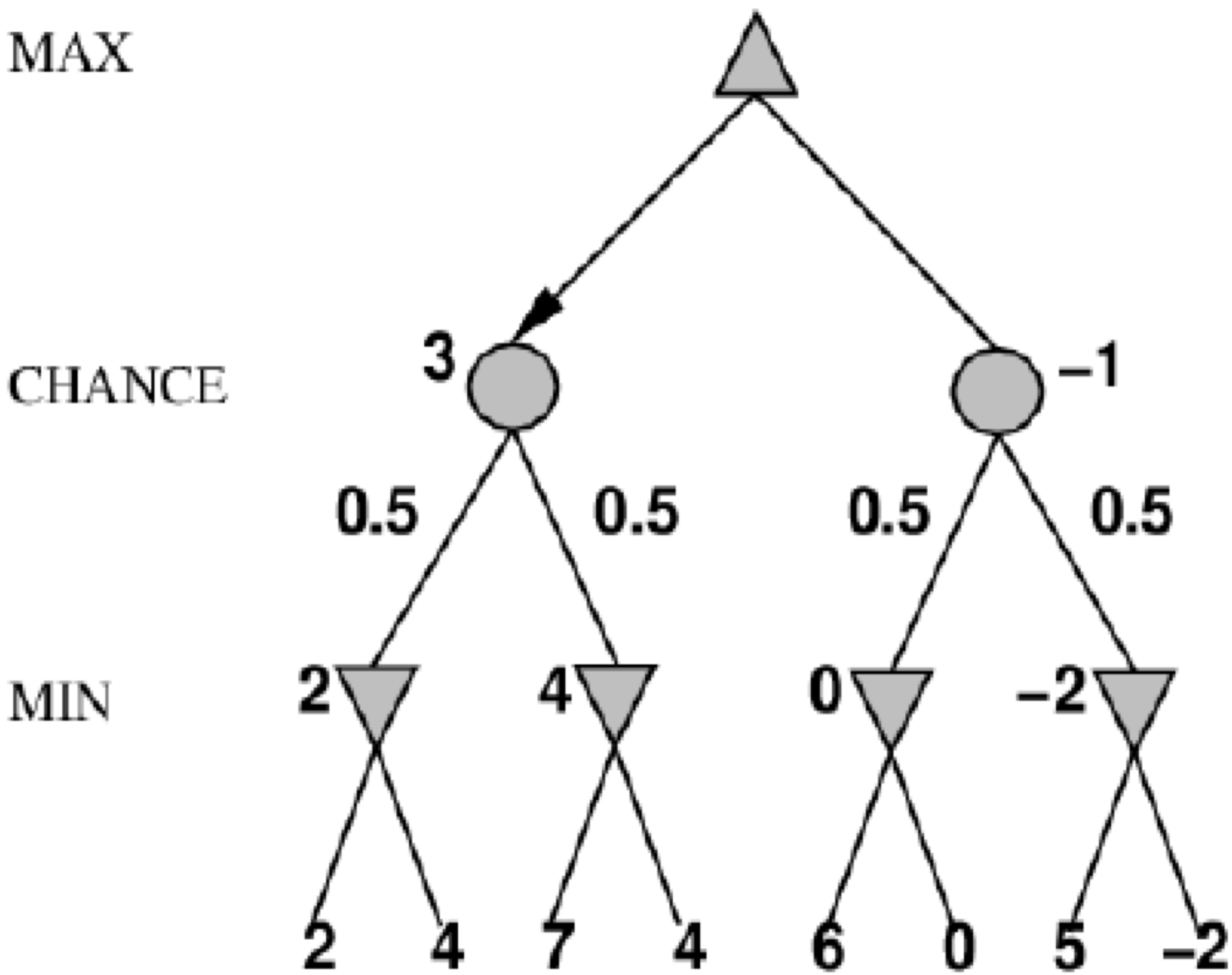
Parallelization

Non-Deterministic Games

E.g. backgammon: dice rolls determine legal moves

We can do Minimax with a extra “chance” layer

Simplified example with coin-flipping instead of dice-rolling



EXPECT-MINIMAX Algorithm

EXPECTIMINIMAX gives perfect play for non-deterministic games

Like MINIMAX, except add chance nodes

- For max node return highest EXPECTIMINIMAX of SUCCESSORS
- For min node return lowest EXPECTIMINIMAX of SUCCESSORS
- For chance node return average of EXPECTIMINIMAX of SUCCESSORS

Here exact values of evaluation function do matter (“probabilities”, “expected gain”, not just order)

α-β pruning possible by taking weighted averages according to probabilities

Games of Imperfect Information

E.g. card games (bridge, hearts, some forms of poker)

- Opponent's initial cards are unknown
- Not quite like non-deterministic games

We can calculate probabilities for each possible deal

- Seems just like one big dice roll at the beginning
- Idea:
  - Compute the minimax value of each action in each deal
  - Then choose action with highest expected value over all deals
  - Special case: an action optimal for all deals, is optimal
- Take weighted average over all possible situations to make decision at the top of the game tree
- Requires a lot of computation. . .