Lecture 5: Game Playing (Adversarial Search)

CS 580 (001) - Spring 2018

Amarda Shehu

Department of Computer Science
George Mason University, Fairfax, VA, USA

February 21, 2018

Search in a multi-agent, competitive environment → Adversarial Search/Game Playing

Mathematical game theory treats any multi-agent environment as a game, with possibly co-operative behaviors (study of economies)

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

Most games studied in AI:

deterministic, **turn-taking**, **two-player**, **zero-sum** games of **perfect information**

Most games studied in AI:
deterministic, **turn-taking**, **two-player**, **zero-sum** games of **perfect information**

zero-sum: utilities of the two players sum to 0 (no win-win)
deterministic: precise rules with known outcomes
perfect information: fully observable

Most games studied in AI:
deterministic, **turn-taking**, **two-player**, **zero-sum** games of **perfect information**

zero-sum: utilities of the two players sum to 0 (no win-win)
deterministic: precise rules with known outcomes
perfect information: fully observable

Search algorithms designed for such games make use of interesting general techniques (meta-heuristics) such as evaluation functions, search pruning, and more.

However, games are to AI what grand prix racing is to automobile design.

Most games studied in AI:
deterministic, **turn-taking**, **two-player**, **zero-sum** games of **perfect information**

zero-sum: utilities of the two players sum to 0 (no win-win)
deterministic: precise rules with known outcomes
perfect information: fully observable

Search algorithms designed for such games make use of interesting general techniques (meta-heuristics) such as evaluation functions, search pruning, and more.

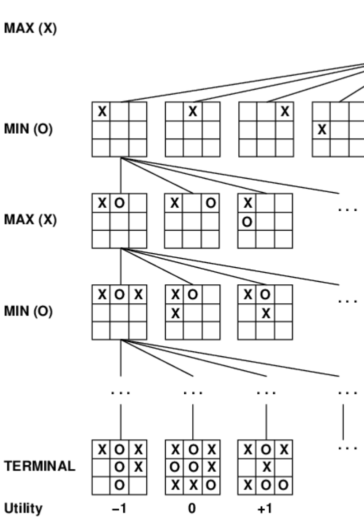
However, games are to AI what grand prix racing is to automobile design.

Our objective: study the three main adversarial search algorithms [minimax, alpha-beta pruning, and expectiminimax] and meta-heuristics they employ

Game Playing as a Search Problem

Two turn-taking agents in a zero-sum game: Max (starts game) and Min

Max's goal is to maximize its utility Min's goal is to minimize Max's utility



Game Playing as a Search Problem

Formal definition of a game as a search problem:

$S_0 \leftarrow$ initial state that specifices how game starts
PLAYER(s) $\leftarrow$ which player has move in state s
ACTIONS(s) $\leftarrow$ returns set of legal moves in state s
RESULT(s, a) $\leftarrow$ transition model that defines result of an action a on a state s
TERMINAL-TEST(s) $\leftarrow$ true on states that are game enders, false otherwise
UTILITY(s, p) $\leftarrow$ utility/objective function defines numeric value for game that ends in terminal state s with player p

Concept of game/search tree valid here

Chess: 35 moves per player $\rightarrow$ branching factor $b = 35$
ends at typically 50 moves $\rightarrow$ $m = 100$
search tree has $35^{100} \approx 10^{40}$ distinct nodes

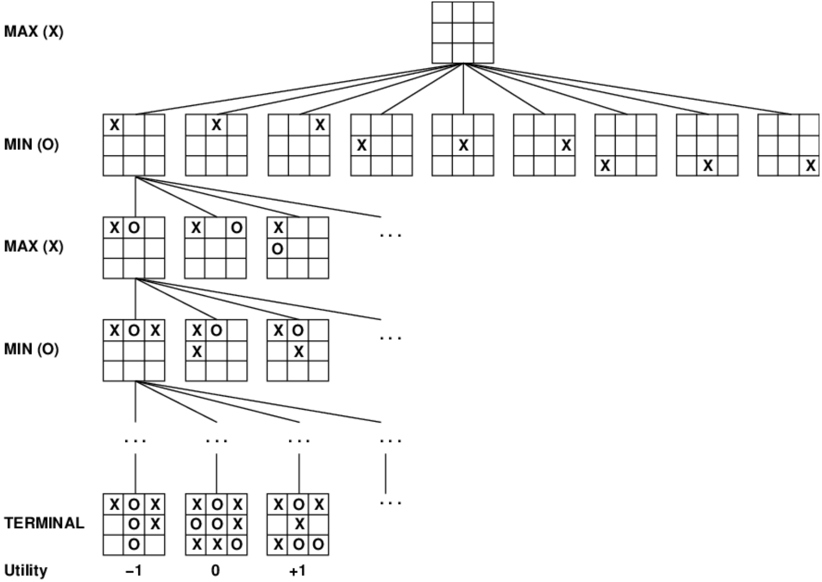
Pruning: how to ignore portions of tree without impacting strategy
Evaluation function: estimate utility of a state without a complete search

Some games too big search trees:
Time limits $\Rightarrow$ unlikely to find goal, must approximate
Many "tricks" (meta-heuristics) employed to look ahead

Early Obsession with Games before Term AI Coined

- 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)
- ...
- Today, Alphabet's deep learning team has a Go-playing program that beats world masters

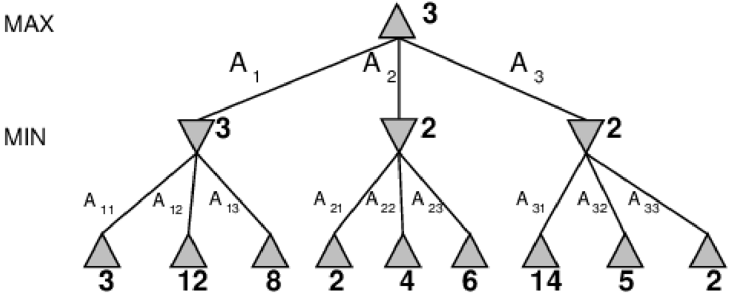
Game Tree (Two-player, Deterministic, Turns)



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:



function MINIMAX-VALUE(*state*) **returns** *minimax-value/utility*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  if NEXT AGENT IS MAX **then return** MAX-VALUE(*state*)
  if NEXT AGENT IS MIN **then return** MIN-VALUE(*state*)

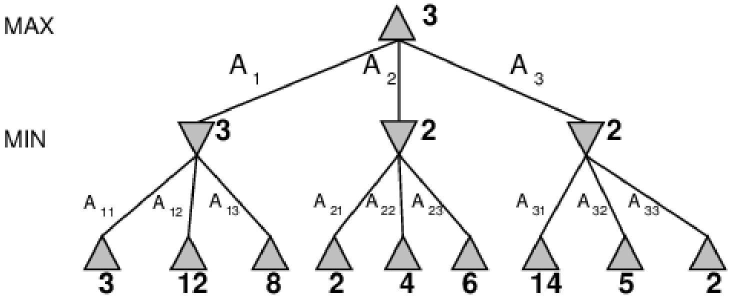
---

function MAX-VALUE(*state*) **returns** *a utility value*
  $v \leftarrow -\infty$
  for each successor of *state*
    do $v \leftarrow$ MAX(v, MINIMAX-VALUE(successor))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  $v \leftarrow \infty$
  for each successor of *state*
    do $v \leftarrow$ MIN(v, MINIMAX-VALUE(successor))
  return v

Class activity: trace Minimax-Value on 2-ply game below
update your v's!



Minimax Decision Algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\text{argmax}_{a \in \text{ACTIONS}}$ MIN-VALUE(RESULT(*state*, *a*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a* in ACTIONS(*state*)
    do $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(*s*, a)))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for *a* in ACTIONS(*state*)
    do $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(*state*, *a*)))
  return v

Properties of Minimax

Complete???

Complete??? Yes, if tree is finite (chess has specific rules for this)

Complete??? Yes, if tree is finite (chess has specific rules for this)

Optimal???

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??

Otherwise even better.

Complete??? Yes, if tree is finite (chess has specific rules for this)

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

Otherwise even better. Example?

Properties of Minimax

Complete??? Yes, if tree is finite (chess has specific rules for this)

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

Otherwise even better. Example? Class activity.

Complete??? Yes, if tree is finite (chess has specific rules for this)

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

Otherwise even better. Example? Class activity.

Consider a simple 2-ply game, with four terminal states with values 10, 10, 9, and 11, in order (from left to right).

DIY & trace on the board

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??? $O(b^m)$

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???

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)

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

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

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

Do we need to explore every path?

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

Do we need to explore every path?

In realistic games, cannot explore the full game tree.

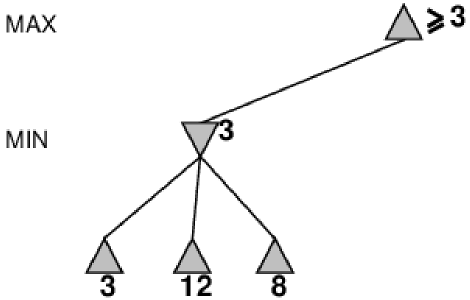
Number of game states MiniMax explores is exponential in the depth of the tree.

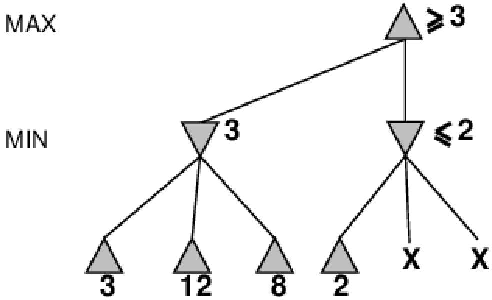
What to do?

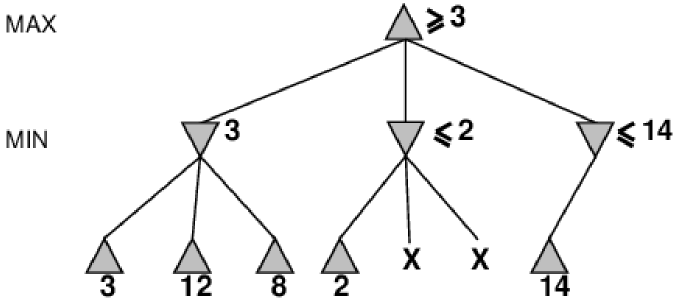
Two options (can be used in combination):

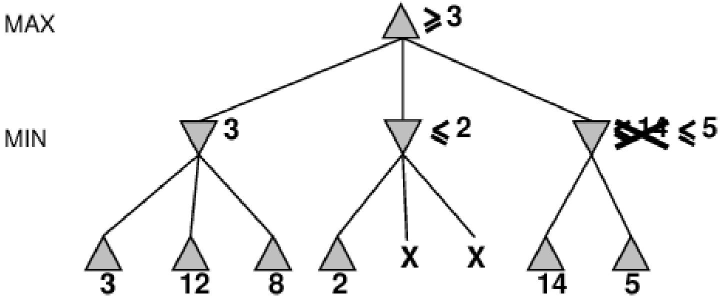
- Remove from consideration entire subtrees
- Find away not to have to reach the leaves to determine the value of a state

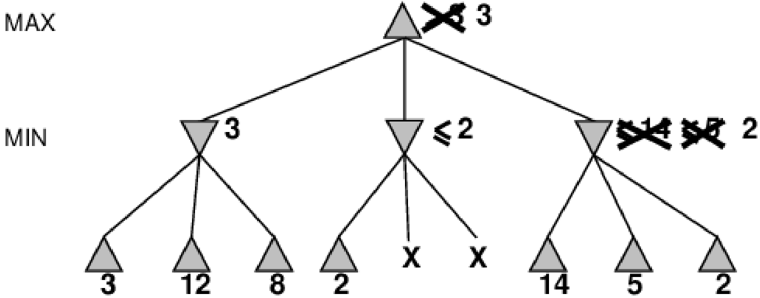
Remove from Consideration Entire Subtrees – α–β Pruning Example

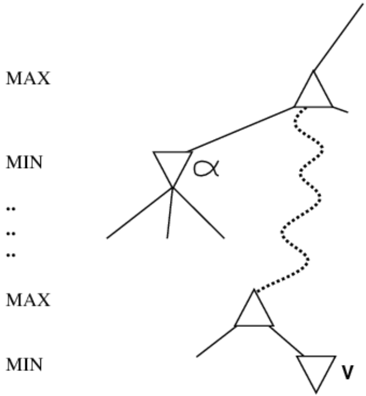



MAX
MIN
≥ 3
3
≤ 2
3
12
8
2
X
X


MAX
MIN
≥ 3
3
≤ 2
≤ 14
3
12
8
2
X
X
14


MAX
MIN
≥ 3
3
≤ 2
≤ 5
3
12
8
2
X
X
14
5


MAX
MIN
3
≤ 2
2
3
3
12
8
2
X
X
14
5
2



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

α : MAX's best option on path to root
β: MIN's best option on path to root

Pruning by Maintaining α and β

function ALPHA-BETA-VALUE(*state*, α, β) **returns** *value/utility*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  if NEXT AGENT IS MAX **then return** MAX-VALUE(*state*, α, β)
  if NEXT AGENT IS MIN **then return** MIN-VALUE(*state*, α, β)

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  $v \leftarrow -\infty$
  for each successor of *state*
    $v \leftarrow$ MAX(v, ALPHA-BETA-VALUE(successor, 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*
  $v \leftarrow \infty$
  for each successor of *state*
    $v \leftarrow$ MIN(v, ALPHA-BETA-VALUE(successor, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

The α–β Pruning Algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, ∞)
  return *a* in ACTIONS(*state*) with value *v*

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, value of best alternative for MAX along the path to *state*
    β, value of best alternative for MIN along the path to *state*

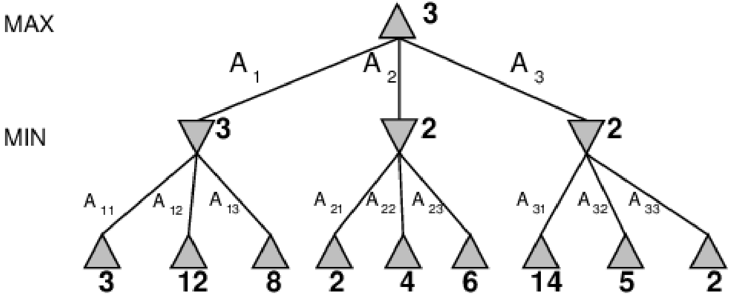
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a* in ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(*v*, MIN-VALUE(RESULT(*state*, *a*), α, β))
    if $v \geq \beta$ **then return** *v*
    $\alpha \leftarrow$ MAX(α, *v*)
  return *v*

---

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

Class activity: trace Alpha-Beta-Pruning on 2-ply game below

update your v's, α's, and β's!



Pruning is an example of **metareasoning** – computing about what to compute

Pruning **does not** affect final result, though intermediate nodes may have wrong values (when subtrees pruned)

Pruning is an example of **metareasoning** – computing about what to compute

Pruning **does not** affect final result, though intermediate nodes may have wrong values (when subtrees pruned)

Good move ordering improves effectiveness of pruning

Properties of α–β

Pruning is an example of **metareasoning** – computing about what to compute

Pruning **does not** affect final result, though intermediate nodes may have wrong values (when subtrees pruned)

Good move ordering improves effectiveness of pruning

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

Pruning is an example of **metareasoning** – computing about what to compute

Pruning **does not** affect final result, though intermediate nodes may have wrong values (when subtrees pruned)

Good move ordering improves effectiveness of pruning

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

With random ordering, time complexity $\approx O(b^{3m/4})$ for moderate b

Pruning is an example of **metareasoning** – computing about what to compute

Pruning **does not** affect final result, though intermediate nodes may have wrong values (when subtrees pruned)

Good move ordering improves effectiveness of pruning

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

With random ordering, time complexity $\approx O(b^{3m/4})$ for moderate b

Unfortunately, 35^{50} for chess is still impossible!

Pruning is an example of **metareasoning** – computing about what to compute

Pruning **does not** affect final result, though intermediate nodes may have wrong values (when subtrees pruned)

Good move ordering improves effectiveness of pruning

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

With random ordering, time complexity $\approx O(b^{3m/4})$ for moderate b

Unfortunately, 35^{50} for chess is still impossible!

Some tricks/meta-heuristics: killer moves first, IDS, remembering states (and their values) in transposition table, and more.

Pruning is an example of **metareasoning** – computing about what to compute

Pruning **does not** affect final result, though intermediate nodes may have wrong values (when subtrees pruned)

Good move ordering improves effectiveness of pruning

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

With random ordering, time complexity $\approx O(b^{3m/4})$ for moderate b

Unfortunately, 35^{50} for chess is still impossible!

Some tricks/meta-heuristics: killer moves first, IDS, remembering states (and their values) in transposition table, and more.

More generally: need to obtain value of a state without reaching leaf states

Properties of α–β

Pruning is an example of **metareasoning** – computing about what to compute

Pruning **does not** affect final result, though intermediate nodes may have wrong values (when subtrees pruned)

Good move ordering improves effectiveness of pruning

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

With random ordering, time complexity $\approx O(b^{3m/4})$ for moderate b

Unfortunately, 35^{50} for chess is still impossible!

Some tricks/meta-heuristics: killer moves first, IDS, remembering states (and their values) in transposition table, and more.

More generally: need to obtain value of a state without reaching leaf states

Standard approach:

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

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

function H-MINIMAX-VALUE(*state, d*) **returns** *h-minimax-value*
  if CUTOFF-TEST(*state, d*) **then return** EVAL(*state*)
  if NEXT AGENT IS MAX **then return** H-MAX-VALUE(*state, d+1*)
  if NEXT AGENT IS MIN **then return** H-MIN-VALUE(*state, d+1*)

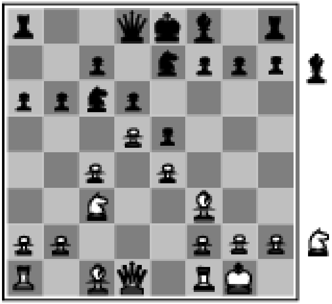
---

function H-MAX-VALUE(*state, d*) **returns** *a utility value*
  $v \leftarrow -\infty$
  for each successor of *state*
    do $v \leftarrow$ MAX(v, H-MINIMAX-VALUE(successor, d))
  return v

---

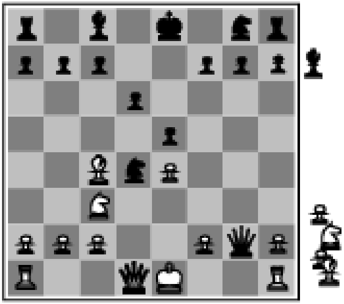
function H-MIN-VALUE(*state, d*) **returns** *a utility value*
  $v \leftarrow \infty$
  for each successor of *state*
    do $v \leftarrow$ MIN(v, H-MINIMAX-VALUE(successor, d))
  return v

Take-home exercise.



Black to move

White slightly better



White to move

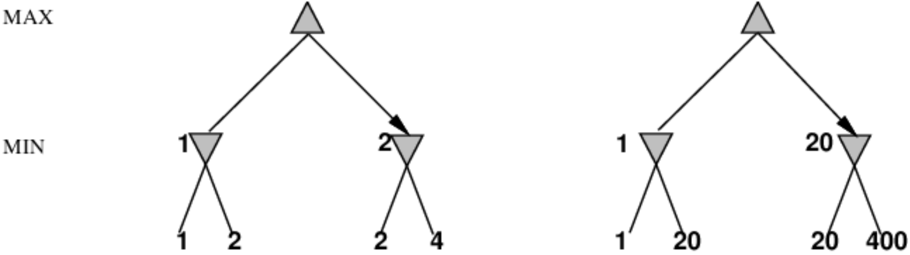
Black winning

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 do not Matter



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

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

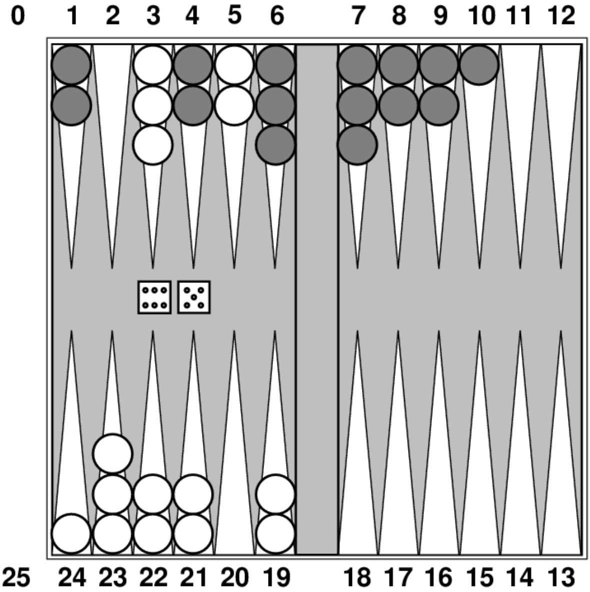
- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

- 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.

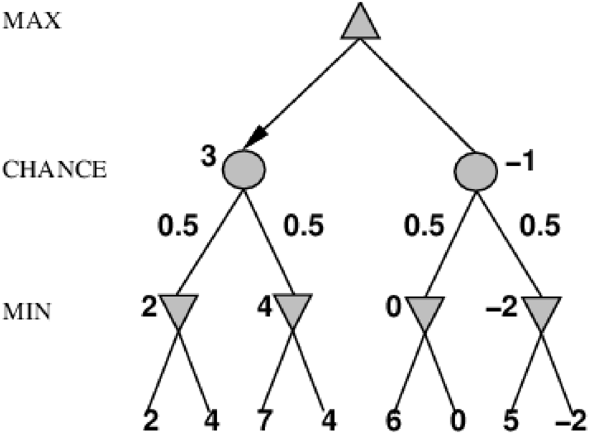
- Othello: human champions refuse to compete against computers, who are too good.

- Go: human champions refused to compete against computers, who **were** too bad. In Go, b goes from 361 to 250 (compared to chess' $b = 35$), so most programs use pattern knowledge bases to suggest plausible moves. Great progress made by AlphaGo via deep learning and playing against itself: now indisputable champion!

Nondeterministic Games: Backgammon



In nondeterministic games, chance introduced by dice, card-shuffling
Simplified example with coin-flipping:



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*)

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

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

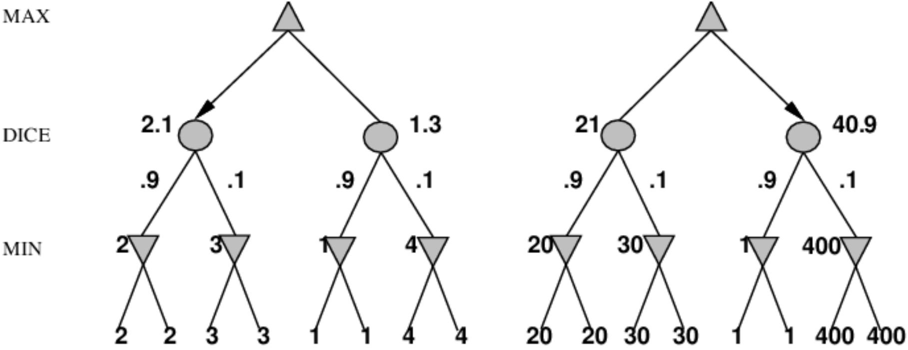
Time complexity: $O(b^m n^m)$, where n is the number of distinct rolls

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 ($d = 2$ in CUTOFF-test) + very good EVAL
$\approx$ world-champion level

Careful with EVAL design: Exact Values DO Matter



Behaviour is preserved only by positive linear transformation of expected utility

Hence EVAL should be proportional to the expected payoff

What kind of Evaluation Functions for Stochastic Games?

Monte Carlo simulation can be used to evaluate a state

From a start state, have the algorithm play games against itself, using random dice rolls

In backgammon, the resulting win percentage is a good-enough approximation of the value of a state

For games with dice, this is called a rollout

For stochastic games other than backgammon, more sophisticated evaluation functions may be designed via machine learning algorithm

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*

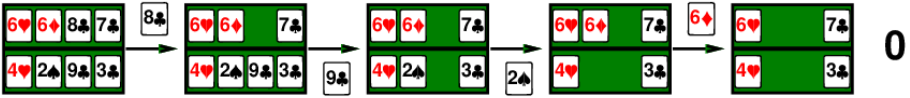
Idea:

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.*

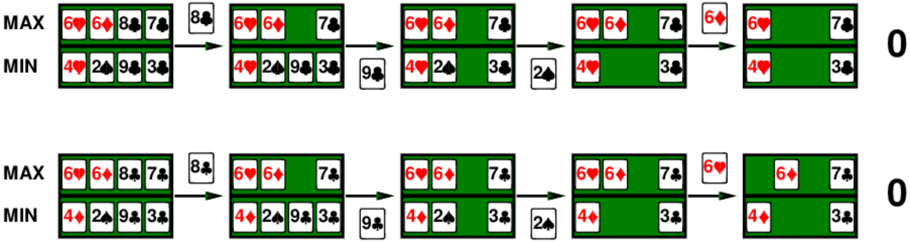
GIB, current best bridge program, approximates this idea by
1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

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

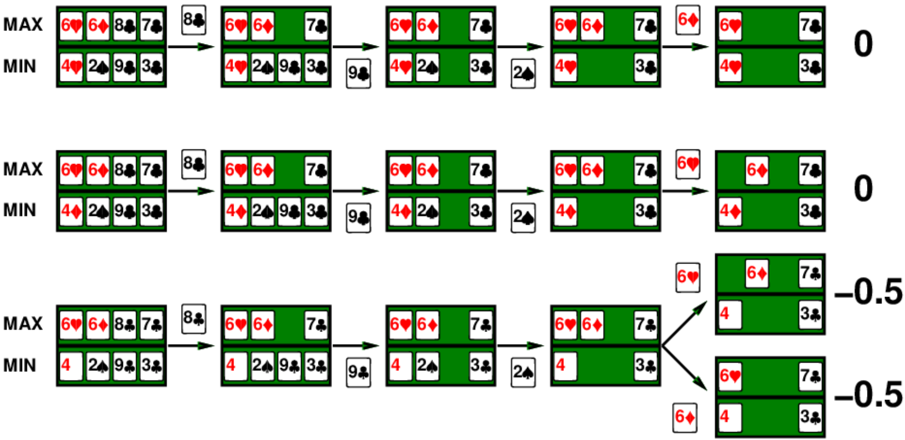


Example

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



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



Commonsense Example

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.

Commonsense Example

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.

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.

Commonsense Example

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.

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.

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

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

Game Playing Summary

Games are fun to work on! (and dangerously obsessive)

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

- ♢ Domain-specific tricks can be generalized to meta-heuristics of possible relevance for search of complex state spaces