21: More adversarial search
Lecture 22: More adversarial search

- Minimax
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!
Tic tac toe as search

MINIMAX
Tic tac toe as search

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

MINIMAX
Tic tac toe as search

Now what?
MINIMAX

Tic tac toe as search

O’s turn!
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 as search

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

Tic tac toe as search
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

- **WIN**: +1
- **TIE**: 0
- **LOSE**: -1

**Idea:**

- Define a function that gives us a “score” for how good each state is.
- Higher scores mean better.
Defining a scoring function

Our (X) turn

What should be the score of this state?
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

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

Opponent’s (O) turn

What should be the score of this state?

-1: opponent can get to a win
MINIMAX

Opponent’s (O) turn

-1

+1

-1
Defining a scoring function

Our (X) turn

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

Our (X) turn

O turn

X turn

What's the score of this state?
Defining a scoring function

Our (X) turn

O turn

X turn

What’s the score of this state?
Defining a scoring function

Our (X) turn

O turn

X turn

What’s the score of this state?
Defining a scoring function

Our (X) turn

O turn

X turn
How can X play optimally?

[Diagram of a decision tree for the game of Tic-Tac-Toe, showing possible moves and outcomes with utility values of -1, 0, and +1 for X.]
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

\[
\text{minimax}(\text{state}) = \begin{cases} 
\text{score}(\text{state}) & \text{if state is a terminal state} \\
\max_{\text{all next states, } s} \text{minimax}(s) & \text{else if MY turn} \\
\min_{\text{all next states, } s} \text{minimax}(s) & \text{else if OPPONENTS turn} 
\end{cases}
\]

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
Minimax Algorithm: An Optimal Strategy

```
minimax(state) =
    if state is a terminal state
        score(state)
    else if MY turn
        over all next states, s: return the maximum of minimax(s)
    else if OPPONENTS turn
        over all next states, s: return the minimum of minimax(s)
```

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?
Minimax Algorithm: An Optimal Strategy

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

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?

DFS!
Baby Nim

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

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

- MAX wins
  \[\text{\color{green}W} = 1.0\]
  \[\text{\color{red}X} = -1.0\]
- MIN wins/
  MAX loses
Take 1 or 2 at each turn. 
Goal: take the last match.
Take 1 or 2 at each turn
Goal: take the last match

MAX wins

\[ \text{W} = 1.0 \]

MIN wins/ MAX loses

\[ \text{W} = -1.0 \]
MINIMAX

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

MAX wins

\[ \text{W} = 1.0 \]

MIN wins/
MAX loses

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

MAX wins

\[ \text{MAX wins} \]

\[ = 1.0 \]

MIN wins/
MAX loses

\[ \text{MIN wins/} \]

\[ \text{MAX loses} \]

\[ = -1.0 \]
Take 1 or 2 at each turn
Goal: take the last match

MAX wins

\[ = 1.0 \]

MIN wins/
MAX loses

\[ = -1.0 \]
Take 1 or 2 at each turn
Goal: take the last match

MAX wins
-1.0

MIN wins/
MAX loses
1.0
MINIMAX

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

MAX wins

\[ = 1.0 \]

MIN wins/
MAX loses

\[ = -1.0 \]
Take 1 or 2 at each turn
Goal: take the last match

MAX wins

W = 1.0

MIN wins/
MAX loses

MAX

W = -1.0
Take 1 or 2 at each turn
Goal: take the last match

MAX wins
\[ \begin{align*}
\text{W} & = 1.0 \\
\text{X} & = -1.0 
\end{align*} \]

MIN wins/
MAX loses

\[ \text{MINIMAX} \]
Take 1 or 2 at each turn
Goal: take the last match

MAX wins

\[ \text{MAX wins} \]

\[ \text{MIN wins/ MAX loses} \]

\[ \text{MAX wins} = 1.0 \]

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

MAX wins

\[ \text{W} = 1.0 \]

MIN wins/
MAX loses

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

MAX wins

- \[ = 1.0 \]

MIN wins/
MAX loses

- \[ = -1.0 \]
Take 1 or 2 at each turn
Goal: take the last match

MINIMAX

could still win, but not optimal!!!

MAX wins

\[
\begin{align*}
\text{MAX wins} & : W = 1.0 \\
\text{MIN wins/} & \text{MAX loses} : \Delta W = -1.0
\end{align*}
\]
Which move should be made: A₁, A₂ or A₃?
MINIMAX

MAX

MIN

0

A_1

A_2

A_3

A_{11} A_{12} A_{13}

A_{21} A_{22} A_{23}

A_{31} A_{32} A_{33}

3 12 8

2 4 6

2 5 14
Properties of minimax

Minimax is optimal! Are we done?
Game state space

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

<table>
<thead>
<tr>
<th>Game</th>
<th>Branching Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>4</td>
</tr>
<tr>
<td>Connect Four</td>
<td>7</td>
</tr>
<tr>
<td>Checkers</td>
<td>10</td>
</tr>
<tr>
<td>Othello</td>
<td>30</td>
</tr>
<tr>
<td>Chess</td>
<td>35</td>
</tr>
<tr>
<td>Go</td>
<td>300</td>
</tr>
</tbody>
</table>
Game state space

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

Boundaries for qualitatively different games...

### Branching Factor Estimates for different two-player games

- Tic-tac-toe: 4
- Connect Four: 7
- Checkers: 10
- Othello: 30
- Chess: 35
- Go: 300
Game state space

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

```
<table>
<thead>
<tr>
<th>Game</th>
<th>Branching Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>4</td>
</tr>
<tr>
<td>Connect Four</td>
<td>7</td>
</tr>
<tr>
<td>Checkers</td>
<td>10</td>
</tr>
<tr>
<td>Othello</td>
<td>30</td>
</tr>
<tr>
<td>Chess</td>
<td>35</td>
</tr>
<tr>
<td>Go</td>
<td>300</td>
</tr>
</tbody>
</table>
```

“solved” games

computer-dominated games

human-dominated games
Game state space

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

“solved” games

<table>
<thead>
<tr>
<th></th>
<th>0 Ply</th>
<th>1 Ply</th>
<th>2 Ply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching Factor Estimates for different two-player games</td>
<td></td>
<td></td>
<td></td>
</tr>
<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>

computer-dominated games

human-dominated games

Is this true?
MINIMAX

Game state space

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


<table>
<thead>
<tr>
<th>Game</th>
<th>Branching Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>4</td>
</tr>
<tr>
<td>Connect Four</td>
<td>7</td>
</tr>
<tr>
<td>Checkers</td>
<td>10</td>
</tr>
<tr>
<td>Othello</td>
<td>30</td>
</tr>
<tr>
<td>Chess</td>
<td>35</td>
</tr>
<tr>
<td>Go</td>
<td>300</td>
</tr>
</tbody>
</table>
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.
Game state space

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

computer-dominated games

Branching Factor Estimates for different two-player games:

<table>
<thead>
<tr>
<th>Game</th>
<th>Branching Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>4</td>
</tr>
<tr>
<td>Connect Four</td>
<td>7</td>
</tr>
<tr>
<td>Checkers</td>
<td>10</td>
</tr>
<tr>
<td>Othello</td>
<td>30</td>
</tr>
<tr>
<td>Chess</td>
<td>35</td>
</tr>
<tr>
<td>Go</td>
<td>300</td>
</tr>
</tbody>
</table>
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

### GAMES STATE SPACE SIZES

<table>
<thead>
<tr>
<th>Game</th>
<th>Branching Factor Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>4</td>
</tr>
<tr>
<td>Connect Four</td>
<td>7</td>
</tr>
<tr>
<td>Checkers</td>
<td>10</td>
</tr>
<tr>
<td>Othello</td>
<td>30</td>
</tr>
<tr>
<td>Chess</td>
<td>35</td>
</tr>
<tr>
<td>Go</td>
<td>300</td>
</tr>
</tbody>
</table>

MINIMAX
Tic Tac Toe evaluation functions

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

Ideas?
**MINIMAX**

**Tic Tac Toe**
Assume MAX is using “X”

\[ EVAL(state) = \]

if \( state \) is win for MAX:
   \(+ \infty\)
if \( state \) is win for MIN:
   \(- \infty\)
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

Ideas?
Assume each piece has the following values:

- **pawn** = 1;
- **knight** = 3;
- **bishop** = 3;
- **rook** = 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 value:
- 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?
Chess EVAL

- Ignores actual positions!
- Actual heuristic functions are often a weighted combination of features.
Chess EVAL

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

- number of pawns
- number of attacked knights
- 1 if king has knighted, 0 otherwise

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 (1991) explored all end-games with 6 pieces
  - one case check-mate required 223 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/](https://www.goobix.com/games/nim/)
Resources

- practice_midterm_2.py

Homework

- Assignment 10 (cont'd)