Game Playing

Why do AI researchers study game playing?

1. It’s a good reasoning problem, formal and nontrivial.

2. Direct comparison with humans and other computer programs is easy.

What Kinds of Games?

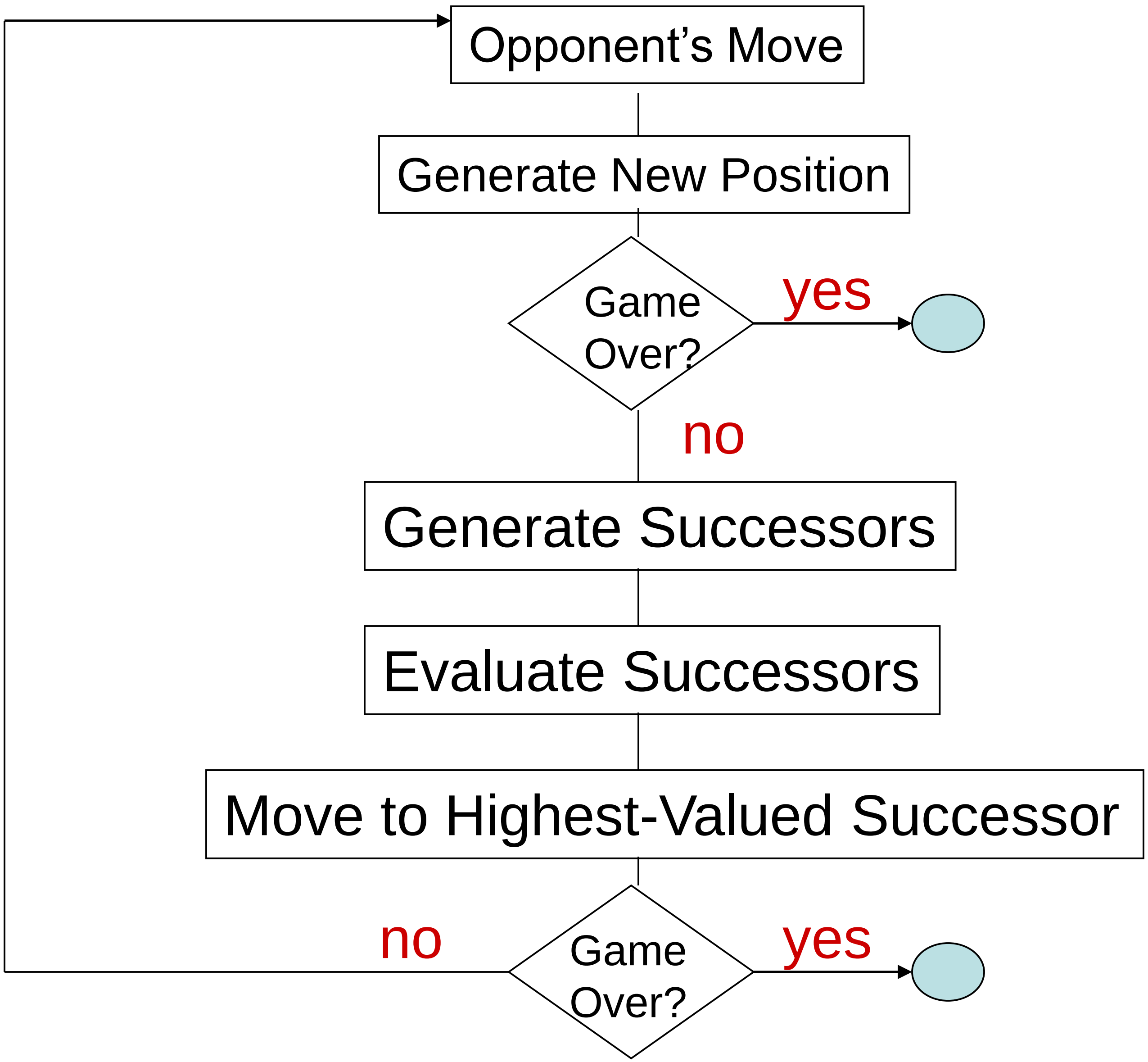
Mainly games of strategy with the following characteristics:

1. Sequence of moves to play
2. Rules that specify possible moves
3. Rules that specify a payment for each move
4. Objective is to maximize your payment

Games vs. Search Problems

- Unpredictable opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

Two-Player Game



Game Tree (2-player, Deterministic, Turns)

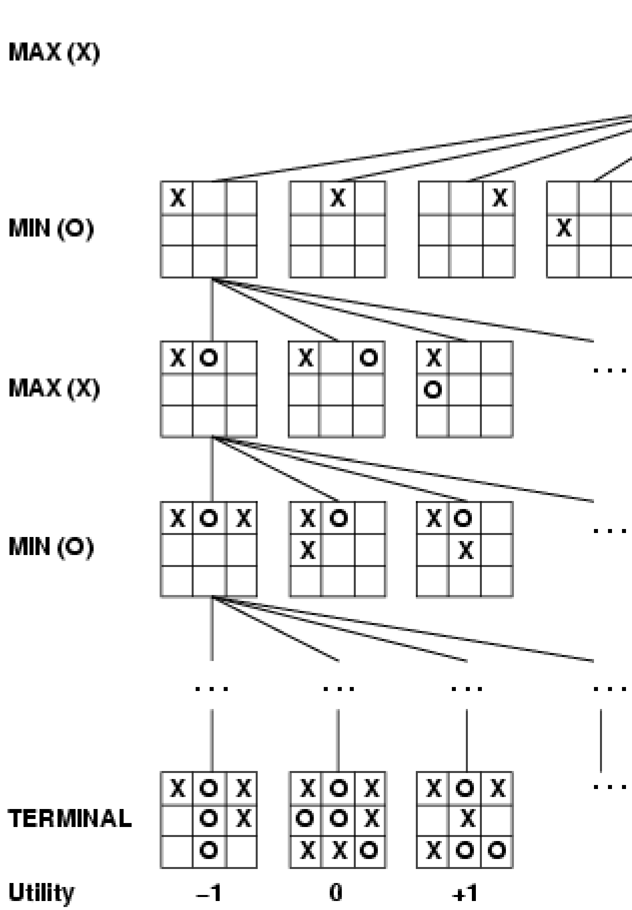
computer's turn

opponent's turn

computer's turn

opponent's turn

leaf nodes are evaluated



The computer is **Max**.
The opponent is **Min**.

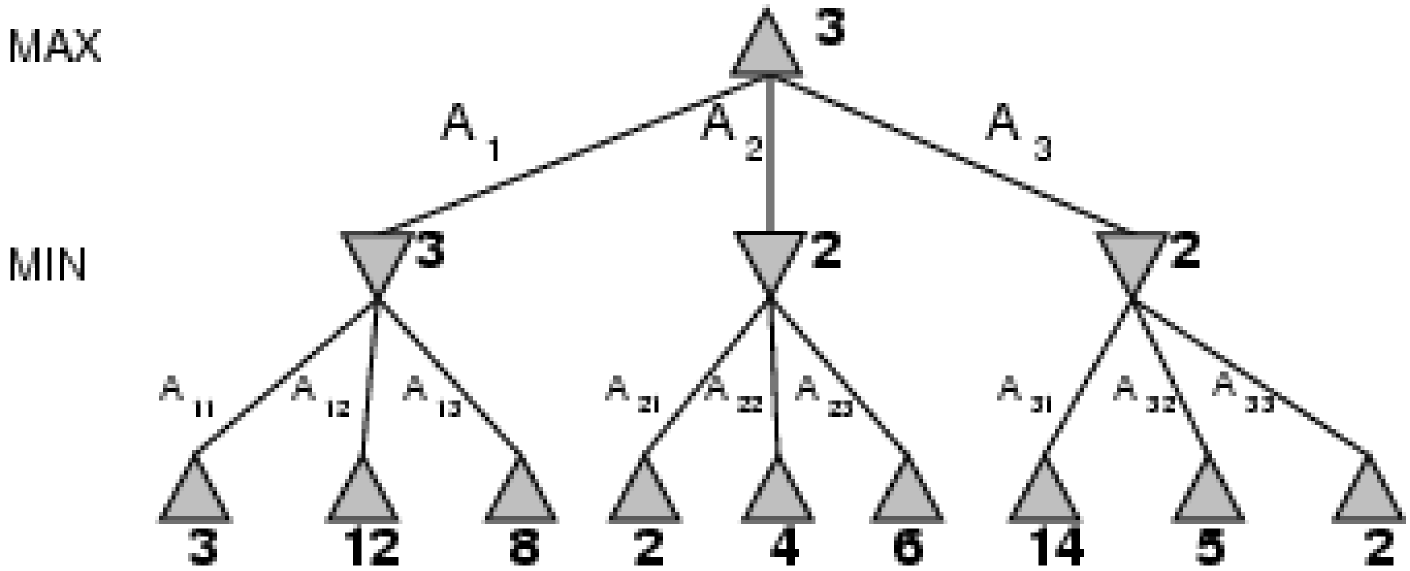
At the leaf nodes, the **utility function** is employed. Big value means good, small is bad.

Mini-Max Terminology

- utility function: the function applied to leaf nodes
- backed-up value
  - of a max-position: the value of its largest successor
  - of a min-position: the value of its smallest successor
- minimax procedure: search down several levels; at the bottom level apply the utility function, back-up values all the way up to the root node, and that node selects the move.

Minimax

- Perfect play for deterministic games
-
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play



Minimax Strategy

- Why do we take the min value every other level of the tree?

- These nodes represent the opponent's choice of move.

- The computer assumes that the human will choose that move that is of least value to the computer.

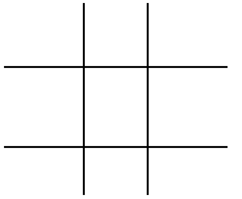
Minimax algorithm

```
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
```

Tic Tac Toe



- Let p be a position in the game
- Define the utility function f(p) by
  - f(p) =
    - largest positive number if p is a win for computer
    - smallest negative number if p is a win for opponent
    - RCDC – RCDO
  - where RCDC is number of rows, columns and diagonals in which computer could still win
  - and RCDO is number of rows, columns and diagonals in which opponent could still win.

Sample Evaluations

- X = Computer; O = Opponent

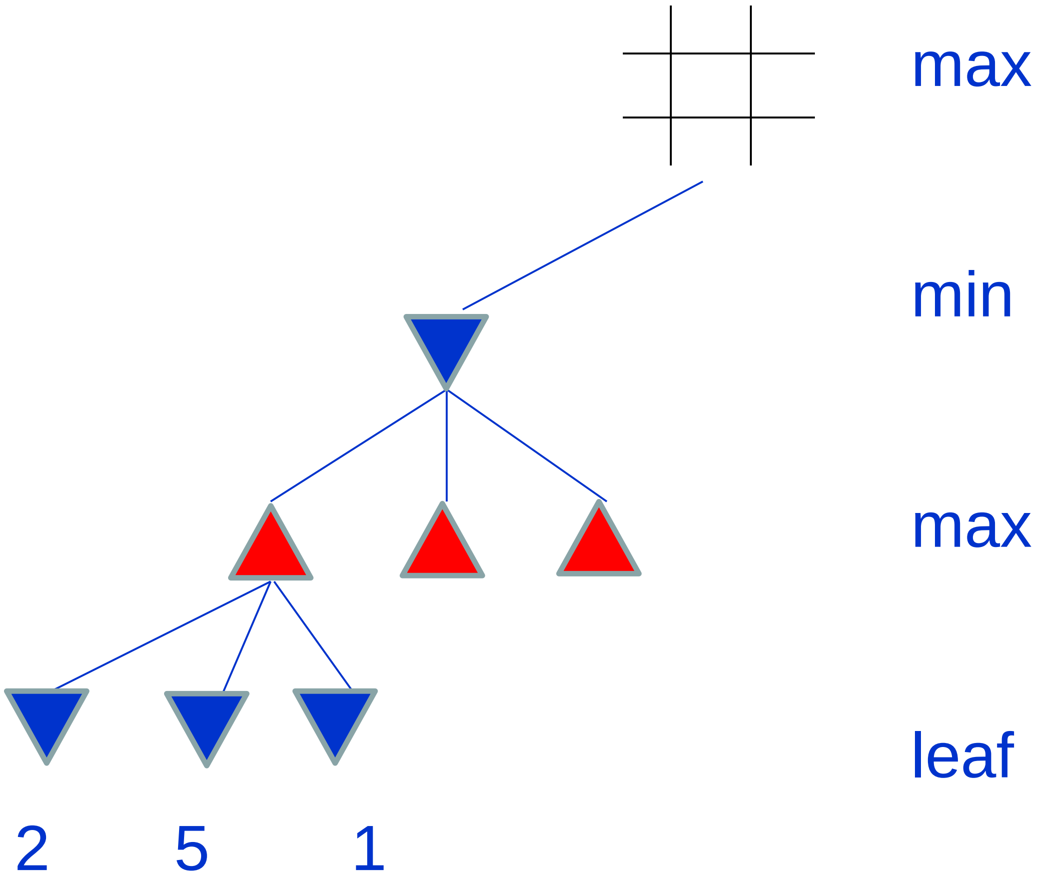
	O	
	X	

O	O	X
X	X	

	X	O
rows		
cols		
diags		

	X	O
rows		
cols		
diags		

Minimax is done depth-first



Properties of Minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (depth-first exploration)
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  → exact solution completely infeasible

  Need to speed it up.

Alpha-Beta Procedure

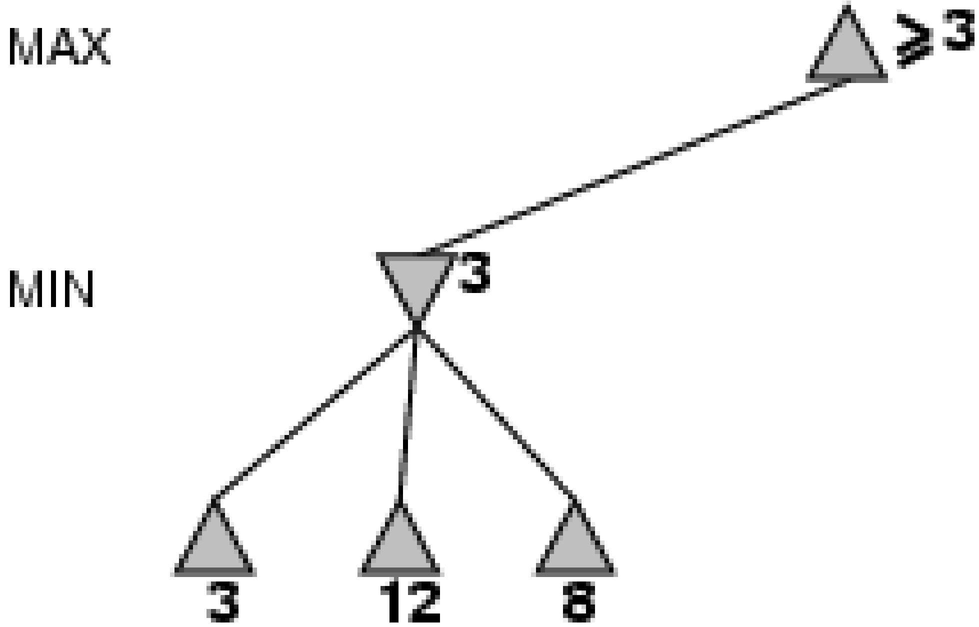
- The alpha-beta procedure can speed up a depth-first minimax search.
- Alpha: a lower bound on the value that a max node may ultimately be assigned

$$v \geq \alpha$$

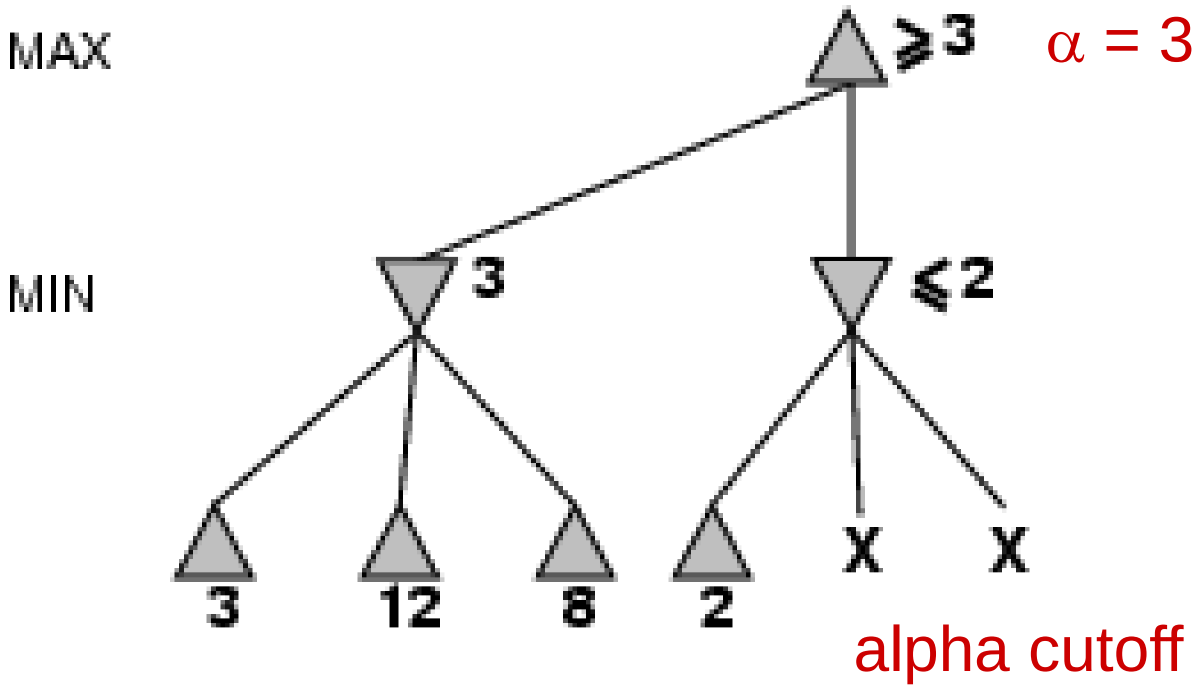
- Beta: an upper bound on the value that a minimizing node may ultimately be assigned

$$v \leq \beta$$

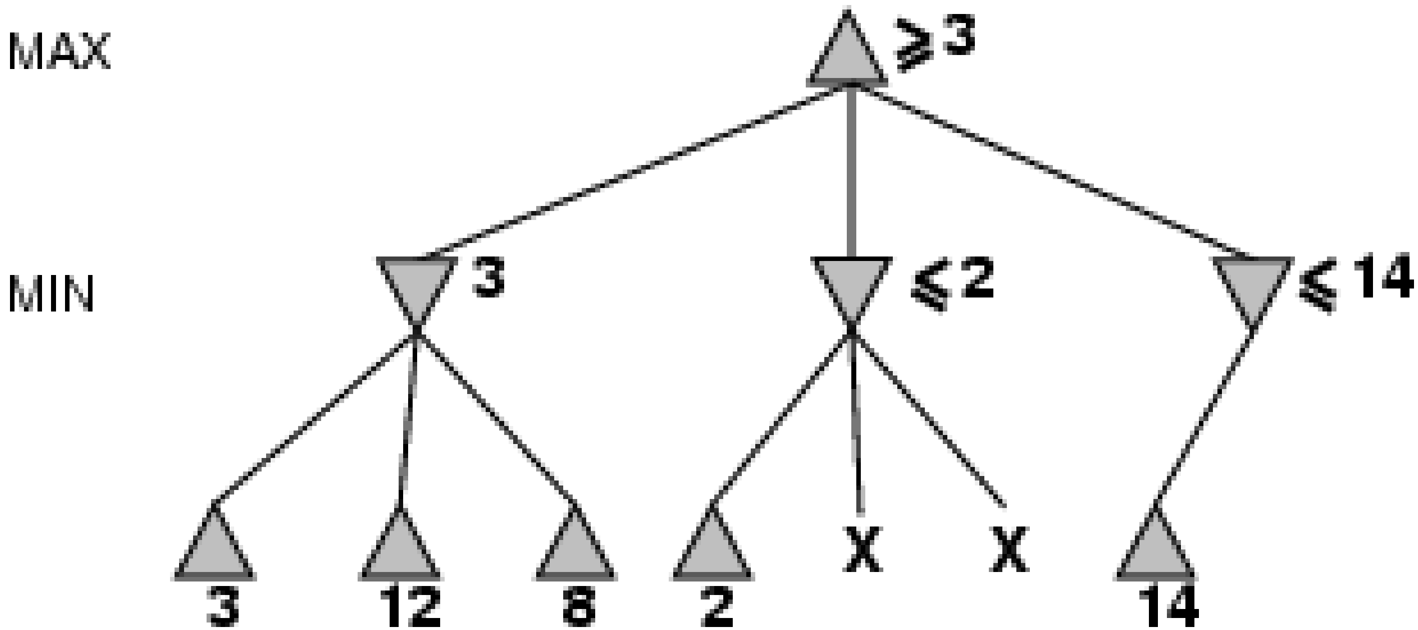
α-β pruning example



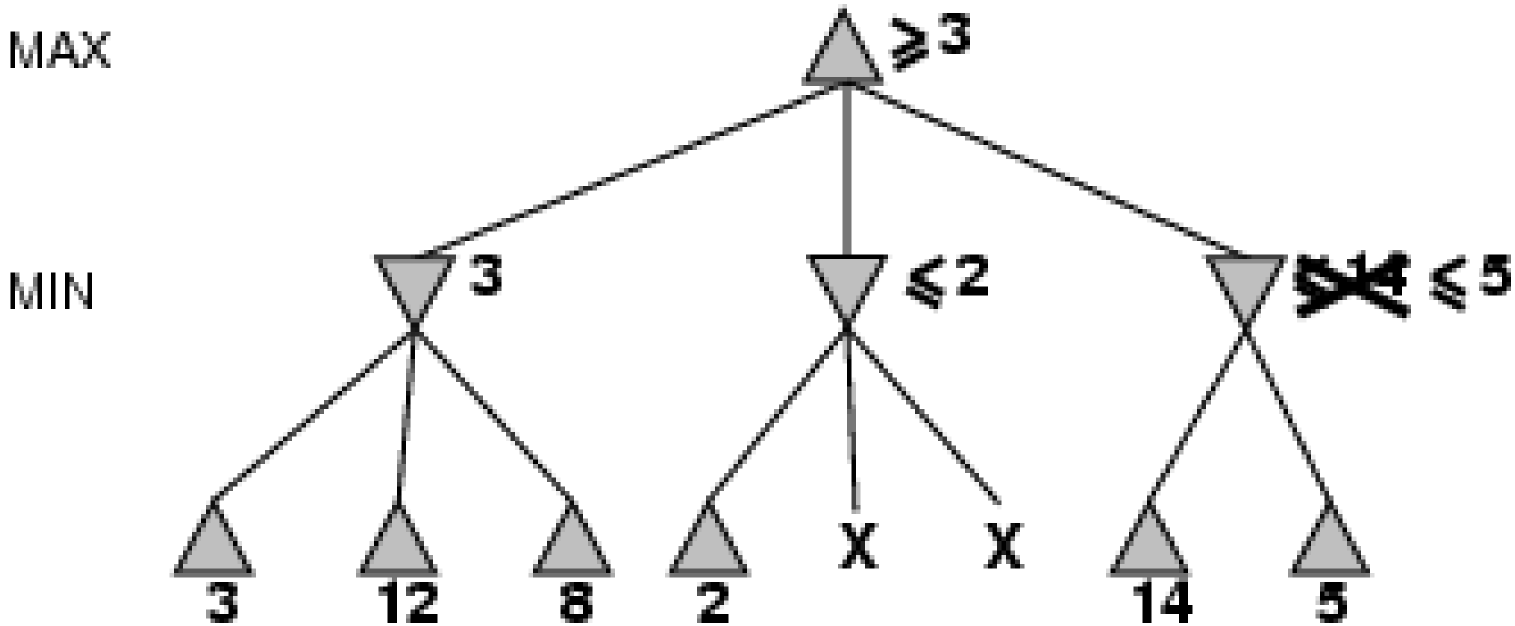
α-β pruning example



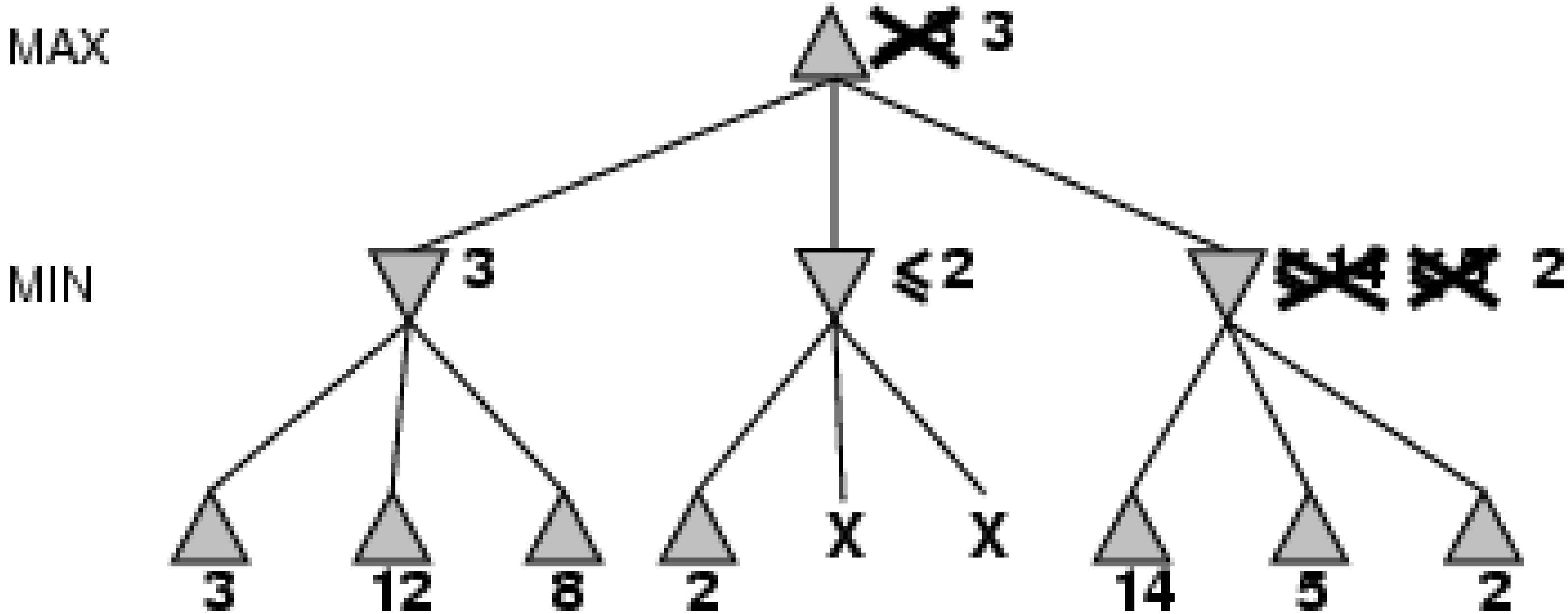
α-β pruning example



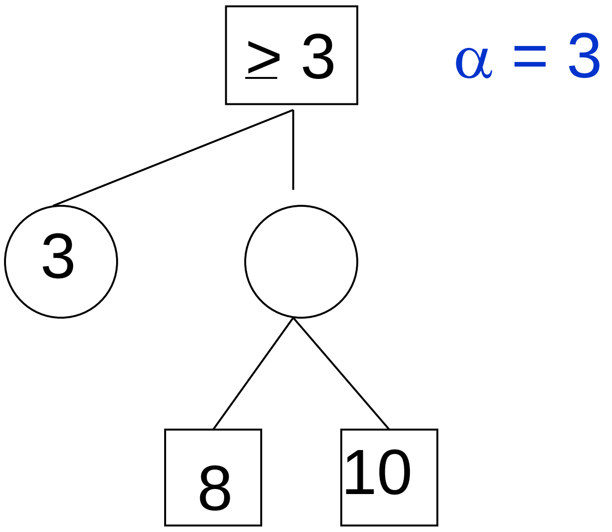
α-β pruning example



α-β pruning example

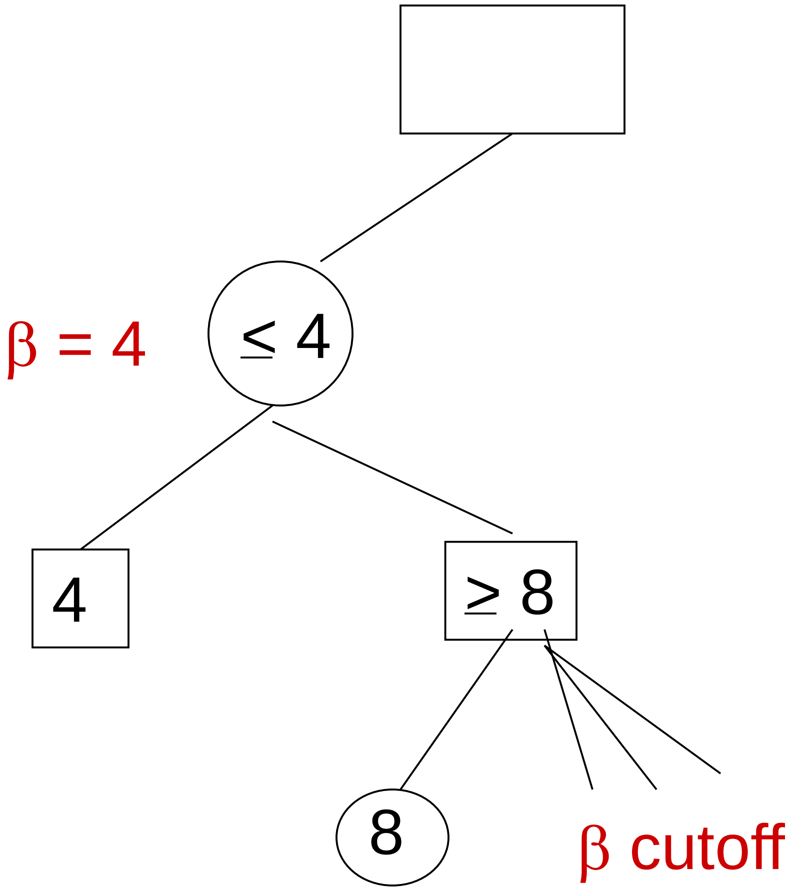


Alpha Cutoff

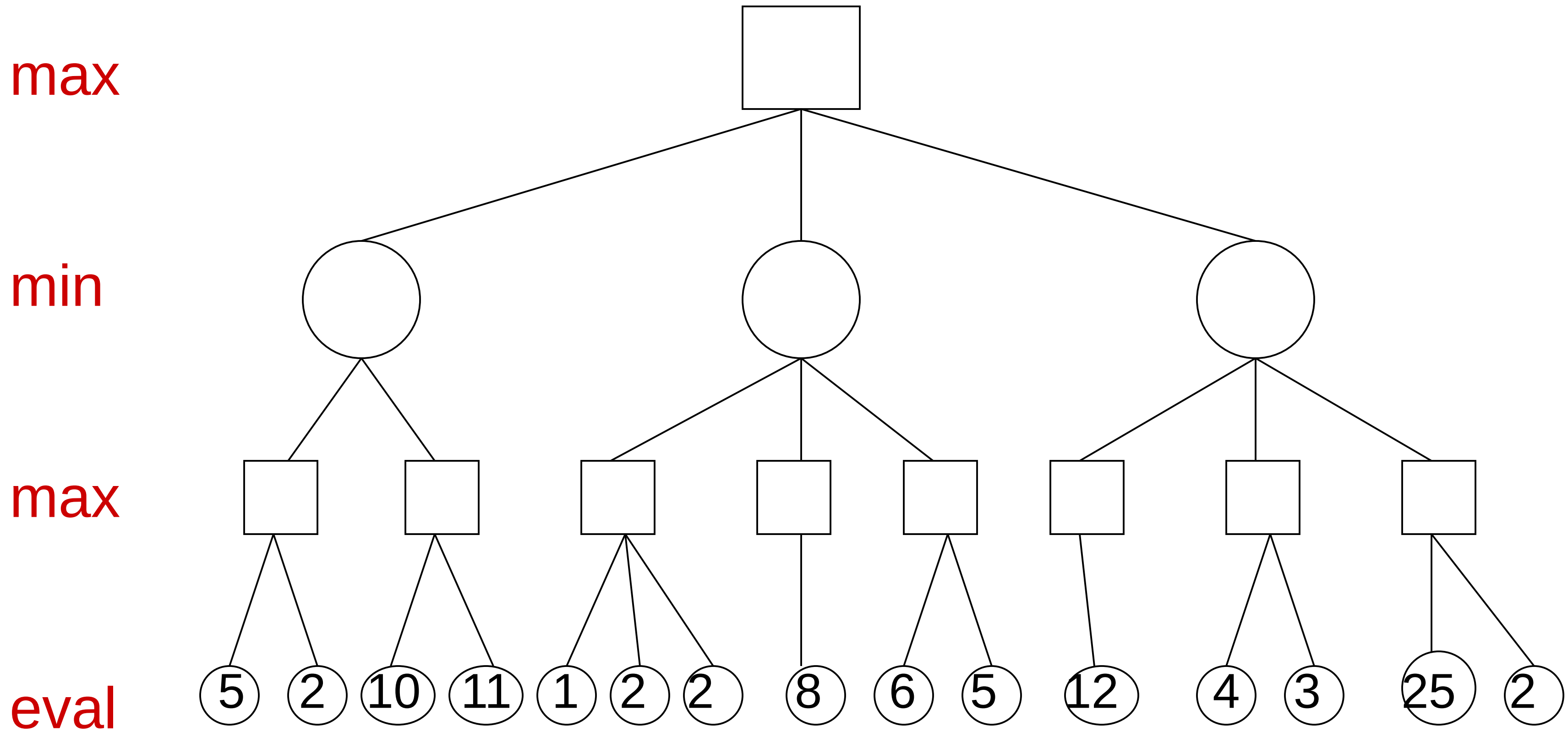


What happens here? Is there an alpha cutoff?

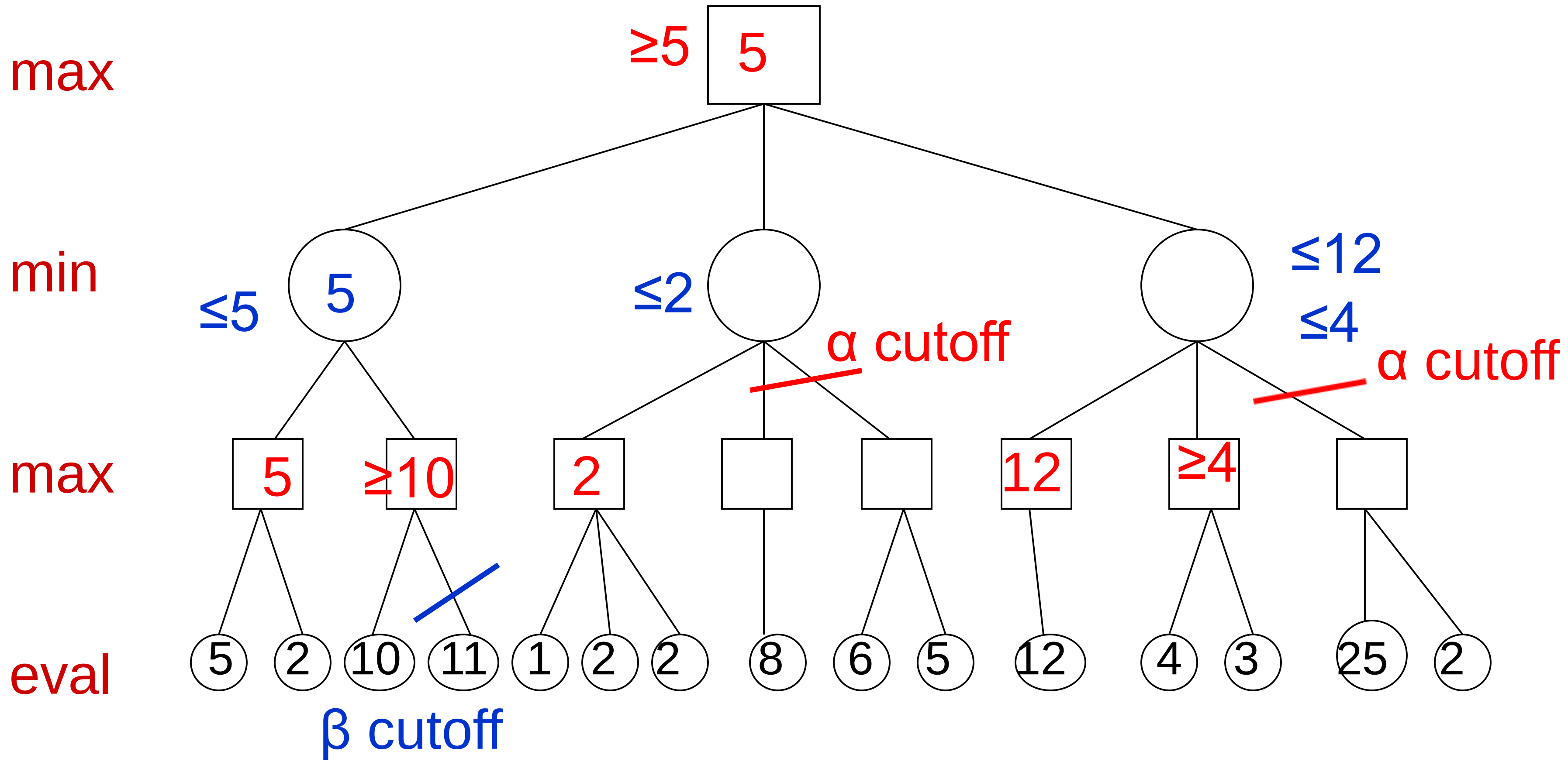
Beta Cutoff



Alpha-Beta Pruning



Alpha-Beta Pruning



Properties of α-β

- Pruning does not affect final result. This means that it gets the exact same result as does full minimax.
-
- Good move ordering improves effectiveness of pruning
-
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
-

The α-β algorithm

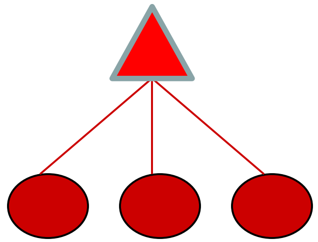
function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ 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*

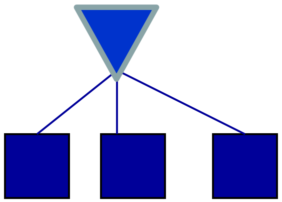
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v cutoff
    $\alpha \leftarrow$ MAX(α, v)
  return v



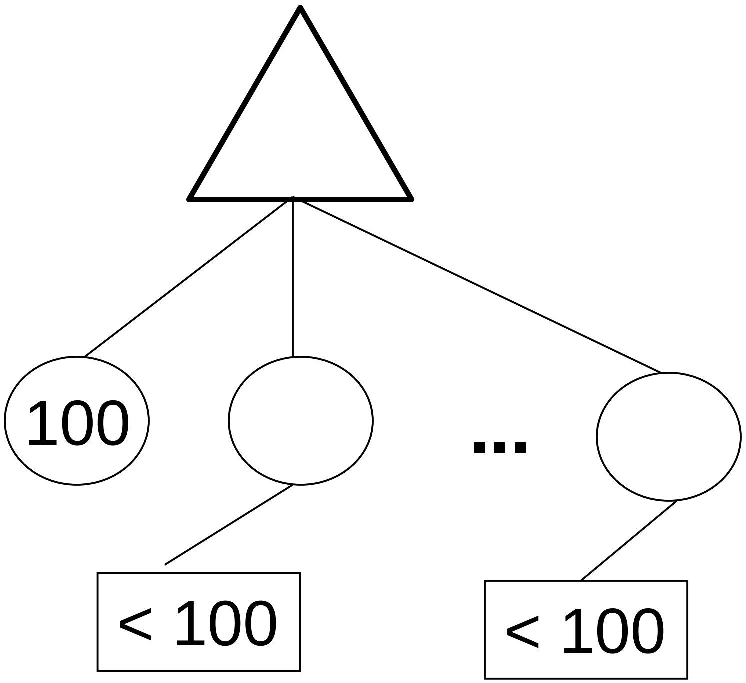
The α-β algorithm

function MIN-VALUE($state, \alpha, \beta$) **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 cutoff
    $\beta \leftarrow$ MIN(β, v)
  return v



When do we get alpha cutoffs?



Shallow Search Techniques

1. limited search for a few levels

2. reorder the level-1 sucessors

3. proceed with α-β minimax search

Additional Refinements

- Waiting for Quiescence: continue the search until no drastic change occurs from one level to the next.
- Secondary Search: after choosing a move, search a few more levels beneath it to be sure it still looks good.
- Book Moves: for some parts of the game (especially initial and end moves), keep a catalog of best moves to make.

Evaluation functions

- For chess/checkers, typically linear weighted sum of features

$$Eval(s) = w_1\ f_1(s) + w_2\ f_2(s) + \ldots + w_n\ f_n(s)$$

- e.g., $w_1 = 9$ with

  $f_1(s)$ = (number of white queens) – (number of black queens), etc.

Example: Samuel's Checker-Playing Program

- It uses a linear evaluation function

  $f(n) = a_1x_1(n) + a_2x_2(n) + ... + a_mx_m(n)$

  For example: $f = 6K + 4M + U$

  - K = King Advantage
  - M = Man Advantage
  - U = Undenied Mobility Advantage (number of moves that Max has that Min can't jump after)

Samuel's Checker Player

- In learning mode
  - Computer acts as 2 players: A and B
  - A adjusts its coefficients after every move
  - B uses the static utility function
  - If A wins, its function is given to B

Samuel's Checker Player

- How does A change its function?

1. Coefficent replacement

$\triangle$ (node) = backed-up value(node) – initial value(node)

if $\triangle$ > 0 then terms that contributed **positively** are given more weight and terms that contributed negatively get less weight

if $\triangle$ < 0 then terms that contributed **negatively** are given more weight and terms that contributed positively get less weight

Samuel's Checker Player

- How does A change its function?

2. Term Replacement

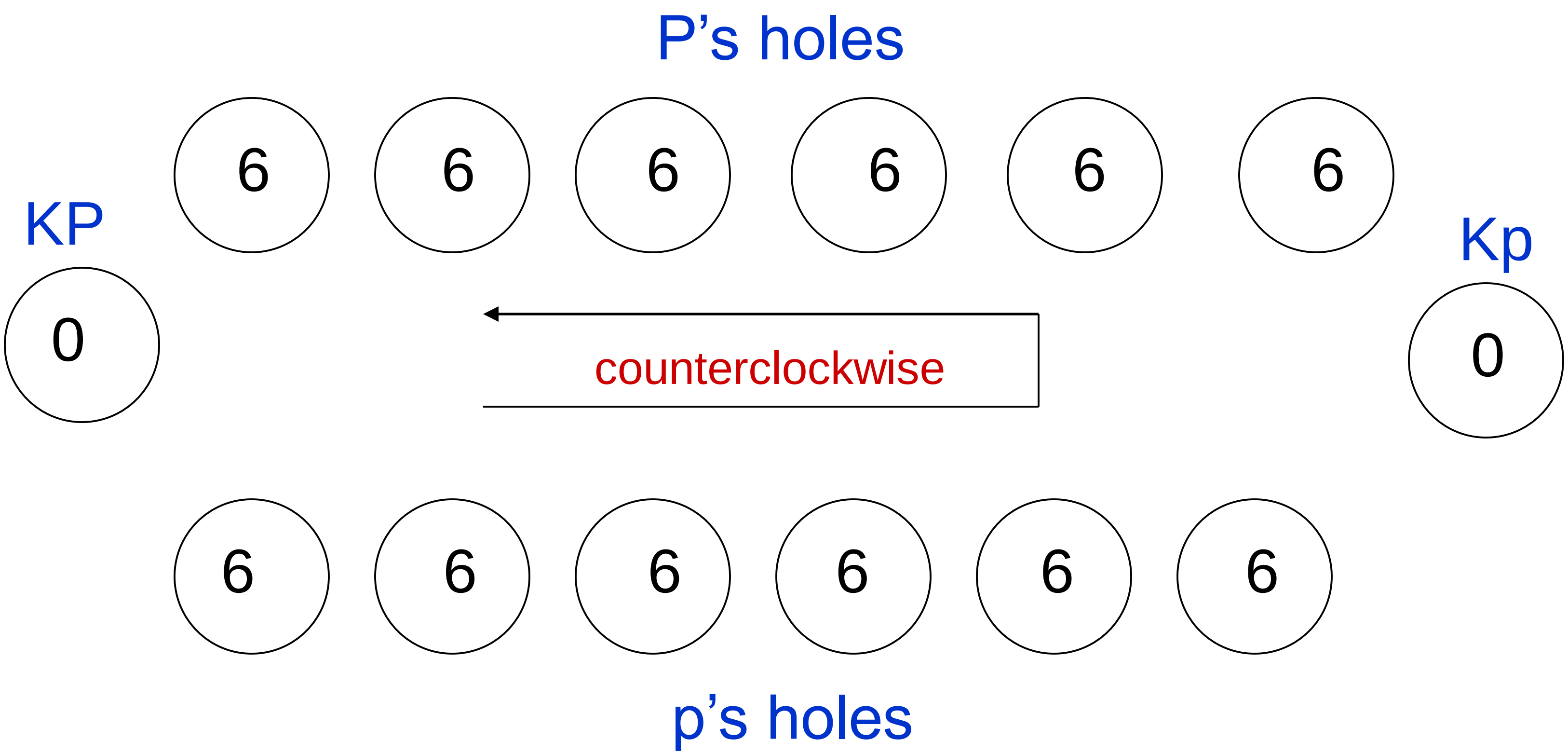
   38 terms altogether

   16 used in the utility function at any one time

   Terms that **consistently correlate low** with the function value are removed and added to the end of the term queue.

   They are replaced by terms from the front of the term queue.

Kalah



To move, pick up all the stones in one of your holes, and put one stone in each hole, starting at the next one, including your Kalah and skipping the opponent’s Kalah.

Kalah

- If the last stone lands in your Kalah, you get another turn.
- If the last stone lands in your empty hole, take all the stones from your opponent's hole directly across from it and put them in your Kalah

  AND put that last stone in your Kalah too.
- If all of your holes become empty, the opponent keeps the rest of the stones.
- The winner is the player who has the most stones in his Kalah at the end of the game.

Cutting off Search

MinimaxCutoff is identical to *MinimaxValue* except

1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

Does it work in practice?

$b^m = 10^6$, $b=35 \rightarrow m=4$

4-ply lookahead is a hopeless chess player!

– 4-ply ≈ human novice
– 8-ply ≈ typical PC, human master
– 12-ply ≈ Deep Blue, Kasparov

Deterministic Games in Practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.

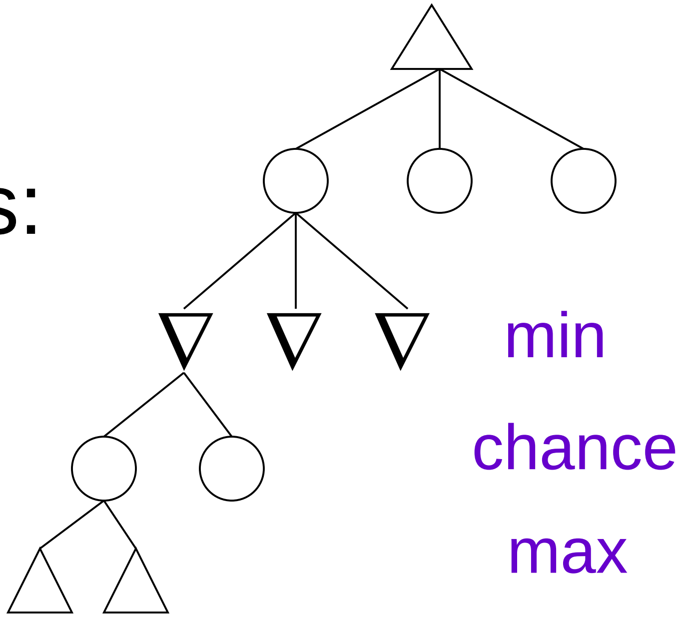
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

- Othello: human champions refuse to compete against computers, who are too good.

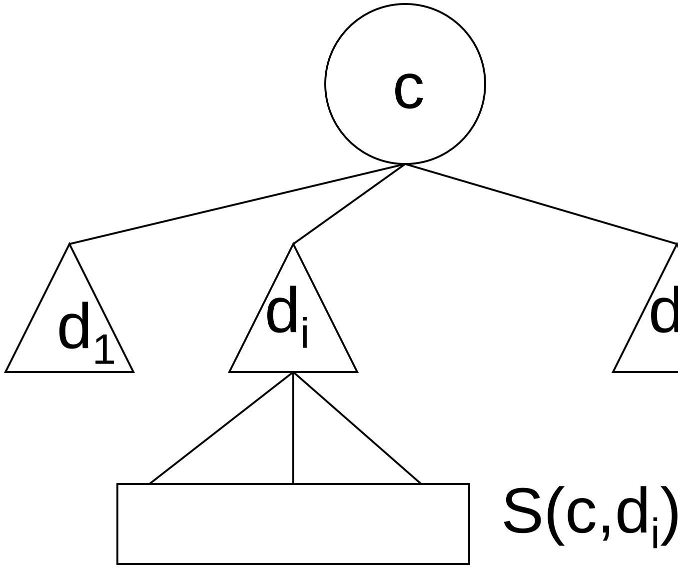
- Go: The game of Go has been a fertile subject of artificial intelligence research for decades, culminating in 2017 with **AlphaGo Master winning three of three games against Ke Jie**, who at the time continuously held the world No. 1 ranking for two years.

Games of Chance

- What about games that involve chance, such as
  - rolling dice
  - picking a card
- Use three kinds of nodes:
  - max nodes
  - min nodes
  - chance nodes



Games of Chance

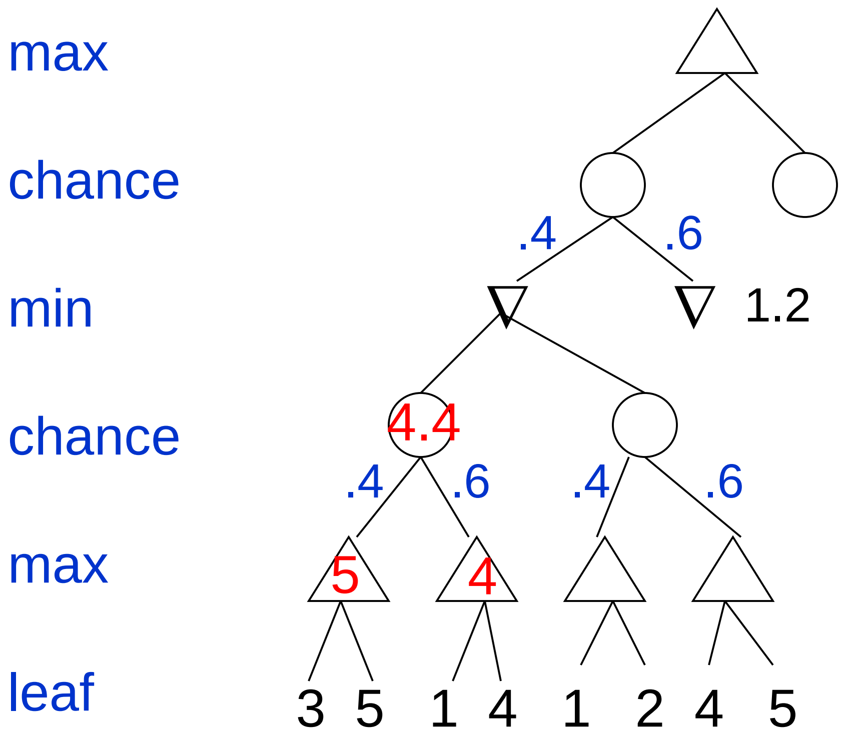


chance node with max children

$$\text{expectimax}(c) = \sum_{i} P(d_i) \max_{s \text{ in } S(c,d_i)} (\text{backed-up-value}(s))$$

$$\text{expectimin}(c') = \sum_{i} P(d_i) \min_{s \text{ in } S(c,d_i)} (\text{backed-up-value}(s))$$

Example Tree with Chance



.4 * 5 + .6 * 4 = 2 + 2.4 = 4.4

Complexity

- Instead of $O(b^m)$, it is $O(b^m n^m)$ where n is the number of chance outcomes.

- Since the complexity is higher (both time and space), we cannot search as deeply.

- Pruning algorithms may be applied.

Summary

- Games are fun to work on!
-
- They illustrate several important points about AI.
-
- Perfection is unattainable → must approximate.
- Game playing programs have shown the world what AI can do.