Announcements

• Homeworks:
  – HW8: Game AI released. Use this set of slides for reference.

• Midterm: grading nearly done.

• Class roadmap:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Thursday, April 1</td>
<td>Games I</td>
</tr>
<tr>
<td><strong>Tuesday, April 6</strong></td>
<td><strong>Games II</strong></td>
</tr>
<tr>
<td>Thursday, April 8</td>
<td>Search I</td>
</tr>
<tr>
<td>Tuesday, April 13</td>
<td>Search II</td>
</tr>
</tbody>
</table>
Outline

• Review of game theory basics
  – Properties, mathematical setup, simultaneous games

• Sequential games
  – Game trees, minimax, search approaches

• Speeding up sequential game search
  – Pruning, heuristics
Review of Games: Multiple Agents

Games setup: **multiple** agents

- Now: interactions between agents
- Still want to maximize utility
- **Strategic** decision making.
Review of Games: Properties

Let’s work through properties of games

• **Number** of agents/players
• State & action spaces: *discrete* or *continuous*
• **Finite** or **infinite**
• **Deterministic** or **random**
• **Sum**: zero or positive or negative
• **Sequential** or **simultaneous**
Review: Prisoner’s Dilemma

**Famous** example from the ‘50s.

Two prisoners A & B. Can choose to betray the other or not.

- A and B both betray, each of them serves two years in prison
- One betrays, the other doesn’t: betrayer free, other three years
- Both do not betray: one year each

**Properties:** 2-player, discrete, finite, deterministic, negative-sum, simultaneous
Review: Normal Form

Mathematical description of simult. games. Has:

• $n$ players $\{1, 2, ..., n\}$

• Player $i$ strategy $a_i$ from $A_i$. **All**: $a = (a_1, a_2, ..., a_n)$

• Player $i$ gets rewards $u_i(a)$ for any outcome
  – **Note**: reward depends on other players!

• Setting: all of these spaces, rewards are known
Review: Example of Normal Form

**Ex:** Prisoner’s Dilemma

<table>
<thead>
<tr>
<th>Player 2</th>
<th>Stay silent</th>
<th>Betray</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stay silent</td>
<td>−1, −1</td>
<td>−3, 0</td>
</tr>
<tr>
<td>Betray</td>
<td>0, −3</td>
<td>−2, −2</td>
</tr>
</tbody>
</table>

• **2 players, 2 actions:** yields 2x2 matrix
• Strategies: \{Stay silent, betray\} (i.e, **binary**)
• Rewards: \{0, -1, -2, -3\}
Review: Dominant Strategies

Let’s analyze such games. Some strategies are better

- Dominant strategy: if \( a_i \) better than \( a_i' \) regardless of what other players do, \( a_i \) is **dominant**

- I.e.,

  \[
  u_i(a_i, a_{-i}) \geq u_i(a_i', a_{-i}) \forall a_i' \neq a_i \text{ and } \forall a_{-i}
  \]

  All of the other entries of \( a \) excluding \( i \)

- Doesn’t always exist!
Review: Equilibrium

\( a^* \) is an equilibrium if all the players do not have an incentive to unilaterally deviate

\[
    u_i(a_i^*, a_{-i}^*) \geq u_i(a_i, a_{-i}^*) \quad \forall a_i \in A_i
\]

- All players dominant strategies -> equilibrium
- Converse doesn’t hold (don’t need dominant strategies to get an equilibrium)
So far, all our strategies are deterministic: “pure”

- Take a particular action, no randomness

Can also randomize actions: “mixed”

- Assign probabilities $x_i$ to each action

$$x_i(a_i), \text{ where } \sum_{a_i \in A_i} x_i(a_i) = 1, x_i(a_i) \geq 0$$

- Note: have to now consider expected rewards
Review: Nash Equilibrium

Consider the mixed strategy $x^* = (x_1^*, ..., x_n^*)$

• This is a **Nash equilibrium** if

$$u_i(x^*_i, x^*_{-i}) \geq u_i(x_i, x^*_{-i}) \quad \forall x_i \in \Delta_{A_i}, \forall i \in \{1, \ldots, n\}$$

Better than doing anything else, “best response”

Space of probability distributions

• Intuition: nobody can **increase expected reward** by changing only their own strategy. A type of solution!
Sequential Games

More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**
- Find strategies: perform search over the tree

- Can still look for Nash equilibrium
  - Or, other criteria like **maximin / minimax**
II-Nim: Example Sequential Game

2 piles of sticks, each with 2 sticks.

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

Two players: Max and Min

If Max wins, the score is +1; otherwise -1
Min’s score is –Max’s
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
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
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
Strategies & Rewards

Let’s stick to zero-sum two-player games

- Strategies: player 1 (Max): s, player 2 (Min): t
- Player 1 (Max): reward $u(s,t)$, player 2 (Min): $-u(s,t)$

- **Max** goal: maximize $u(s,t)$

- Goal: find strategies $s$, $t$ that do this.
Minimax Theorem

Famous result of von Neumann

- Says: there are strategies $s^*$ and $t^*$ and a value $u^*$, the minimax value so that
  - If Min uses $t^*$, then Max’s reward $\leq u^*$ (i.e., $\max_s u(s, t^*) = u^*$)
  - If Max uses $s^*$, then Max’s reward $\geq u^*$ (i.e., $\min_t u(s^*, t) = u^*$)

- So: $u(s^*, t^*) = u^*$

- Also: if game has perfect information, there are pure strategies $s^*, t^*$ that satisfy the result
Back to our game tree

• Write down all the pure strategies (e.g., the big tree) and select the $s^*$ and $t^*$

$$s^* = \arg \max_{s \in S} \min_{t \in T} u(s, t) \quad t^* = \arg \min_{t \in T} \max_{s \in S} u(s, t)$$

• Big search, since for branching factor $b$, height $h$, need to look at $\sim b^h$ strategies
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
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!
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`
`\[ \alpha := -\infty \]
`\quad \text{for each } s' \text{ in } \text{Succ}(s)`
`\quad \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`
`\[ \beta := \infty \]
`\quad \text{for each } s' \text{ in } \text{Succs}(s)`
`\quad \beta := \min( \beta , \text{Max-value}(s'))`
`return \beta`

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

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

max

min

max

min

$\alpha = -\infty$
Minimax algorithm in execution

max

A

\[ \alpha = -\infty \]

\[ \beta = +\infty \]

min

C

D

E

F

G

\[ \text{max} \]

\[ \text{min} \]

H

I

\[ \text{max} \]

\[ \text{min} \]
The execution on the terminal nodes is omitted.
Minimax algorithm in execution

max

$\alpha = -\infty$

$\beta = 100$

min

max

min
\textbf{Minimax algorithm in execution}

\begin{itemize}
\item max
\item $\beta=100$
\item min
\item max
\item min
\end{itemize}

\begin{itemize}
\item $\alpha=100$
\end{itemize}
Minimax algorithm in execution

\[ \alpha = 100 \]

\[ \beta = +\infty \]

(max) A

\[ \min \]

C 200

D 100

E 120

F 20

(max) B

\[ \beta = +\infty \]

(min) S

A 100

C 200

D 100

E 120

F 20

G

H 150

I 100

min
Minimax algorithm in execution

\[ \alpha = 100 \quad \beta = 120 \]
Minimax algorithm in execution

$S$

$\alpha = 100$

$\beta = 20$

$max$

$min$

$max$

$min$
Minimax algorithm in execution

max

min

max

min
Minimax algorithm in execution

max

min

max

min
Minimax algorithm in execution

max

min

α=100

β=20

α=150
Minimax algorithm in execution

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

max

α=100

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
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 ( \(s\) is a terminal state )
then return ( terminal value of \(s\) )
else for each \(s'\) in Succs(\(s\))
    \(\alpha := \max(\alpha, \text{Min-value}(s', \alpha, \beta))\)
    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 ( \(s\) is a terminal state )
then return ( 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 number 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 are yield 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 $e(x)$

```python
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

- $e(x)$ often 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!
Summary

• Review of game theory
  – Properties, Mathematical formulation for simultaneous games Normal form, dominance, equilibria, mixed vs pure

• Sequential games
  – Game trees, minimax value, minimax algorithm

• Improving our search
  – Using heuristics, pruning, random search
Acknowledgements: Developed from materials by Yingyu Liang (University of Wisconsin), James Skrentny (University of Wisconsin), inspired by Haifeng Xu (UVA) and Dana Nau (University of Maryland).