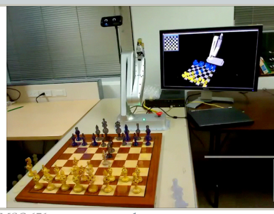
Game Playing

Ch. 5.1-5.3, 5.4.1, 5.5



On to Games

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

Questions from reading?

We've seen search problems where other agents' 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
- Many problems can be formalized as games

State-of-the-art

"A computer can't be intelligent; one could never beat a human at _____"

- **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)
  - Now computers play computers
- **Checkers**: "Chinook" (sigh), an AI program with a *very large* endgame database, is world champion, can provably never be beaten. Retired 1995.

State-of-the-art

- **Bridge**: "Expert-level" AI, but no world champions
  - "computer bridge world champion *Jack* played seven top Dutch pairs … and two reigning European champions.
  - A total of 196 boards were played. Jack defeated three out of the seven pairs (including the Europeans). Overall, the program lost by a small margin (359 versus 385)." (2006)
  - Bridge is stochastic: the computer has imperfect information.
- **Go**

"A computer can't be intelligent; one could never beat a human at _____"

AlphaGo Master defeated Ke Jie by three to zero during its 60 straight wins in the online games at the end of 2016 and beginning of 2017.

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

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

39

39

Typical Games

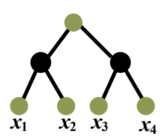
- 2-person game
- Players alternate moves
- Easiest games are:
  - **Zero-sum**: one player's loss is the other's gain
  - **Fully observable**: both players have access to complete information about the state of the game.
  - **Deterministic**: No chance (e.g., dice) involved
- Tic-Tac-Toe, Checkers,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

44

44

How to Play (How to Search)

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

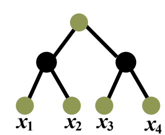


45

45

How to Play (How to Search)

- Key problems:
  - Representing the "board" (game state)
    - We've seen that there are different ways to make these choices
  - Generating all legal next boards
    - That can get ugly
  - **Evaluating a position**



46

46

Evaluation Function

- **Evaluation function** or **static evaluator** is used to evaluate the "goodness" of a game *position* (state)
- Zero-sum assumption allows *one* evaluation function to describe goodness of a board for *both* players
  - One player's gain of n means the other loses n
  - How?

47

47

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

48

48

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

49

Evaluation function examples

- Most evaluation functions are specified as a **weighted sum** of position features:
  - $f(n) = w_1 * feat_1(n) + w_2 * feat_2(n) + ... + w_n * feat_k(n)$
- 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, …

50

Evaluation Function: the Idea

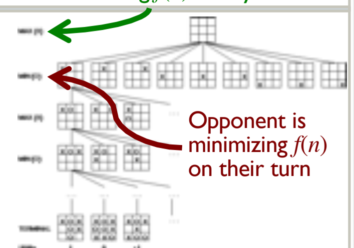
- **I** am always trying to reach the **highest** value
- **You** are always trying to reach the **lowest** value
- Captures everyone's goal in a single function
  - $\boldsymbol{f(n)} >> \mathbf{0}$: position n good for me and bad for you
  - $\boldsymbol{f(n)} << \mathbf{0}$: position n bad for me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

52

Game trees

I am maximizing $f(n)$ on my turn

- Problem spaces for typical games are represented as trees
- Player must decide best single **move** to make next
- Root node = current board configuration
- Arcs = possible legal moves for a player



53

Game trees

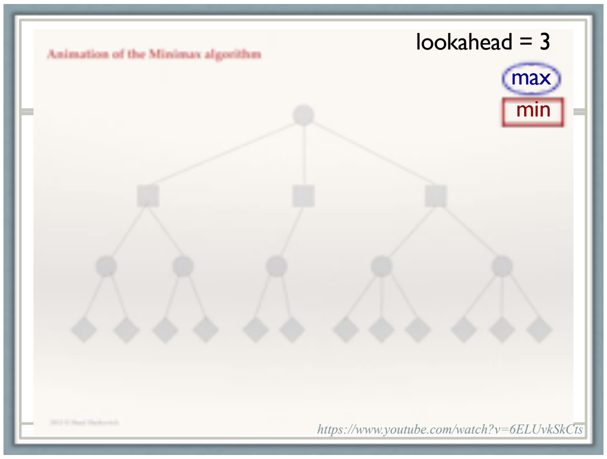
- **Static evaluator function**
  - Rates a board position
  - f(board) = R, with f>0 for me, f<0 for you
- If it is **my turn** to move:
  - Root is labeled "**MAX**" node
  - Otherwise it is a "**MIN**" node (**opponent's turn**)
- Each level's nodes are all MAX or all MIN
- Nodes at level i are opposite those at level i +1

54

Minimax Procedure

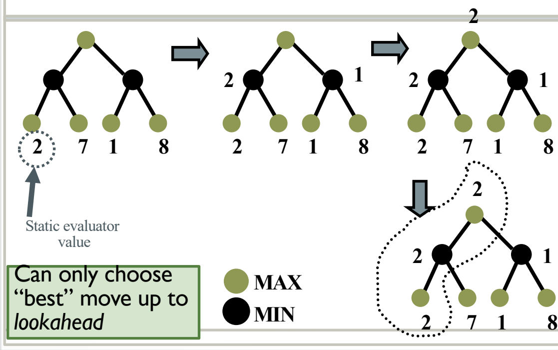
- Create start node: MAX node, current board state
- Expand nodes down to a **depth** of *lookahead*
- Apply evaluation function at each leaf node
- "Back up" values for each non-leaf node until a value is computed for the root node
  - MIN: backed-up value is **lowest** of children's values
  - MAX: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

55



56

Minimax Algorithm



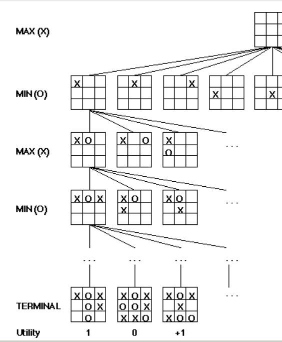
57

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

58

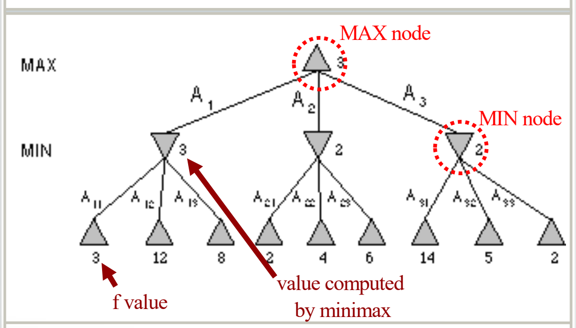
Partial Game Tree for Tic-Tac-Toe



- $f(n) = +1$ 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.

59

Minimax Tree



60

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

61

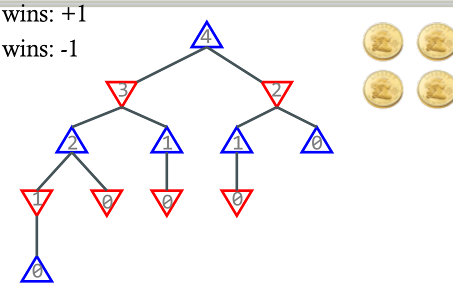
Games 2

Expectiminimax
Alpha-beta Pruning

62

Nim Game Tree

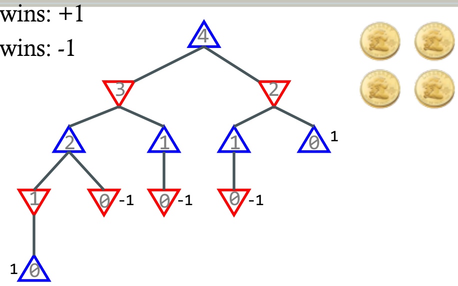
Player 1 wins: +1
Player 2 wins: -1



63

Nim Game Tree

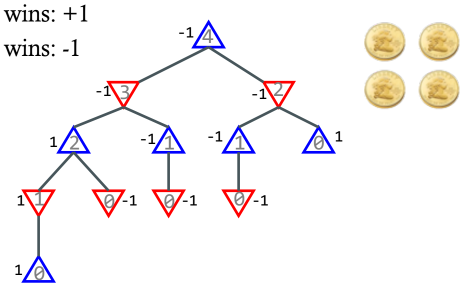
Player 1 wins: +1
Player 2 wins: -1



64

Nim Game Tree

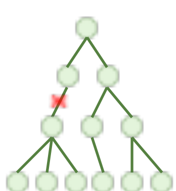
Player 1 wins: +1
Player 2 wins: -1



65

Improving Minimax

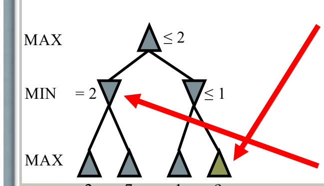
- Basic problem: must examine a number of states that is exponential in d!
- Solution: judicious **pruning** of the search tree
- "Cut off" whole sections that **can't** be part of the best solution
  - Or, sometimes, **probably won't**
  - Can be a completeness vs. efficiency tradeoff, esp. in stochastic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



66

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



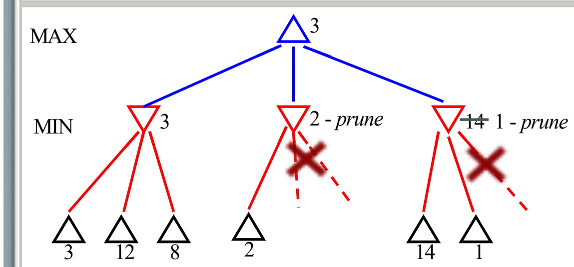
- We don't need to compute the value at this node.
- No matter what it is, it can't affect the value of the root node.
- Because the MAX player will choose this value.

67

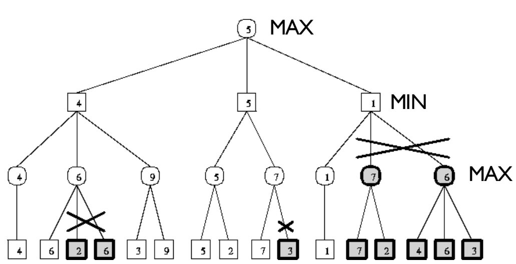
Alpha-Beta Pruning

- Traverse search tree in *depth-first order*
- At each **MAX** node n, α(n) = maximum value found so far
- At each **MIN** node n, β(n) = minimum value found so far
  - α starts at -∞ and increases, β starts at +∞ 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:
  - α(n) ≥ β(i) for some MIN node ancestor i of n
- **α–cutoff**:
  - Stop searching below MIN node n if:
  - β(n) ≤ α(i) for some MAX node ancestor i of n

Alpha-beta Example (*b*=3)



Alpha-Beta Pruning

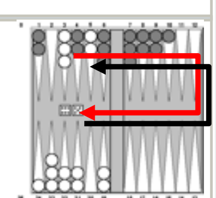


Alpha-Beta Pruning: Exercise

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!

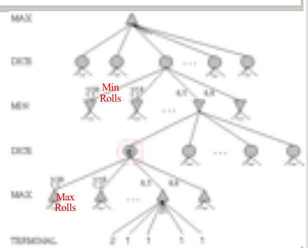
Games of Chance



- Backgammon: 2-player 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

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!



75

Game Trees with Chance

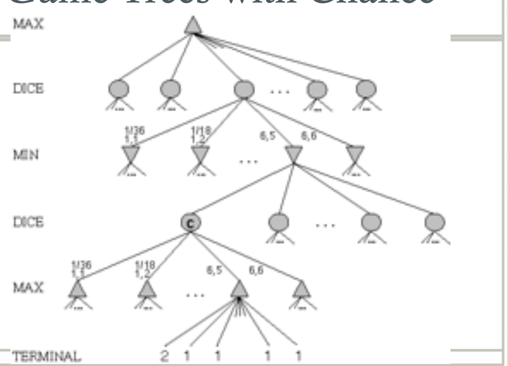
- Use minimax to compute values for MAX and MIN nodes
- Use **expected values** for chance nodes
- Over a max node, as in C:

  expectimax(C) = $\sum_i(\mathrm{P}(d_i) * \mathrm{maxvalue}(i))$
- Over a min node:

  expectimin(C) = $\sum_i(\mathrm{P}(d_i) * \mathrm{minvalue}(i))$

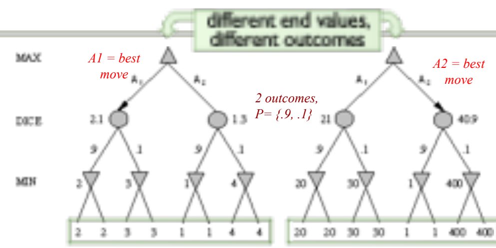
76

Game Trees with Chance



77

Meaning of the Evaluation Function



- Dealing with probabilities and expected values means being careful with "meaning" of values returned by the static evaluator
- "Relative-order preserving" (as here) change won't change minimax, but could change the decision with chance nodes

78

Exercise: 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)**

79