Game playing

in competitive multi-agent problems

Reminder: Taxi example. How do you define multi-agent problems?

Outline

- Games
- Perfect play
  - – minimax decisions
  - – α–β pruning
- Resource limits and approximate evaluation
- Games of chance
- Games of imperfect information

From the textbook: "Game playing was one of the first tasks undertaken by AI. ... to the point that machines have surpassed humans in ... The main exception is Go, in which computers perform at the amateur level".
search tree of chess: 35^100 in average.

Games vs. search problems

"Unpredictable" opponent $\Rightarrow$ solution is a strategy
specifying a move for every possible opponent reply

Time limits $\Rightarrow$ unlikely to find goal, must approximate

Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

Require the ability to make some decision even when calculating the optimal decision is infeasible.

Types of games

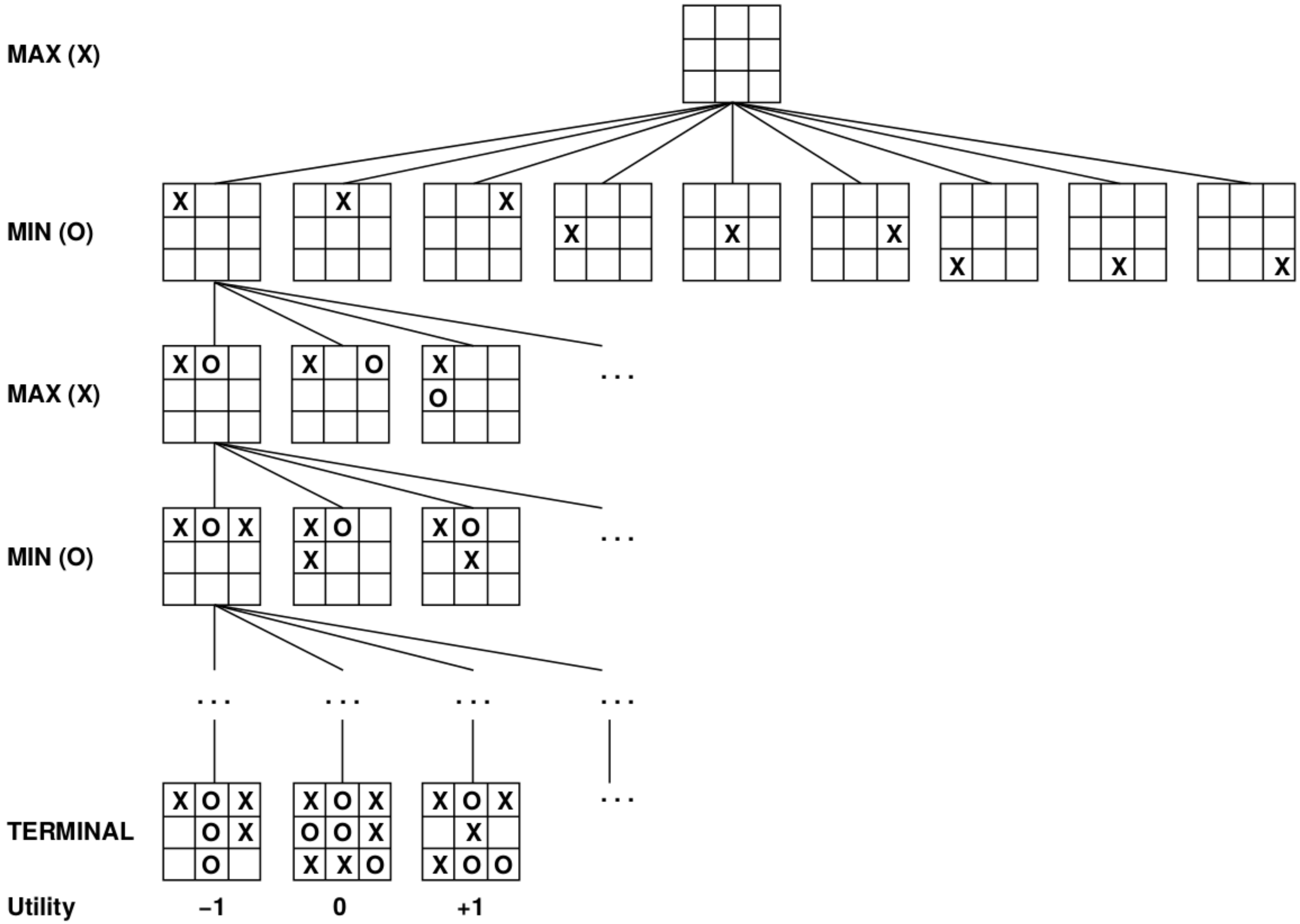
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Define game formally:
- initial state
- successor function
- terminal test
- utility function

"zero-sum" games are considered here.

2-players: MAX and MIN. Max moves first

Game tree (2-player, deterministic, turns)



minimax value:High values are assumed to be good for MAX (&bad for MIN).
MAX should find a "contingent" strategy.

Optimal strategy: leads to outcomes at least as good as any other strategy when one is playing a perfect opponent.

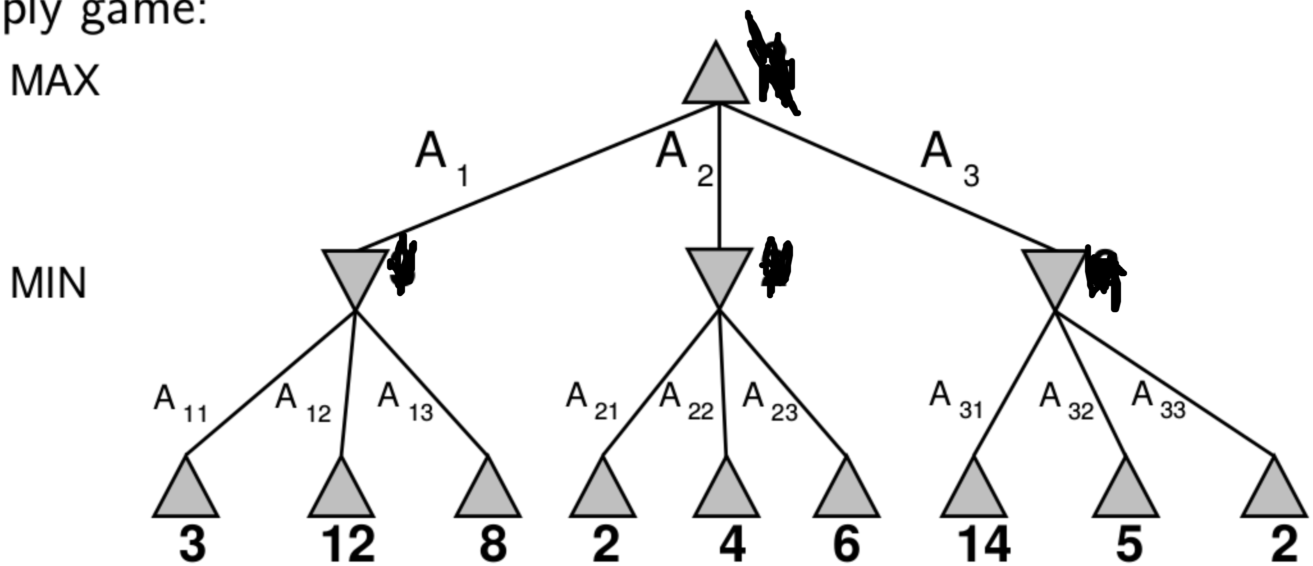
Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
= best achievable payoff against best play

assuming both players play optimally.

E.g., 2-ply game:



MAX prefers to a state with max value, MIN prefers min

MINIMAX-VALUE (n) = Utility (n) if n is terminal
max MINIMAX-VAL (s) where s is successor and n is MAX node
min MINIMAX-VAL (s) where s is successor and n is MIN node

How about minimax decision at the root?

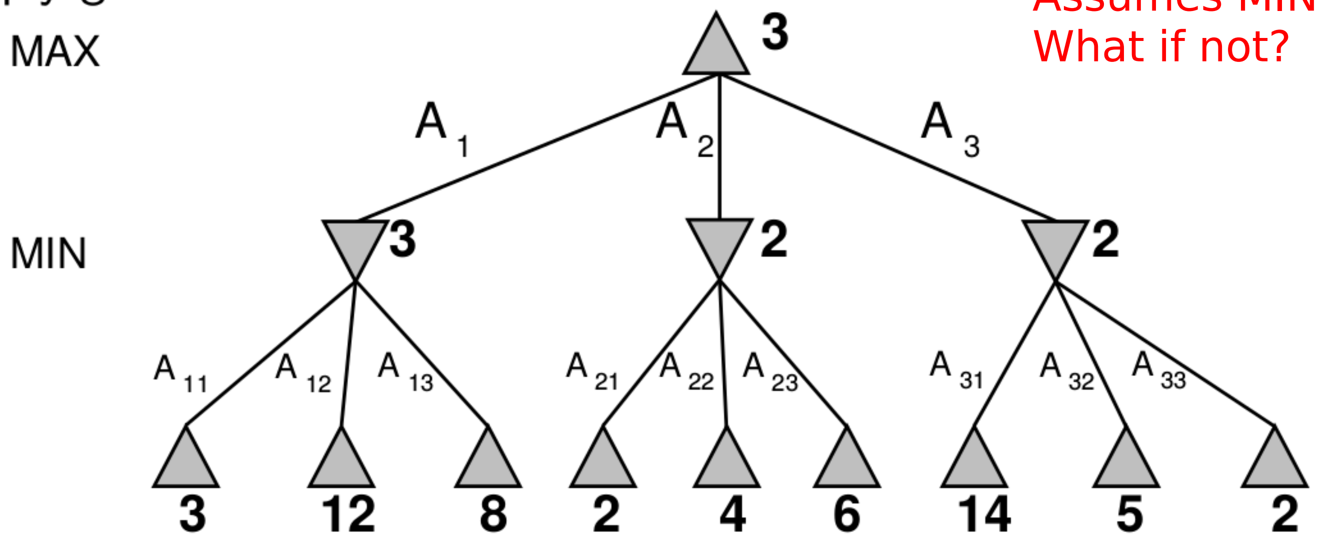
Minimax

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:

Assumes MIN plays optimal.
What if not?



How about minimax decision at the root?

Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
   inputs: state, current state in game

   return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
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
```

Properties of minimax

Complete??

Properties of minimax

Complete?? Only if tree is finite

Optimal??

Properties of minimax

Complete?? Yes, if tree is finite

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity??

What type of exploration?

Properties of minimax

Complete?? Yes, if tree is finite

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity??

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

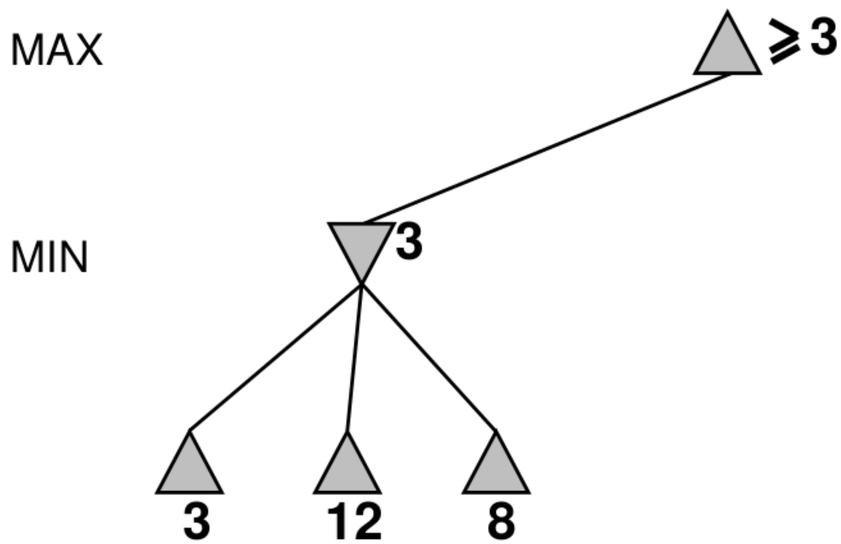
$\Rightarrow$ exact solution completely infeasible

2 questions:

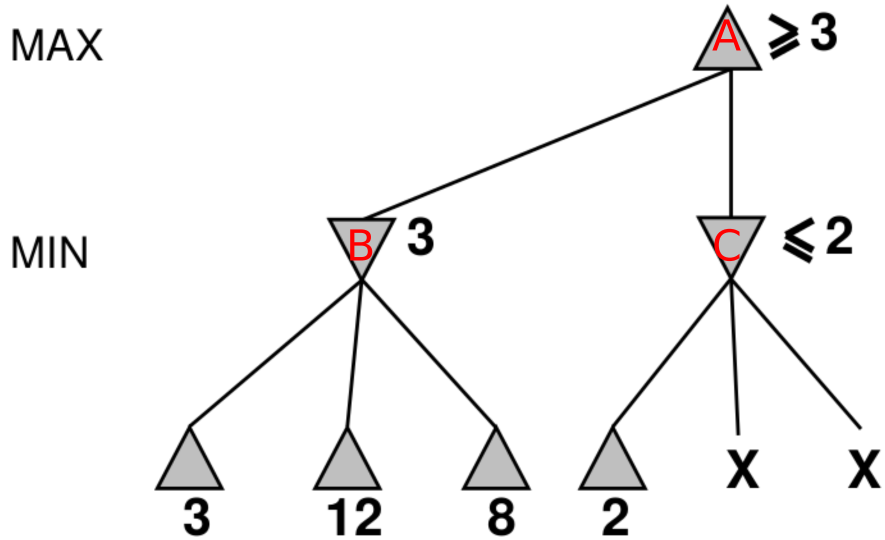
But do we need to explore every path?

How about multi-player games?

α–β pruning example

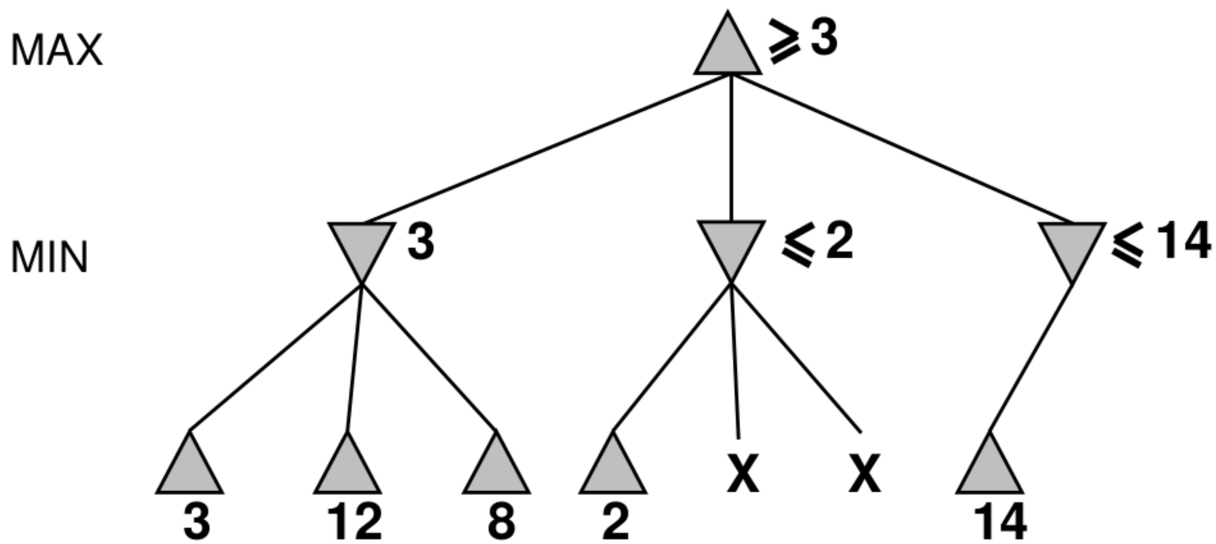


α–β pruning example

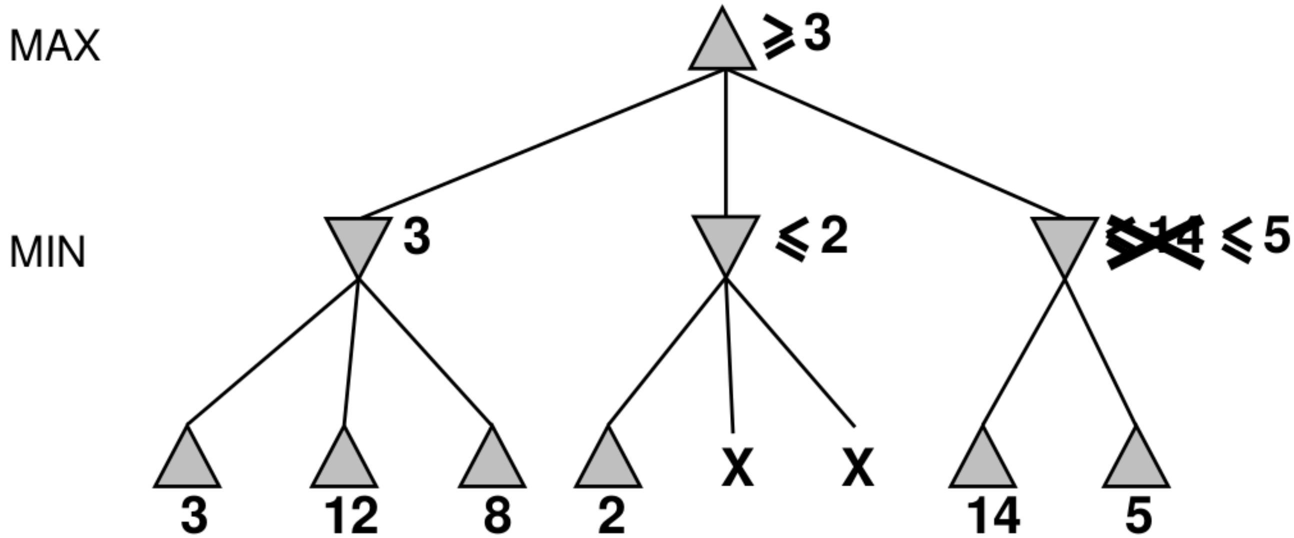


A would never choose C.

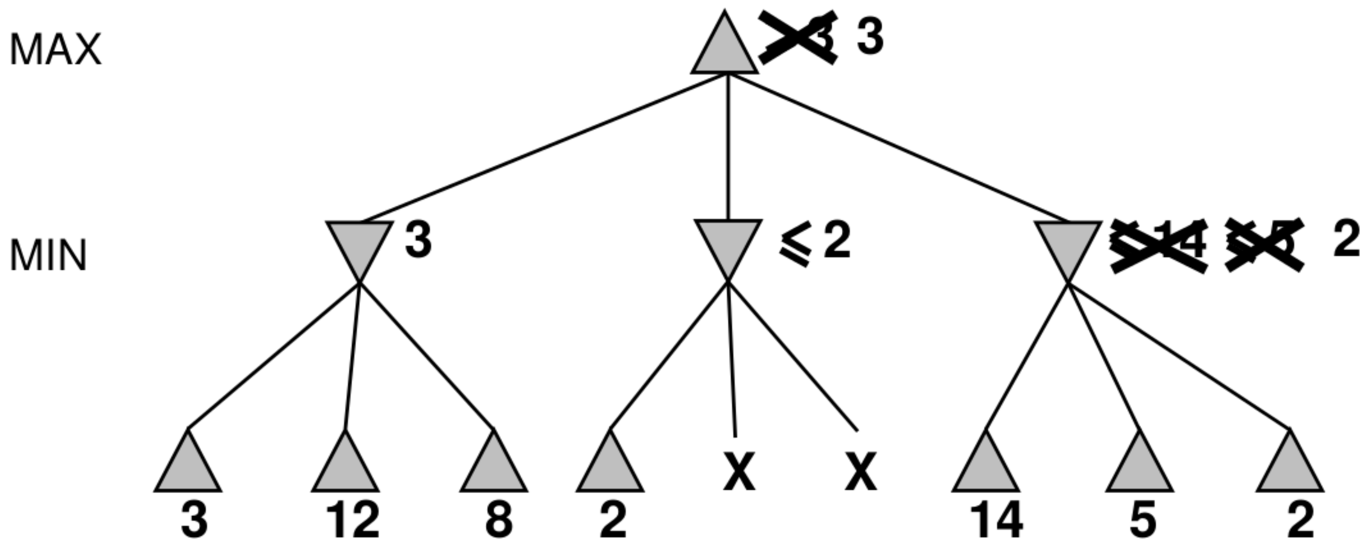
α–β pruning example



α–β pruning example



α–β pruning example



The α–β algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(a, *state*))

---

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

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

alpha: the value of the best (highest) choice found at any choice point along the path for MAX
beta: the value of the best (lowest) choice found at any choice point along the path for MIN

Properties of α–β

Pruning **does not** affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$
$\Rightarrow$ **doubles** solvable depth

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

Unfortunately, 35^{50} is still impossible!

Resource limits

Standard approach:

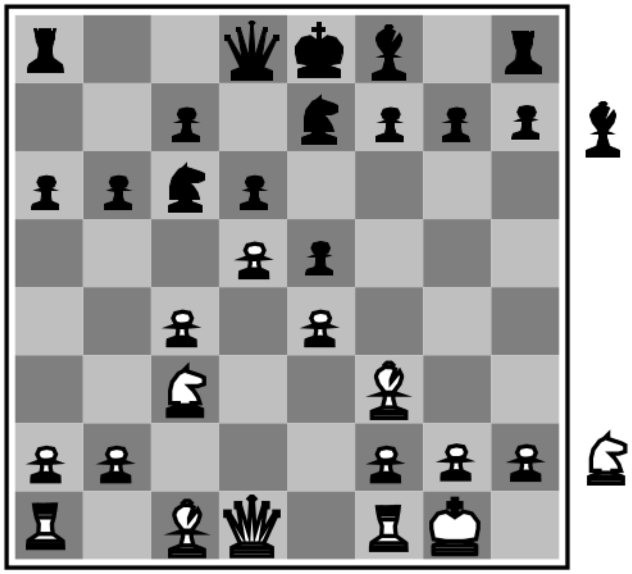
- Use CUTOFF-TEST instead of TERMINAL-TEST
  e.g., depth limit (perhaps add quiescence search)
- Use EVAL instead of UTILITY
  i.e., evaluation function that estimates desirability of position

Suppose we have 100 seconds, explore 10^4 nodes/second
$\Rightarrow$ 10^6 nodes per move $\approx 35^{8/2}$
$\Rightarrow$ α–β reaches depth 8 $\Rightarrow$ pretty good chess program

1- EVAL should order the terminal nodes the same way as utility
2- Computation time should be short
3- Should be strongly correlated with the actual chances of winning

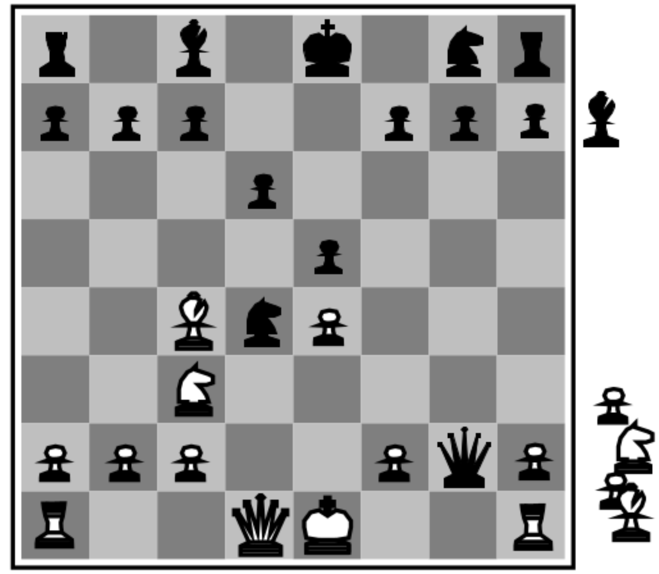
EVAL (x) = .. for the chess?
- find some categories, find expected value of winning for each category
.... too many categories, a lot of experience.
- what else?

Evaluation functions



Black to move

White slightly better



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.

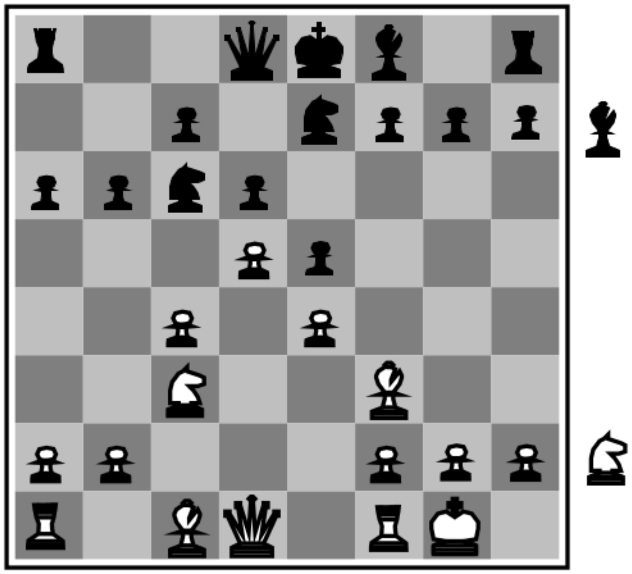
pawn: 1
knight or bishop: 3
rook: 5

Try to include the following information: a pair of bishops worth more than twice the value of the bishop
No rule information is included in the evaluation..

Where to cut-off?
fixed depth? more robust: iterative deepening until time limit?
apply only in quiescent positions: unlikely exhibit significant changes in the value.

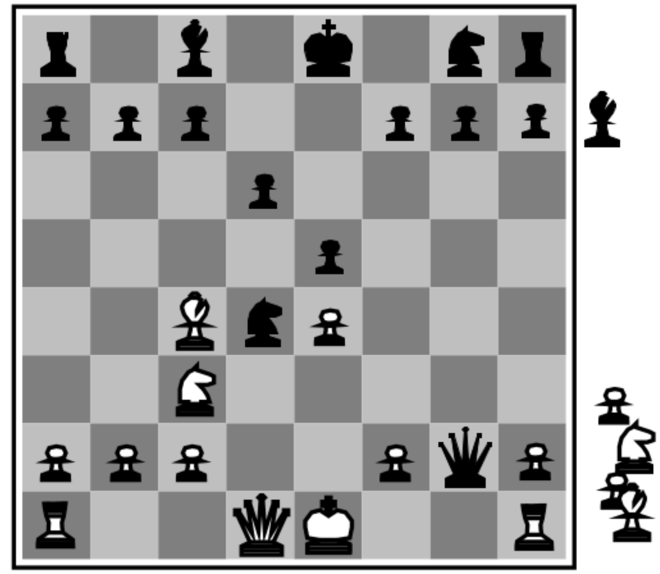
--> quiescence search

Evaluation functions



Black to move

White slightly better



White to move

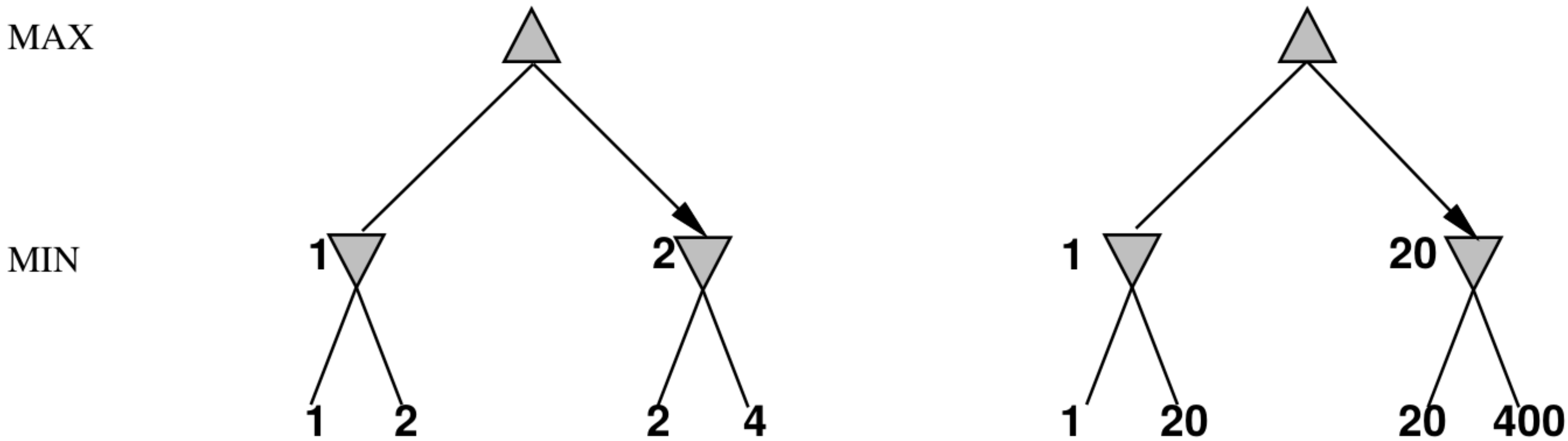
Black winning

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



Behaviour is preserved under any **monotonic** transformation of EVAL

Only the order matters:
payoff in deterministic games acts as an ordinal utility function

Deterministic games 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.

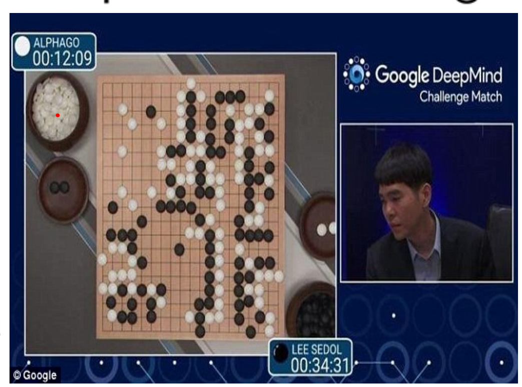
Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

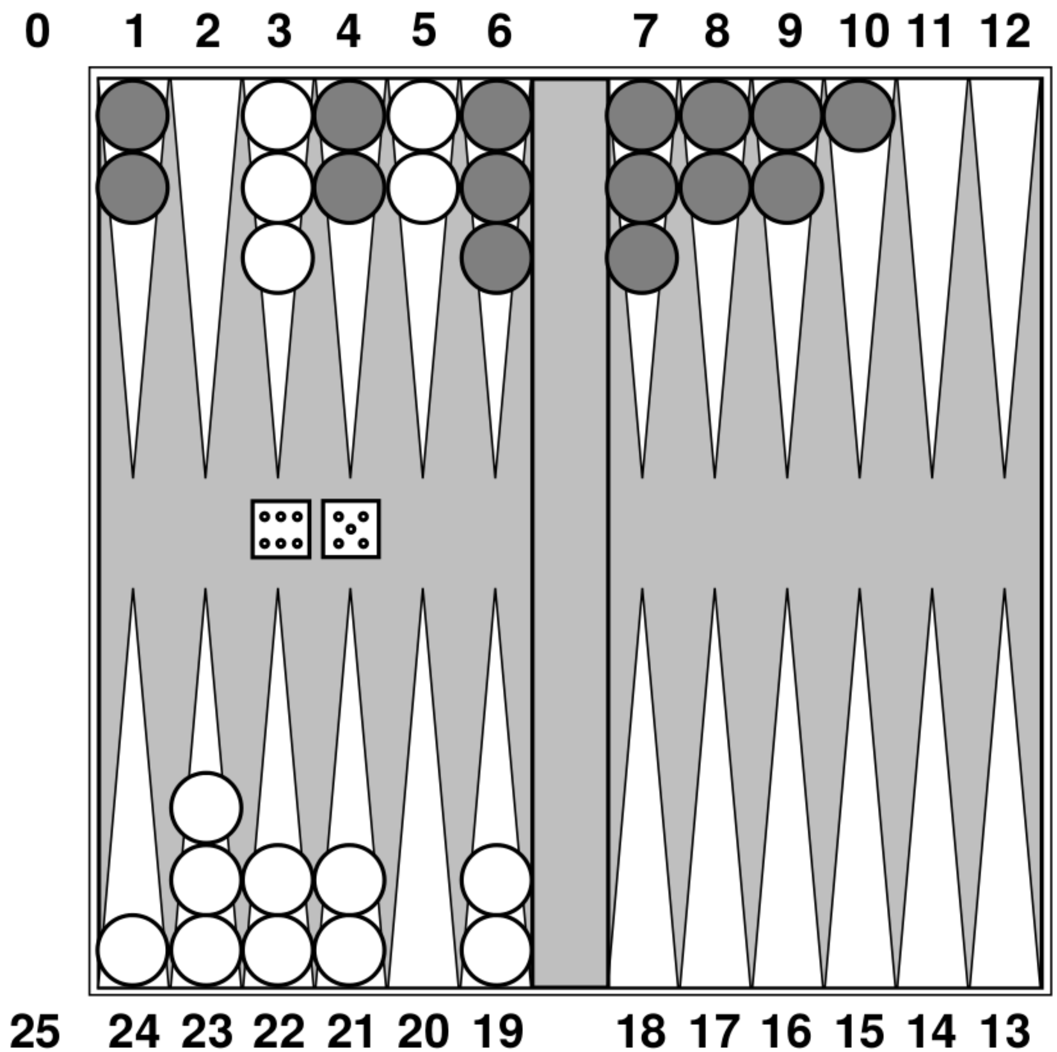
Not anymore..

AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol

DeepMind's artificial intelligence astonishes fans to defeat human opponent and offers evidence computer software has mastered a major challenge

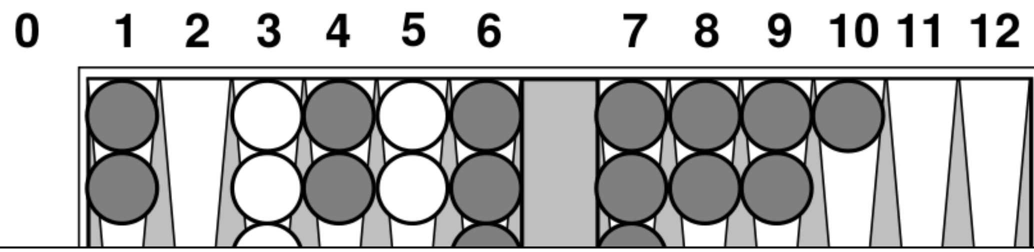


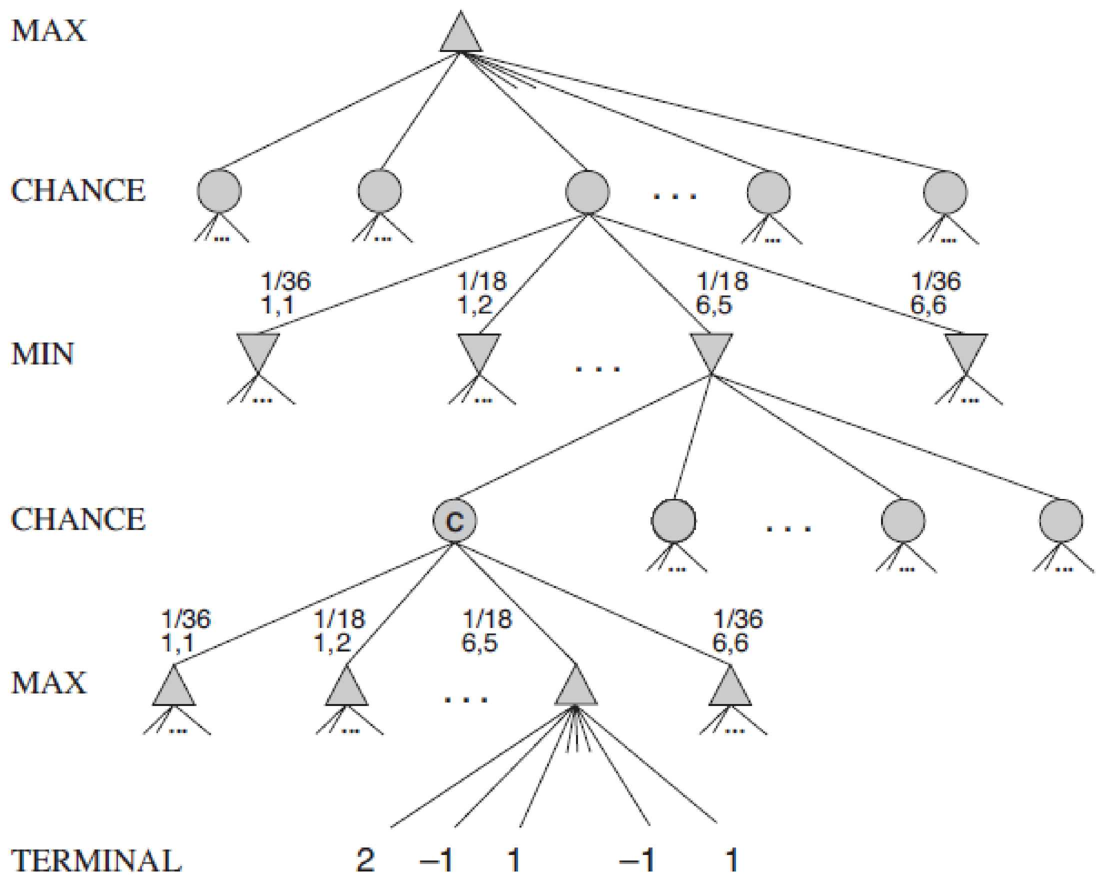
Nondeterministic games: backgammon



Combine luck and skill!
How is MIN-MAX tree handled?

Nondeterministic games: backgammon





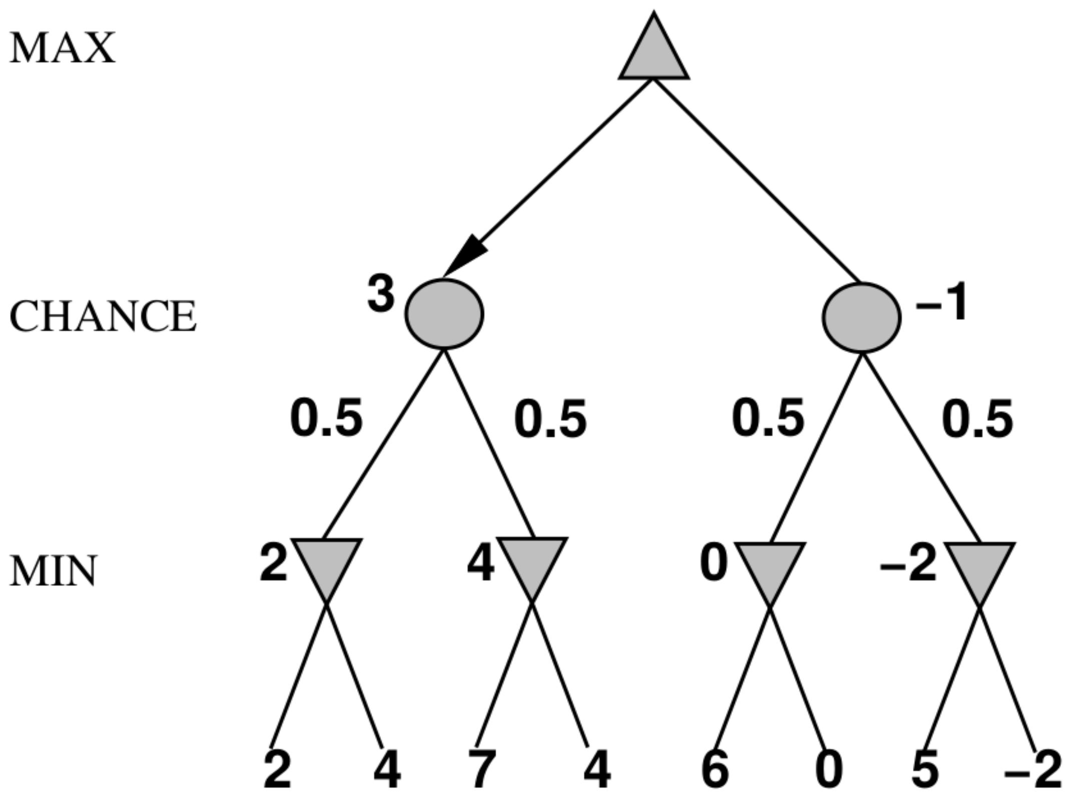
b=21

Figure 5.11 Schematic game tree for a backgammon position.

Nondeterministic games in general

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

Simplified example with coin-flipping:



how should we adapt the minimax algorithm?

max: return highest
min: return lowest
chance: ?

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)
. . .
```

Nondeterministic games in practice

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

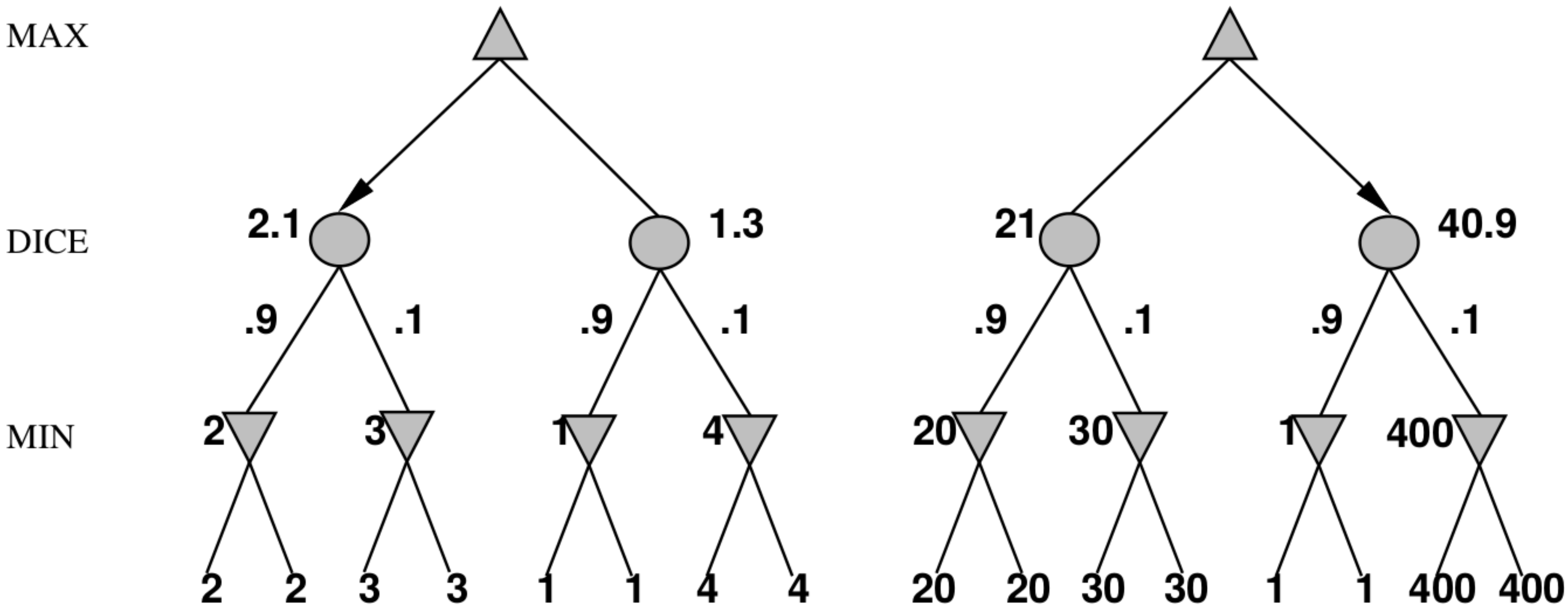
As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

TDGammon uses depth-2 search + very good Eval
$\approx$ world-champion level

With minimax, satisfying the same ordering was sufficient.

Digression: Exact values DO matter



Behaviour is preserved only by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff

Games of imperfect information

E.g., card games, where opponent's initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game*

Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals*

Special case: if an action is optimal for all deals, it's optimal.*

GIB, current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average