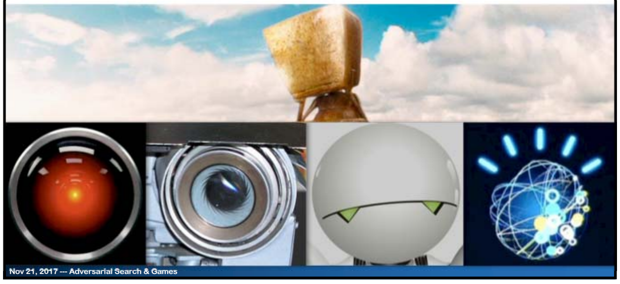
Computer Vision group heidelberg

IWR Interdisciplinary Center for Scientific Computing

Ruprecht-Karls-Universität Heidelberg

Artificial Intelligence

Prof. Björn Ommer
HCI & IWR

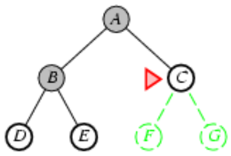


Outline – Game Playing & Adversarial Search

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

Games vs. search problems

- "Unpredictable" opponent → specifying a move for every possible opponent reply



- Time limits → unlikely to find goal, must approximate

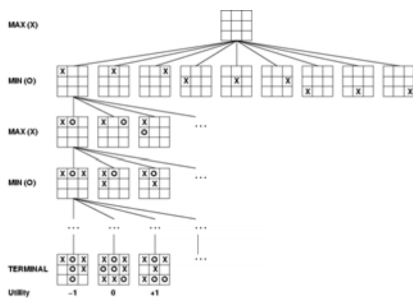
Games vs. search problems

- Brute-force, adding pruning strategies, … automatically learning evaluation strategies:
  - 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)
  - …
  - DeepBlue beating world champion in chess, 1997
  - Deep Learning: AlphaGo beating champion in Go, 2016

Types of games

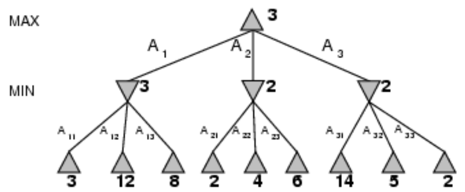
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Game tree (2-player, deterministic, turns)



Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



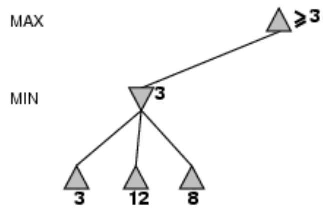
Minimax algorithm

```
function MINIMAX-DECISION(state) 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
```

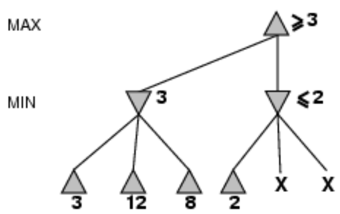

Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (depth-first exploration)
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  → exact solution completely infeasible (35^{50})

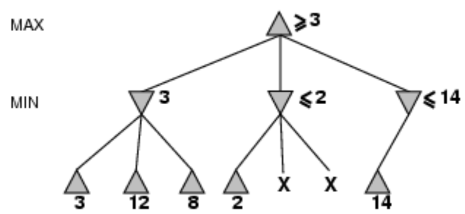
α-β pruning example



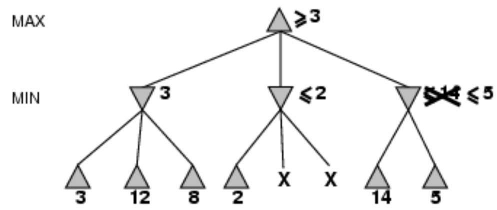
α-β pruning example



α-β pruning example

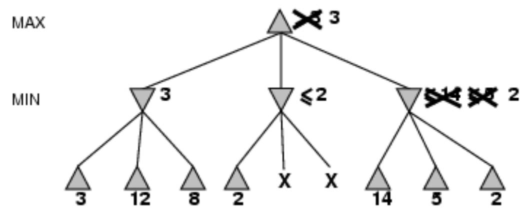


α-β pruning example



UNIVERSITÄT HEIDELBERG

α-β pruning example



UNIVERSITÄT HEIDELBERG

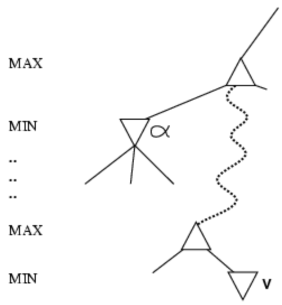
Properties of α-β

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

UNIVERSITÄT HEIDELBERG

Why is it called α-β?

- **α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max***
- **If *v* is worse than α, *max* will avoid it**
  → prune that branch
- **Define β similarly for *min***



UNIVERSITÄT HEIDELBERG

The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in SUCCESSORS(state) with value v

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
```


UNIVERSITÄT HEIDELBERG

The α-β algorithm

```
function MIN-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 ← MIN(v, MAX-VALUE(s, α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Resource limits

Suppose we have 100 secs, explore 10^4 nodes/sec

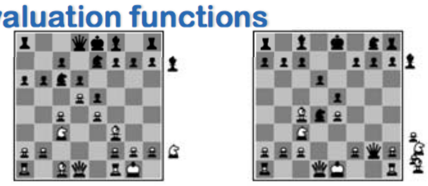
→ 10^6 nodes per move ~$35^{8/2}$

→ α-β reaches depth 8 → pretty good chess program

Standard approach:

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

Evaluation functions

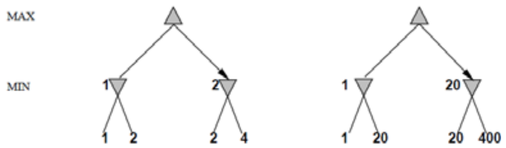


Black to move
White slightly better

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)



- 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

***MinimaxCutoff* is identical to *MinimaxValue* except**

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

Does it work in practice?

$b^m = 10^6$, $b=35$ → $m=4$

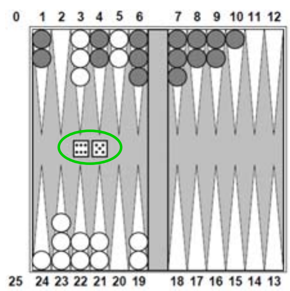
4-ply lookahead is a hopeless chess player!

- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov

Deterministic games in practice

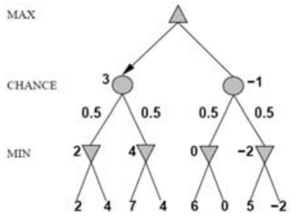
- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
- Chess: Deep Blue defeated human world champion Garry Kasparov 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*.
- *Until 2016:* 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. *However, see later slide…*

Nondeterministic games: backgammon



Nondeterministic games in general

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



Algorithm for nondeterministic games

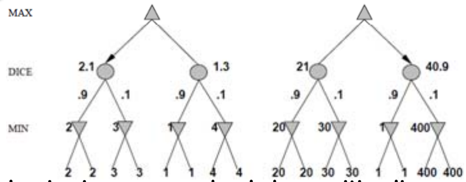
- Expectiminimax gives perfect play
- Just like Minimax, except we must also handle chance nodes:

```
. . .
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 20 legal moves (can be 6,000 with 1-1 roll)
  -
- As depth increases, probability of reaching a given node shrinks
  ⇨ value of lookahead is diminished
- α-β pruning is much less effective
- TDGammon uses depth-2 search + very good Eval ⇨ world-champion level

Digression: Exact values DO matter



- Behavior 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 (best bridge program for long) approximates this idea by
  1) generating 100 deals consistent with bidding information
  2) picking the action that wins most tricks on average

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, exploration, …
  - Signalling to one's partner
  - Acting randomly to minimize information disclosure

Summary

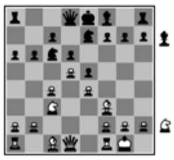
- Games are fun to work on!
- They illustrate several important points about AI
  - perfection is unattainable → must approximate
  - good idea to think about what to think about
  - uncertainty constrains the assignment of values to states
  - optimal decisions depend on information state, not real state
- Games are to AI as grand prix racing is to automobile design

- … recent improvements…

AlphaGo zero

Intelligent Game Playing

- Learning optimal actions



x $\quad$ $y=f(x;w)$

- Learn a function *f* with some parameters *w* to predict next move *y*

➪ Machine learning and esp. deep learning