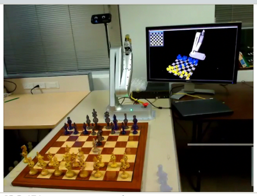
Game Playing

AI Class 8 — Ch. 5.1-5.3, 5.4.1, 5.5



Today's Class

- Tail end of Constraint Satisfaction
- Game playing
  - Framework
- Game trees
  - Minimax
  - Alpha-beta pruning
  - Adding randomness

We've seen multi-agent systems, and search problems where another agent's moves need to be taken into account – but what if they are actively moving *against* us?

Why Games?

- Clear criteria for success
- Offer an opportunity to study problems involving {hostile / adversarial / competing} agents.
- Interesting, hard problems which require minimal setup
- Often define very large search spaces
  - chess 35^{100} nodes in search tree, 10^{40} legal states
- Historical reasons
- Fun! (Mostly.)

State-of-the-art

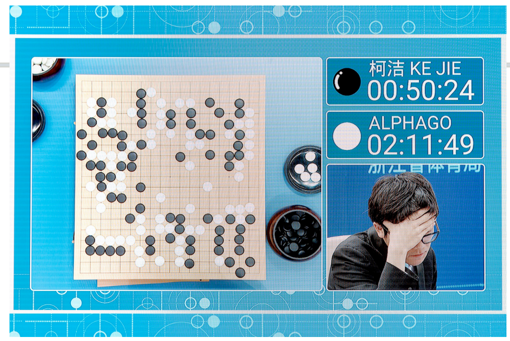
- **Chess**:
  - Deep Blue beat Gary Kasparov in 1997
  - Garry Kasparov vs. Deep Junior (Feb 2003): tie!
  - Kasparov vs. X3D Fritz (November 2003): tie!
  - Deep Fritz beat world champion Vladimir Kramnik (2006)
- **Checkers**: Chinook (an AI program with a *very large* endgame database) is the world champion and can provably never be beaten. Retired in 1995.
- **Bridge**: "Expert-level" AI, but no world champions

State-of-the-art: Go

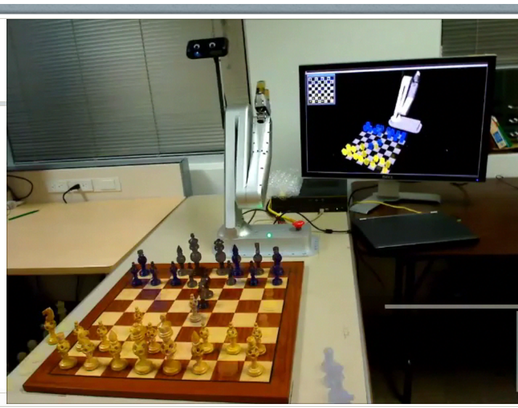
- Computers finally got there: **AlphaGo!**
  - Made by Google DeepMind in London
- 2015: Beat a professional Go player without handicaps
- 2016: Beat a 9-dan professional without handicaps
- **2017: Beat Ke Jie, #1 human player**
- 2017: DeepMind published AlphaGo Zero
  - No human games data
  - Learns from playing itself
  - Better than AlphaGo in 3 days of playing

Chinook

- World Man-Machine Checkers Champion, developed by researchers at the University of Alberta.
- Earned this title by competing in human tournaments, winning the right to play for the world championship, eventually defeating the best players in the world.
- Play it! http://www.cs.ualberta.ca/~chinook
- Developers have fully analyzed the game of checkers, and can provably *never* be beaten
  - (http://www.sciencemag.org/cgi/content/abstract/1144079v1)



www.wired.com/2017/05/googles-alphago-levels-board-games-power-grids



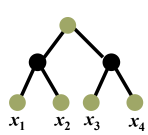
Typical Games

- 2-person game
- Players alternate moves
- **Zero-sum**: one player's loss is the other's gain
- **Perfect information**: both players have access to complete information about the state of the game. No information is hidden from either player.
- **Deterministic**: No chance (e.g., dice) involved
- Examples: Tic-Tac-Toe, Checkers, Chess, Go, Nim, Othello
- Not: Bridge, Solitaire, Backgammon, ...

12

How to Play (How to Search)

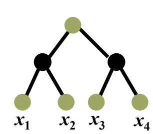
- Obvious approach:
  - From **current game state:**
  - Consider all the legal moves you can make
  - Compute new position resulting from each move
  - Evaluate each resulting position
  - Decide which is best
  - Make that move
  - Wait for your opponent to move
  - Repeat



13

How to Play (How to Search)

- Key problems:
  - Representing the "board"
    - What does that mean in, e.g., bridge?
  - Generating all legal next boards
  - **Evaluating a position**



14

Evaluation function

- **Evaluation function** or **static evaluator** is used to evaluate the "goodness" of a game position
  - Unlike heuristic search, where evaluation function is a positive estimate of cost from start node to a goal, passing through *n*
- Zero-sum assumption allows *one* evaluation function to describe goodness of a board for *both* players (how?)
  - $f(n)$ >> 0: position *n* good for me and bad for you
  - $f(n)$ << 0: position *n* bad for me and good for you
  - $f(n) = 0 \pm \varepsilon$: position *n* is a neutral position
  - $f(n) = +\infty$: win for me
  - $f(n) = -\infty$: win for you

Evaluation Function: the Idea

- **I** am always trying to reach the **highest** value
- **You** are always trying to reach the **lowest** value
- Captures everyone's goal in a single function
  - $f(n)$ >> 0: position *n* good for me and bad for you
  - $f(n)$ << 0: position *n* bad for me and good for you
  - $f(n) = 0 \pm \varepsilon$: position *n* is a neutral position
  - $f(n) = +\infty$: win for me
  - $f(n) = -\infty$: win for you

Evaluation function examples

- Example of an evaluation function for Tic-Tac-Toe:
  - $f(n)$ = [#3-lengths open for ×] - [#3-lengths open for O]
  - A 3-length is a complete row, column, or diagonal
- Alan Turing's function for chess
  - $f(n) = w(n)/b(n)$
  - $w(n)$ = sum of the **point value** of white's pieces
  - $b(n)$ = sum of black's

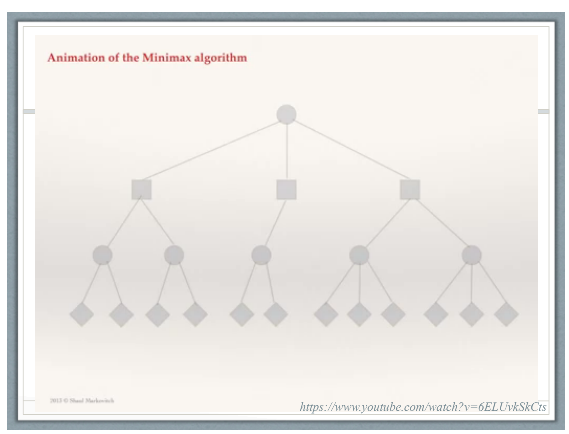
Evaluation function examples

- Most evaluation functions are specified as a **weighted sum** of position features:
  - $f(n) = w_1 * feat_1(n) + w_2 * feat_2(n) + \ldots + w_n * feat_k(n)$
- Example features for chess: piece count, piece placement, squares controlled, …
- Deep Blue had over **8000** features in its nonlinear evaluation function!

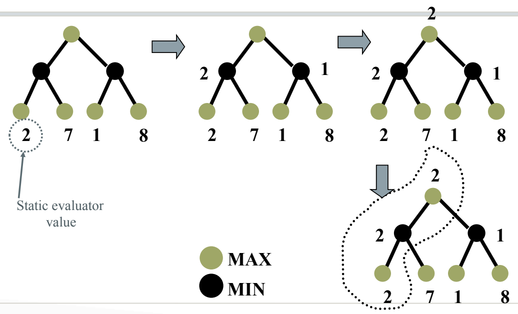
square control, rook-in-file, x-rays, king safety, pawn structure, passed pawns, ray control, outposts, pawn majority, rook on the 7th blockade, restraint, trapped pieces, color complex, …

Minimax Procedure

- Create start node: MAX node, current board state
- Expand nodes down to a **depth** of *lookahead*
- Apply evaluation function at each leaf node
- "Back up" values for each non-leaf node until a value is computed for the root node
  - MIN: backed-up value is **lowest** of children's values
  - MAX: backed-up value is **highest** of children's values
- Pick operator associated with the child node whose backed-up value set the value at the root



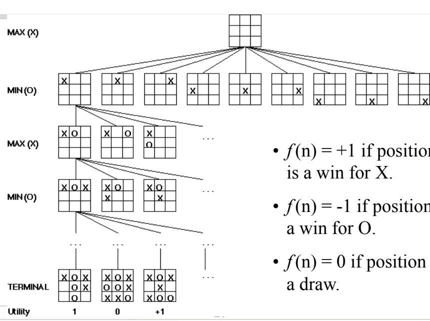
Minimax Algorithm



Example: Nim

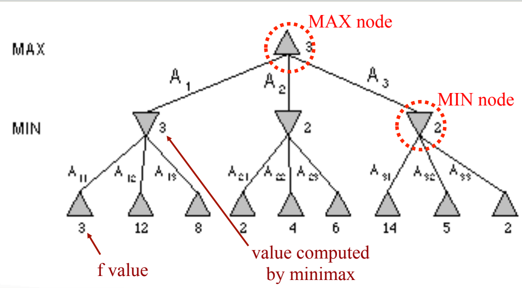
- In Nim, there are a certain number of objects (coins, sticks, etc.) on the table – we'll play 7-coin Nim
- Each player in turn has to pick up either one or two objects
- Whoever picks up the last object loses

Partial Game Tree for Tic-Tac-Toe



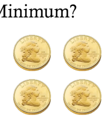
- $f(n) = +1$ if position is a win for X.
- $f(n) = -1$ if position is a win for O.
- $f(n) = 0$ if position is a draw.

Minimax Tree



Nim Game Tree

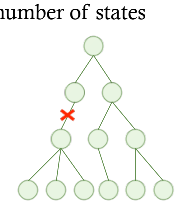
- **In-class exercise:**
- Draw minimax search tree for 4-coin Nim
- Things to consider:
  - What's your start state?
  - What's the maximum depth of the tree? Minimum?
- Pick up either one or two objects
- Whoever picks up the last object loses



29

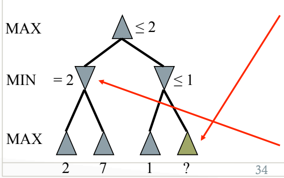
Improving Minimax

- Basic problem: must examine a number of states that is exponential in *d* !
- Solution: judicious **pruning** of the search tree
- "Cut off" whole sections that **can't** be part of the best solution
  - Or, sometimes, **probably won't**
  - Can be a completeness vs. efficiency tradeoff, esp. in stochastic problem spaces



Alpha-Beta Pruning

- We can improve on the performance of the minimax algorithm through **alpha-beta pruning**
  - Basic idea: *"If you have an idea that is surely bad, don't take the time to see how truly awful it is."* – Pat Winston

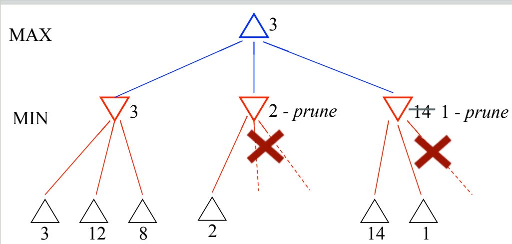


- We don't need to compute the value at this node.
- No matter what it is, it can't affect the value of the root node.
- Because the MAX player will choose this value.

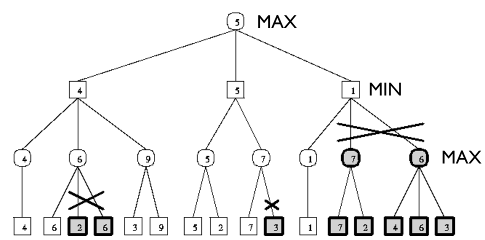
Alpha-Beta Pruning

- Traverse search tree in *depth-first order*
- At each **MAX** node n, $\alpha(n)$ = maximum value found so far
- At each **MIN** node n, $\beta(n)$ = minimum value found so far
  - α starts at $-\infty$ and increases, β starts at $+\infty$ and decreases
- **β−cutoff**: Given a MAX node *n*,
  - Cut off search below *n* (i.e., don't look at any more of *n*'s children) if:
  - $\alpha(n) \geq \beta(i)$ for some MIN node ancestor *i* of *n*
- **α−cutoff**:
  - Stop searching below MIN node *n* if:
  - $\beta(n) \leq \alpha(i)$ for some MAX node ancestor *i* of *n*

Alpha-beta Example (b=3)



Alpha-Beta Pruning

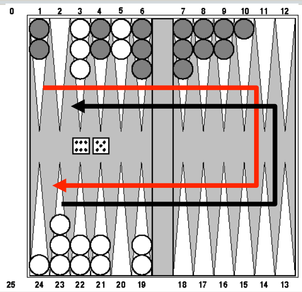


Effectiveness of Alpha-Beta

- Alpha-beta is guaranteed to:
  - Compute the same value for the root node as minimax
  - With ≤ computation
- **Worst case:** nothing pruned
  - Examine b^d leaf nodes
  - Each node has b children and a d-ply search is performed
- **Best case:** examine only $(2b)^{d/2}$ leaf nodes.
  - So you can search twice as deep as minimax!
  - When each player's best move is the first alternative generated
- In Deep Blue, empirically, alpha-beta pruning took average branching factor from ~35 to ~6!

Games of Chance

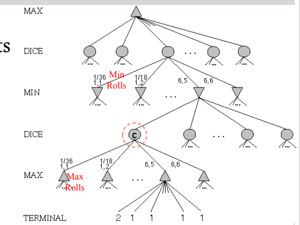
- Backgammon: a two-player game with uncertainty
- Players roll dice to determine what moves to make
- White has just rolled 5 and 6 and has four legal moves:
  - 5-10, 5-11
  - 5-11, 19-24
  - 5-10, 10-16
  - 5-11, 11-16
- Good for decision making in adversarial problems with skill *and* luck



41

Game Trees with Chance

- **Chance nodes** (circles) represent random events
- For a random event with N outcomes:
  - Chance node has N distinct children
  - Each has a probability
- Example:
  - Rolling 2 dice → 21 distinct outcomes
  - Not all equally likely!



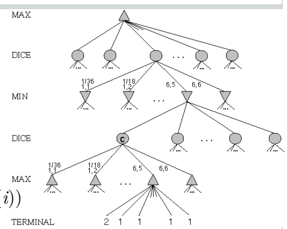
42

Game Trees with Chance

- Use minimax to compute values for MAX and MIN nodes
- Use **expected values** for chance nodes
- Over a max node, as in C:

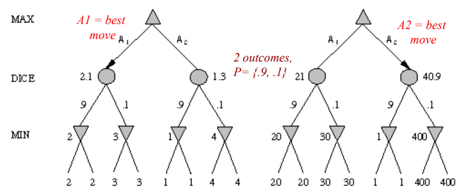
  expectimax(C) = Σ_i(P(d_i) * maxvalue(i))
- Over a min node:

  expectimin(C) = Σ_i(P(d_i) * minvalue(i))



43

Meaning of the Evaluation Function



- Dealing with probabilities and expected values means we have to be careful about the "meaning" of values returned by the static evaluator.
- A "relative-order preserving" change of values would not change decision of minimax, but could change the decision with chance nodes.

44

Example: Oopsy-Nim

- Starts out like Nim
  - Each player in turn has to pick up either one or two objects
  - Sometimes (probability = 0.25), when you try to pick up two objects, you drop them both
  - Picking up a single object always works

- Question: Why can't we draw the entire game tree?
- **Exercise: Draw the 4-ply game tree (2 moves per player)**

Nim Game Tree

- **In-class exercise:**
- Draw minimax search tree for 4-coin Nim
- Things to consider:
  - What's your start state?
  - What's the maximum depth of the tree? Minimum?

46