Outline

◇ Perfect play

◇ Resource limits

◇ α–β pruning

◇ Games of chance

◇ Games of imperfect information

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

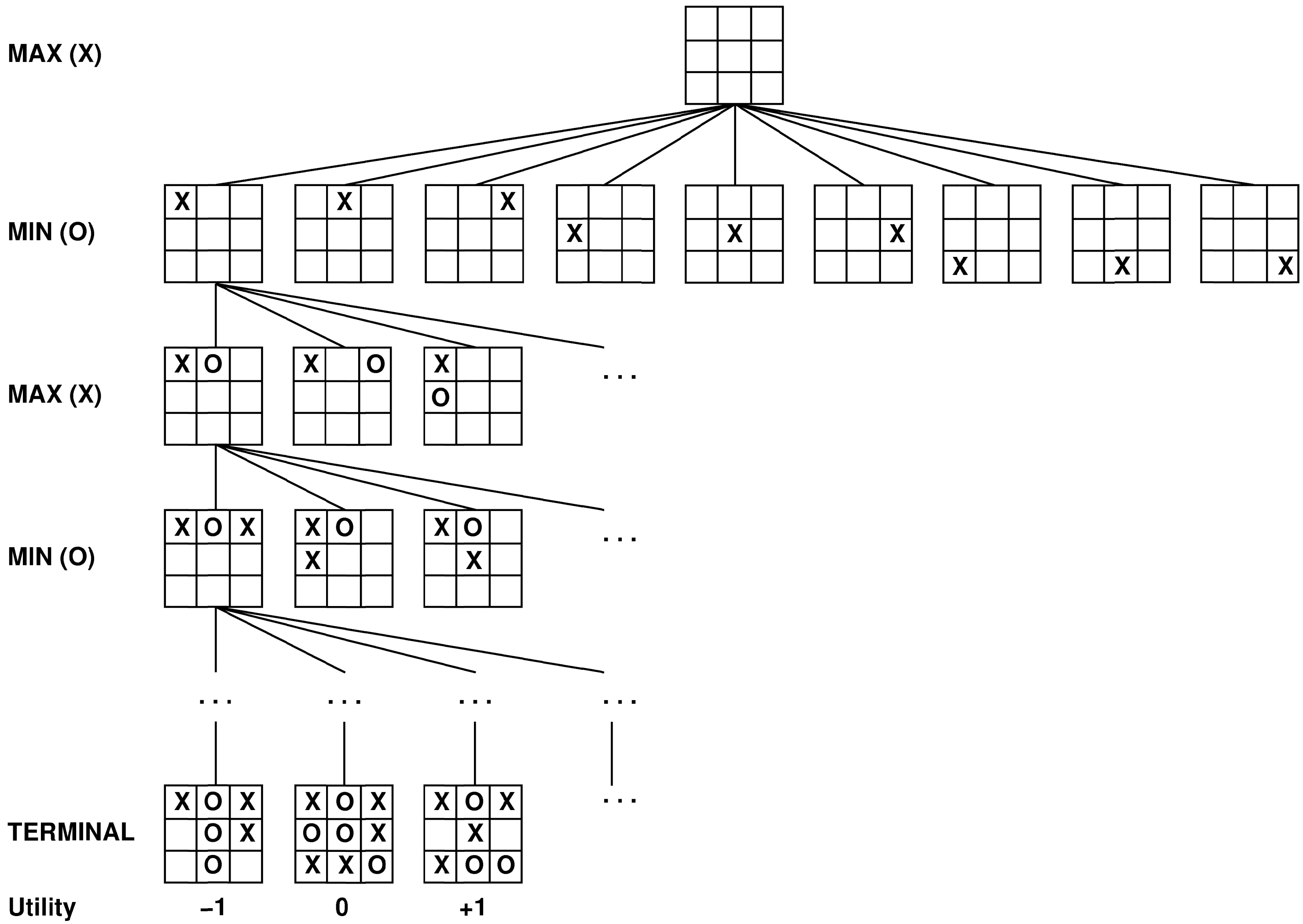
Plan of attack:

- 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)

Types of games

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

Game tree (2-player, deterministic, turns)

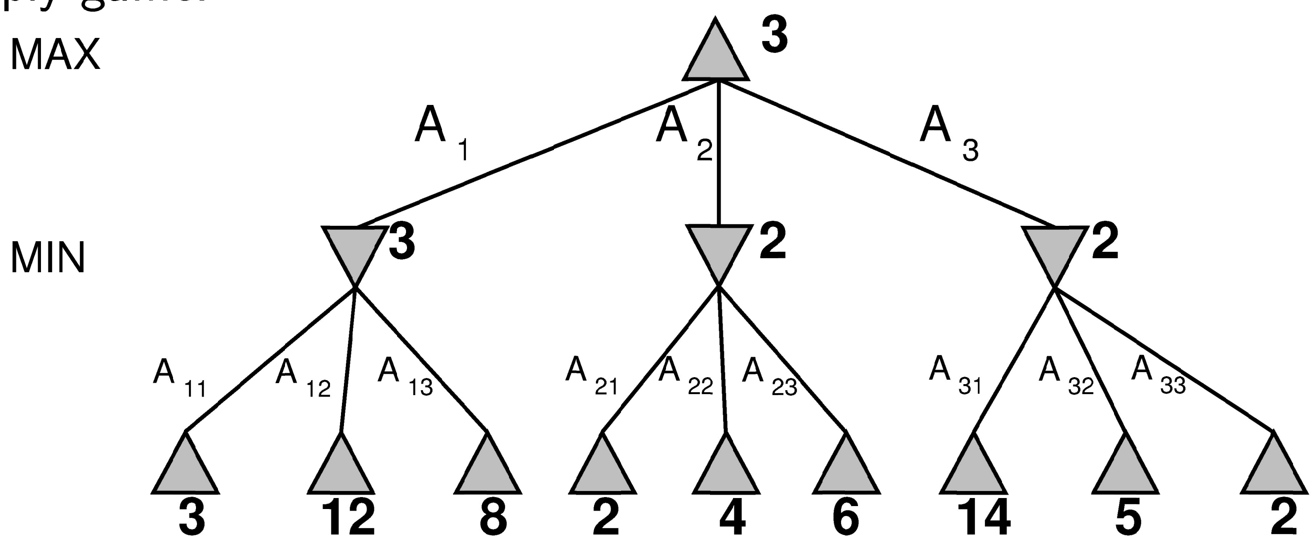


Minimax

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:



Minimax algorithm

```
function MINIMAX-DECISION(state, game) returns an action
   action, state ← the a, s in SUCCESSORS(state)
            such that MINIMAX-VALUE(s, game) is maximized
   return action
---------------------------------------------------------------
function MINIMAX-VALUE(state, game) returns a utility value
   if TERMINAL-TEST(state) then
      return UTILITY(state)
   else if MAX is to move in state then
      return the highest MINIMAX-VALUE of SUCCESSORS(state)
   else
      return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

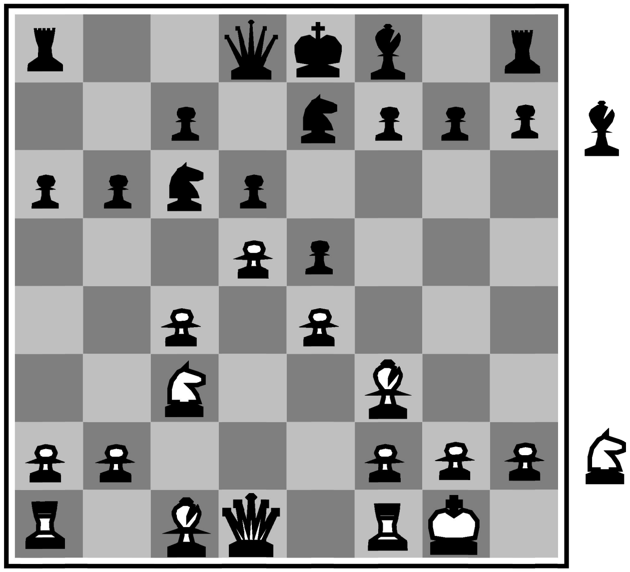
Resource limits

Suppose we have 100 seconds, explore 10^4 nodes/second
$\Rightarrow 10^6$ nodes per move

Standard approach:

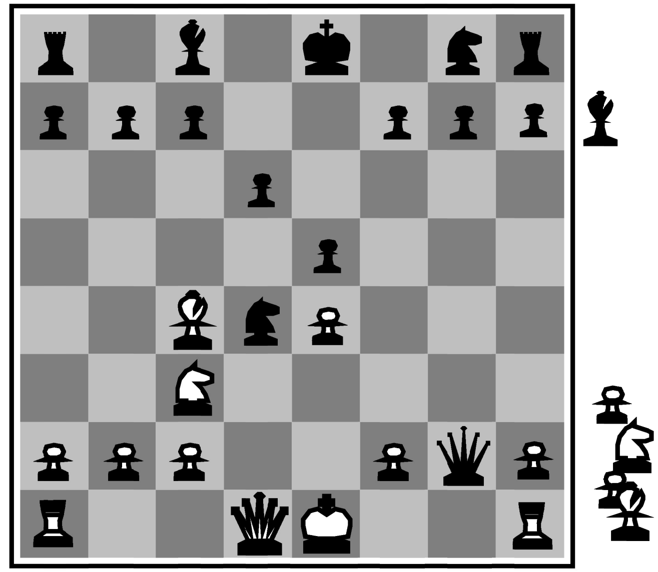
- *cutoff test*
  e.g., depth limit (perhaps add *quiescence search*)
- *evaluation function*
  $=$ estimated desirability of position

Evaluation functions



Black to move

White slightly better



White to move

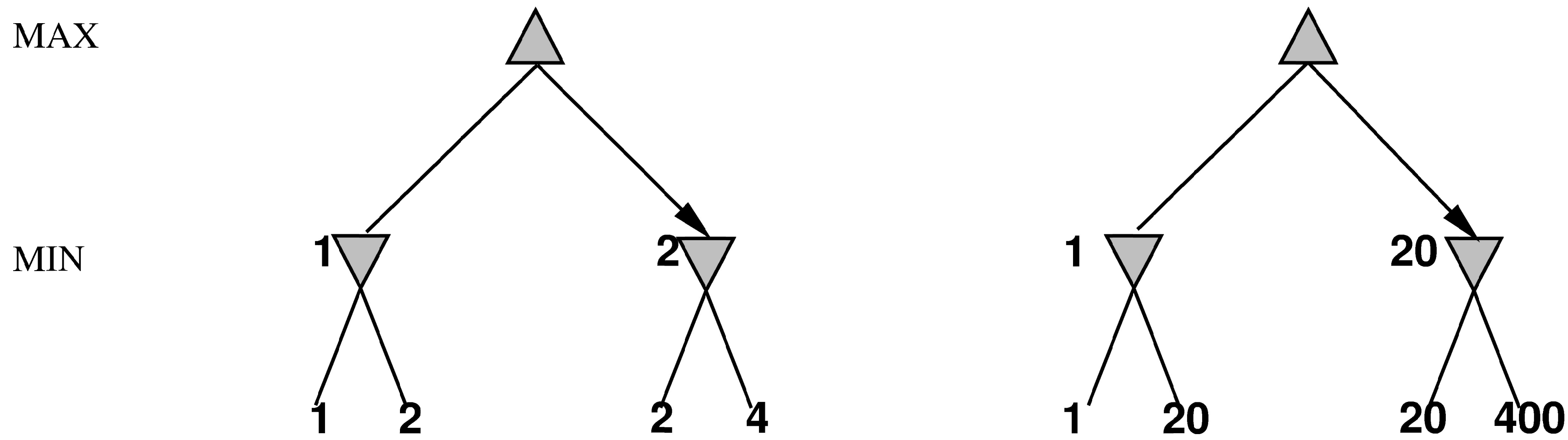
Black winning

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

Cutting off search

MinimaxCutoff is identical to MinimaxValue except

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

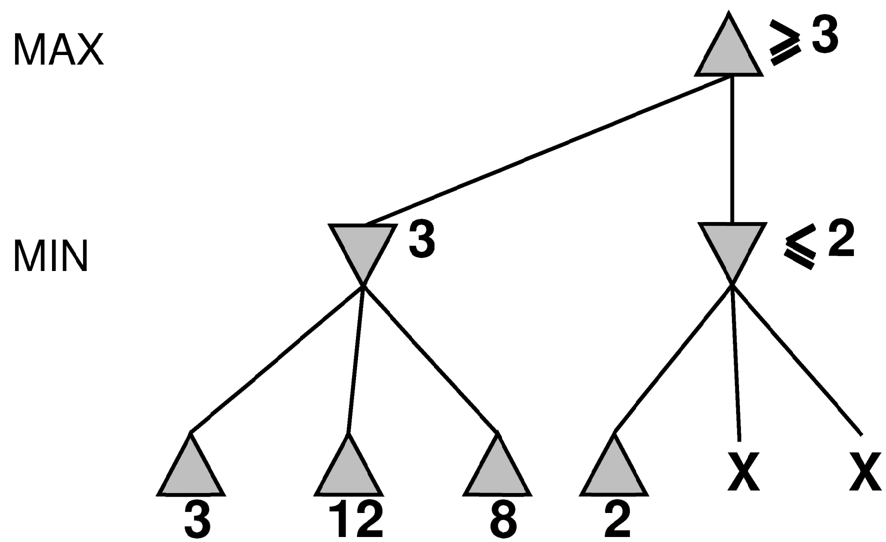
Does it work in practice?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

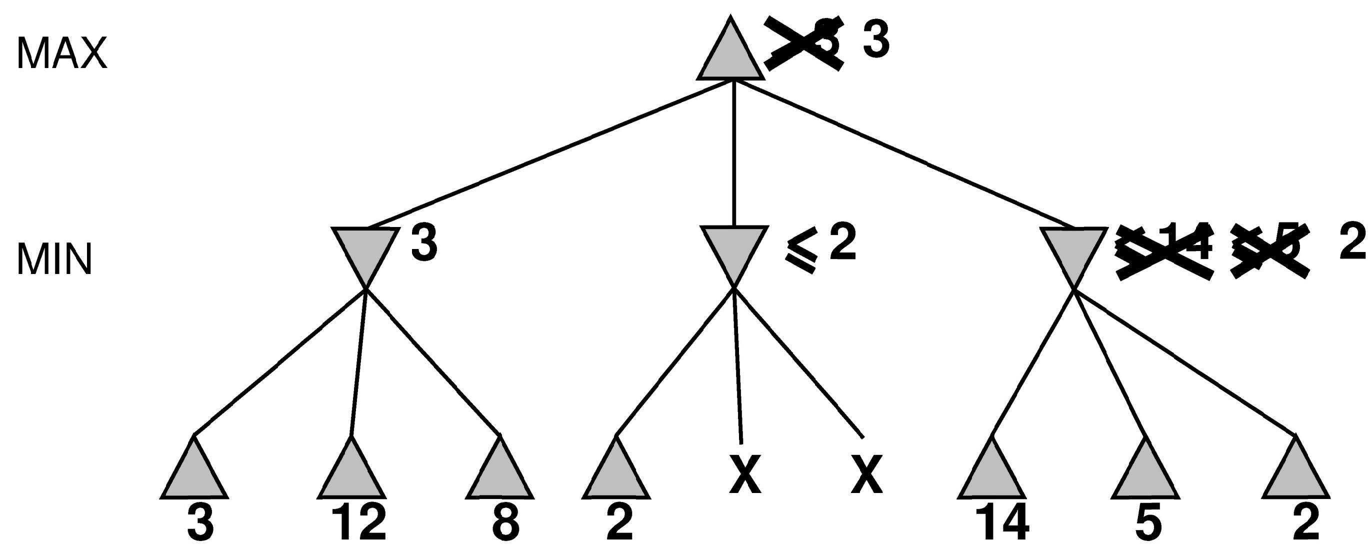
4-ply lookahead is a hopeless chess player!

4-ply $\approx$ human novice
8-ply $\approx$ typical PC, human master
12-ply $\approx$ Deep Blue, Kasparov

α–β pruning example



α–β pruning example



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* depth of search
$\Rightarrow$ 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 *metareasoning*)

Deterministic games in practice

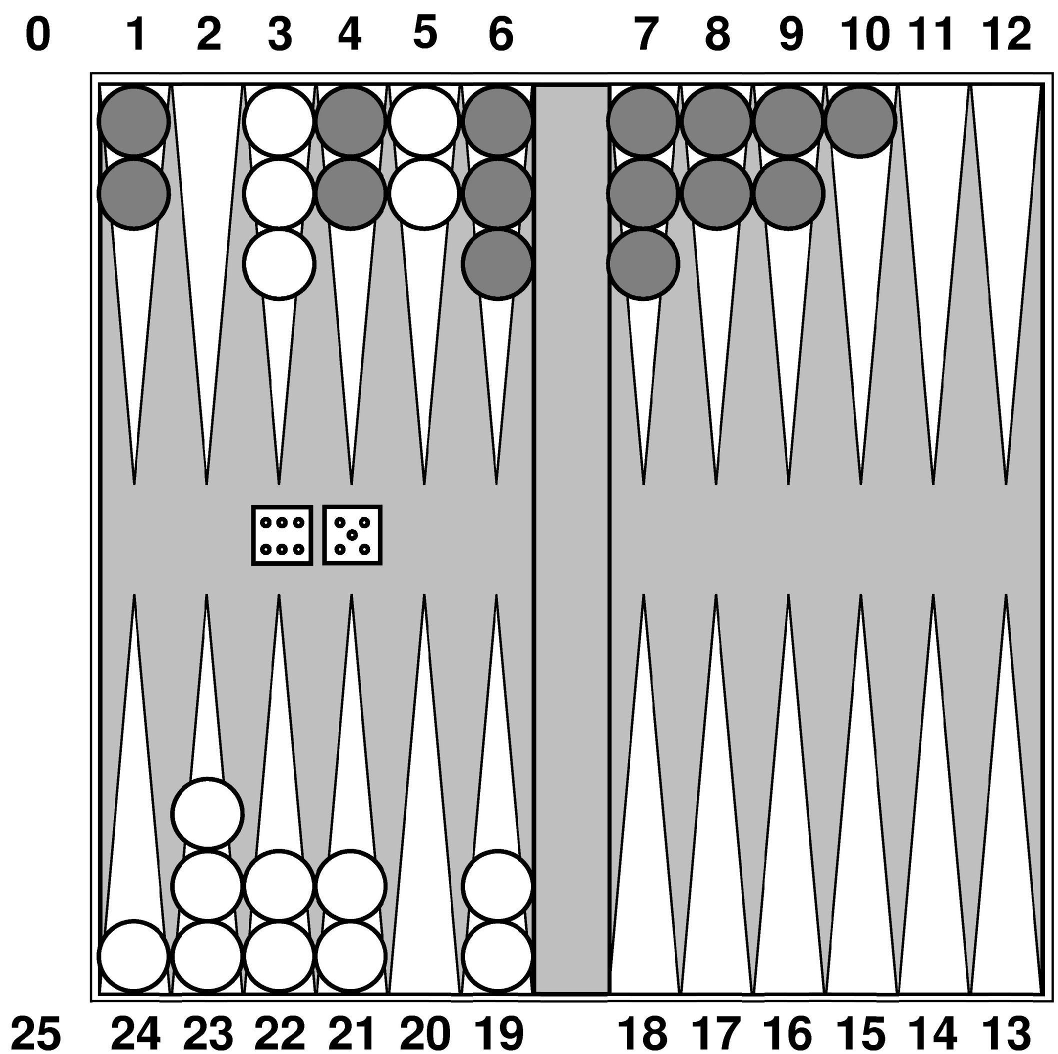
Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

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.

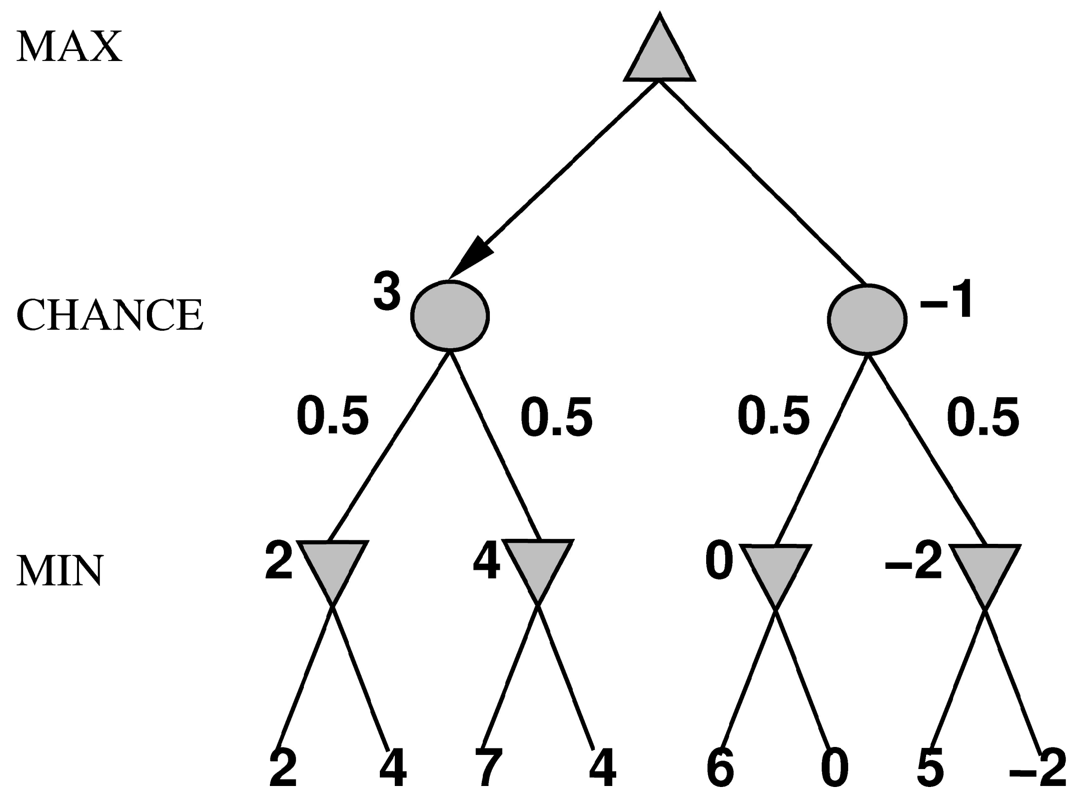
Nondeterministic games: backgammon



Nondeterministic games in general

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

Simplified example with coin-flipping:



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

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*

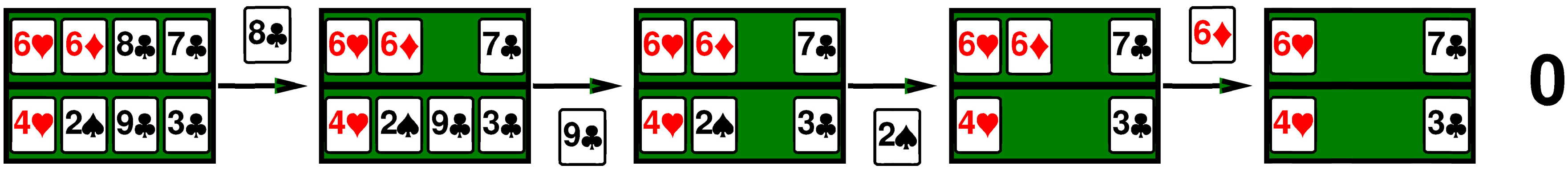
Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals*

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

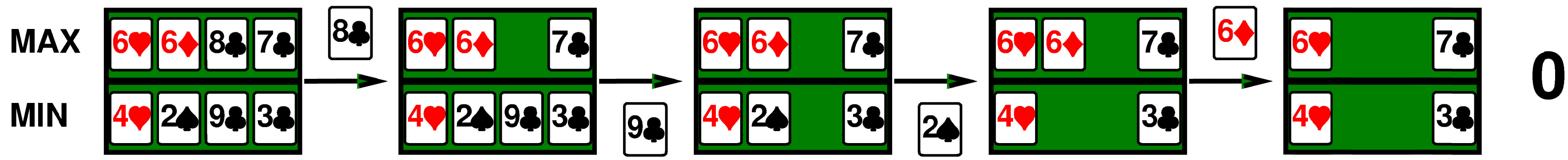
Example

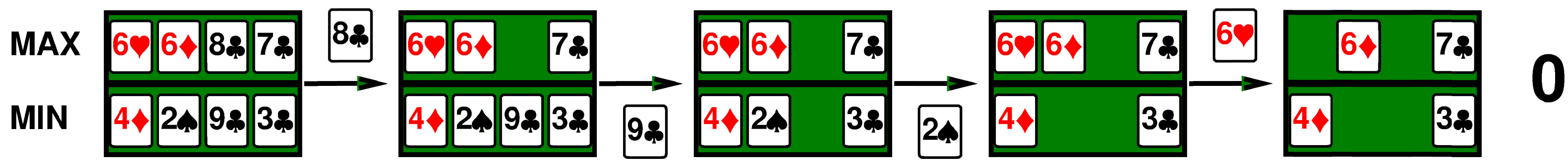
Four-card bridge/whist/hearts hand, MAX to play first



Example

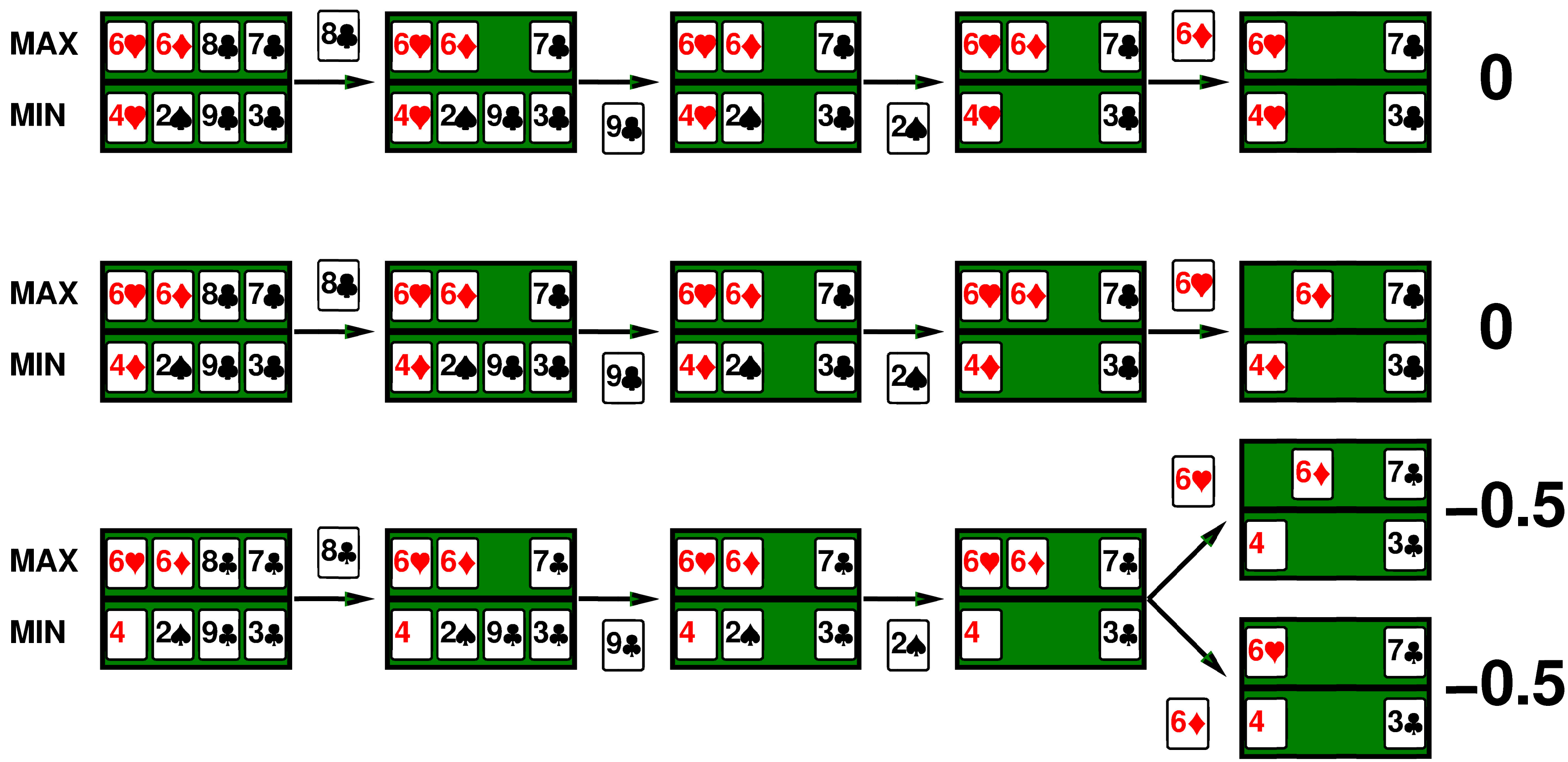
Four-card bridge/whist/hearts hand, MAX to play first





Example

Four-card bridge/whist/hearts hand, MAX to play first



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

◇ uncertainty constrains the assignment of values to states

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