Adversarial Search

CS51A
David Kauchak, Joseph C. Osborn
Fall 2019

Some material borrowed from:
Sara Owsley Sood and others
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
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…

Important question: Is strategic thinking the same as intelligence?
Humans and computers have different relative strengths in these games:
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.
Strategic thinking $\neq$ intelligence

How could you figure out how humans approach playing chess?

humans

good at evaluating the strength of a board for a player
How humans play games...

An experiment 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/~hsstffg/frg-research/chess_expertise/
Tic Tac Toe as search

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
Tic Tac Toe as search

\[
\begin{array}{ccc}
X & X & O \\
O & O & O \\
X & O & X \\
\end{array}
\]

\[
\begin{array}{ccc}
X & X & O \\
X & O & O \\
X & O & X \\
\end{array}
\]
How can we pose this as a search problem?
Tic Tac Toe as search
Tic Tac Toe as search
Tic Tac Toe as search

Eventually, we’ll get to a leaf

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

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

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 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

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?

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

Opponent’s (O) turn

What should be the score of this state?

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

Opponent’s (O) turn

+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 +1

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

Our (X) turn

What’s the score of this state?
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?
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 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

Is this optimal?
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?

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

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

MIN wins/
MAX loses

W = 1.0

W = -1.0

W
Baby Nim

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

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

MIN wins/
MAX loses
\[ \text{\texttriangle} = -1.0 \]
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

\[
\text{W} = 1.0
\]

MIN wins/
MAX loses

\[
\text{W} = -1.0
\]
Baby Nim

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

MAX wins

W = 1.0

MIN wins/ MAX loses

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

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

MAX wins

\[
\begin{align*}
\text{\ding{115}} &= 1.0 \\
\text{\ding{116}} &= -1.0
\end{align*}
\]

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
Baby Nim

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

MAX wins

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

MIN wins/
MAX loses

\[ \text{MIN wins/ MAX loses} = -1.0 \]
Baby Nim

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

MAX wins

MAX

MIN

MAX

MIN

MAX

MIN

MAX

MIN

MAX

MIN

MAX

MIN

MAX

MIN

MAX

MIN
**Baby Nim**

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

MAX wins

\[ \begin{align*}
\text{MAX wins} &\quad \downarrow = 1.0 \\
\text{MIN wins/MAX loses} &\quad \uparrow = -1.0
\end{align*} \]
Baby Nim

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

MAX wins

MIN wins
MAX loses
**Baby Nim**

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

Which move?

- **MAX wins**
  - Green triangle: \(= 1.0\)
  - Green triangle with white: \(-1.0\)

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

**MAX wins**
- \(= 1.0\)
- \(-1.0\)

**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

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

Which move should be made: $A_1$, $A_2$ or $A_3$?
Minimax example 2

MAX

MIN

A1

A2

A3

A11 A12 A13

A21 A22 A23

A31 A32 A33

3 12 8

2 4 6

2 5 14
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...

Branching Factor Estimates for different two-player games

- Tic-tac-toe: 4
- Connect Four: 7
- Checkers: 10
- Othello: 30
- Chess: 35
- Go: 300
Games State Space Sizes

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

<table>
<thead>
<tr>
<th>Branching Factor Estimates for different two-player games</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
</tr>
<tr>
<td>Connect Four</td>
</tr>
<tr>
<td>Checkers</td>
</tr>
<tr>
<td>Othello</td>
</tr>
<tr>
<td>Chess</td>
</tr>
<tr>
<td>Go</td>
</tr>
</tbody>
</table>

Boundaries for qualitatively different games...
Games State Space Sizes

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

- CHINOOK (2007)
  - Can search entire space
  - Computer-dominated

- Computer-dominated

- Human-dominated

- Can't

Is this true?
Games State Space Sizes

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


“solved” games

Can search entire space

CHINOOK (2007)

computer-dominated

Can’t

Branching Factor Estimates for different two-player games

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

What do we do?
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?

Branching Factor Estimates for different two-player games

<table>
<thead>
<tr>
<th>Game</th>
<th>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>

computer-dominated
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

Branching Factor Estimates for different two-player games

<table>
<thead>
<tr>
<th>Game</th>
<th>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>
Tic Tac Toe evaluation functions

O X X
O X

Ideas?
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)

\[
\begin{array}{ccc}
X & O \\
\end{array}
\]

= 6 - 4 = 2

\[
\begin{array}{ccc}
O & X & X \\
O & \\
\end{array}
\]

= 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;

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

\[ = 31 - 36 = -5 \]
Chess EVAL

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?
Chess EVAL

Ignores actual positions!

Actual heuristic functions are often a weighted combination of features

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + w_3 f_3(s) + ...$$
**Chess EVAL**

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

- 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
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/