Game Playing
AI Class 8 — Ch. 5.1-5.3, 5.4.1, 5.5

Cynthia Matuszek – CMSC 671

Based on slides by Marie desJardin, Francisco Iacobelli

Today’s Class

• Game playing
  • State of the art and resources
  • Framework
• Game trees
  • Minimax
  • Alpha-beta pruning
  • Adding randomness

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

Adversarial search = Games!

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 35^106 nodes in search tree, 10^10 legal states
• Historical reasons
• Fun! (Mostly.)

State-of-the-art

• How good are computer game players?
  • Chess:
    • Deep Blue beat Garry Kasparov in 1997
    • Garry Kasparov vs. Deep Junior (Feb 2003) saw
      http://www.buchempfehlung.de/spieleempfehlung/Kasparov-DeepJunior/spiele.html
    • 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
  • Go: Computer players have finally reached tournament-level play
    • AlphaGo beat Ke Jie (No.1 world player) in 2017
  • Bridge: “Expert-level” computer players exist (but no world champions yet?)
  • Good places to learn more:
    • http://www.cs.ualberta.ca/~games/
    • http://www.cs.unimasa.nl/ioga

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.iaces.cs.rpi.edu/papers/ICAOJS93.pdf)
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, ...

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 and repeat

- **Key problems are:**
  - Representing the “board”
  - Generating all legal next boards
  - Evaluating a position

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 it
  - Zero-sum assumption allows one evaluation function to describe goodness of a board for both players (how?)
  - $f(n) >> 0$: position is good for me and bad for you
  - $f(n) << 0$: position is bad for me and good for you
  - $f(n) = 0$: position 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:
  - $f(n) = [\#3-lengths open for \times] - [\#3-lengths open for O]$
  - A 3-length is a complete row, column, or diagonal
- Alan Turing’s function for chess
  - $f(n) = w(n)/b(n)$
  - $w(n) = \text{sum of the point value of white’s pieces}$
  - $b(n) = \text{sum of black’s}$
Evaluation function examples

- Most evaluation functions are specified as a weighted sum of position features:
  \[ f(n) = w_1 \cdot \text{feat}_1(n) + w_2 \cdot \text{feat}_2(n) + \ldots + w_n \cdot \text{feat}_n(n) \]

- Example features for chess: piece count, piece placement, squares controlled, ...

- Deep Blue had over 8000 features in its nonlinear evaluation function!

Game trees

- Problem spaces for typical games are represented as trees

- Player must decide best single move to make next

- Root node = the current board configuration

- Arcs = possible legal moves for a player

Game trees

- Static evaluator function
  - Rates a board position
  - \( f(\text{board}) = R \) with \( f>0 \) “white” (me), \( f<0 \) for black (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 \( i+1 \)

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

MAX node

MIN node

f value

value computed by minimax

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

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.
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
- \( \alpha \) starts at $-\infty$ and increases, \( \beta \) starts at $+\infty$ and decreases
- \( \beta \)-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 $n$
- \( \alpha \)-cutoff:
  - Stop searching below MIN node $n$ if:
    - $\beta(n) \leq \alpha(i)$ for some MAX node ancestor $i$ of $n$

Effectiveness of Alpha-Beta

- Alpha-beta is guaranteed to:
  - Compute the same value for the root node as minimax
  - With \( \leq \) computation
- Worst case: nothing pruned, examine \( b^d \) leaf nodes, where each node has $b$ children and a $d$-ply search is performed
- Best case: examine only \( (2b)^{d/2} \) leaf nodes.
  - Result is 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

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 \( \rightarrow \) 21 distinct outcomes
  - Not all equally likely!
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:
    \[ \text{expectimax}(C) = \sum_i (P(d_i) \times \text{maxvalue}(i)) \]
  - Over a min node:
    \[ \text{expectimin}(C) = \sum_i (P(d_i) \times \text{minvalue}(i)) \]

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.

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 2-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?