Admin

Assignment 10

Midterm 2 on Monday
- From dictionaries (2/21) through informed search (4/11)
- Practice problems available
- 2 page “cheat” sheet
- Will try and have an additional mentor session for midterm questions over the weekend

Tic Tac Toe as search

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

Now what?
Tic Tac Toe as search

Eventually, we'll get to a state without any options

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

Problem: we don't know what O will do
Minimizing risk
The computer doesn’t know what move O (the opponent) will make

It can assume 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

Our (X) turn

What should be the score of this state?

+1: we can get to a win
Defining a scoring function

Opponent’s (O) turn

What should be the score of this state?

-1: opponent can get to a win

Our (X) turn

What should be the score of this state?
Defining a scoring function

Our (X) turn

What's the score of this state?

O turn

X turn

+1

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?

When it's my turn, pick the highest scoring state

When it's the opponent's turn, assume the lowest scoring state (from my perspective)

If we can reach the end games, we can percolate these answers all the way back up
How can X play optimally?

Start from the bottom 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

Is this optimal?

Minimax Algorithm: An Optimal Strategy

\[
\text{minimax}(\text{state}) =
\begin{align*}
\text{if state is a terminal state} & \quad \text{score(state)} \\
\text{else if MY turn} & \quad \text{over all next states, s: return the maximum of minimax(s)} \\
\text{else if OPPONENTS turn} & \quad \text{over all next states, s: return the minimum of minimax(s)}
\end{align*}
\]

Uses recursion to compute the “value” of each state

Searches down to the leaves, then the values are “backed up” through the tree as the recursion finishes

What type of search is this?

What does this assume about how MIN will play? What if this isn’t true?

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
\textcolor{green}{\downarrow} = 1.0
\textcolor{red}{\downarrow} = -1.0
MIN wins/
MAX loses
Take 1 or 2 at each turn
Goal: take the last match

MAX wins
Green triangle = 1.0
Red triangle = -1.0

MIN wins/
MAX loses

MAX wins
Green triangle = 1.0
Red triangle = -1.0

MIN wins/
MAX loses

MAX wins
Green triangle = 1.0
Red triangle = -1.0

MIN wins/
MAX loses

MAX wins
Green triangle = 1.0
Red triangle = -1.0

MIN wins/
MAX loses

MAX wins
Green triangle = 1.0
Red triangle = -1.0

MIN wins/
MAX loses

MAX wins
Green triangle = 1.0
Red triangle = -1.0

MIN wins/
MAX loses

MAX wins
Green triangle = 1.0
Red triangle = -1.0

MIN wins/
MAX loses
Baby Nim

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

MAX wins
\[= 1.0\]

\[= -1.0\]

MIN wins/
MAX loses

Which move?
Baby Nim

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

MAX wins
\[ \downarrow = 1.0 \]

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

could still win, but not optimal!!!

Minimax example 2

Which move should be made: A₁, A₂ or A₃?

Properties of minimax

Minimax is optimal!

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

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

AlphaGo (created by Google), in April 2016 beat one of the best Go players:

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

Games State Space Sizes

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

<table>
<thead>
<tr>
<th></th>
<th>0 Ply</th>
<th>1 Ply</th>
<th>2 Ply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connect Four</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checkers</td>
<td></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>

computer-dominated
Tic Tac Toe evaluation functions

Example Tic Tac Toe EVAL

Tic Tac Toe
Assume MAX is using “X”

EVAL(state) =

if state is win for MAX:

+ ¥

if state is win for MIN:

- ¥

else:

(number of rows, columns and diagonals available to MAX) -

(number of rows, columns and diagonals available to MIN)

= 6 - 4 = 2

= 4 - 3 = 1

Chess evaluation functions

Chess EVAL

Assume each piece has the following value:

pawn = 1;

knight = 3;

bishop = 3;

rock = 5;

queen = 9;

EVAL(state) =

sum of the value of white pieces –

sum of the value of black pieces

= 31 - 36 = -5
Assume each piece has the following values:
- pawn = 1;
- knight = 3;
- bishop = 3;
- rook = 5;
- queen = 9;

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

Any problems with this?

Ignores actual positions!

Actual heuristic functions are often a weighted combination of features:

\[ \text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + w_3 f_3(s) + \ldots \]

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

- 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

Nim

K piles of coins

On your turn you must take one or more coins from one pile

Player that takes the last coin wins

Example:
https://www.goobix.com/games/nim/