# Further Solution Concepts Computational Issues

S&LB §3.4.5, 3.4.7, 4.1, 4.2.3, 4.6

CMPUT 654: Modelling Human Strategic Behaviour

## Assignment #1

- Assignment #1 is released today
- Due **October 1** before lecture

See the website under Assignments (or on the Schedule)

### Recap: Solution Concepts

- Maxmin strategies maximize an agent's guaranteed payoff
- Minmax strategies minimize the other agent's payoff as much as possible
- The Minimax Theorem:
  - Maxmin and minmax strategies are the only Nash equilibrium strategies in zero-sum games
  - Every Nash equilibrium in a zero-sum game has the **same payoff**
- **Dominated strategies** can be removed **iteratively** without strategically changing the game (too much)
- Rationalizable strategies are any that are a best response to some rational belief

### Lecture Outline

- Recap & Logistics 1.
- 2.  $\varepsilon$ -Nash Equilibrium
- 3. Correlated Equilibrium
- Linear Programming 4.
- 5. Computing Nash Equilibrium
- 6. Computing Correlated Equilbrium

- In a Nash equilibrium, agents best respond perfectly •
- What if they are indifferent to very small gains in utility?
  - Could reflect modelling error (e.g., unmodelled cost of computational effort)

#### **Definition:**

For any  $\varepsilon > 0$ , a strategy profile s is an  $\varepsilon$ -Nash equilibrium if, for all agents *i* and strategies  $s'_i \neq s_i$ ,

## Equilibrium

- $\mathcal{U}_i(S_i, S_{-i}) \geq \mathcal{U}_i(S'_i, S_{-i}) \boldsymbol{\epsilon}.$

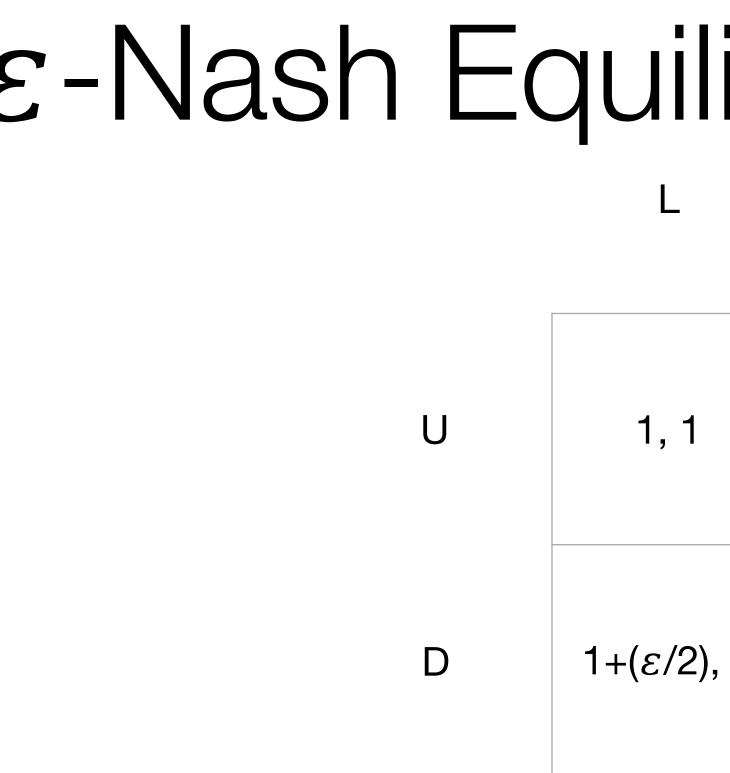
#### **Questions:**

For a given  $\epsilon > 0$ ,

1. Is an  $\varepsilon$ -Nash equilibrium guaranteed to exist?

2. Is more than one  $\varepsilon$ -Nash equilibrium guaranteed to exist?





- $\bullet$ 
  - $\bullet$ computes  $\varepsilon$ -Nash equilibrium
- **arbitrarily far** from Nash equilibrium payoffs.

### $\varepsilon$ -Nash Equilibrium Example

R

	0, 0
), 1	500, 500

### **Questions:**

What are the **Nash** equilibria of this game?

### 2. What are the $\varepsilon$ -Nash equilibria of this game?

Every Nash equilibrium is surrounded by a region of  $\varepsilon$ -Nash equilibria

Every **numerical algorithm** for computing Nash equilibrium actually

• However, the reverse is not true! Payoffs from an  $\varepsilon$ -Nash equilibrium can be

### Correlated Equilibrium Examples

	Ballet	Soccer
Ballet	2, 1	0, 0
Soccer	0, 0	1, 2

	Go	Wait
Go	-10, -10	1, 0
Wait	0, 1	-1, -1

- lacksquareutility of 2/3
- strategy equilibrium to play
  - Each would get utility of 1.5
- $\bullet$ 
  - is not necessary in general

In the unique mixed strategy equilibrium of Battle of the Sexes, each player gets a

If the players could first observe a coin flip, they could coordinate on which pure

• **Fairer** than either pure strategy equilibrium, and **Pareto dominates** the mixed strategy equilibrium

**Correlated equilibrium** is a solution concept in which agents get private, potentially-correlated **signals** before choosing their action

In both of these example, each agent sees the **same signal** perfectly, but that

## Correlated Equilibrium

#### **Definition:**

Given an *n*-agent game G = (N, A, u), a correlated equilibrium is a tuple  $(v, \pi, \sigma)$ , where

- $v = (v_1, \dots, v_n)$  is a tuple of random variables with domains  $(D_1, \dots, D_n)$ ,
- $\pi$  is a joint distribution over v,
- $\sigma = (\sigma_1, \dots, \sigma_n)$  is a vector of mappings  $\sigma_i : D_i \to A_i$ , and
- for every agent *i* and mapping  $\sigma': D_i \to A_i$ ,

 $d \in D_1 \times \cdots \times D_n$ 

**Question:** Why do the  $\sigma_i$ 's map to pure strategies instead of mixed strategies?

 $\pi(d)u_{i}(\sigma_{1}(d_{1}),...,\sigma_{n}(d_{n})) \geq \sum \pi(d)u_{i}(\sigma_{1}(d_{1}),...,\sigma_{i}(d_{i}),...,\sigma_{n}(d_{n}))$  $d \in D_1 \times \cdots \times D_n$ 



### Correlated Equilibrium Properties

#### **Theorem:**

For every **Nash equilibrium**, there exists a corresponding correlated equilibrium in which each action profile appears with the same frequency. (**how?**)

### **Theorem:** be realized in some correlated equilibrium. (how?)

Any **convex combination** of correlated equilibrium payoffs can

### Correlated Equilibrium Another Example

- In our example correlated equilibria, each agent best-responded to the other at every signal
  - This is not a requirement of a correlated equilibrium
- Consider this correlated equilibrium, with  $D_1 = \{x, y, z\}$  and  $D_2 = \{m, r\}$ :

$$\pi \left[ (x,m) \right] = .25 \qquad \sigma_r(x)$$

$$\pi\left[(y,m)\right] = .25 \qquad \sigma_r(y)$$

$$\pi\left[(z,r)\right] = .5 \qquad \sigma_r(z) =$$

- **Question:** Does the column player best-respond at each signal? lacksquare
- **Question:** What are the marginal probabilities for each player's actions?
- **Question:** What would happen if the agents played **mixed strategies** with those marginal probabilities?

- $= X \qquad \sigma_c(m) = M$ = Y
- $\sigma_r(r) = R$ =ZХ

Soccer	0, 0	
	Go	
Go	-10, -10	

Ballet

Wait

Μ

-10, -10	
0, 1	-

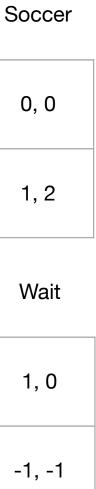
R

Ballet

2, 1

0,8	3,6	-9,1
0,2	3,9	-12,1(
1,0	0,-2	7,7

Ζ





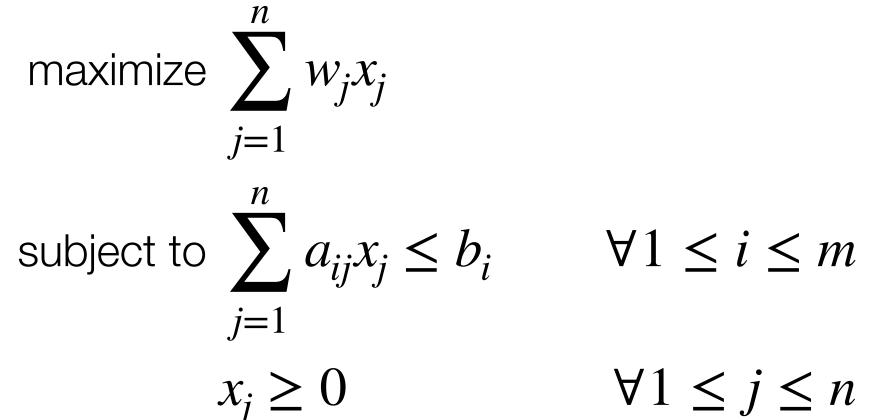
## Linear Programming

#### **Definition:**

A linear program consists of

- A set of real-valued variables  $\{x_1, \ldots, x_n\}$
- A linear objective function defined by weights  $\{w_1, \ldots, w_n\}$
- A set of linear **constraints** of the form  $\sum_{i=1}^{n} a_i x_i \le b$

#### Sample:



 $\forall 1 \leq j \leq n$ 

### Linear Program Properties

- Linear programs can be solved in **polynomial time** by generic algorithms (e.g., ellipsoid algorithm)
  - So writing a problem as a linear program constitutes a **proof** that it is solvable in polynomial time
- Negating weights  $w_i$  allows us to minimize or maximize the objective
- Negating constraint coefficients  $a_{ij}$  allows for greater-than-or-equal constraints
- Providing both greater-than-or-equal and less-than-or-equal constraints allows for equality constraints
- Cannot always express strict inequalities (although there are tricks)

$$\begin{array}{ll} \text{maximize } \sum_{j=1}^{n} w_{j} x_{j} \\ \text{subject to } \sum_{j=1}^{n} a_{ij} x_{j} \leq b_{i} & \forall 1 \leq \\ x_{j} \geq 0 & \forall 1 \leq \end{array}$$

 $\leq i \leq m$  $\leq j \leq n$ 

### Computing Nash Equilibrium

- **computationally hard** (*PPAD*-complete)
  - Even for two-player games!

• The problem of computing a Nash equilibrium is known to be

But there are some **special cases** that we can compute efficiently

### Computing Nash Equilibrium: Zero-Sum Games

- minimize  $U_1^*$  $a_2 \in A_2$  $\sum s_2(a_2) = 1$  $a_2 \in A_2$ 
  - $s_2(a_2) \ge 0$
- This linear program computes  $U_1^*$ , player 1's minmax value, and  $s_2$ , player 2's minmax strategy against player 1
  - By the minimax theorem, this is player 2's equilibrium strategy
- Compute player 1's equilibrium strategy analogously

subject to  $\sum u_1(a_1, a_2) s_2(a_2) \le U_1^*$   $\forall a_1 \in A_1$ 

 $\forall a_2 \in A_2$ 

### Computing Maxmin Strategies: Two-Player, General-Sum Games

- We can efficiently compute the maxmin strategies for agents in a two-player **zero-sum game**
- The maxmin strategy for an agent in a **general-sum game** is their best response to an imaginary agent that is **trying to hurt them**
- To compute player 1's maxmin strategy in a general-sum game:
  - 1. Construct a zero-sum game from player 1's payoffs,
  - 2. Find player 1's minmax strategy in the **constructed game** (using the program from the previous slide)

### Computing Nash Equilibrium: Two-Player, General Sum Games

- Finding an equilibrium in general is hard
- But if we already know the **support** of the equilibrium, then we can compute it efficiently in a two-player game:

$$\sum_{\substack{a_{-i} \in \sigma_{-i} \\ a_{-i} \in \sigma_{-i}}} s_{-i}(a_{-i})u_i(a_i, a_{-i}) = v_i$$

$$\sum_{\substack{a_{-i} \in \sigma_{-i} \\ s_i(a_i) \ge 0 \\ s_i(a_i) = 0}$$

$$\sum_{\substack{a_i \in A_i \\ s_i(a_i) = 1}} s_i(a_i) = 1$$

 $\forall i \in \{1,2\}, a_i \in \sigma_i$ 

 $\forall i \in \{1,2\}, a_i \notin \sigma_i$ 

 $\forall i \in \{1,2\}, a_i \in \sigma_i$  $\forall i \in \{1,2\}, a_i \notin \sigma_i$  $\forall i \in \{1,2\}$ 

### **Questions:**

- 1. Why can't we just set  $\sigma_i = A_i$  for every agent and solve once?
- 2. Why can't we just try every possible support?
- Why wouldn't this 3. work for *n*-player games?



### Computing Nash Equilibrium: General-Sum *n*-Player Games

- In theory, computing an equilibrium in *n*-player games and two-player games have equal computational complexity
- In practice, **two-player** games tend to be faster to solve:
  - Lemke-Howson pivoting algorithm based on a linear complementarity program
- For *n*-player games, **homotopy-following** methods:
  - Construct a family of parameterized **perturbations** of the game, with t = 0 being a trivial game with a known equilibrium, and t = 1 being the original game
  - Move t along [0,1], adjusting the equilibrium as you go, until you reach t = 1

### Computing Correlated Equilibrium

- *n*-player games
- over action profiles
  - are their own actions in that profile
- sufficient to find a correlated equilibrium (**why?**)

Correlated equilibria can be found efficiently even in general-sum,

• Every correlated equilibrium induces a probability distribution

 Corresponds to a correlated equilibrium where Nature randomly chooses an action profile, and the agent's signals

• So finding a **distribution over action profiles** in which each agent would always prefer to play their recommended action is

### Computing Correlated Equilibrium in Polynomial Time

 $a \in A | a_i \in a$   $a \in A | a_i \in a$  $p(a) \geq 0$  $\sum p(a) = 1$  $a \in A$ 

We could find the social-welfare-optimizing correlated equilibrium by adding an **objective function**:

 $\sum p(a)u_i(a) \ge \sum p(a)u_i(a'_i, a_{-i}) \qquad \forall i \in N, \ a_i, a'_i \in A_i$ 

 $\forall a \in A$ 

maximize  $\sum p(a) \sum u_i(a)$  $i \in N$  $a \in A$ 

## Summary

- $\epsilon$ -Nash equilibria: stable when agents have no deviation that gains them more than  $\epsilon$
- Correlated equilibria: stable when agents have signals from a possiblycorrelated randomizing device
- Linear programs are a flexible encoding that can always be solved in polytime
- Finding a Nash equilibrium is computationally hard in general
- **Special cases** are efficiently computable:
  - Nash equilibria in zero-sum games
  - Maxmin strategies (and values) in two-player games
  - Correlated equilibrium