Lecture 4

Adversarial Search

Marco Chiarandini

Deptartment of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

Outline

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

Multiagent environments

Multi agent environments:

- cooperative
- competitive ➨ adversarial search in games

AI game theory (combinatorial game theory)

- deterministic
- turn taking
- two players
- zero sum games = utility values equal and opposite
- perfect information
- agents are restricted to a small number of actions described by rules

“Classical” game theory includes cooperation, chance, imperfect knowledge, simultaneously moves and they tend to represent real-life decision making situations.

Types of Games

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

Games vs. search problems

"Unpredictable" opponent ⇒ solution is a strategy
specifying a move for every possible opponent reply ➨ contingency stratery

Optimal strategy: the one that leads to outcomes at least as good as any other strategy when one is playing an infallibile opponent

Search problem ➨ game tree

- initial state: game tree
- successor function: game rules
- terminal test (is the game over)
- utility function, gives a value for terminal nodes (eg, +1, -1)

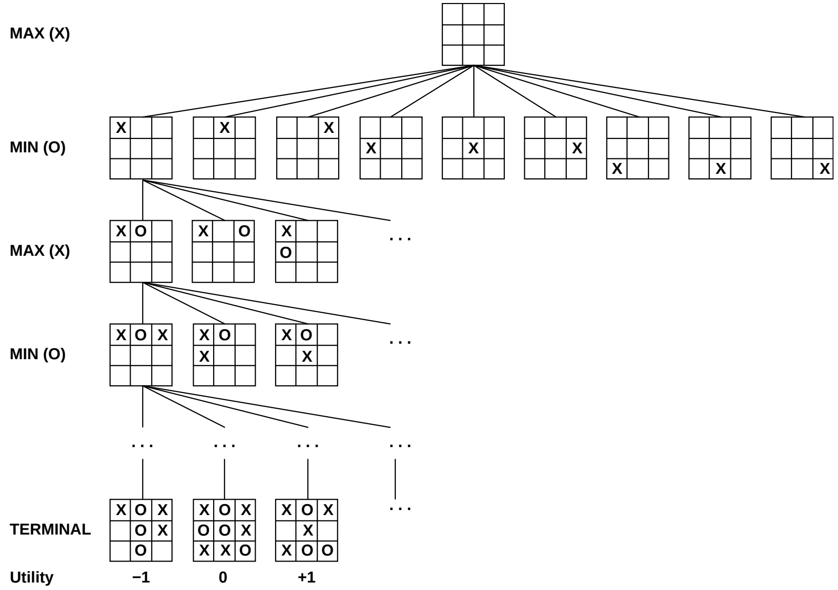
Terminology:

Two players called MAX and MIN.

MAX searches the search tree.

Ply: one turn taken by one of the players from "reply". [A. Samuel 1959]

Game tree (2-player, deterministic, turns)



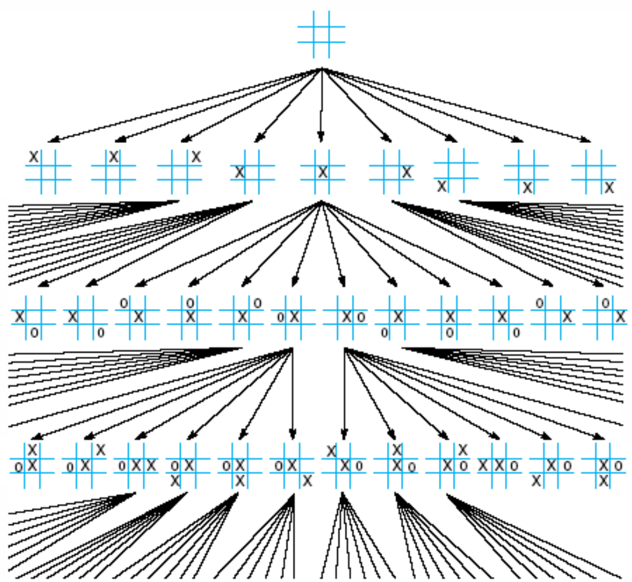
Measures of Game Complexity

- state-space complexity: number of legal game positions reachable from the initial position of the game.

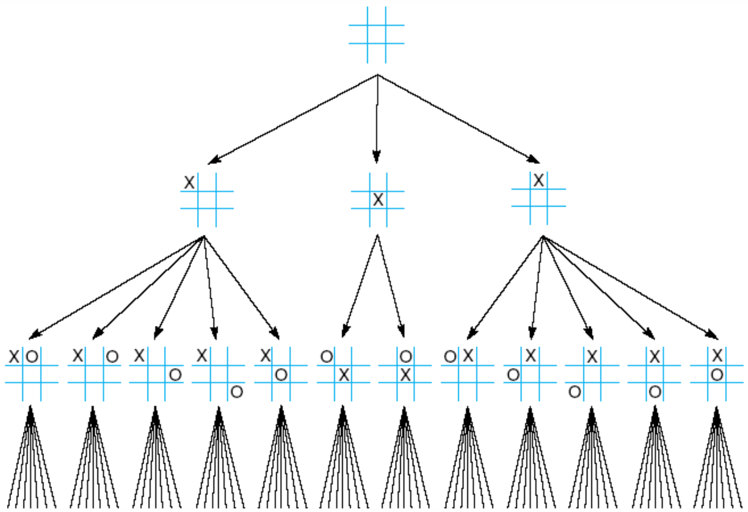
  an upper bound can often be computed by including illegal positions
  Eg, TicTacToe:
  $3^9 = 19.683$
  5.478 after removal of illegal
  765 essentially different positions after eliminating symmetries

- game tree size: total number of possible games that can be played: number of leaf nodes in the game tree rooted at the game's initial position.

  Eg: TicTacToe:
  $9! = 362.880$ possible games
  255.168 possible games halting when one side wins
  26.830 after removal of rotations and reflections



First three levels of the tic-tac-toe state space reduced by symmetry: $12 \times 7!$



Measures of Game Complexity

- game-tree complexity: number of leaf nodes in the smallest full-width decision tree that establishes the value of the initial position.
  A full-width tree includes all nodes at each depth.
  estimates the number of positions to evaluate in a minimax search to determine the value of the initial position.

  approximation: game's average branching factor to the power of the number of plies in an average game.
  Eg.: chess For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  $\Rightarrow$ exact solution completely infeasible

- computational complexity applies to generalized games (eg, $n \times n$ boards)
  Eg: TicTacToe:
  $m \times n$ board k in a row solved in $DSPACE(mn)$ by searching the entire game tree

Historical view

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

Plan of attack:

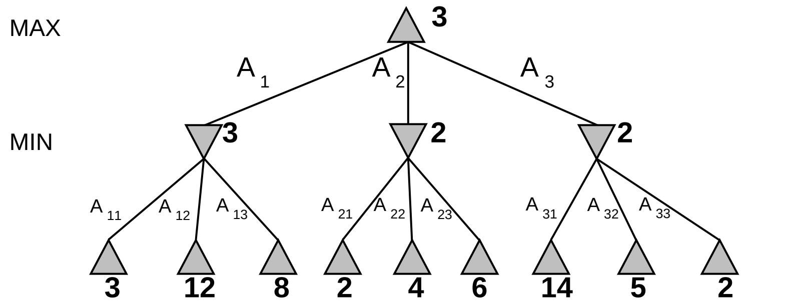
- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play - MINIMAX - (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy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

Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value ($\rightsquigarrow$utility for MAX)
= 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 (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)

But do we need to explore every path?

Resource limits

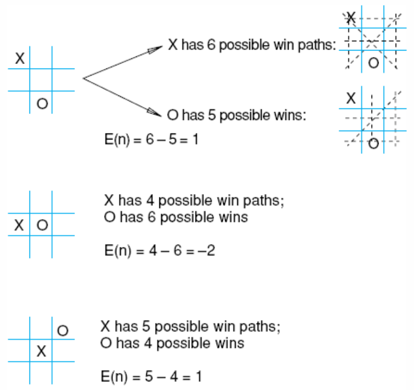
Standard approaches:

- n-ply lookahead: depth-limited search
- heuristic descent
- heuristic cutoff
  1. Use Cutoff-Test instead of Terminal-Test
     e.g., depth limit (perhaps add quiescence search)
  2. 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 $\approx 35^{8/2}$

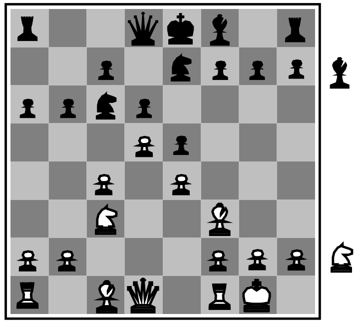
Heuristic Descent

Heuristic measuring conflict applied to states of tic-tac-toe



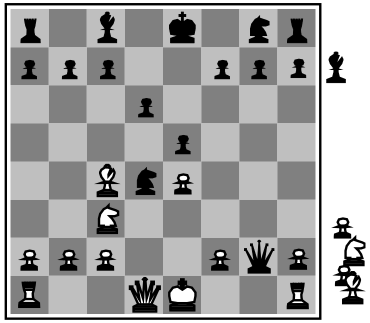
Heuristic is E(n) = M(n) – O(n)
where M(n) is the total of My possible winning lines
O(n) is total of Opponent's possible winning lines
E(n) is the total Evaluation for state n

Evaluation functions



Black to move

White slightly better



White to move

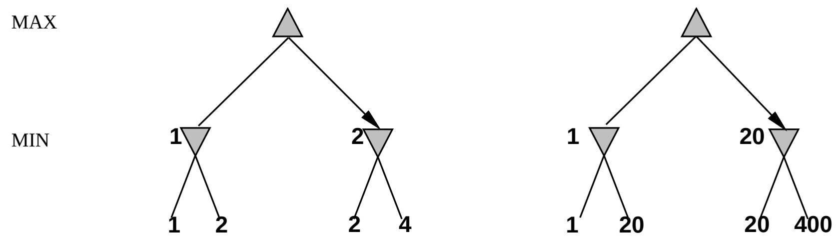
Black winning

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



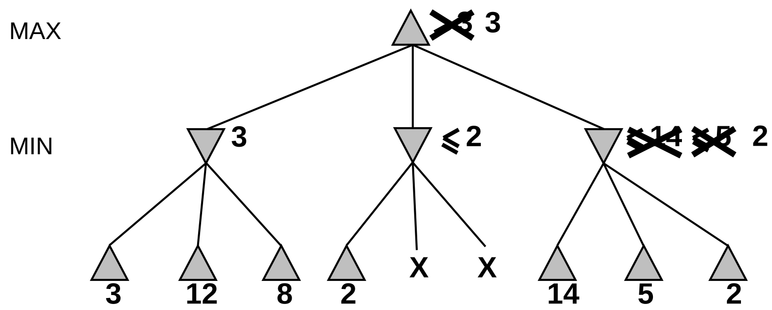
Behaviour is preserved under any **monotonic** transformation of Eval

Only the order matters:

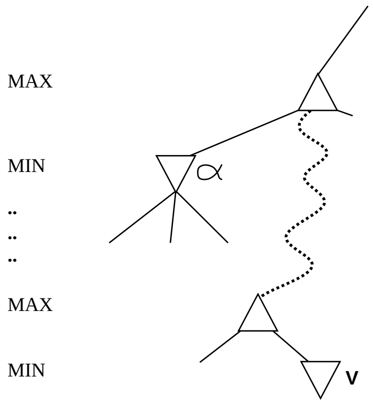
payoff in deterministic games acts as an ordinal utility function

Example

α–β pruning example



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, 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*)

$v \leftarrow -\infty$	$v \leftarrow \infty$
for *a*, *s* in Successors(*state*) **do**	**for** *a*, *s* in Successors(*state*) **do**
$v \leftarrow$ Max(v, Min-Value(s, α, β))	$v \leftarrow$ Min(v, Max-Value(s, α, β))
if $v \geq \beta$ **then return** v	**if** $v \leq \alpha$ **then return** v
$\alpha \leftarrow$ Max(α, v)	$\beta \leftarrow$ Min(β, v)
return v	**return** v

---

function Min-Value(*state*, α, β) **returns** *a utility value*
  same as Max-Value but with roles of α, β reversed

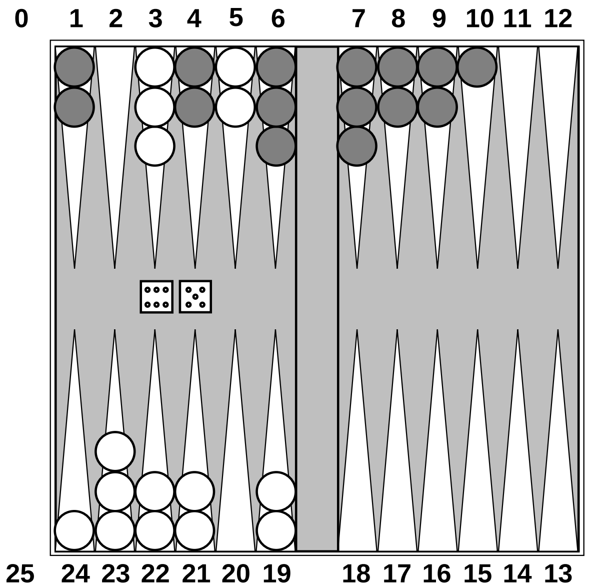
Properties of α–β

- 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!

Deterministic games in practice

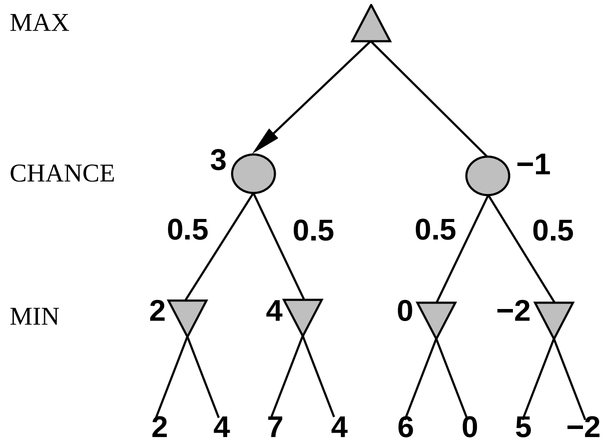
- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.
- Chess: Deep Blue defeated human world champion Gary Kasparov in a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



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*)
...

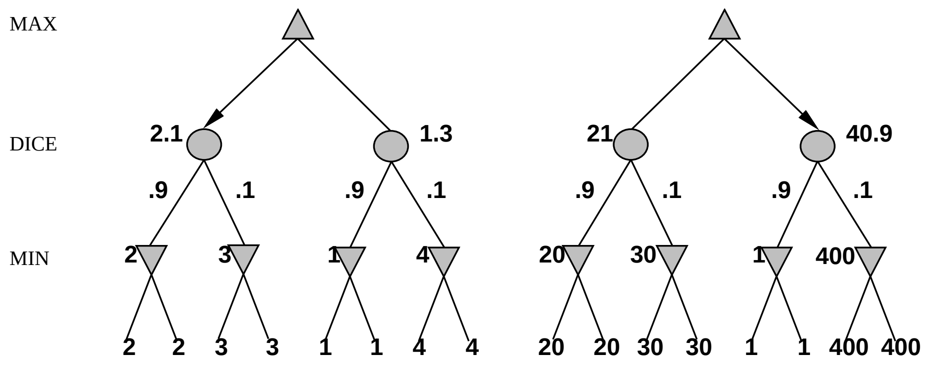
Nondeterministic games in practice

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

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

- As depth increases, probability of reaching a given node shrinks
  ⇒ 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



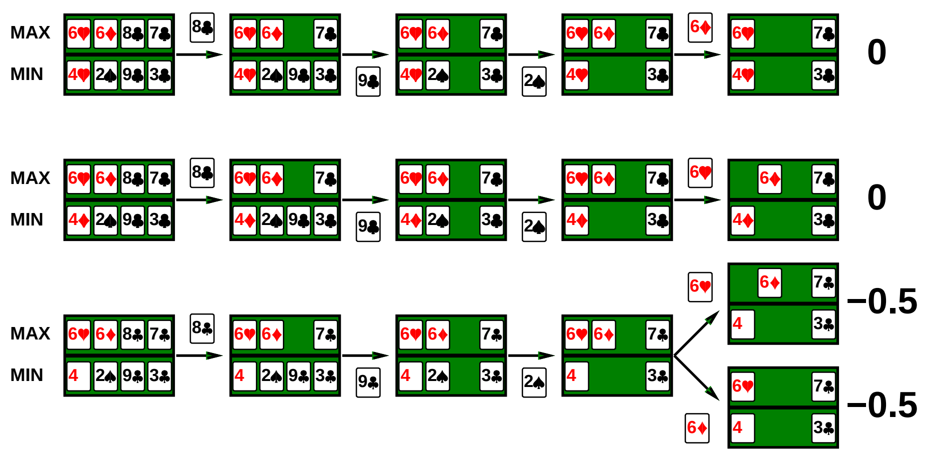
Behaviour is preserved only by positive linear transformation of Eval
Hence Eval should be proportional to the expected payoff

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

Example

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.

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

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