Minimax strategies, alpha beta pruning

Lirong Xia

Rensselaer

Friday, February 3, 2017
Reminder

- Project 1 due tonight
  - Makes sure you DO NOT SEE “ERROR: Summation of parsed points does not match”
- Project 2 due in two weeks
How to find good heuristics?

- No really mechanical way
  - art more than science
- General guideline: relaxing constraints
  - e.g. Pacman can pass through the walls
- Mimic what you would do
Last class

- Arc consistency
- CSP on tree graphs
- Linear programming
- The “sum 2” game
A simple form of propagation makes sure all arcs are consistent:

- If V loses a value, neighbors of V need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment
- Might be time-consuming
Limitations of Arc Consistency

- After running arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)
CSP for tree graph

- Stage 1: moving upward, cross out the values that cannot work with the subtree below that node
- Stage 2: if a value remains at the root, there is a solution: go downward to pick a solution
Given

- Variables $x$: a row vector of $m$ positive real numbers
- Parameters (fixed)
  - $c$: a row vector of $m$ real numbers
  - $b$: a column vector of $n$ real numbers
  - $A$: an $n \times m$ real matrix

Solve

$$\max \ cs^T$$

$$s.t. \ Ax^T \leq b, \ x \geq 0$$

Solutions

- $x$ is a feasible solution, if it satisfies all constraints
- $x$ is an optimal solution, if it maximizes the objective function among all feasible solutions
“Sum to 2” game

- Player 1 moves, then player 2, finally player 1 again
- Move = 0 or 1
- Player 1 wins if and only if all moves together sum to 2

Player 1’s utility is in the leaves; player 2’s utility is the negative of this
Today’s schedule

- Adversarial game
- Minimax search
- Alpha-beta pruning algorithm
Adversarial Games

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - The MAX player maximizes result
  - The MIN player minimizes result

- Minimax search:
  - A search tree
  - Players alternate turns
  - Each node has a minimax value: best achievable utility against a rational adversary
Computing Minimax Values

- This is DFS

- Two recursive functions:
  - `max-value` maxes the values of successors
  - `min-value` mins the values of successors

- Def `value (state)`:
  - If the state is a terminal state: return the state’s utility
  - If the agent at the state is MAX: return `max-value(state)`
  - If the agent at the state is MIN: return `min-value(state)`

- Def `max-value(state)`: similar to `max-value`

  Initialize max = $-\infty$
  For each successor of state:
  - Compute `value(successor)`
  - Update max accordingly
  return max

- Def `min-value(state)`: similar to `max-value`
Minimax Example
Tic-tac-toe Game Tree

MAX (X)

MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility

-1 0 +1
Renju

- 15*15
- 5 horizontal, vertical, or diagonal in a row win
- no double-3 or double-4 moves for black
- otherwise black’s winning strategy was computed
  - L. Victor Allis 1994 (PhD thesis)
Minimax Properties

- Time complexity?
  - \( O(b^m) \)

- Space complexity?
  - \( O(bm) \)

- For chess,
  - Exact solution is completely infeasible \( b \approx 35, \ m \approx 100 \)
  - But, do we need to explore the whole tree?
Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Guarantee of optimal play is gone
Evaluation Functions

- Functions which scores non-terminals

- Ideal function: returns the minimax utility of the position

- In practice: typically weighted linear sum of features:
  \[ Evals(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) \]

- e.g. \( f_1(s) = \left( \# \text{ white queens} - \# \text{ black queens} \right) \), etc.
Minimax with limited depth

- Suppose you are the MAX player
- Given a depth $d$ and current state
- Compute value(state, $d$) that reaches depth $d$
  - at depth $d$, use a evaluation function to estimate the value if it is non-terminal
Improving minimax: pruning
Pruning in Minimax Search

- An ancestor is a MAX node
  - already has an option than my current solution
  - my future solution can only be smaller
**Alpha-beta pruning**

- **Pruning** = cutting off parts of the search tree (because you realize you don’t need to look at them)
  - When we considered A* we also pruned large parts of the search tree

- **Maintain**
  - $\alpha$ = value of the best option for the MAX player along the path so far
  - $\beta$ = value of the best option for the MIN player along the path so far
  - Initialized to be $\alpha = -\infty$ and $\beta = +\infty$

- **Maintain and update** $\alpha$ and $\beta$ for each node
  - $\alpha$ is updated at MAX player’s nodes
  - $\beta$ is updated at MIN player’s nodes
Alpha-Beta Pruning

- General configuration
  - We’re computing the MIN-VALUE at n
  - We’re looping over n’s children
  - n’s value estimate is dropping
  - α is the best value that MAX can get at any choice point along the current path
  - If n becomes worse than α, MAX will avoid it, so can stop considering n’s other children
  - Define β similarly for MIN
  - α is usually smaller than β
    - Once α >= β, return to the upper layer
Alpha-Beta Pruning Example

\[ \alpha \text{ is MAX's best alternative here or above} \]
\[ \beta \text{ is MIN's best alternative here or above} \]
Alpha-Beta Pruning Example

Starting $\alpha / \beta$

Raising $\alpha$

Lowering $\beta$

Raising $\alpha$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
Alpha-Beta Pseudocode

function MAX-VALUE(state) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    \( v \leftarrow -\infty \)
    for \( a, s \) in Successors(state) do \( v \leftarrow \text{Max}(v, \text{Min-Value}(s)) \)
    return \( v \)

function MAX-VALUE(state, \( \alpha, \beta \)) returns a utility value
    inputs: state, current state in game
            \( \alpha \), the value of the best alternative for \( \text{MAX} \) along the path to state
            \( \beta \), the value of the best alternative for \( \text{MIN} \) along the path to state
    if Terminal-Test(state) then return Utility(state)
    \( v \leftarrow -\infty \)
    for \( a, s \) in Successors(state) do
        \( v \leftarrow \text{Max}(v, \text{Min-Value}(s, \alpha, \beta)) \)
        if \( v \geq \beta \) then return \( v \)
        \( \alpha \leftarrow \text{Max}(\alpha, v) \)
    return \( v \)
Alpha-Beta Pruning Properties

- This pruning has no effect on final result at the root
- Values of intermediate nodes might be wrong!
  - Important: children of the root may have the wrong value
- Good children ordering improves effectiveness of pruning
- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Your action looks smarter: more forward-looking with good evaluation function
  - Full search of, e.g. chess, is still hopeless…
Project 2

- **Q1:** write an evaluation function for (state, action) pairs
  - the evaluation function is for this question only
- **Q2:** minimax search with arbitrary depth and multiple MIN players (ghosts)
  - evaluation function on states has been implemented for you
- **Q3:** alpha-beta pruning with arbitrary depth and multiple MIN players (ghosts)
Recap

- Minimax search
  - with limited depth
  - evaluation function
- Alpha-beta pruning
- Project 1 due midnight today
- Project 2 due in two weeks