Adversarial Search

CS30
David Kauchak
Spring 2016

A quick review of search

Problem solving via search:
• To define the state space, define three things:
  – is_goal
  – next_states
  – starting_state

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

Why should we study games?

Clear success criteria

Important historically for AI

Fun 😊

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

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

Admin

• Assignment 10 out
  – May work in groups of up to 4 people
  – Due Sunday 4/24 (though, don’t wait until the weekend to finish!)
### 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?

### Strategic thinking ≠ intelligence

Humans and computers have different relative strengths in these games:

- ?
- ?
Strategic thinking ≠ 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/~hstffg/frg-research/chess_expertise/

Tic Tac Toe as search

Example of eye movements (presentation time = 5 seconds)

Master's eye movements
Novice's eye movements

If we want to write a program to play tic tac toe, what question are we trying to answer?

Given a state (i.e. board configuration), what move should we make!
How can we pose this as a search problem?

Eventually, we’ll get to a leaf

How does this help us?

Try and make moves that move us towards a win, i.e. where there are leaves with a WIN.
Tic Tac Toe

X’s turn

O’s turn

X’s turn

Problem: we don’t know what O will do

I’m X, what will ‘O’ do?

O’s turn

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. Why?

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
Defining a scoring function

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X X O</td>
</tr>
<tr>
<td>X O O</td>
</tr>
<tr>
<td>X O X</td>
</tr>
</tbody>
</table>

Defining a scoring function

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X X O</td>
</tr>
<tr>
<td>X O O</td>
</tr>
<tr>
<td>X O X</td>
</tr>
</tbody>
</table>

Defining a scoring function

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X X O</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Defining a scoring function

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X X O</td>
</tr>
<tr>
<td>O X O</td>
</tr>
<tr>
<td>O X O</td>
</tr>
</tbody>
</table>

Defining a scoring function

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X X O</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Defining a scoring function

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X X O</td>
</tr>
<tr>
<td>O X O</td>
</tr>
<tr>
<td>O X O</td>
</tr>
</tbody>
</table>

Defining a scoring function

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X X O</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Idea:**
• define a function that gives us a “score” for how good each state is for us
• higher scores mean better for us

**WIN**
+1: we can get to a win

**TIE**
0

**LOSE**
-1: we can get to a win
Defining a scoring function

Our (X) turn

What should be the score of this state?

0: If we play perfectly and so does O, the best we can do is a tie (could do better if O makes a mistake)
How can X play optimally?

Start from the leaves and propagate the score up:
- if X’s turn, pick the move that maximizes the utility
- if O’s turn, pick the move that minimizes the utility

Minimax Algorithm: An Optimal Strategy

\[
\text{minimax(state)} = \begin{cases} 
\text{Utility(state)} & \text{if state is a terminal state} \\
\text{max(minimax(next state))} & \text{if MY turn} \\
\text{min(minimax(next state))} & \text{if OPPONENTS turn}
\end{cases}
\]

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

What move should I take?
Baby Nim

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

MAX wins
\[
\text{\(\downarrow \)} = 1.0
\]

MIN wins/
MAX loses

\[
\text{\(\uparrow \)} = -1.0
\]

Baby Nim

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

MAX wins
\[
\text{\(\downarrow \)} = 1.0
\]

MIN wins/
MAX loses

\[
\text{\(\uparrow \)} = -1.0
\]
Take 1 or 2 at each turn
Goal: take the last match

MAX wins
\[ \downarrow = 1.0 \]
MIN wins/
MAX loses
\[ \uparrow = -1.0 \]
Take 1 or 2 at each turn
Goal: take the last match

MAX wins
\(\downarrow = 1.0\)

MIN wins/
MAX loses
\(\triangle = -1.0\)

Could still win,
but not optimal!!!
Minimax example 2

MAX

MIN

Properties of minimax

Minimax is optimal!

Are we done?

Games State Space Sizes

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

Hydra at home in the United Arab Emirates…

Boundaries for qualitatively different games…

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

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

AlphaGo (created by Google), just in April beat one of the best Go players:


What do we do?

Pruning helps get a bit deeper
For many games, still can’t search the entire tree

Now what?
Games State Space Sizes

Pruning helps get a bit deeper
For many games, still can’t search the entire tree

Go as deep as you can:
- estimate the score/quality of the state (called an evaluation function)
- use that instead of the real score

<table>
<thead>
<tr>
<th></th>
<th>1 Ply</th>
<th>2 Ply</th>
<th>1 Ply</th>
<th>2 Ply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Connect Four</td>
<td>7</td>
<td>49</td>
<td>11</td>
<td>121</td>
</tr>
<tr>
<td>Checkers</td>
<td>10</td>
<td>100</td>
<td>30</td>
<td>900</td>
</tr>
<tr>
<td>Othello</td>
<td>30</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chess</td>
<td>35</td>
<td>875</td>
<td>35</td>
<td>24625</td>
</tr>
<tr>
<td>Go</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Branching Factor Estimates for different two-player games

Computer-dominated

Tic Tac Toe evaluation functions

Ideas?

Example Tic Tac Toe EVAL

Tic Tac Toe
Assume MAX is using “X”

\[ \text{EVAL(state)} = \begin{cases} +\infty & \text{if state is win for MAX:} \\ -\infty & \text{if state is win for MIN:} \\ \text{(number of rows, columns and diagonals available to MAX)} - \text{(number of rows, columns and diagonals available to MIN)} & \text{else} \end{cases} \]

\[ \begin{array}{c|c|c} \hline & X & O \\ \hline X & & \\ \hline \end{array} \Rightarrow 6 - 4 = 2 \]

\[ \begin{array}{c|c|c} \hline O & X & X \\ \hline O & & \\ \hline \end{array} \Rightarrow 4 - 3 = 1 \]

Chess evaluation functions

Ideas?
Chess EVAL

Assume each piece has the following values:
- pawn = 1;
- knight = 3;
- bishop = 3;
- rook = 5;
- queen = 9;

$$EVAL(state) = \text{sum of the value of white pieces} - \text{sum of the value of black pieces}$$

= 31 - 36 = -5

Any problems with this?

Chess EVAL

Ignores actual positions!

Actual heuristic functions are often a weighted combination of features

$$EVAL(s) = w_1f_1(s) + w_2f_2(s) + w_3f_3(s) + ...$$

A feature can be any numerical information about the board
- as general as the number of pawns
- to specific board configurations

Deep Blue: 8000 features!
history/end-game tables

History
- keep track of the quality of moves from previous games
  - use these instead of search

end-game tables
- do a reverse search of certain game configurations, for example all board configurations with king, rook and king
- tells you what to do in any configuration meeting this criterion
- if you ever see one of these during search, you lookup exactly what to do

end-game tables

Devastatingly good

Allows much deeper branching
- for example, if the end-game table encodes a 20-move finish and we can search up to 14
  - can search up to depth 34

Stiller (1996) explored all end-games with 5 pieces
- one case check-mate required 262 moves!

Knoval (2006) explored all end-games with 6 pieces
- one case check-mate required 517 moves!

Traditional rules of chess require a capture or pawn move within 50 or it’s a stalemate

Opening moves

At the very beginning, we’re the farthest possible from any goal state

People are good with opening moves

Tons of books, etc. on opening moves

Most chess programs use a database of opening moves rather than search