Adversarial Search

CS311
David Kauchak
Spring 2013

Some material borrowed from:
Sara Owsley Sood and others

http://www.youtube.com/watch?v=LcPWEMwGJvQ

A quick review of search

Rational thinking via search – determine a plan of actions by searching from starting state to goal state

Uninformed search vs. informed search
– what’s the difference?
– what are the techniques we’ve seen?
– pluses and minuses?

Heuristic design
– admissible?
– dominant?

Admin

• Reading/book?
• Assignment 2
  – On the web page
  – 3 parts
  – Anyone looking for a partner?
  – Get started!
• Written assignments
  – Make sure to look at them asap!
  – Post next written assignment soon

• On the web page
  – Anyone looking for a partner?
  – Get started!
• Written assignments
  – Make sure to look at them asap!
  – Post next written assignment soon
Why should we study games?

Clear success criteria

Important historically for AI

Fun 😊

Good application of search
  - hard problems (chess $35^{100}$ nodes in search tree, $10^{40}$ legal states)

Some real-world problems fit this model
  - game theory (economics)
  - multi-agent problems

Types of games

What are some of the games you’ve played?

Types of games: game properties

single-player vs. 2-player vs. multiplayer

Fully observable (perfect information) vs. partially observable

Discrete vs. continuous

real-time vs. turn-based

deterministic vs. non-deterministic (chance)

Strategic thinking ≠ intelligence

For reasons previously stated, two-player games have been a focus of AI since its inception…

Begs the question: Is strategic thinking the same as intelligence?
Humans and computers have different relative strengths in these games:

- **Humans**: good at evaluating the strength of a board for a player.
- **Computers**: good at looking ahead in the game to find winning combinations of moves.

How could you figure out how humans approach playing chess?

An experiment (by deGroot) was performed in which chess positions were shown to novice and expert players:

- Experts could reconstruct these perfectly.
- Novice players did far worse.

Random chess positions (not legal ones) were then shown to the two groups:

- Experts and novices did just as badly at reconstructing them!
People are still working on this problem...

http://people.brunel.ac.uk/~hssfflg/fig-research/chess_expertise/

Tic Tac Toe as search

How can we pose this as a search problem?

Tic Tac Toe as search
Tic Tac Toe as search

Eventually, we’ll get to a leaf

The **UTILITY** of a state tells us how good the states are.

Defining the problem

- **INITIAL STATE** – board position and the player whose turn it is
- ** Successor Function** – returns a list of (move, next state) pairs
- **Terminal Test** – is game over? Are we in a terminal state?
- **Utility Function** – (objective or payoff func) gives a numeric value for terminal states (ie – chess – win/lose/draw +1/-1/0, backgammon +192 to -192)

Games’ Branching Factors

- On average, there are ~35 possible moves that a chess player can make from any board configuration…

Boundaries for **qualitatively different games**…
Games’ Branching Factors

- On average, there are ~35 possible moves that a chess player can make from any board configuration...

```
CHINOOK (2007)
```

<table>
<thead>
<tr>
<th>“solved” games</th>
<th>0 Ply</th>
<th>1 Ply</th>
<th>2 Ply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connect Four</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checkers</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Othello</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chess</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Go</td>
<td>300</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Games vs. search problems?

Opponent!
- unpredictable/uncertainty
- deal with opponent strategy

Time limitations
- must make a move in a reasonable amount of time
- can’t always look to the end

Path costs
- not about moves, but about UTILITY of the resulting state/winning

Back to Tic Tac TOe

```
X's turn
```

```
O's turn
```

```
X's turn
```

```
...
```

```
...
```

I’m X, what will ‘O’ do?

```
X  X  O
O  X  O
X
```

```
O's turn
```

```
X  X  O
O  X  O
X  O
```

```
X  O
```

```
X  O
```
Minimizing risk

The computer doesn’t know what move \(O\) (the opponent) will make.

It can assume, though, that it will try and make the best move possible.

Even if \(O\) actually makes a different move, we’re no worse off.

Optimal Strategy

An **Optimal Strategy** is one that is at least as good as any other, no matter what the opponent does.

- If there’s a way to force the win, it will.
- Will only lose if there’s no other option.

How can \(X\) play optimally?

Start from the leaves and propagate the utility up:

- if \(X\)’s turn, pick the move that maximizes the utility.
- if \(O\)’s turn, pick the move that minimizes the utility.

Is this optimal?
Minimax Algorithm: An Optimal Strategy

- Uses recursion to compute the "value" of each state
- Proceeds to the leaves, then the values are "backed up" through the tree as the recursion unwinds
- What type of search is this?
- What does this assume about how MIN will play? What if this isn’t true?

```
def minimax(state):
    for all actions in actions(state):
        return the a with the largest minValue(result(state,a))

def maxVal(state):
    if state is terminal:
        return utility(state)
    else:
        return the a with the largest maxVal(result(state,a))
        for all actions a in actions(state):
            value = max(value, maxVal(result(state,a)))
        return value

def minValue(state):
    if state is terminal:
        return utility(state)
    else:
        return the a with the smallest maxVal(result(state,a))
        for all actions a in actions(state):
            value = min(value, maxVal(result(state,a)))
        return value
```

ME: Assume the opponent will try and minimize value, maximize my move

OPPONENT: Assume I will try and maximize my value, minimize his/her move

Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

What move should I take?
Take 1 or 2 at each turn
Goal: take the last match

MAX wins
\[
\begin{array}{c}
\downarrow \\
= 1.0
\end{array}
\]
MIN wins/
MAX loses
\[
\begin{array}{c}
\downarrow \\
= -1.0
\end{array}
\]
Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

MAX wins
\(= 1.0\)

MIN wins/ MAX loses
\(= -1.0\)
Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

MAX wins
\[ = 1.0 \]

MIN wins/
MAX loses
\[ = -1.0 \]

Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

MAX wins
\[ = 1.0 \]

MIN wins/
MAX loses
\[ = -1.0 \]

Baby Nim

could still win,
but not optimal!!!

Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

MAX wins
\[ = 1.0 \]

MIN wins/
MAX loses
\[ = -1.0 \]
**Minimax example 2**

Which move should be made: $A_1$, $A_2$ or $A_3$?

---

**Properties of minimax**

Minimax is optimal!

Are we done?
- For chess, $b = 35$, $d \approx 100$ for reasonable games → exact solution completely infeasible
- Is minimax feasible for Mancala or Tic Tac Toe?
  - Mancala: 6 possible moves, average depth of 40, so $6^{40}$ which is on the edge
  - Tic Tac Toe: branching factor of 4 (on average) and depth of 9... yes!

Ideas?
- pruning!
- improved state utility/evaluation functions
Pruning: do we have to traverse the whole tree?

Minimax example 2

Minimax example 2

Minimax example 2

Prune!

Any others if we continue?
Minimax example 2

Max

MIN

4 12 7 10 3 2

prune!

Alpha-Beta pruning

An optimal pruning strategy
– only prunes paths that are suboptimal (i.e., wouldn’t be chosen by an optimal playing player)
– returns the same result as minimax, but faster

As we go, keep track of the best and worst along a path
– alpha = best choice we’ve found so far for MAX
– beta = best choice we’ve found so far for MIN

Alpha-Beta pruning

alpha = best choice we’ve found so far for MAX

Using alpha and beta to prune:
– We’re examining MIN’s options for a ply
– To do this, we’re examining all possible moves for MAX. If we find a value for one of MAX’s moves that is less than alpha, return. (MIN could do better down this path)

MIN

MAX

return if any < alpha

Alpha-Beta pruning

beta = best choice we’ve found so far for MIN

Using alpha and beta to prune:
– We’re examining MAX’s options for a ply
– To do this, we’re examining all possible moves for MIN. If we find a value for one of MIN’s possible moves that is greater than beta, return. (MIN won’t end up down here)

MIN

MAX

return if any > beta
Alpha-Beta pruning

Do DFS until we reach a leaf:

What do we know?

alpha = best choice we've found so far for MAX
beta = best choice we've found so far for MIN

3

3

3

≤ 3
What do we know?

alpha = best choice we’ve found so far for MAX
beta = best choice we’ve found so far for MIN
What do we know?

alpha = best choice we’ve found so far for MAX
beta = best choice we’ve found so far for MIN

Prune!
What do we know?

alpha = best choice we've found so far for MAX
beta = best choice we've found so far for MIN

\[ \begin{align*}
3 & \leq 2 \\
3 & \leq 5
\end{align*} \]
def maxValue(state, alpha, beta):
    if state is terminal:
        return utility(state)
    else:
        value = -∞
        for all actions a in actions(state):
            value = max(value, minValue(result(state,a), alpha, beta))
        if value >= beta:
            return value
        alpha = max(alpha, value) # update alpha
        return value

We’re making a decision for MAX.
• When considering MIN’s choices, if we find a value that is greater than beta, stop, because MIN won’t make this choice
• If we find a better path than alpha, update alpha

alpha = best choice we’ve found so far for MAX
beta = best choice we’ve found so far for MIN

def minValue(state, alpha, beta):
    if state is terminal:
        return utility(state)
    else:
        value = +∞
        for all actions a in actions(state):
            value = min(value, maxValue(result(state,a), alpha, beta))
        if value <= alpha:
            return value
        beta = min(beta, value) # update beta
        return value

We’re making a decision for MIN.
• When considering MAX’s choices, if we find a value that is less than alpha, stop, because MAX won’t make this choice
• If we find a better path than beta for MIN, update beta

alpha = best choice we’ve found so far for MAX
beta = best choice we’ve found so far for MIN

Baby NIM2: take 1, 2 or 3 sticks

Effectiveness of pruning
Notice that as we gain more information about the state of things, we’re more likely to prune

What affects the performance of pruning?
– key: which order we visit the states
– can try and order them so as to improve pruning
Effectiveness of pruning

If perfect state ordering:
- $O(b^m)$ becomes $O(b^{m/2})$
- We can solve a tree twice as deep!

Random order:
- $O(b^m)$ becomes $O(b^{3m/4})$
- still pretty good

For chess using a basic ordering
- Within a factor of 2 of $O(b^{m/2})$