1 Computation of Equilibria

1.1 Review

Nash Algorithm Given a simultaneous-move game, find a mixed strategy Nash equilibrium.

Two-player zero-sum games

- Maximin Strategy: \( \hat{s}_1 \in \arg \max_{s_1} [\min_{a_2 \in A_2} u_1(s_1, a_2)] \)
- Minimax Strategy: \( s_1 \in \arg \min_{s_1} [\max_{a_2 \in A_2} u_2(s_1, a_2)] \)
- Maximin and minimax value
- Minimax Theorem: In any two-player, zero-sum game,
  - For each player the set of maximin strategies is identical to the set of minimax strategies.
  - Any maximin strategy for player 1 and any maximin strategy for player 2 is a NE, and these correspond to all NE's.
  - Maximin value = minimax value = expected utility.

General-sum games

- Finding PSNE's for multiple-player general-sum games
- Iterated Elimination of Strictly Dominated Actions
- Support Enumeration Method

1.2 Practice

1. Comprehension Questions

(a) How do you find a MSNE for a two-player zero-sum games? Why is this feasible and why this does not apply to two-player general-sum games?

(b) Briefly explain why the maximin strategy can be solved for both players separately?

(c) Briefly explain why the set of minimax strategies and the set of maximin strategies are identical?
(d) Is it easy to find a pure strategy DSE in a multiple player general sum game? What about finding a pure Nash Equilibrium? If you had to write some code to find the pure NE, how would you do it? Comment on the complexity.

This is polynomial in the payoff matrix, whose size is, however, exponential in \( n \) and \( m \).

(e) Consider a 2 player general-sum game where player 1 has 2 possible actions and player 2 has 3 possible actions. How many different pairs of support exist when looking

2. 2-player general-sum game. Take the following game:

<table>
<thead>
<tr>
<th></th>
<th>Player 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Left</td>
</tr>
<tr>
<td>Up</td>
<td>(6,4)</td>
</tr>
<tr>
<td>Player 1 Down</td>
<td>(4,3)</td>
</tr>
</tbody>
</table>

(a) Find a pure strategy Nash equilibrium.

(b) Given that there exists a mixed strategy Nash equilibrium in the support UD,MR, formulate the LFP which could then be used to solve for this mixed Nash equilibrium. You do not need to solve the LFP.

(c) Briefly explain why we do not insist positive probability for strategies within the support, and why we’re able to do this?

2 Correlated equilibrium

1. Computation complexity of CE

(a) How to find a CE of a multiple-agent general-sum game?

(b) Is the complexity polynomial in the size of the input? Is it polynomial in the number of agents \( n \) or the number of actions \( m \)?

(c) Why is this easier than finding MSNE’s?

2. 3 Player Pollution Game. Assume there are 3 players and a river that is polluted. Cleaning the river costs 1 for each player and the benefit from a clear river is 3 for each player. However, at least two players are needed to clean the river, otherwise the effort is wasted. Denote “cleaning” as C and “do not clean” as D.

(a) For a mixed equilibrium, suppose agents 1, 2 and 3 are cleaning the river with probabilities \( p_1, p_2 \) and \( p_3 \) respectively \( s_i(C) = p_i \) and \( s_i(D) = 1 - p_i \), write the inequalities that define the MSNE for player 1.

(b) For a correlated equilibrium with \( p \) over the action profiles, write the inequalities that define the CE for player 2.

(c) Briefly explain why computing CE can be solved with LFP, whereas computing MSNE cannot.

(d) Any idea about how to solve for the MSNE’s?

(e) Can we cancel \( p_i \)’s and \( 1 - p_i \)’s from both sides of the inequalities of the MSNE? Why or why not?
3 The Class PPAD

3.1 Review

Reduction Intuitively, a reduction between two problems \( X \) and \( Y \) transforms an instance \( I_X \) of one problem into an instance \( I_Y \) of the other problem, usually in polynomial time. Then if we can solve \( Y \) in polynomial-time, we can also solve \( X \) in polynomial time via the reduction, solving the resulting instance of \( Y \), then taking the inverse reduction.

Complexity Classes

- **P**: yes-no decision problems that can be solved in worst case polynomial time, e.g. linear programming, 2SAT, max flow
- **NP**: verifiable in polynomial time
- **NP-hard**: “very hard” problems, not necessarily in \( NP \), and at least as hard as the hardest in \( NP \), e.g subset sum problem
- **NP-complete**: problems in \( NP \) and \( NP-hard \) (any problem in \( NP \) can be reduced to it in polynomial-time), e.g. traveling salesman (decision version)
- **PPAD**: total search problems that can be reduced to \textsc{End-of-the-Line}.
- **PPAD-hard**: problems to which \textsc{End-of-the-Line} can be reduced
- **PPAD-complete**: problems in \( PPAD \) and \( PPAD-hard \)

\textbf{Nash is PPAD-hard}

1. Equilibrium of two player general sum games - This can be found using linear feasibility programs given some supports. Because one needs to check all possible sets of supports, and there are an exponential number of possible supports, this algorithm has an exponential run time.
2. Finding Nash Equilibrium (NASH) in general sum games is **PPAD-Complete**. This means that a Nash Equilibrium definitely exists for every finite game, but it is conjectured that some instances might take exponential time to solve because it is believed **PPAD-Complete** cannot be solved in worst case polynomial time.

### 3.2 Practice

1. Explain briefly what does it mean for a problem to be **P**? **NP**? **NP-hard**?

2. Explain what does it mean for **Nash** to be in **PPAD**, in terms of how easy it is to find a solution and whether a solution exists.

3. The Battle of the Sexes

<table>
<thead>
<tr>
<th>Player1</th>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>3,2</td>
<td>0,0</td>
</tr>
<tr>
<td>D</td>
<td>0,0</td>
<td>2,3</td>
</tr>
</tbody>
</table>

(a) List all possible strategy labels

(b) Draw PPAD graph and find all Nash equilibrium

4. Gadgets

(a) For the multiplication gadget introduced in the Example 3.9 in the textbook and reproduced below, why is it important to design the payoff of agent 4 such that it gets 1 when it takes the opposite action of agent 3 and 0 otherwise?

\[
\begin{align*}
 u_3(a_3) &= \begin{cases}
 a_1 \cdot a_2, & \text{if } a_3 = 0 \\
 a_4, & \text{if } a_3 = 1
\end{cases} \\
 u_4(a_4) &= \begin{cases}
 1 - a_3, & \text{if } a_4 = 1 \\
 a_3, & \text{if } a_4 = 0
\end{cases}
\end{align*}
\]

(b) Is it important for agent 4 to know the value of \(a_1a_2\)? Why?

(c) Design a gadget with 4 agents 1, 2, 3, and 4 to implement \(a_4 = \max(a_1 - a_2, 0)\). Agents 1 and 2 are inputs and agent 4 gets payoff 1 when playing the opposite action as agent 3.