GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

E016350 - Artificial Intelligence

Lecture 14

Solving problems by searching

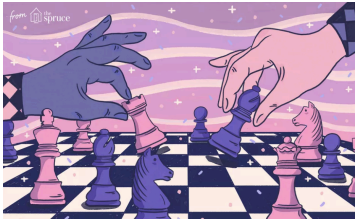
Game playing

Aleksandra Pizurica

Ghent University
Fall 2024

Overview

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



These slides are based on: S. Russel and P. Norvig: *Artificial Intelligence: A Modern Approach*, (Fourth Ed.), http://aima.cs.berkeley.edu/ and the slides of D. Klein and P. Abbeel (course Introduction to Artificial Intelligence), http://ai.berkeley.edu/

Games as a playground of AI

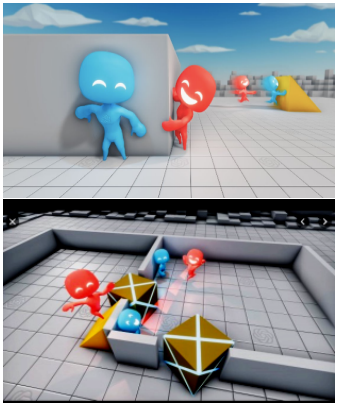
Games are interesting for AI because:

- They are hard to solve. E.g.,
  - Chess: average branching factor $b = 35$; often 50 moves per player $35^{100} \approx 10^{154}$ nodes (out of these, about 10^{40} distinct nodes)
  - Go: far more complex ($b \approx 250$, $d \approx 150$)
- A decision has to be made even when calculating the optimal decision is infeasible $\rightarrow$ ideas to make best possible use of time
- Inefficiency is often penalized severely (like in the real world)

Lee Sedol playing against AlphaGo (2016)

Games vs. search problems

- Multiagent environments – each agent needs to consider the actions of other agents and how they affect its welfare
- The **unpredictability of other agents** $\rightarrow$ **contingencies** in problem solving
- Competitive environments = agents' goals in conflict $\rightarrow$ **adversarial search**
- Mathematical game theory views any multiagent environment as a game (no matter if cooperative or competitive agents)

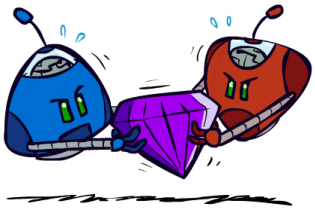


OpenAI: multiagent hide & seek

Types of games

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

Zero-sum games



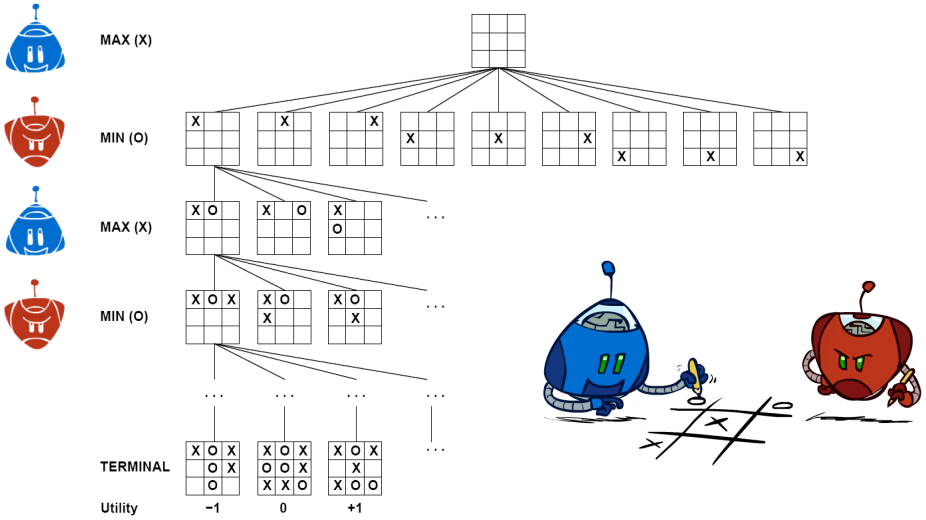
Zero-Sum Games

- Agents have opposite utilities (values on outcomes)
- Think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

General Games

- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible

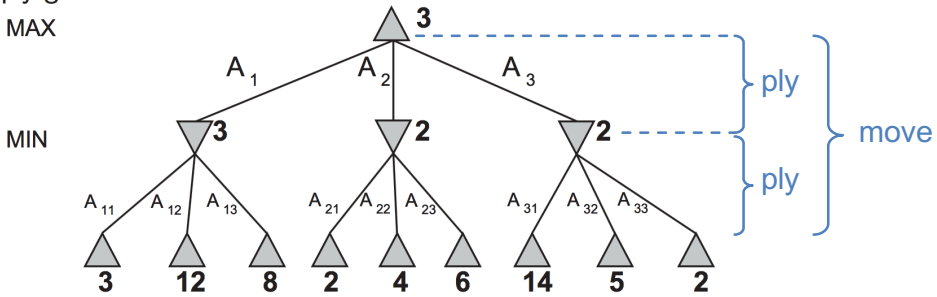
Game tree



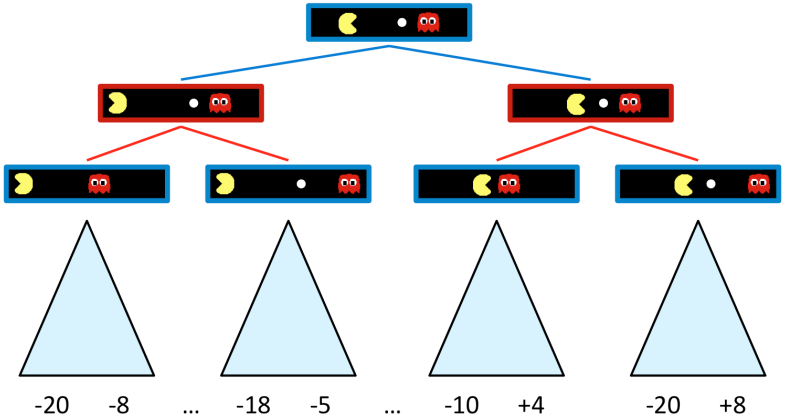
Minimax

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

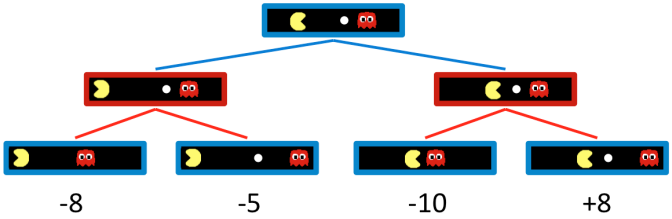
E.g., 2-ply game:



Example: Pacman game tree



Example: Pacman game tree



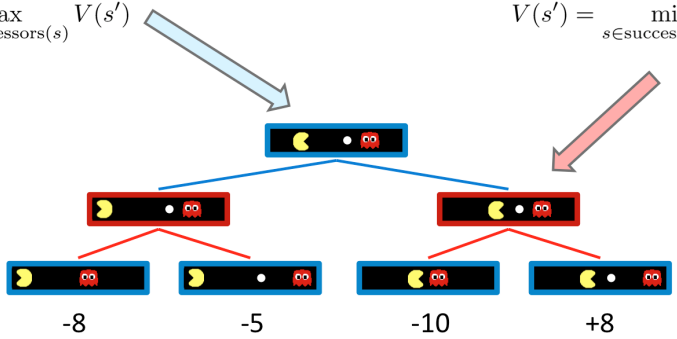
Example: Pacman game tree and minimax

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

$$V(s) = \text{known}$$

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*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX(v, MIN-VALUE(s))
  return v

function MIN-VALUE(*state*) **returns** *a utility value*
  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))
  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 (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??
Time 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??

b – branching factor; m – maximum depth of the game tree

Properties of minimax

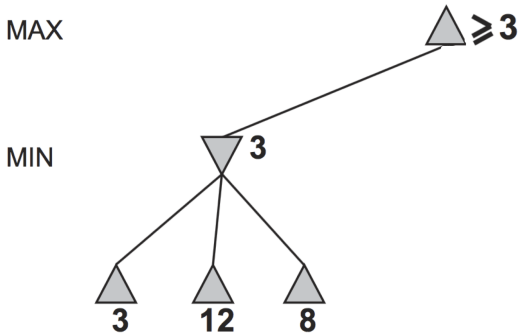
Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??
Time complexity?? $O(b^m)$
Space complexity?? $O(bm)$ (depth-first exploration)
For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games

$\Rightarrow$ exact solution completely infeasible

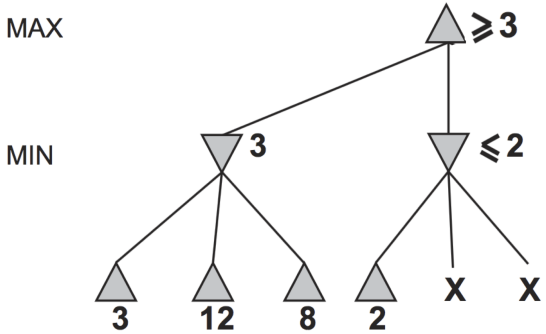
But do we need to explore every path?

b – branching factor; m – maximum depth of the game tree

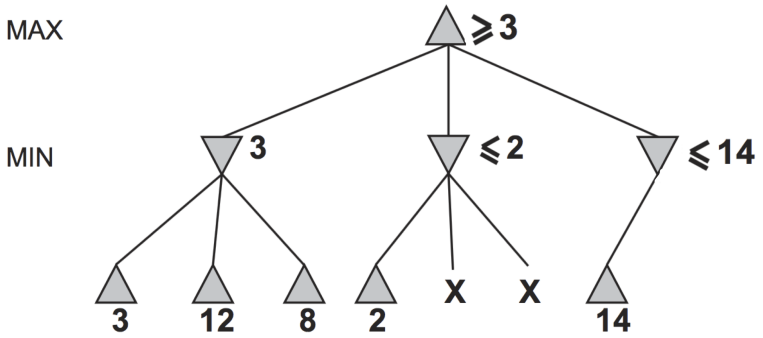
α–β pruning example



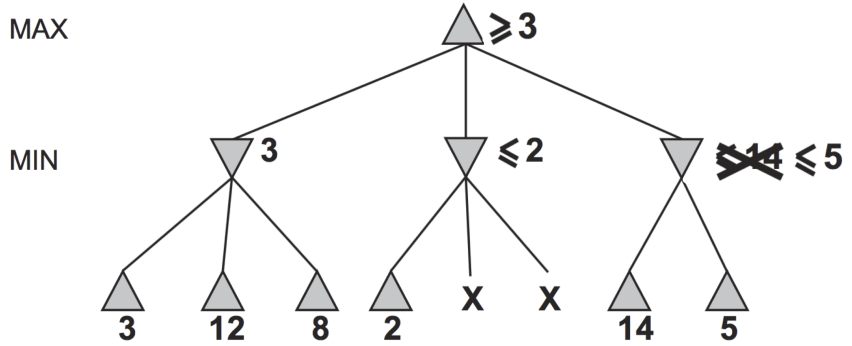
α–β pruning example



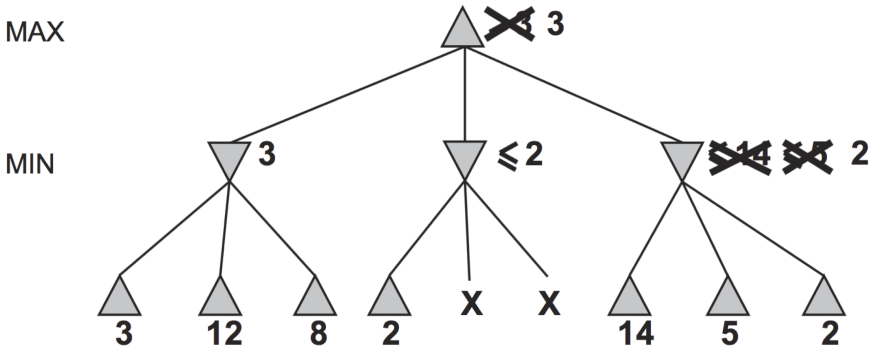
α–β pruning example



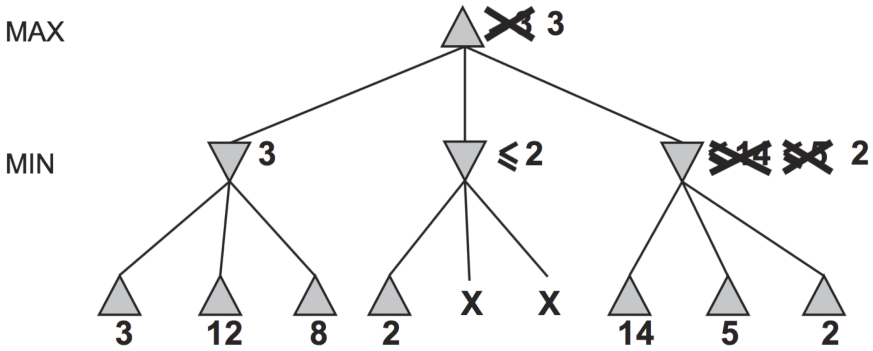
The α–β algorithm



α–β pruning example

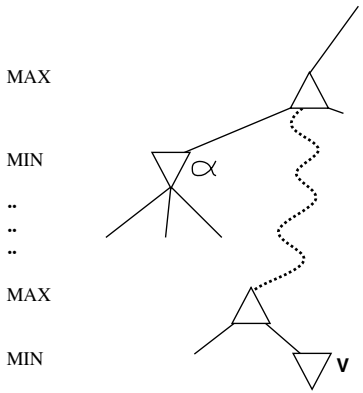


α–β pruning example



$$Minimax(root) = \max(\min(3,12,8), \min(2,x,y), \min(14,5,2))$$
$$= \max(3, \underbrace{\min(2,x,y)}_{\leq 2}, 2) = 3$$

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 $\Rightarrow$ prune that branch
Define β similarly for MIN

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 ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← 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
```

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!

Dynamic move ordering (try first the moves that were best in the past – 'killer moves')

Resource limits

Standard approach (refer to the $\alpha - beta$ algorithm):

- 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

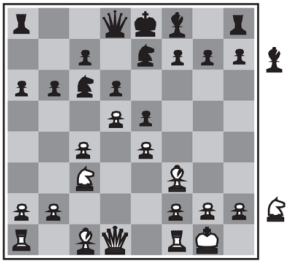
Suppose we have 100 seconds per move, explore 10^4 nodes/second

$\Rightarrow 10^6$ nodes per move $\approx 35^{8/2}$

$\Rightarrow \alpha$–β reaches depth 8 $\Rightarrow$ pretty good* chess program

*average chess player: $6 - 8$ plies; expert level ~ 10 plies; grand master > 40 plies (i.e., grand master can calculate > 20 moves ahead)

Evaluation functions

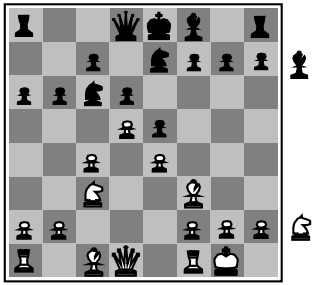


E.g., in chess, consider

- relative values of pieces (queen: 9, rook: 5, bishop: 3, knight: 3, pawn: 1)
- positions of pieces (mobility, protection, rooks should be placed on open files)
- pawn structure (backward, isolated and doubled pawns are weak)
- king safety

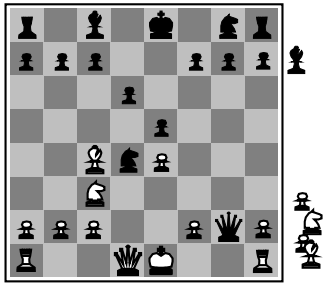
Claude Shannon (1950): Programming a computer for playing chess

Evaluation functions



Black to move

White slightly better



White to move

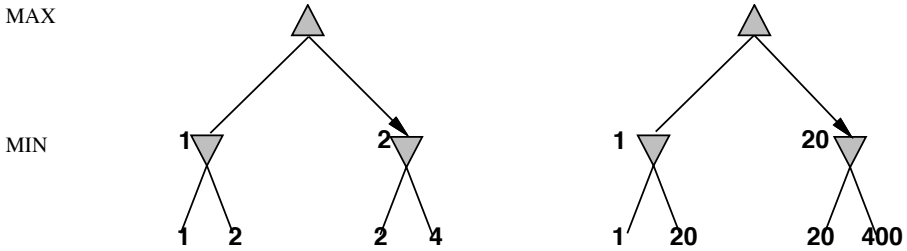
Black winning

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

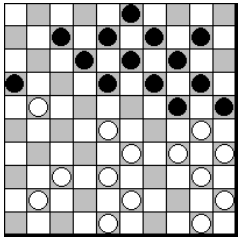
Checkers ($\sim 5 \times 10^{20}$ positions): *Chinook* ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions with 8 or fewer pieces on the board, a total of 443,748,401,247 positions.


TECHNOLOGY

How Checkers Was Solved

The story of a duel between two men, one who dies, and the nature of the quest to build artificial intelligence

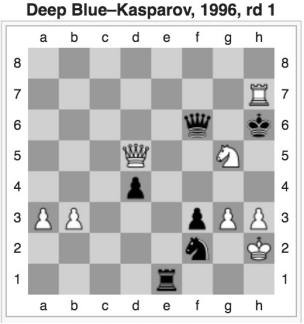
ALEXIS C. MADRIGAL JUL 19, 2017



J. Schaeffer et al: Checkers is Solved, Science, 2007.

Deterministic games in practice: Chess

Chess: *Deep Blue* (IBM) defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue used very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

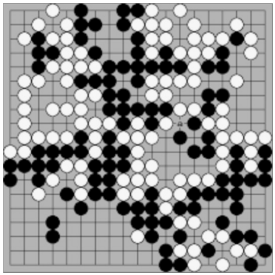


M. Campbell et al: Deep Blue, Artificial Intelligence, 2002.
Google DeepMind: AlphaZero (2017)

Deterministic games in practice: Go

Go: *AlphaGo* (Google DeepMind) defeated a 9-dan master and 18-time world champion Lee Sedol in 2016.

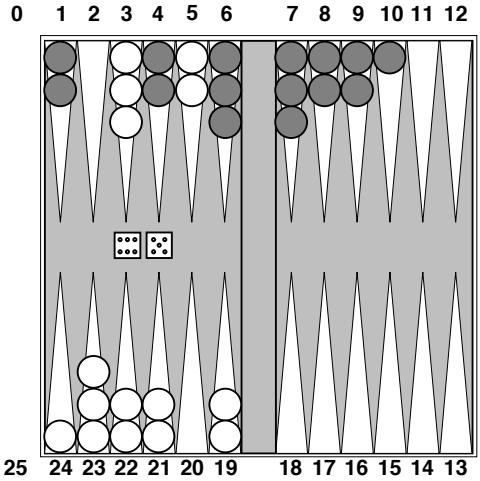
- Go is exemplary of many difficulties faced by AI:
  - Challenging decision-making task
  - Intractable search space
  - Complex optimal solution - cannot be approximated directly by value functions.
- Main Challenges:
  - Enormous combinatorial complexity and
  - Long term influence of moves.
- *AlphaGo* combines Monte Carlo tree search (MCTS) with deep learning



A strategy game, 19x19 grid.
Goal: surround more territory than the opponent

D. Silver et al: Mastering the game of Go with deep neural networks and tree search, Nature, 2016.

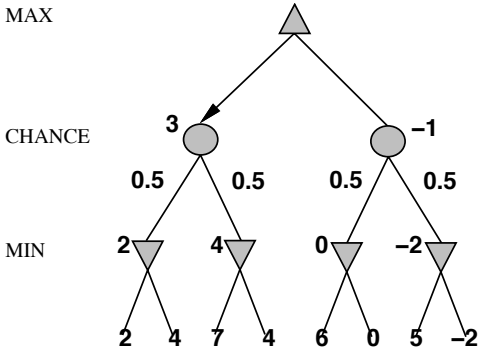
Nondeterministic games: backgammon



Nondeterministic games in general

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

Simplified example with coin-flipping:



Can only estimate the expected value of a state → **expecti-minimax** strategy, also called **expectimax** strategy

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*)

...

Algorithm for nondeterministic games

Time complexity of expectiminimax $O(b^m n^m)$, where n is the number of distinct rolls

If the rolls would be known in advance for the rest of the game $O(b^m)$

But then no chance involved! Expectimax considers all possible dice-rolls

Dice rolls increase the effective branching factor: 21 possible rolls with 2 dice

Backgammon $\approx$ 20 legal moves (can be even 4000 with dice rolls that are doubles)

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

As depth increases, probability of reaching a given node shrinks

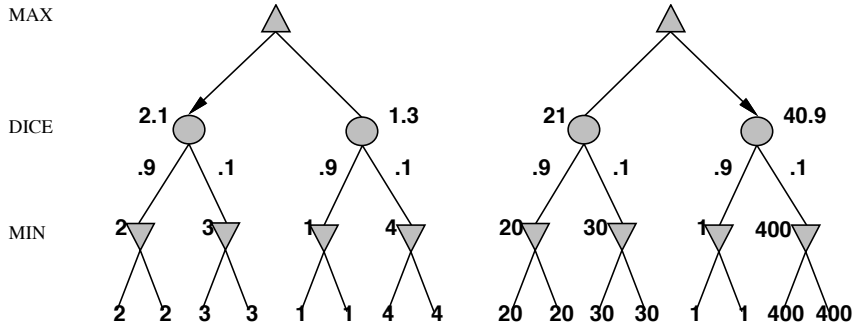
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

TDGammon uses depth-2 search + very good Eval

$\approx$ world-champion level

Digression: Exact values DO matter



An order-preserving transformation on leaf values can change the best move!
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

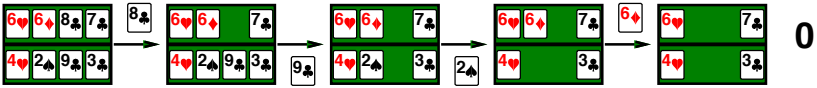
Seems just like having one big dice roll at the beginning of the game*
Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals*

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

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

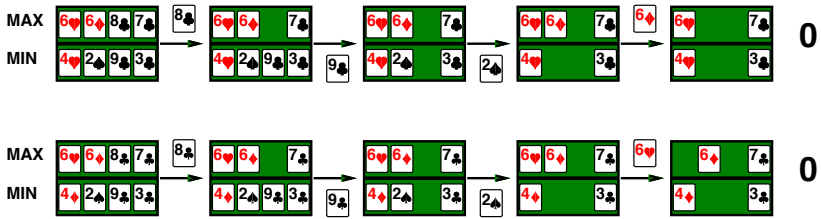
Games of imperfect information

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



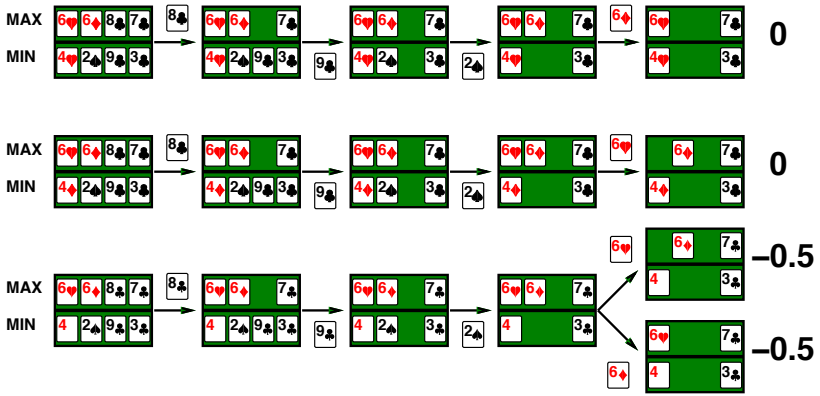
Games of imperfect information

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



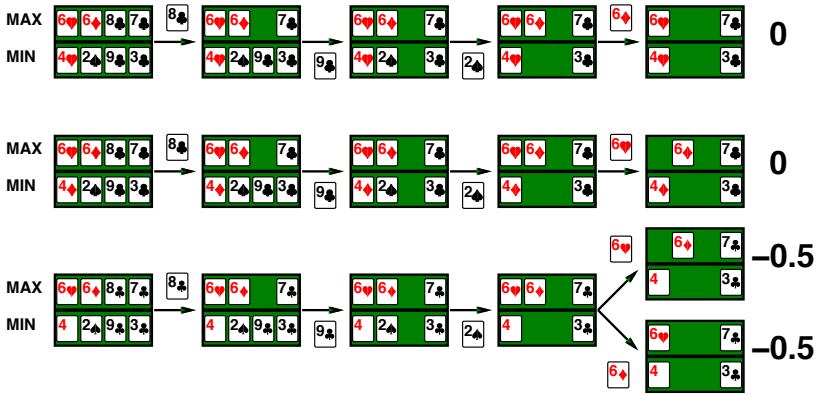
Games of imperfect information

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



Games of imperfect information

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

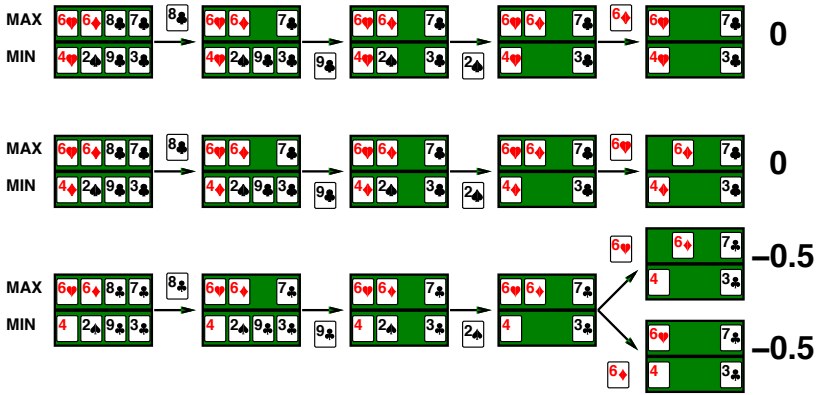


Suppose each deal s occurs with probability $P(s)$. Then the move we want is:

$$\arg\max_a \sum_s P(s)\textsc{Minimax}(\textsc{Result}(s,a))$$

Games of imperfect information

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



Monte Carlo approximation (with a representative random sample of N deals):

$$\arg\max_a \frac{1}{N}\sum_{i=1}^{N} \textsc{Minimax}(\textsc{Result}(s_i, a))$$

Does averaging over clairvoyance always work?

DAY 1

Road A leads to a small heap of gold pieces
Road B leads to a fork:
take the left fork and you'll find a mound of jewels;
take the right fork and you'll be run over by a bus.

DAY 2

Road A leads to a small heap of gold pieces
Road B leads to a fork:
take the left fork and you'll be run over by a bus;
take the right fork and you'll find a mound of jewels.

DAY 3

Road A leads to a small heap of gold pieces
Road B leads to a fork:
guess correctly and you'll find a mound of jewels;
guess incorrectly and you'll be run over by a bus.

Proper analysis

* Intuition that the value of an action is the average of its values in all actual states is **WRONG**

With partial observability, value of an action depends on the information state or belief state the agent is in

The agent can generate and search a tree of information states

Leads to rational behaviors such as

- Acting to obtain information
- Signalling to one's partner
- Acting randomly to minimize information disclosure

Summary

Games are fun to work on! (and dangerous)
They illustrate several important points about AI

- perfection is unattainable $\Rightarrow$ must approximate
- good idea to think about what to think about
- uncertainty constrains the assignment of values to states
- optimal decisions depend on information state, not real state

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