Outline

• Sequential-move games
  – Game trees, minimax, search approaches
• Speeding up sequential-move game search
  – Pruning, heuristics
Sequential-Move Games

More complex games with multiple moves
- Instead of normal form, **extensive form**
- Represent with a **tree**
- **Rewards at leaves**
- Find strategies: perform search over the tree
- Nash equilibrium still well-defined
  - Backward induction
II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

• Each player takes one or more sticks from pile
• Take last stick: lose

(ii, ii)

• Two players: Max and Min
• If Max wins, its score is $+1$; otherwise $-1$
• Min’s score is $-\text{Max}’s$ (two-player zero-sum)
• Use Max’s as the score of the game
Game Trajectory

(ii, ii)
Game Trajectory

(ii, ii)

Max takes one stick from one pile

(i, ii)
Game Trajectory

(ii, ii)

Max takes one stick from one pile

(i, ii)

Min takes two sticks from the other pile

(i,-)
Game Trajectory

(ii, ii)

Max takes one stick from one pile

(i, ii)

Min takes two sticks from the other pile

(i,-)

Max takes the last stick

(-,-)

Max gets score -1
Game tree for II-Nim

Two players:
Max and Min

Convention: score is w.r.t. the first player Max. Min’s score = – Max

Max wants the largest score
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score

Symmetry (i ii) = (ii i)
Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Game tree for II-Nim

Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Two players: 
Max and Min

Max wants the largest score
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score

Game tree for II-Nim
Two players: 
Max and Min

Max wants the largest score 
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Two players: 
Max and Min

Max wants the largest score
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Game tree for II-Nim

Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score
Two players: Max and Min

Max wants the largest score
Min wants the smallest score

The first player always loses, if the second player plays optimally!
Q 2.1: We are playing a game where Player A goes first and has 4 moves. Player B goes next and has 3 moves. Player A goes next and has 2 moves. Player B then has one move.

How many nodes are there in the minimax tree, including termination nodes (leaves)?

• A. 23
• B. 65
• C. 41
• D. 2
Q 2.1: We are playing a game where Player A goes first and has 4 moves. Player B goes next and has 3 moves. Player A goes next and has 2 moves. Player B then has one move.

How many nodes are there in the minimax tree, including termination nodes (leaves)?

- A. 23
- B. 65
- C. 41
- D. 2
Q 2.1: We are playing a game where Player A goes first and has 4 moves. Player B goes next and has 3 moves. Player A goes next and has 2 moves. Player B then has one move.

How many nodes are there in the minimax tree, including termination nodes (leaves)?

• A. 23
• B. 65 \((1 + 4 + 4\times3 + 4\times3\times2 + 4\times3\times2 = 65\). Note the root and leaf nodes.)
• C. 41
• D. 2
Q 2.2: During minimax tree search, must we examine every node?

- A. Always
- B. Sometimes
- C. Never
**Break & Quiz**

**Q 2.2:** During minimax tree search, must we examine every node?

- A. Always
- **B. Sometimes**
- C. Never
Q 2.2: During minimax tree search, must we examine every node?

- **A. Always** (No: consider layer \( k \), where we take the max of all the mins of its children at layer \( k+1 \). If the current value of a min node at \( k+1 \) already smaller than the current max, we don’t need to continue the minimization.)
- **B. Sometimes**
- **C. Never** (No: the event above may simply not happen).
Our Approach So Far

We find the minimax value/strategy bottom up

• Minimax value: score of terminal node when both players play optimally
  – Max’s turn, take max of children
  – Min’s turn, take min of children

• Can implement this as depth-first search: minimax algorithm
Minimax Algorithm

function Max-Value(s)
inputs:
s: current state in game, Max about to play
output: best-score (for Max) available from s

if ( s is a terminal state )
then return ( terminal value of s )
else
    \( \alpha := -\text{infinity} \)
    for each \( s' \) in Succ(s)
        \( \alpha := \max( \alpha, \text{Min-value}(s') ) \)

return \( \alpha \)

function Min-Value(s)
output: best-score (for Min) available from s

if ( s is a terminal state )
then return ( terminal value of s )
else
    \( \beta := \text{infinity} \)
    for each \( s' \) in Succs(s)
        \( \beta := \min( \beta, \text{Max-value}(s') ) \)

return \( \beta \)

Time complexity?
- \( O(b^m) \)

Space complexity?
- \( O(bm) \)
Minimax algorithm in execution

\[ \alpha = -\infty \]

max

min

max

min
Minimax algorithm in execution

\[
\begin{align*}
\text{max} & \quad \alpha = -\infty \\
\text{min} & \quad \beta = +\infty
\end{align*}
\]
The execution on the terminal nodes is omitted.
Minimax algorithm in execution

\[
\begin{align*}
\text{max} & \quad \beta = 100 \\
\text{min} & \quad \alpha = -\infty \\
\text{max} & \quad \min
\end{align*}
\]
Minimax algorithm in execution

max

\[ \alpha = 100 \]

\[ \beta = 100 \]

min

max

min

C (200)

D (100)

E (120)

F (20)

G

H (150)

I (100)
Minimax algorithm in execution

\[ \max \]

\[ \min \]

\[ \max \]

\[ \min \]
Minimax algorithm in execution

\[
\begin{align*}
\text{max} & \quad \alpha = 100 \\
\text{min} & \quad \beta = 120 \\
\text{max} & \\
\text{min} &
\end{align*}
\]
Minimax algorithm in execution

max

A

100

min

C

200

D

100

E

120

F

20

max

α=100

min

G

H

150

I

100

β=20

S
Minimax algorithm in execution

max

S

α=100

B

β=20

G

α=-∞

min

max

min

A

100

C

200

D

100

E

120

F

20

H

150

I

100
Minimax algorithm in execution

max

min

max

min

α=100

β=20

α=150
Minimax algorithm in execution

max

min

max

min
Minimax algorithm in execution

max

min

max

min

$S$

$B$

$E$

$D$

$C$

$F$

$G$

$H$

$I$

$A$

$S$

$B$

$E$

$F$

$G$

$H$

$I$

$A$

$max \quad α=100$

$min \quad β=20$

$A$

$B$

$E$

$F$

$G$

$H$

$I$

$α=100$

$β=20$

$100$

$200$

$100$

$120$

$20$

$150$

$100$

$150$

$100$
Minimax algorithm in execution

max

min

max

min
Can We Do Better?

One **downside**: we had to examine the entire tree

An idea to speed things up: **pruning**

- Goal: want the same minimax value, but faster
- We can get rid of bad branches
- Same principle as quiz question
Alpha-beta pruning

function Max-Value \((s, \alpha, \beta)\)
inputs:
  \(s\): current state in game, Max about to play
  \(\alpha\): best score (highest) for Max along path to \(s\)
  \(\beta\): best score (lowest) for Min along path to \(s\)
output: \(\min(\beta, \text{best-score (for Max) available from } s)\)
  if \((s \text{ is a terminal state})\)
  then return \((\text{terminal value of } s)\)
  else for each \(s'\) in Succ(s)
    \(\alpha := \max(\alpha, \text{Min-value}(s', \alpha, \beta))\)
    if \((\alpha \geq \beta)\) then return \(\beta\) /* alpha pruning */
  return \(\alpha\)

function Min-Value\((s, \alpha, \beta)\)
output: \(\max(\alpha, \text{best-score (for Min) available from } s)\)
  if \((s \text{ is a terminal state})\)
  then return \((\text{terminal value of } s)\)
  else for each \(s'\) in Succs(s)
    \(\beta := \min(\beta, \text{Max-value}(s', \alpha, \beta))\)
    if \((\alpha \geq \beta)\) then return \(\alpha\) /* beta pruning */
  return \(\beta\)

Starting from the root:
Max-Value(root, -\(\infty\), +\(\infty\))
How effective is alpha-beta pruning?

• Depends on the order of successors!
  – Best case, the # of nodes to search is $O(b^{m/2})$
  – Happens when each player's best move is the leftmost child.
  – The worst case is no pruning at all.

• In DeepBlue, the average branching factor was about 6 with alpha-beta instead of 35-40 without.
Minimax With Heuristics

Note that long games may require huge computation

- To deal with this: limit $d$ for the search depth
- **Q:** What to do at depth $d$, but no termination yet?
  - **A:** Use a heuristic evaluation function $e(x)$

```
function MINIMAX(x, d) returns an estimate of x’s utility value
  inputs: x, current state in game
  d, an upper bound on the search depth
  if x is a terminal state then return Max’s payoff at x
  else if d = 0 then return $e(x)$
  else if it is Max’s move at x then
    return max{MINIMAX(y, d−1) : y is a child of x}
  else return min{MINIMAX(y, d−1) : y is a child of x}
```

Credit: Dana Nau
Heuristic Evaluation Functions

• $e(x)$ can be any computable function of $x$; e.g. a weighted sum of features (like our linear models)

\[
e(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x)
\]

• Chess example: $f_i(x) =$ difference between number of white and black, with $i$ ranging over piece types.
  – Set weights according to piece importance
  – E.g., $1(\# \text{white pawns} - \# \text{black pawns}) + 3(\#\text{white knights} - \# \text{black knights})$
Going Further

• Monte Carlo tree search (MCTS)
  – Uses random sampling of the search space
  – Choose some children (heuristics to figure out #)
  – Record results, use for future play
  – Self-play

• AlphaGo and other big results!
From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.

- Player A has 4 pure strategies:
  - A’s strategy I: \((1 \rightarrow L, 4 \rightarrow L)\)
  - A’s strategy II: \((1 \rightarrow L, 4 \rightarrow R)\)
  - A’s strategy III: \((1 \rightarrow R, 4 \rightarrow L)\)
  - A’s strategy IV: \((1 \rightarrow R, 4 \rightarrow R)\)

- Player B has 3 pure strategies:
  - B’s strategy I: \((2 \rightarrow L, 3 \rightarrow R)\)
  - B’s strategy II: \((2 \rightarrow M, 3 \rightarrow R)\)
  - B’s strategy III: \((2 \rightarrow R, 3 \rightarrow R)\)

- How many pure strategies if each player can see \(N\) states, and has \(b\) moves at each state?
The matrix normal form is the game value matrix indexed by each player’s strategies.

<table>
<thead>
<tr>
<th></th>
<th>B-I</th>
<th>B-II</th>
<th>B-III</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-I</td>
<td>7</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>A-II</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>A-III</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>A-IV</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Another example of normal form

- How many pure strategies does A have?
- How many does B have?
- What is the matrix form of this game?
• How many pure strategies does A have? 4
  A-I (1→L, 4→L)  A-II (1→L, 4→R)  A-III (1→R, 4→L)  A-IV (1→R, 4→R)
• How many does B have? 4
  B-I (2→L, 3→L)  B-II (2→L, 3→R)  B-III (2→R, 3→L)  B-IV (2→R, 3→R)
• What is the matrix form of this game?
Minimax in Matrix Normal Form

- Player A: for each strategy, consider all B’s counter strategies (a row in the matrix), find the minimum value in that row. Pick the row with the maximum minimum value.
- Here maximin=5
Minimax in Matrix Normal Form

- Player B: find the maximum value in each column. Pick the column with the minimum maximum value.
- Here minimax = 5

Fundamental game theory result (proved by von Neumann):

In a 2-player, zero-sum game of perfect information (sequential moves), Minimax==Maximin. And there always exists an optimal pure strategy for each player.

<table>
<thead>
<tr>
<th></th>
<th>B-I</th>
<th>B-II</th>
<th>B-III</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-I</td>
<td>7</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>A-II</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>A-III</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>A-IV</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Minimax in Matrix Normal Form

• We can also check for mutual best responses

<table>
<thead>
<tr>
<th></th>
<th>B-I</th>
<th>B-II</th>
<th>B-III</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-I</td>
<td>7</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>A-II</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>A-III</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>A-IV</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Minimax in Matrix Normal Form

Interestingly, A can tell B in advance what strategy A will use (the maximin), and this information will not help B! Similarly B can tell A what strategy B will use. In fact A knows what B’s strategy will be. And B knows A’s too. And A knows that B knows … The game is at an equilibrium.

Fundamental game theory result (proved by von Neumann):
In a 2-player, zero-sum game of perfect information, Minimax = Maximin. And there always exists an optimal pure strategy for each player.