CMSC 474, Introduction to Game Theory

Perfect-Information Extensive Form Games

Mohammad T. Hajiaghayi
University of Maryland
The Sharing Game

Suppose agents 1 and 2 are two children

Someone offers them two cookies, but only if they can agree how to share them

Agent 1 chooses one of the following options:
- Agent 1 gets 2 cookies, agent 2 gets 0 cookies
- They each get 1 cookie
- Agent 1 gets 0 cookies, agent 2 gets 2 cookies

Agent 2 chooses to accept or reject the split:
- Accept => they each get their cookies(s)
- Otherwise, neither gets any

<table>
<thead>
<tr>
<th>2's move</th>
<th>1's move</th>
<th>0-2</th>
<th>1-1</th>
<th>2-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>(0,0)</td>
<td>(2,0)</td>
<td>(0,0)</td>
<td>(1,1)</td>
<td>(0,0)</td>
</tr>
</tbody>
</table>
Extensive Form

- The sharing game is a game in **extensive form**
  - A game representation that makes the temporal structure explicit
  - Doesn’t assume agents act simultaneously
- Extensive form can be converted to normal form
  - So previous results carry over
  - But there are additional results that depend on the temporal structure
- In a perfect-information game, the extensive form is a **game tree**:
  - **Choice (or nonterminal) node**: place where an agent chooses an action
  - **Edge**: an available **action** or **move**
  - **Terminal node**: a final outcome
  - At each terminal node $h$, each agent $i$ has a utility $u_i(h)$
Notation from the Book (Section 4.1)

- \( H = \{ \text{nonterminal nodes} \} \)
- \( Z = \{ \text{terminal nodes} \} \)
- If \( h \) is a nonterminal node, then
  - \( \rho(h) = \) the player to move at \( h \)
  - \( \chi(h) = \{ \text{all available actions at } h \} \)
  - \( \sigma(h,a) = \) node produced by action \( a \) at node \( h \)
  - \( h\text{'s children or successors} = \{ \sigma(h,a) : a \in \chi(h) \} \)
- If \( h \) is a node (either terminal or nonterminal), then
  - \( h\text{'s history} = \) the sequence of actions leading from the root to \( h \)
  - \( h\text{'s descendants} = \) all nodes in the subtree rooted at \( h \)
- The book doesn’t give the nodes names
  - The labels tell which agent makes the next move
Pure Strategies

- Pure strategy for agent \( i \) in a perfect-information game:
  - Function telling what action to take at every node where it’s \( i \)’s choice
    - i.e., every node \( h \) at which \( \rho(h) = i \)
- The book specifies pure strategies as lists of actions
  - Which action at which node?
  - Either assume a canonical ordering on the nodes, or use different action names at different nodes

Sharing game:

- Agent 1 has 3 pure strategies: \( S_1 = \{2-0, 1-1, 0-2\} \)
- Agent 2 has 8 pure strategies:
  - \( S_2 = \{(yes, yes, yes), (yes, yes, no), (yes, no, yes), (yes, no, no), (no, yes, yes), (no, yes, no), (no, no, yes), (no, no, no)\} \)
Extensive form vs. normal form

- Every game tree corresponds to an equivalent normal-form game
- The first step is to get all of the agents’ pure strategies
- Each pure strategy for \( i \) must specify an action at every node where it’s \( i \)’s move
- Example: the game tree shown here
  - Agent 1 has four pure strategies:
    - \( s_1 = \{(A, G), (A, H), (B, G), (B, H)\} \)
      - Mathematically, \( (A, G) \) and \( (A, H) \) are different strategies, even though action \( A \) makes the G-versus-H choice irrelevant
  - Agent 2 also has four pure strategies:
    - \( s_2 = \{(C, E), (C, F), (D, E), (D, F)\} \)
Extensive form vs. normal form

- Once we have all of the pure strategies, we can rewrite the game in normal form.
- Note that payoffs come from that of the unique leaf which will be accessible from the root.
- Converting to normal form introduces redundancy:
  - 16 outcomes in the payoff matrix, versus 5 outcomes in the game tree.
  - Payoff (3,8) occurs:
    - once in the game tree
    - four times in the payoff matrix.
- This can cause an exponential blowup.
**Nash Equilibrium**

- **Theorem.** Every perfect-information game in extensive form has a pure-strategy Nash equilibrium
  - This theorem has been attributed to Zermelo (1913), but there’s some controversy about that

- **Intuition:**
  - Agents take turns, and everyone sees what’s happened so far before making a move
  - So never need to introduce randomness into action selection to find an equilibrium

- In our example, there are three pure-strategy Nash equilibria
Nash Equilibrium

- The concept of a Nash equilibrium can be too weak for use in extensive-form games.

- Recall that our example has three pure-strategy Nash equilibria:
  - \{(A,G), (C,F)\}
  - \{(A,H), (C,F)\}
  - \{(B,H), (C,E)\}

- Here is \{(B,H), (C,E)\} with the game in extensive form.
Given a perfect-information extensive-form game $G$, the **subgame** of $G$ rooted at node $h$ is the restriction of $G$ to the descendants of $h$.

Now we can define a refinement of a Nash equilibrium:

**A subgame-perfect equilibrium (SPE)** is a strategy profile $s$ such that for every subgame $G'$ of $G$, the restriction of $s$ to $G'$ is a Nash equilibrium of $G'$.

- Since $G$ itself is a subgame of $G$, every SPE is also a Nash equilibrium.

Every perfect-information extensive-form game has at least 1 SPE:

- Can prove this by induction on the height of the game tree.
Example

- Recall that we have three Nash equilibria:
  - \{ (A, G), (C, F) \}
  - \{ (A, H), (C, F) \}
  - \{ (B, H), (C, E) \}

- Consider this subgame:
  - For agent 1, 
    - \( G \) strictly dominates \( H \)
  - Thus \( H \) can’t be part of a Nash equilibrium

- This excludes \{ (A, H), (C, F) \} and \{ (B, H), (C, E) \}
- Just one subgame-perfect equilibrium
  - \{ (A, G), (C, F) \}
Backward Induction

- To find subgame-perfect equilibria, we can use **backward induction**
- Identify the Nash equilibria in the bottom-most nodes
  - Assume they’ll be played if the game ever reaches these nodes
- For each node $h$, recursively compute a vector $v_h = (v_{h1}, ..., v_{hn})$ that gives every agent’s equilibrium utility
  - At each node $h$,
    - If $i$ is the agent to move, then $i$’s equilibrium action is to move to a child $h'$ of $h$ for which $i$’s equilibrium utility $v_{h'i}$ is highest
To find subgame-perfect equilibria, we can use **backward induction**.

- Identify the Nash equilibria in the bottom-most nodes
  - Assume they’ll be played if the game ever reaches these nodes

**procedure** backward-induction($h$)

```plaintext
if $h \in Z$ then return $u(h)$

$bestv = (-\infty, \ldots, -\infty)$

forall $a \in \chi(h)$ do

  $v = \text{backward-induction}(\sigma(h,a))$

  if $v[\rho(h)] > bestv[\rho(h)]$ then $bestv = v$

return $bestv$
```

---

**Diagram**

```
    1
   / \  \
  A   B
 / \\  / \ \
C   D E F
/ \ (3,8) (8,3) (5,5) 1 (2,10)
(3,8) (2,10) (2,10) (1,0)
```

---

**Backward Induction**

- (3,8) - (8,3) - (5,5) - (2,10) - (1,0)
The Centipede Game

- The players move in alternation
  - Player 1 makes the first move
  - Each player can go either Left or Right
- At each terminal node, the numbers are how many pieces of chocolate you’ll get
  - Next to each nonterminal node, I’ve put the SPE payoffs in red
A Problem with Backward Induction

- Can extend the centipede game to any length
- The only SPE is for each agent always to move Left
- But this isn’t intuitively appealing
- Seems unlikely that one would want to choose Left near the start of the game
  - If the agents continue the game for several moves, they’ll both get higher payoffs
- In lab experiments, subjects continue to choose Right until near the end of the game
Now consider a constant-sum version of the centipede game.

At every node, $u_2 = 5 - u_1$.
I need two more volunteers to play a constant-sum version of the centipede game.

At every node, $u_2 = 5 - u_1$.

Instead of having increasing payoffs for both players, the sum of their payoffs is always the same.

In this case, backward induction gives much more accurate results.
The Minimax Algorithm

- In constant-sum games, only need to compute agent 1’s SPE utility, $u_1$
  - $u_2 = c - u_1$
- From the Minimax Theorem,
  - at each node,
    - agent 1’s minmax value
    - = agent 1’s maxmin value
    - = agent 1’s SPE utility

procedure minimax($h$)

  if $h \in Z$ then return $u_1(h)$
  else if $\rho(h) = 1$ then return $\max_{a \in \chi(h)} u_1(\sigma(h,a))$
  else return $\min_{a \in \chi(h)} u_1(\sigma(h,a))$