CSCI 3230

Fundamentals of Artificial Intelligence

Chapter 5, Sect 1–5

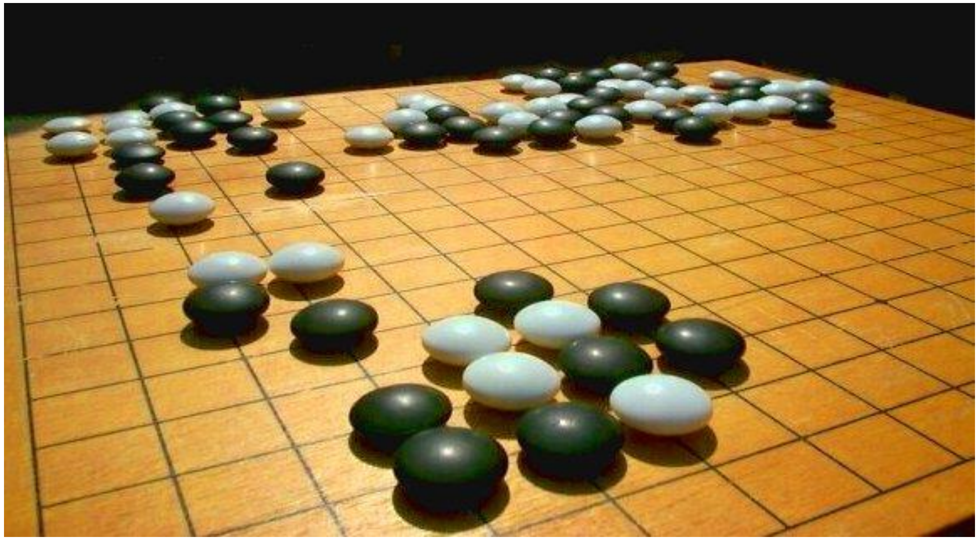
GAME PLAYING

Adver'sarial Search

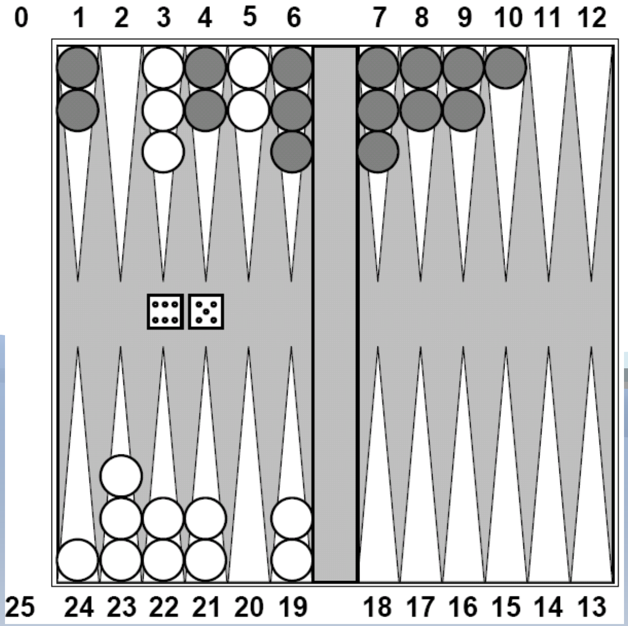
Fall 2019

Game theory – commercial & economical applications; business/real war game theory; bargaining strategies; dynamic/static;

Checkers



Go



Backgammon

A B C D E F G H
1 2 3 4 5 6 7 8

Othello

Outline

- Games as Search Problems
- Optimal Decision Games
- Resource limits
- $\alpha-\beta$ pruning
- Games of chance

Games vs. search problems

"Unpredictable" opponent ⇒ solution is a strategy specifying a move for every possible opponent reply

Time limits ⇒ unlikely to find goal, must approximate

History of attack:

- Considered possibility of computer's chess (Babbage, 1846), 173 yrs ago
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948, Shannon, 1950)
  zu-ser we-ner
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)

Games vs. search problems

- Deep Blue beat G. Kasparov in 1997
- AlphaGo uses CNN & Monte Carlo Tree beat Lee Sedol on 12 Mar 2016
- Then, Ke Jie 柯潔, ranked #1 in the world, May 2017
- AlphaGo Zero-use reinforcement learning (play against itself)

  beat AlphaGo 100:0 (no training examples) Oct 2017.

Games as search problems

- The state of a game: easy to represent; has a small number of well-defined actions. Game playing: an idealization of worlds in which hostile agents act to diminish one’s well-being.

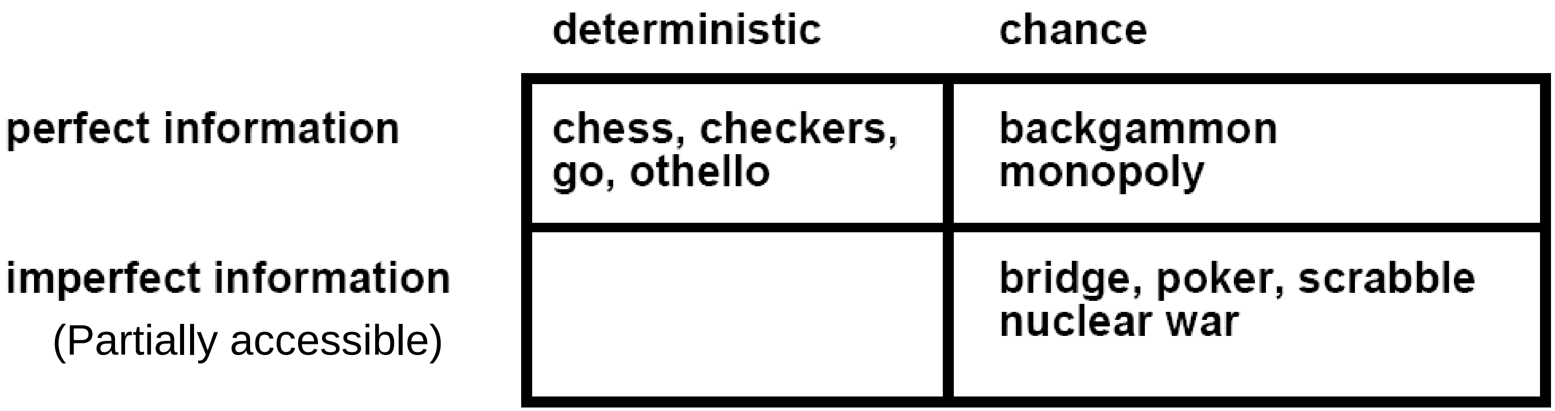
Easy to represent, indicating intelligence

- Why games so special in AI? ∵usually much too hard to solve.
- Chess, average branching factor of about 35, 50 moves by each player, so the search tree= ~35^{100} nodes (“only” ~10^{40} different legal positions).
- GO board:19x19=361; b≈250; d≈211; ~250^{211} nodes

Games as search problems

- Games more like the real world than the standard search problems.
- Games penalize inefficiency very severely.
- A chess program 10% less effective in using its available time probably will be beaten into the ground.
- Pruning: to ignore portions of the search tree that make no difference to the final choice.
- Heuristic evaluation functions approximate the true utility of a state without doing a complete search.

Types of games



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

Perfect Decisions in Two-Person Games

Consider a game with 2 players, MAX and MIN. MAX moves 1st.

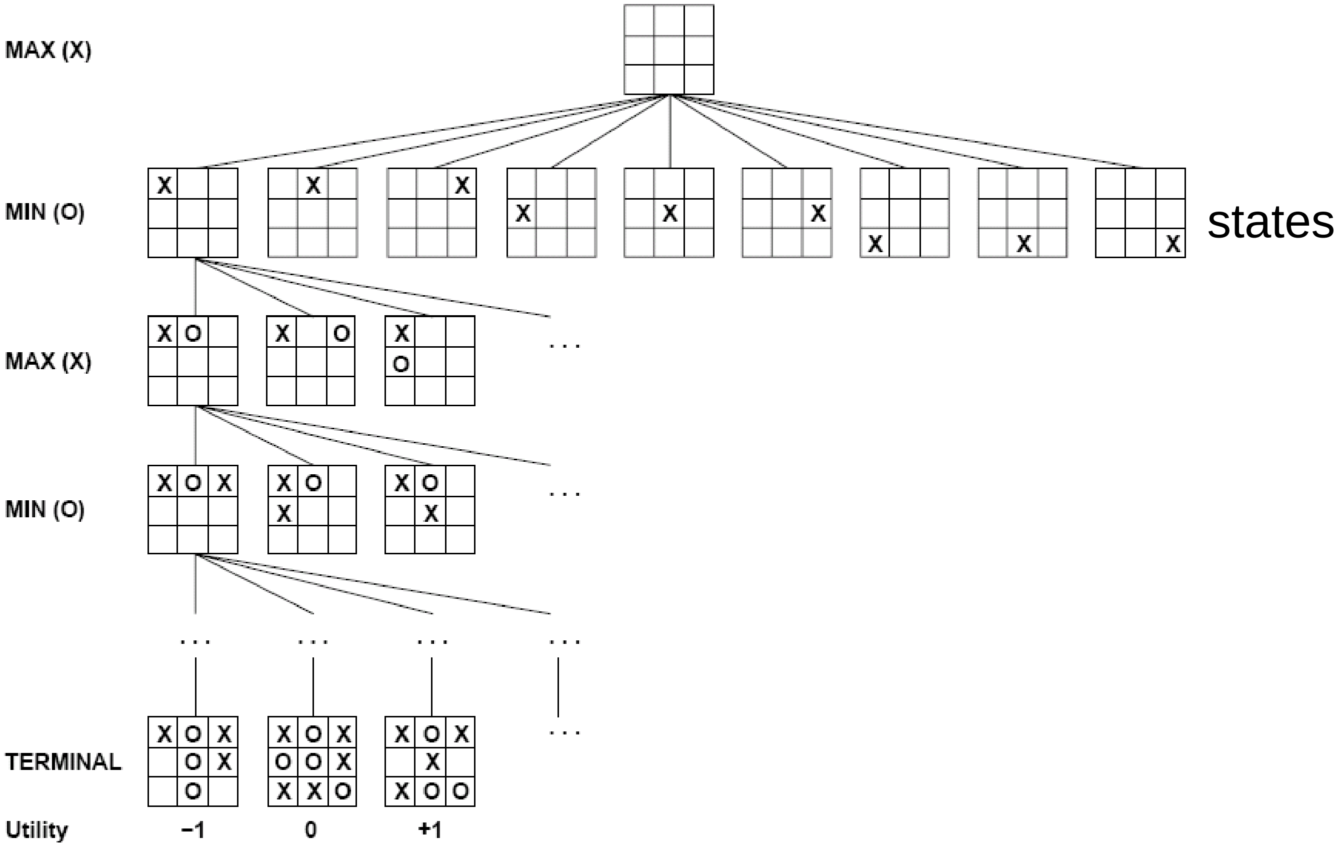
A game defined as a search problem with the following components:

- The initial state includes the board position and an indication of who to move. (All other possible states).
- A set of operators define the legal moves. *b*
- A terminal test determines the terminal states (game over).
- A utility function (or a payoff function) gives numeric outcomes of a game. In chess: win, loss or draw, represented by +1, -1 or 0.

Others: a wider range of possible outcomes; e.g. backgammon payoff range: +192 to -192; Majong

⇒ Max to find a strategy leading to a winning terminal state regardless of what MIN does; i.e., finds the correct move by MAX for each possible move by MIN.

Game tree (2-player, deterministic, turns)



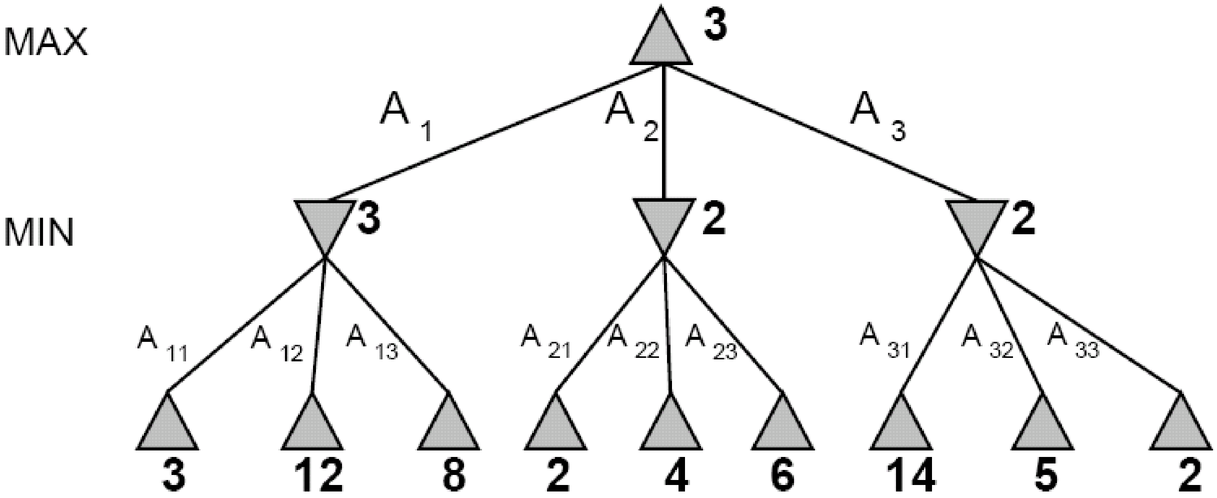
A (partial) search tree for the game of Tic-Tac-Toe.
Top node – initial state.
Utilities assigned by the rules of the game to the terminal states.

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 (1move) game:

(Recursive for n-ply)



A 2-ply game tree generated by the minimax algorithm. The terminal nodes show the utility value for MAX computed by the utility function. The utilities of the other nodes are computed by the minimax algorithm from the utilities of their successors. MAX's best move is A_1; Min's best reply is A_{11}. ?

Minimax algorithm

```
function Minimax-Decision(state, game) returns an action
    inputs: state, current state in game
    v ← Max-Value(state) //For Max nodes to start
    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 each s in Successors(state) do //all possible moves
        v ← Max(v, Min-Value(s)) //recursive call
    return v
```

```
function Min-Value(state) returns  a utility value
    if Terminal-Test(state) then return Utility(state)
    v ← ∞
    for each s in Successors(state) do
        v ← Min(v, Max-Value(s)) //recursive call
    return v
```

Recursive ⇒ ? search

Goto α-β

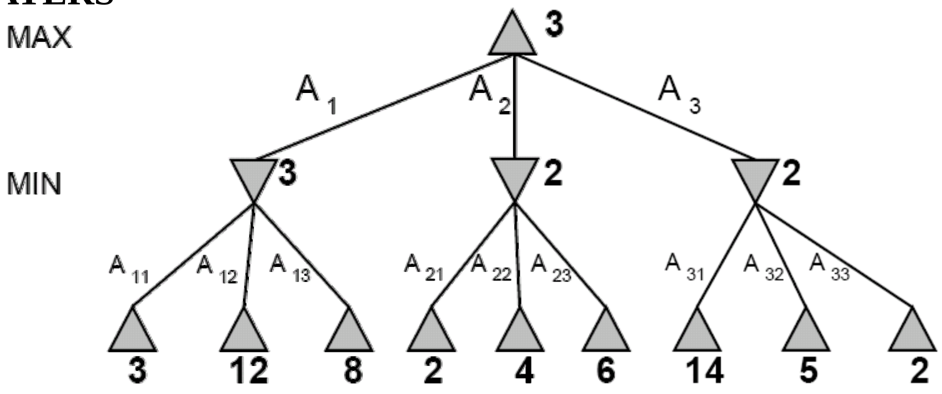
EXAMPLE for MINIMAX Decision

$v \leftarrow$ MAX-VALUE (state)		
Call MAX-VALUE $v \leftarrow -\infty$ for each s A_1 $v \leftarrow$ MAX (v, MIN-VALUE (s))	A_2 $v \leftarrow$ MAX (v, MIN-VALUE (s))	A_3 ………….
MIN-VALUE $v \leftarrow \infty$ for each s A_{11} A_{12} A_{13} $v \leftarrow$ MIN (v, MAX-VALUE (s))* $v = 3 \leftarrow$ MIN (∞, 3) (3, 12) (3, 8) return $v = 3$	MIN-VALUE $v \leftarrow \infty$ for each s A_{21} A_{22} A_{23} $v \leftarrow$ MIN (v, MAX-VALUE (s))** $v = 2 \leftarrow$ MIN (∞, 2) (2, 4) (2, 6) return $v = 2$	…………. return $v = 2$
Return to MAX-VALUE $v \leftarrow$ MAX (v, MIN-VALUE (s)) $v = 3 \leftarrow$ MAX ($v = -\infty$, MIN-VALUE = $v = 3$)	Return to MAX-VALUE $v \leftarrow$ MAX (v, MIN-VALUE (s)) $v = 3 \leftarrow$ MAX ($v = 3$, MIN-VALUE = $v = 2$)	…………. $v = 3 \leftarrow$MAX ($v = 3$, $v = 2$)

Note: * TERMINAL-TEST in MAX-VALUE is "YES", ∴ return UTILITY(s) = 3, 12, 8

**** TERMINAL-TEST in MAX-VALUE is "YES", ∴ return UTILITY(s) = 2, 4, 6**

TERMINAL-TEST CAN COME IN DEEP LAYERS



Properties of minimax

Complete	Yes, if tree is finite (chess has specific rules for this) NB a finite strategy can exist even in an infinite tree!
Optimal	Yes, against an optimal opponent. Otherwise??
Time	$O(b^m)$, m: max depth of search tree; ??
Space	O(bm) (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
⇒ exact solution completely infeasible

Resource limits

Minimax assumes the program has time to search all the way to terminal states, usually not practical.

Suppose we have 100 seconds, explore 10^4 nodes/second

⇒ 10^6 nodes per move (chess: about 10^{40} different legal positions, **m=4** out of 100 required).

Standard approach:

- Cutoff test (🡐 terminal test)

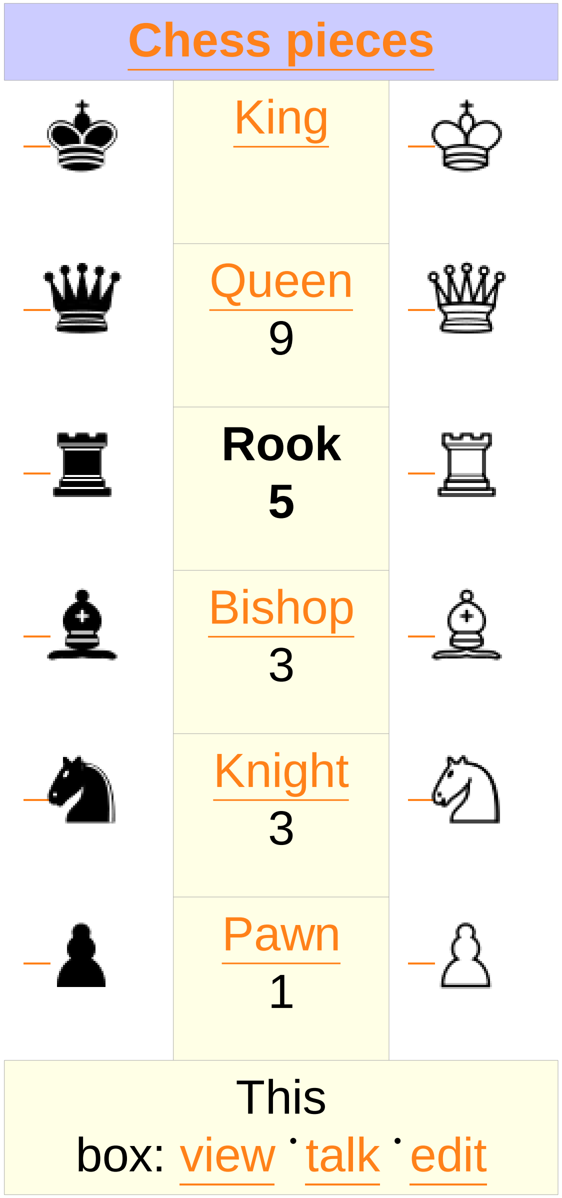
  E.g. Depth limit (perhaps add quiescence search)

- Evaluation function (🡐Utility function) (cf heuristics in A*)

  = estimated desirability of position

Evaluation functions, EVAL

- An evaluation function estimates the **expected utility** of a given position.
- E.g. introductory chess books give an approximate material value for each piece: each pawn is worth 1, a knight or bishop is 3, a rook 5, and the queen 9.
- The performance of a game-playing program extremely dependent on the quality of its EVAL.
- If inaccurate, it guides the program toward positions that are apparently "good", but in fact disastrous.



Chess pieces		
♚	King	♔
♛	Queen 9	♕
♜	**Rook** **5**	♖
♝	Bishop 3	♗
♞	Knight 3	♘
♟	Pawn 1	♙

This box: view · talk · edit

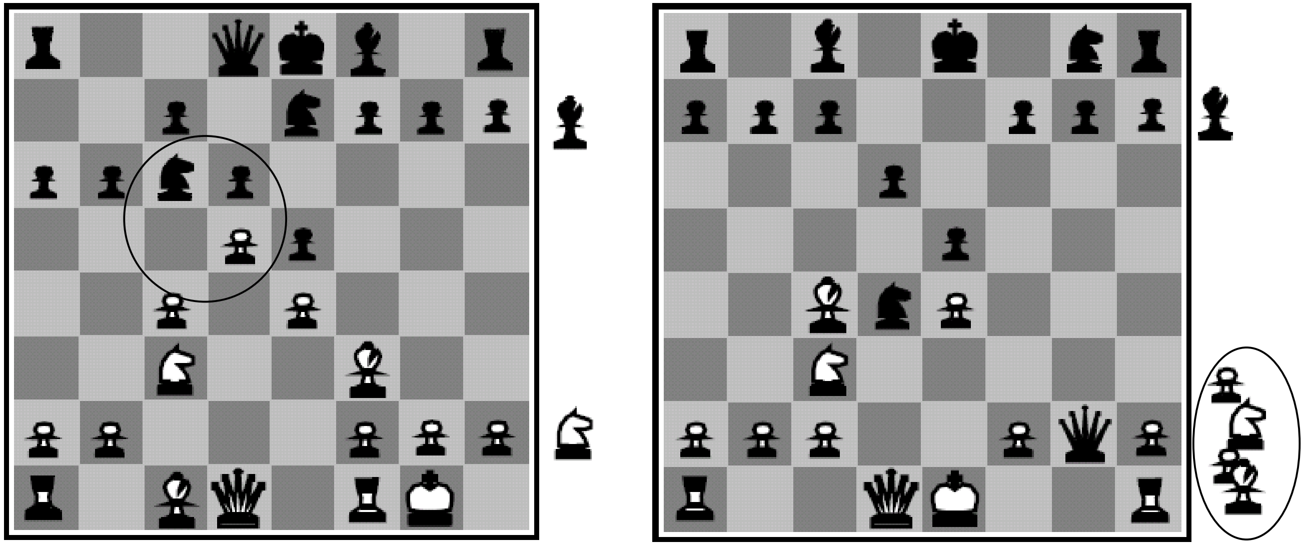
Evaluation functions, EVAL

- How exactly do we measure quality?
  1. The evaluation function must agree with the utility function on terminal states. Cf heuristic function
  2. It must not take too long!
  3. EVAL should accurately reflect the actual chances of winning.
- The material advantage evaluation function assumes that the value of a piece can be judged independently of the other pieces present on the board.
- This kind of EVAL is called a weighted linear function, expressed as

$$w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

where the w's are the weights, and the f's are the features of the particular position (e.g. w_1 is 1 for the pawn & f_1 is no. of pawns etc.)

Evaluation functions



Black to move

White slightly better

not accounted for in the linear Eval fn

White to move

Black winning

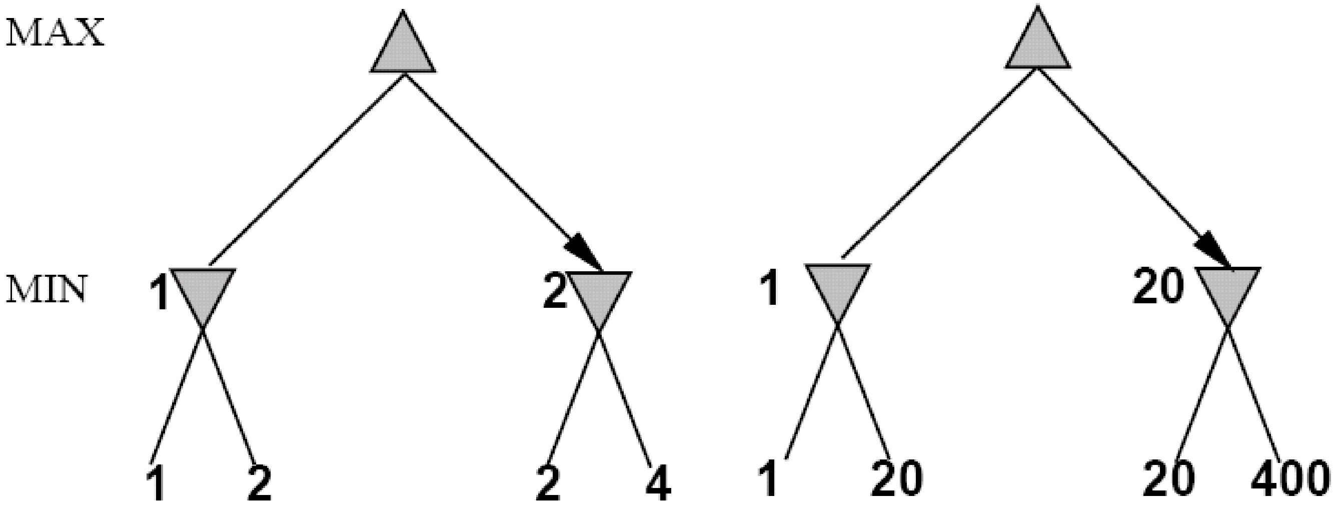
For chess, typically linear weighted sum of features.

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

e.g. $w_1 = 9$

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

Digression: Exact values don't matter



not change direction/order

Behavior is preserved under any monotonic transformation of EVAL.

Only the order matters:

payoff in deterministic games acts as an ordinal utility function.

Cutting off search

Minimax Cutoff is identical to Minimax Value except

1. Terminal? is replaced by Cutoff?
2. Utility is replaced by EVAL

Does it work in practice?

b^m: 10^6 nodes/move, $b = 35 \Rightarrow m = 4$

4 ply look ahead is a hopeless chess player!

4 ply ≈ human novice (: ‘n-all-vis)

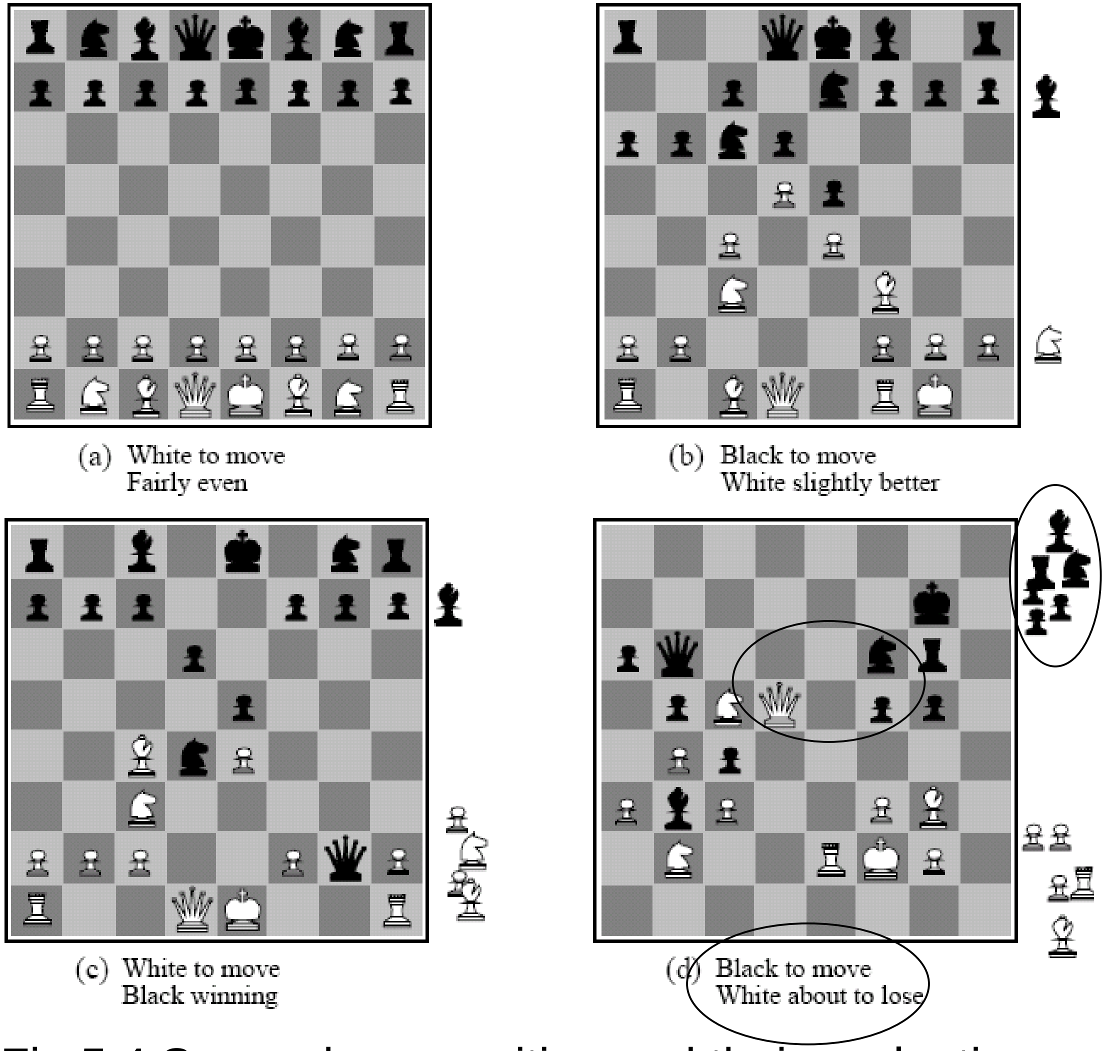
8 ply ≈ typical PC, human master

12 ply ≈ Deep Blue, Kasparov

Cutting off search

- The simplest approach to control the amount of search is to set a fixed depth limit, *d*. ?
- *d* is set by the time limit of the game.
- A more robust approach: apply iterative deepening. When time runs out, returns the move selected by the deepest completed search.??
- Sometime disastrous consequences because of the approximate nature of EVAL.
- Consider the simple evaluation function based on material advantage.
  (see fig 5.4(d) mistaken white is winning if stop too short – White Queen is to be taken.)

Cutting off search



(a) White to move
Fairly even

(b) Black to move
White slightly better

(c) White to move
Black winning

(d) Black to move
White about to lose

White seems
to be winning
by taking more
pieces, (1 knight)
but …

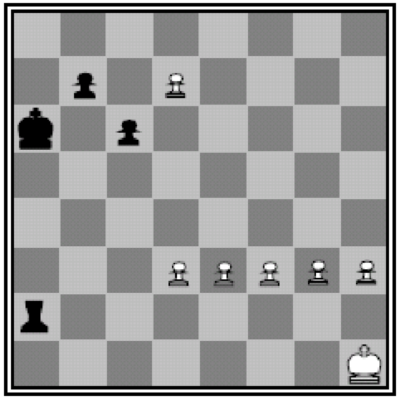
Fig 5.4 Some chess position and their evaluations.

Qui'escence search

- A more sophisticated cutoff test needed. The evaluation function only applied to quiescent positions, i.e., unlikely to exhibit wild swings in value in the near future.
- E.g., positions with favorable captures are not quiescent.
- Expand non-quiescent positions until quiescent positions are reached – called a quiescence search;
- Sometimes restricted to consider only certain types of moves, such as capture moves, that quickly resolve the uncertainties in the position.

Horizon problem

- The horizon problem is more difficult to estimate. It arises when the program is facing a move by the opponent that causes serious damage and is ultimately unavoidable.
- (The stalling moves push the inevitable queening move "over the horizon" to a place where it cannot be detected.)



Black to move

Fig 5.5 The horizon problem. A series of checks by the black rook forces the inevitable queening move by white "over the horizon" and makes this position (state) look like a slight advantage for black, when it is really a sure win for white. ***One strategy to mitigate: singular extension - remember "clearly better" single moves***

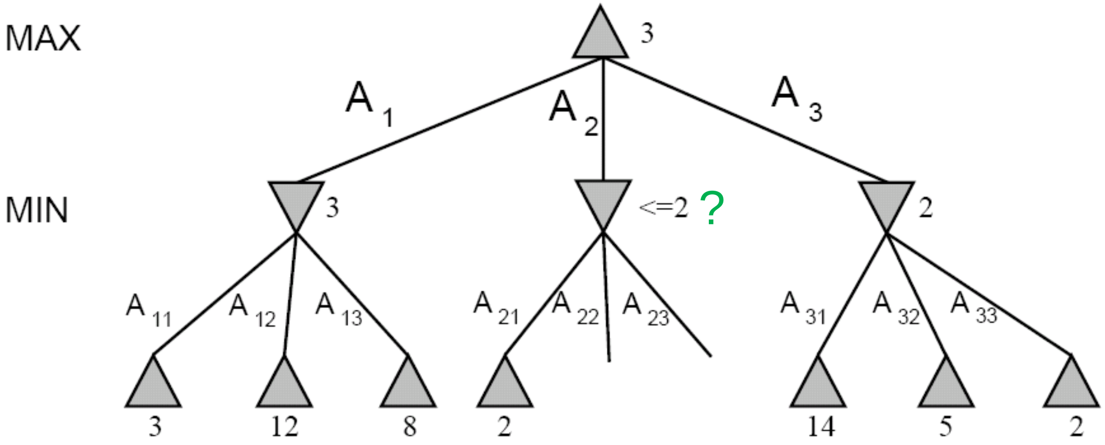
Alpha-Beta pruning

- Possible to compute the correct minimax decision without looking at every node in the search tree.
- The process of eliminating a branch of the search tree is called pruning the search tree.
- The technique is called alpha-beta pruning.
- Returns the same moves as minimax, but prunes away branches not influencing the final decision.

Alpha-Beta pruning

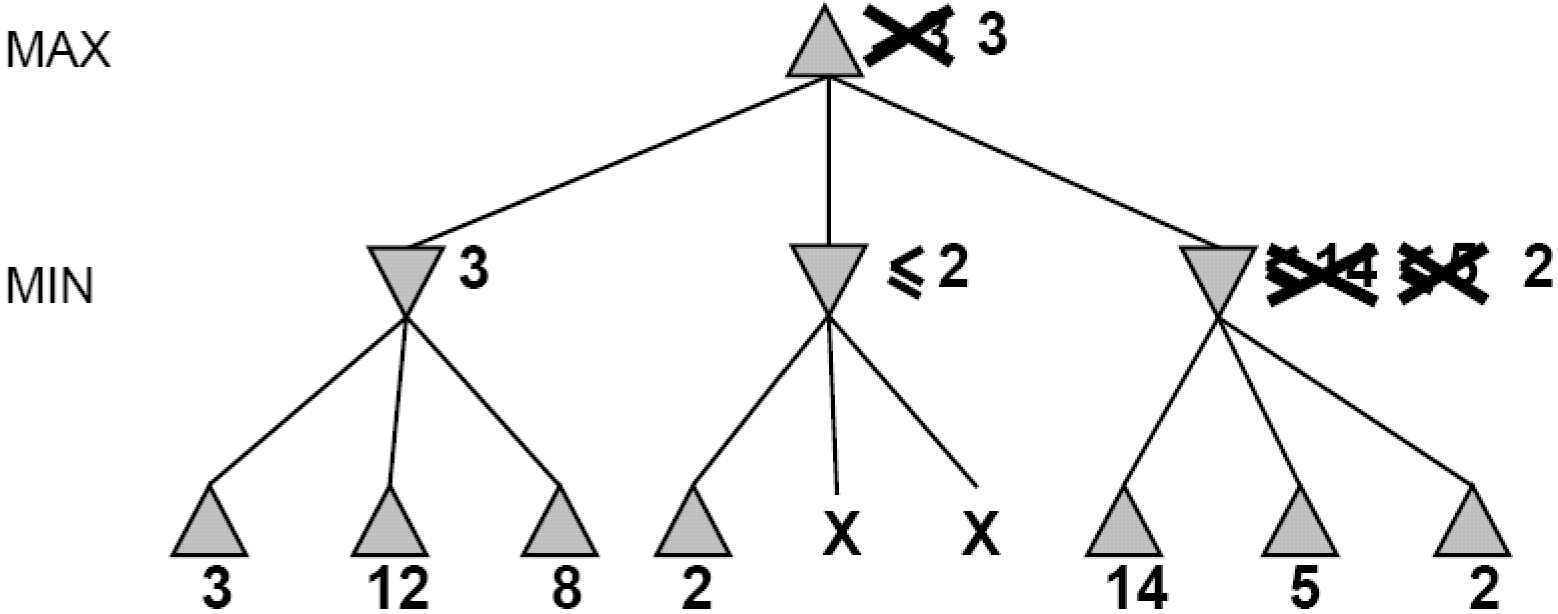
General Principle:

Consider a node *n* somewhere in the tree such that Player has a choice of moving to that node. If Player has a better choice m either at the parent node of *n*, or at any choice point further up, then *n* will never be reached in actual play. So once we know enough about *n* to reach this conclusion, we can prune it.



The 2-ply game tree as generated by alpha-beta.

α–β pruning example

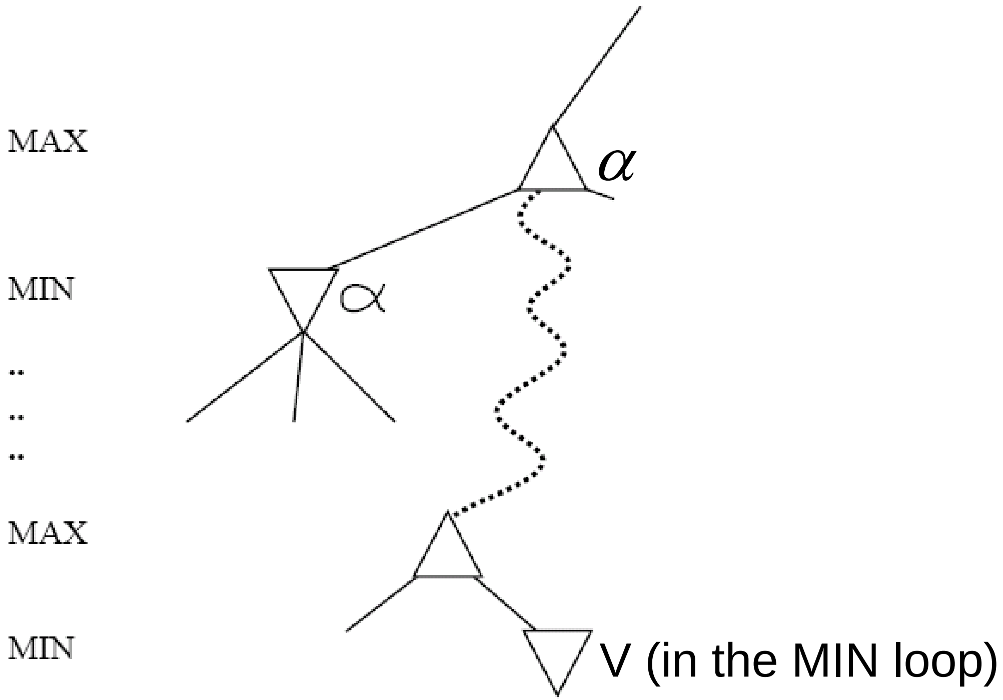


?Re-ordering branches?
Heuristics: types of moves.
Memorize previous search steps, pruning values

Properties of α–β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering", time complexity = $O(b^{m/2})$
  ⇒ doubles depth of search
  ⇒ can easily reach depth 8 and play good chess
- A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)

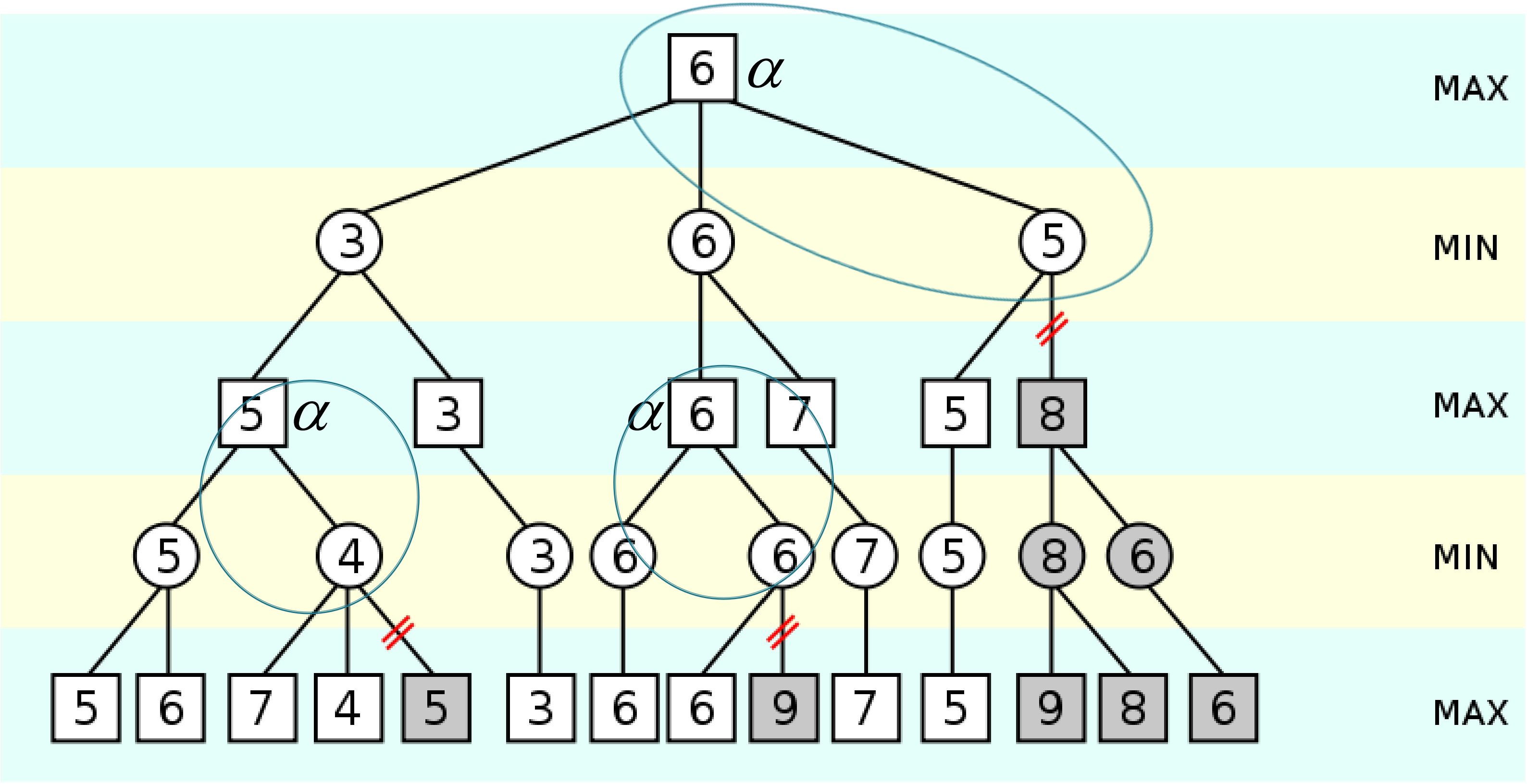
Why is it called α–β



α is the best value (to MAX) found so far off the current path

If V is worse than α, MAX will avoid it ⇒ prune the branch

Define β similarly for MIN


6 α
MAX
3
6
5
MIN
5 α
3
α 6
7
5
8
MAX
5
4
3
6
6
7
5
8
6
MIN
5
6
7
4
5
3
6
6
9
7
5
9
8
6
MAX

The α–β algorithm

```
function Alpha-Beta-Search (state) returns an action
   inputs: state, current state in game

   $v \leftarrow$ Max-Value(state, $-\infty$, $+\infty$) //for Max nodes to start, initialize $\alpha, \beta$
   return the action in Successors(state) with value $v$

function Max-Value(state, $\alpha, \beta$) returns a utility value
   inputs: state, current state in game
               $\alpha$, the value of the best alternative for MAX along the path to state
               $\beta$, 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 each s in Successors(state) do
       $v \leftarrow$ Max($v$, Min-Value($s, \alpha, \beta$))
       if $v \geq \beta$ then return $v$  //nodes follow are pruned by jumping out of the for loop
       $\alpha \leftarrow$ Max($\alpha, v$) //update $\alpha$
    return $v$

function Min-Value(state, $\alpha, \beta$) returns a utility value
   inputs: state, current state in game
               $\alpha$, the value of the best alternative for MAX along the path to state
               $\beta$, 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 each s in Successors(state) do
       $v \leftarrow$ Min($v$, Max-Value($s, \alpha, \beta$))
       if $v \leq \alpha$ then return $v$  //nodes follow are pruned by jumping out of the for loop
       $\beta \leftarrow$ Min($\beta, v$) //update $\beta$
    return $v$
```

Goto Minimax

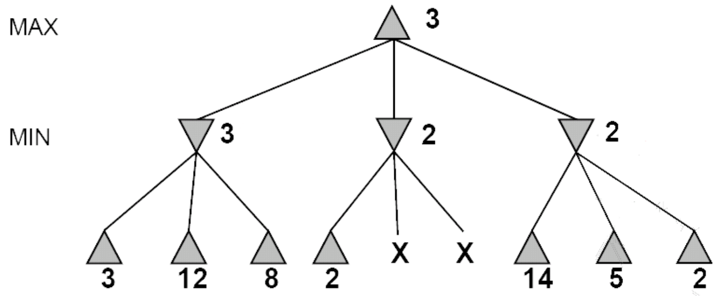
EXAMPLE for α-β Pruning

Initialize $\alpha = -\infty$; $\beta = \infty$		
Call MAX-VALUE $v \leftarrow -\infty$ **for each s** A_1 $v \leftarrow$ **MAX (v, MIN-VALUE (…α, β))**	A_2 $v \leftarrow$ **MAX (v, MIN-VALUE (…$\alpha = 3$, β))**	A_3 ….
MIN-VALUE $v \leftarrow \infty$ for each s A_{11} A_{12} A_{13} $v \leftarrow$ **MIN (v, MAX-VALUE (…α, β))*** $v = 3 \leftarrow$ **MIN (∞, 3) (3, 12) (3, 8)** $v = 3 \leq \alpha = -\infty$? *//pruning?* $\beta = 3 \leftarrow$ **MIN ($\beta = \infty$, v)** *//update β* **return** $v = 3$ *(β won't be returned)*	**MIN-VALUE** $v \leftarrow \infty$ for each s A_{21} / ~~A_{22}~~ / ~~A_{23}~~ $v \leftarrow$ **MIN (v, MAX-VALUE (…α, β))** $v = 2 \leftarrow$ **MIN ($v = \infty$, MAX-VALUE=2)**** $v = 2 \leq \alpha = 3$ *** ***return*** $v = 2$	….. $v = 2 \leftarrow$ **MIN (∞, 14)** **(14, 5) (5, 2)** …..
Return to MAX-VALUE $v \leftarrow$ **MAX (v, MIN-VALUE (…α, β))** $v = 3 \leftarrow$ **MAX ($v = -\infty$, MIN-VALUE = $v = 3$)** $v = 3 \geq \beta = \infty$? $\alpha = 3 \leftarrow$ **MAX ($\alpha = -\infty$, $v = 3$)**	Return to **MAX-VALUE** $v \leftarrow$ **MAX (v, MIN-VALUE (…$\alpha = 3$, β))** $v = 3 \leftarrow$ **MAX ($v = 3$, MIN-VALUE = $v = 2$)** $v = 3 \geq \beta = \infty$? $\alpha = 3 \leftarrow$ **MAX ($\alpha = 3$, $v = 3$)**	…. $v = 3 \leftarrow$ **MAX ($v = 3$, MIN-VALUE = $v = 2$)**

Note: * TERMINAL-TEST in MAX-VALUE is "YES", ∴ return UTILITY(s) = 3, 12, 8

**** TERMINAL-TEST in MAX-VALUE is "YES", ∴ return UTILITY(s) = 2**

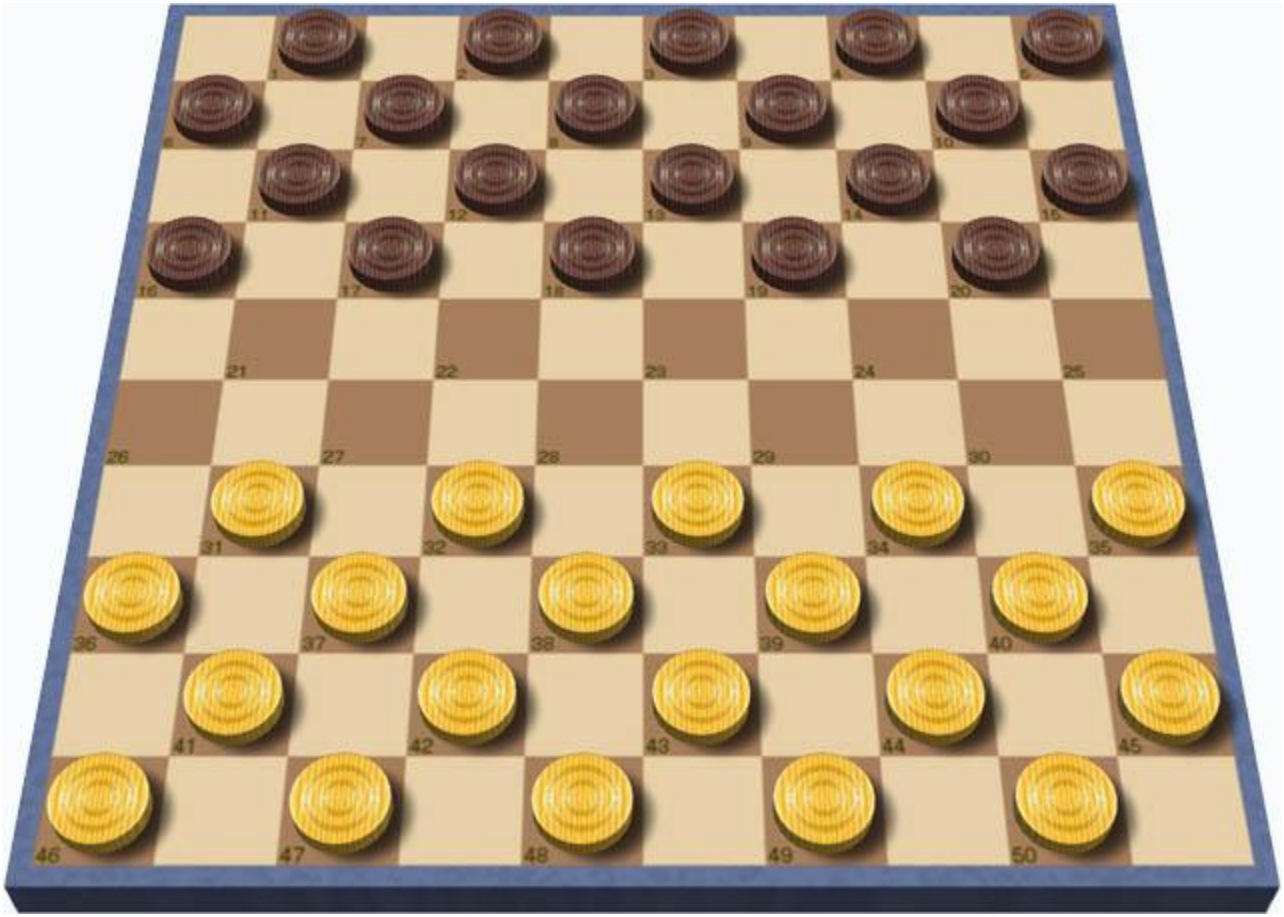
***** Jump out of the "for each s loop" and effectively prune A_{22} & A_{23}**



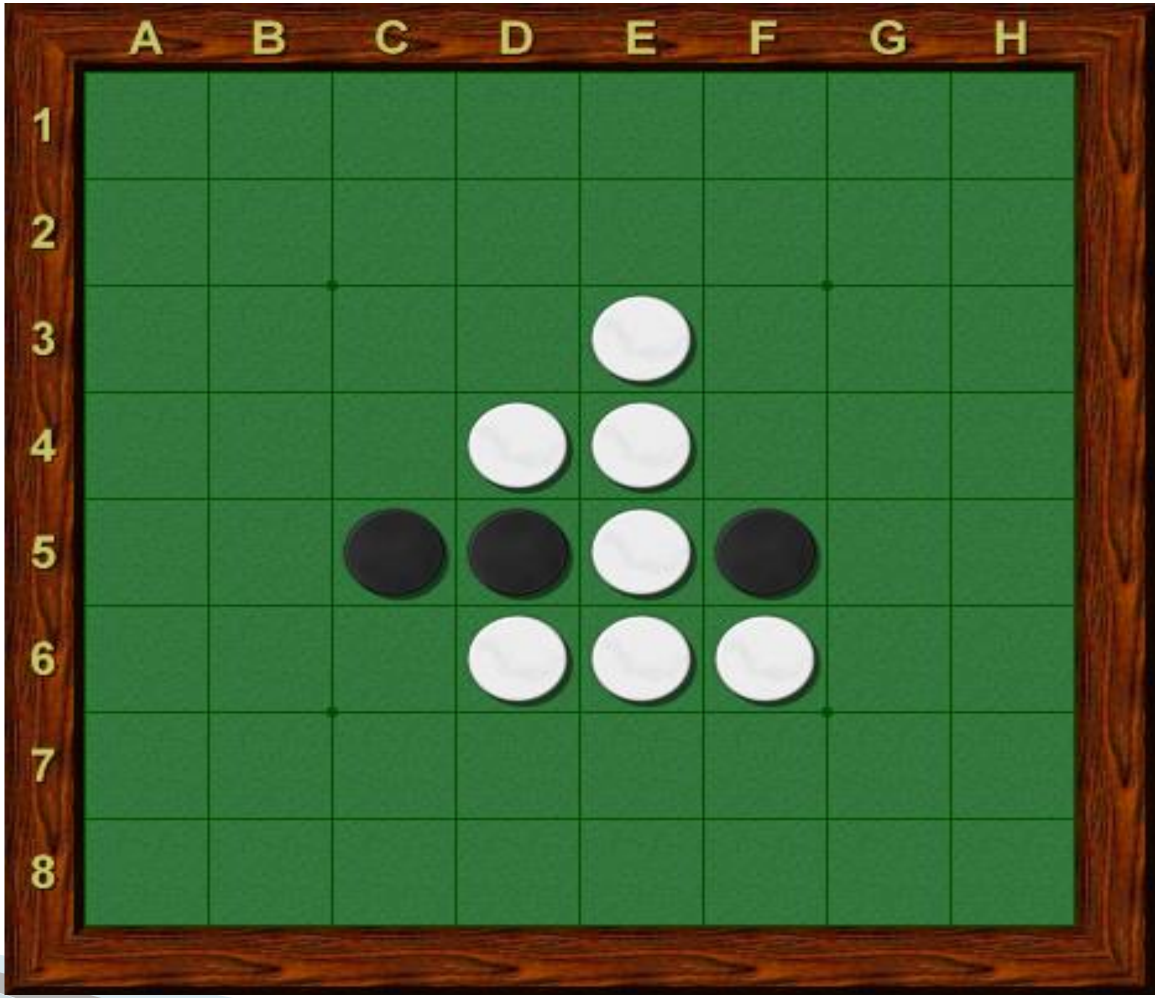
Deterministic in practice

- Checkers: Chinook ended 40-year-reign of human world champion. Marion Tinsley in 1994 used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.
- Chess: Deep Blue defeated human world champion Gary 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. ⇨ Blue-gene
- Othello: human champions refuse to compete against computers, who are too good.
- Go: In Go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves. World champion beaten 2017

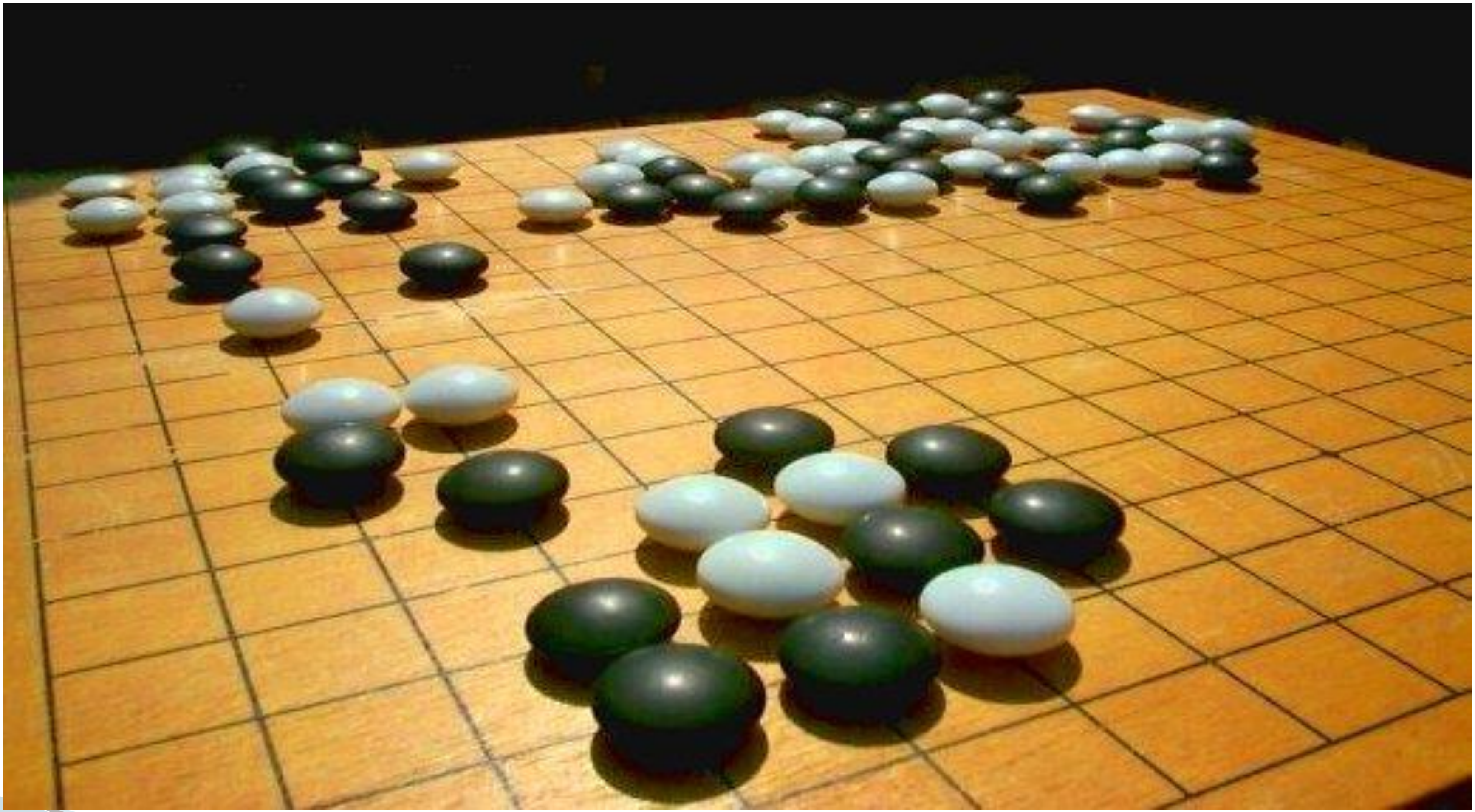
Checkers


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50

Othello

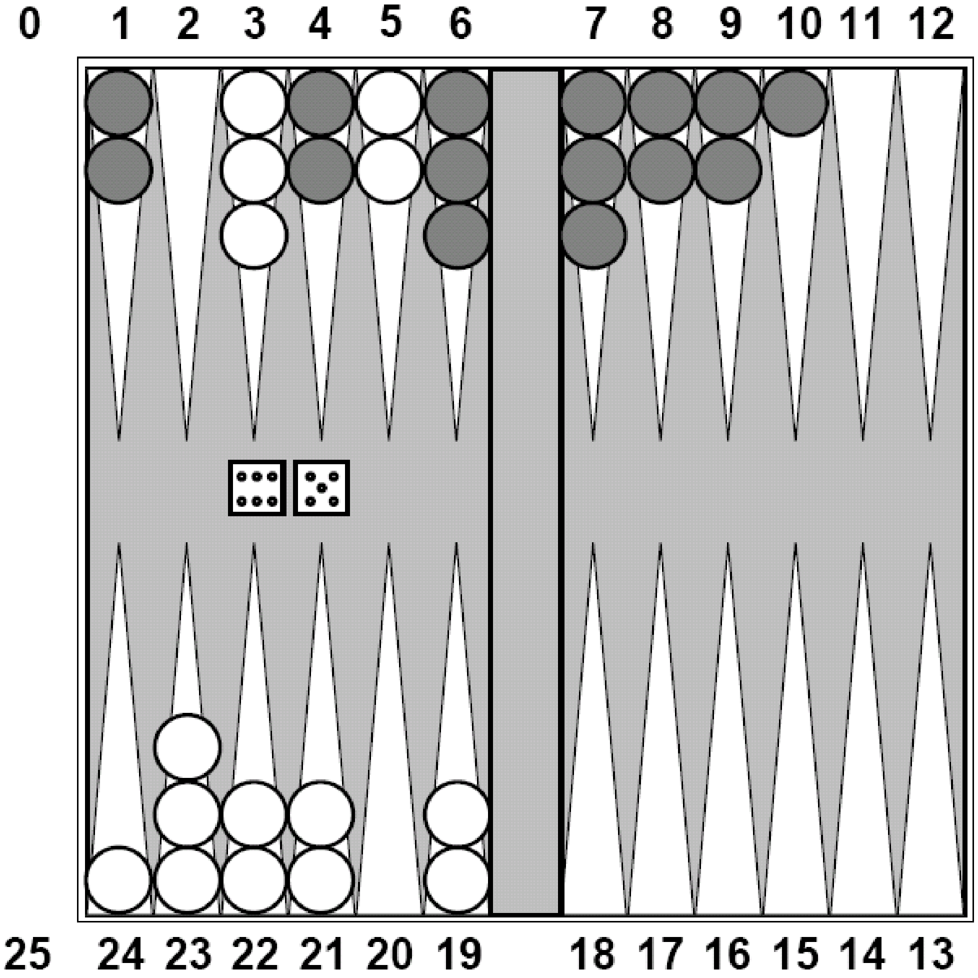


Go 圍棋



Chinese chess invented 204 BC; GO ~2337 BC, recorded ~2500 ago;

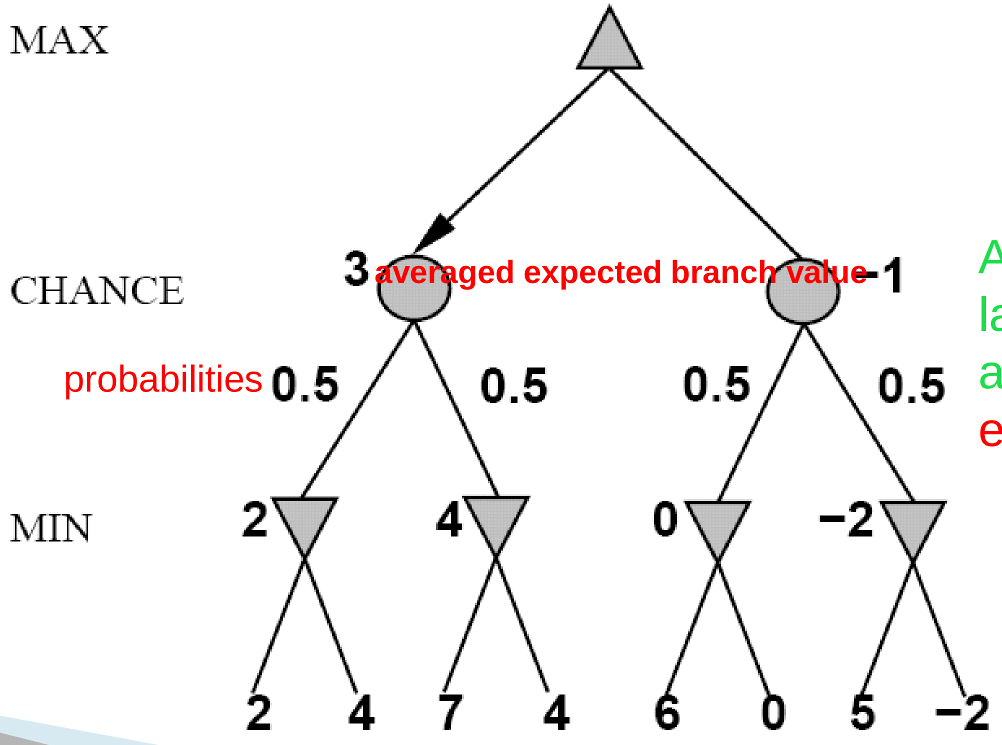
Nondeterministic games: backgammon



Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:



Add a chance node layer in between MIN and MAX; averaged expected branch value

Algorithm for nondeterministic games

ExpectiMinimax gives perfect play

Just like Minimax, except we must also handle chance nodes:

```
…
if state is a Max node then
    return the highest ExpectiMinimax-value of Successors(state)
if state is a Min node then
    return the lowest ExpectiMinimax-value of Successors(state)
if state is a chance node then
    return average of ExpectiMinimax-value of Successors(state)
…
```

Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- Perfection is unattainable ⇒ must approximate
- Good idea to think about what to think about e.g. pruning
- Uncertainty constrains the assignment of values to states

Games are to AI as grand prix racing is to automobile design