Computing the Solution Concepts

Game Theory

MohammadAmin Fazli

TOC

- Computing the Nash equilibria of simple games
- An introduction to LP
- Computing the Nash equilibria of two-player, zero-sum games
- PPAD Complexity Class
- Computing the Nash equilibria of two-player, general-sum games
- Computing the Nash equilibria of n-player, general-sum games
- Reading:
  - Chapter 4 of the MAS book
  - Thomas Ferguson lecture on LP
  - Christos Papadimitriou lecture on the complexity of finding a Nash equilibrium

Computing Nash Equilibria in Simple Games

- We will learn that it’s hard in general
- Finding Pure Nash equilibria is easy especially in simple games
- Finding Mixed Nash equilibria is hard but it’s easy when you can guess the support
- Example: For BoS, let’s look for an equilibrium where all actions are part of the support (see the blackboard)

$$\begin{aligned} u_1(B) &= u_1(F) \\ 2p + 0(1-p) &= 0p + 1(1-p) \\ p &= \frac{1}{3} \end{aligned}$$

$$\begin{aligned} u_2(B) &= u_2(F) \\ q + 0(1-q) &= 0q + 2(1-q) \\ q &= \frac{2}{3} \end{aligned}$$

	B	F
B	$2, 1$	$0, 0$
F	$0, 0$	$1, 2$

Computing Nash Equilibria in Simple Games

- Example: Ignacio Palacios-Heurta (2003) “Professionals Play Minimax”, Review of Economic Studies, Volume 70, pp 395-415
  - See the blackboard

$Kicker/Goalie$	$Left$	$Right$
$Left$	$.58, .42$	$.95, .05$
$Right$	$.93, .07$	$.70, .30$

	$Goalie$ $Left$	$Goalie$ $Right$	$Kicker$ $Left$	$Kicker$ $Right$
$Nash\ Freq.$	$.42$	$.58$	$.38$	$.62$
$Actual\ Freq.$	$.42$	$.58$	$.40$	$.60$

Removal of Dominated Strategies

- Iterated Removal of Strictly Dominated Strategies (From Chapter 2)

	L	C	R
U	$3, 0$	$2, 1$	$0, 0$
M	$1, 1$	$1, 1$	$5, 0$
D	$0, 1$	$4, 2$	$0, 1$

	L	C
U	$3, 0$	$2, 1$
M	$1, 1$	$1, 1$
D	$0, 1$	$4, 2$

	L	C
U	$3, 0$	$2, 1$
D	$0, 1$	$4, 2$

	C
U	$2, 1$
D	$4, 2$

Removal of Dominated Strategies

- Iterated Removal of Strictly Dominated Strategies (From Chapter 2)

	L	C	R
U	3, 1	0, 1	0, 0
M	1, 1	1, 1	5, 0
D	0, 1	4, 1	0, 0

	L	C
U	3, 1	0, 1
M	1, 1	1, 1
D	0, 1	4, 1

	L	C
U	3, 1	0, 1
D	0, 1	4, 1

M is dominated by the mixed strategy that selects *U* and *D* with equal probability.

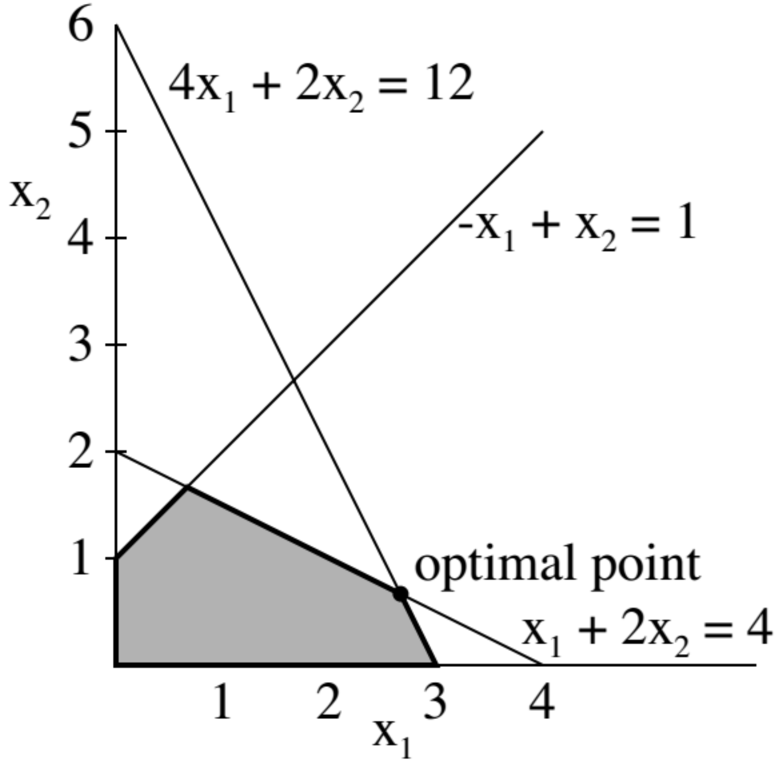
Removal of Dominated Strategies

- This process preserves Nash equilibria.
  - It can be used as a preprocessing step before computing an equilibrium
  - Some games are solvable using this technique - those games are dominance solvable.
  - The order of removal is not important
- Removing Weakly dominated strategies:
  - At least one equilibrium preserved.
  - Order of removal can matter.

Linear Programming

- Find numbers x_1, x_2 that maximize the sum $x_1 + x_2$ subject to the constraints $x_1 \geq 0$ and $x_2 \geq 0$ and

$$\begin{aligned} x_1 + 2x_2 &\leq 4 \\ 4x_1 + 2x_2 &\leq 12 \\ -x_1 + x_2 &\leq 1 \end{aligned}$$



The Standard Maximum LP Problem

- Find and n-vector, $x = (x_1, x_2, \ldots, x_n)^T$ to maximize

$$\boldsymbol{c}^T\boldsymbol{x} = c_1x_1 + \cdots + c_nx_n$$

Subject to the constraints

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\leq b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\leq b_m \end{aligned} \qquad (\text{or } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b})$$

$$x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0 \qquad (\text{or } \boldsymbol{x} \geq \boldsymbol{0})$$

The Standard Minimum LP Problem

- Find an m-vector, $y = (y_1, \dots, y_m)$, to minimize

$$\boldsymbol{y}^T\boldsymbol{b} = y_1 b_1 + \cdots + y_m b_m$$

Subject to the constraints

$$\begin{aligned} y_1 a_{11} + y_2 a_{21} + \cdots + y_m a_{m1} &\geq c_1 \\ y_1 a_{12} + y_2 a_{22} + \cdots + y_m a_{m2} &\geq c_2 \\ &\vdots \\ y_1 a_{1n} + y_2 a_{2n} + \cdots + y_m a_{mn} &\geq c_n \end{aligned} \qquad (\text{or } \boldsymbol{y}^T\boldsymbol{A} \geq \boldsymbol{c}^T)$$

$$y_1 \geq 0, y_2 \geq 0, \dots, y_m \geq 0 \qquad (\text{or } \boldsymbol{y} \geq \boldsymbol{0})$$

Duality

- The dual of the standard maximum problem

$$\text{maximize } \boldsymbol{c}^T\boldsymbol{x}$$
$$\text{subject to the constraints } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \text{ and } \boldsymbol{x} \geq 0$$

is defined to be the standard minimum problem

$$\text{minimize } \boldsymbol{y}^T\boldsymbol{b}$$
$$\text{subject to the constraints } \boldsymbol{y}^T\boldsymbol{A} \geq \boldsymbol{c}^T \text{ and } \boldsymbol{y} \geq 0$$

$$\begin{aligned} &\text{maximize } x_1 + x_2 \\ &\begin{array}{rcrcr} x_1 &+& 2x_2 &\leq& 4 \\ 4x_1 &+& 2x_2 &\leq& 12 \\ -x_1 &+& x_2 &\leq& 1. \end{array} \end{aligned}$$

$$\begin{aligned} &\text{minimize } 4y_1 + 12y_2 + y_3 \\ &\begin{array}{rcrcrcr} y_1 &+& 4y_2 &-& y_3 &\geq& 1 \\ 2y_1 &+& 2y_2 &+& y_3 &\geq& 1 \end{array} \end{aligned}$$

LP Optimality Facts

- **Polynomial Time Algorithm:** LPs are solvable in polynomial time
- **Weak Duality Theorem:** If x is feasible for the standard maximum problem and if y is feasible for its dual then $c^T x \leq y^T b$
- **Strong Duality Theorem:** If a standard linear programming problem is bounded feasible, then so is its dual, their values are equal, and there exists optimal vectors for both problems.
- **The Equilibrium Theorem:** Let x^* and y^* be feasible vectors for a standard maximum problem and its dual respectively. Then x^* and y^* are optimal if, and only if,

$$y_i^* = 0 \text{ for all i for which } \sum_{j=1}^{n} a_{ij} x_j^* < b_i$$

and

$$x_j^* = 0 \text{ for all j for which } \sum_{i=1}^{m} y_i^* a_{ij} > c_j$$

Computing Nash Equilibria in Two-players Zero-sum Games

- The minmax theorem tells us that U_1^* holds constant in all equilibria and that it is the same as the value that player 1 achieves under a minmax strategy by player 2.

$$
\begin{aligned}
\text{minimize} \quad & U_1^* \\
\text{subject to} \quad & \sum_{k \in A_2} u_1(a_1^j, a_2^k) \cdot s_2^k \leq U_1^* && \forall j \in A_1 \\
& \sum_{k \in A_2} s_2^k = 1 \\
& s_2^k \geq 0 && \forall k \in A_2
\end{aligned}
$$

Computing Nash Equilibria in Two-players Zero-sum Games

- We can construct a linear program to give us player 1's mixed strategies. This program reverses the roles of player 1 and player 2 in the constraints; the objective is to *maximize* U_1^*, as player 1 wants to maximize his own payoffs. This corresponds to the *dual* of player 2's program.

$$
\begin{aligned}
\text{maximize} \quad & U_1^* \\
\text{subject to} \quad & \sum_{j \in A_1} u_1(a_1^j, a_2^k) \cdot s_1^j \geq U_1^* && \forall k \in A_2 \\
& \sum_{j \in A_1} s_1^j = 1 \\
& s_1^j \geq 0 && \forall j \in A_1
\end{aligned}
$$

Computing Nash Equilibria in Two-players Zero-sum Games

- LP with slack variables (needed for next slides)

$$
\begin{aligned}
\text{minimize} \quad & U_1^* & \\
\text{subject to} \quad & \sum_{k \in A_2} u_1(a_1^j, a_2^k) \cdot s_2^k + r_1^j = U_1^* & \forall j \in A_1 \\
& \sum_{k \in A_2} s_2^k = 1 & \\
& s_2^k \geq 0 & \forall k \in A_2 \\
& r_1^j \geq 0 & \forall j \in A_1
\end{aligned}
$$

An Introduction to the Related Complexity Concepts

- Complexity class **NP:** The class of all search problems. A search problem A is a binary predicate A(x, y) that is efficiently (in polynomial time) computable and balanced (the length of x and y do not differ exponentially). Intuitively, x is an instance of the problem and y is a solution. The search problem for A is this:

"Given x, find y such that A(x, y), or if no such y exists, say "no"."

- SAT = SAT(ϕ, x): given a Boolean formula ϕ in conjunctive normal form (CNF), find a truth assignment x which satisfies ϕ, or say "no" if none exists.
- Nash = Nash(G,(x, y)): given a game G, find mixed strategies (x, y) such that (x, y) is a Nash equilibrium of G, or say "no" if none exists. Nash is in **NP**, since for a given set of mixed strategies, one can always efficiently check if the conditions of a Nash equilibrium hold or not.

An Introduction to the Related Complexity Concepts

- Reduction: We say problem A reduces to problem B if there exist two functions f and g mapping strings to strings such that
  - f and g are efficiently computable functions, i.e. in polynomial time in the length of the input string;
  - if x is an instance of A, then f (x) is an instance of B such that:
    - x is a "no" instance for problem A if and only if f (x) is a "no" instance for problem B
    - $B(f(x), y) \Rightarrow A(x, g(y))$
- X-completeness: A problem in class X is X-complete if all problems in X reduce to it.
  - NP-Complete problems: The hardest problems in class NP.

Nash-Equilibria & NP-Completeness

- So, is it NP-complete to find a Nash equilibrium?
  - NO, since a solution is guaranteed to exist…
- However, it is NP-complete to find a “tiny” bit more info than a Nash equilibrium; e.g., the following are NP-complete:
  - **(Uniqueness)** Given a game *G*, does there exist a unique equilibrium in *G*?
  - **(Pareto optimality)** Given a game *G*, does there exist a strictly Pareto efficient equilibrium in *G*?
  - **(Guaranteed payoff)** Given a game *G* and a value *v*, does there exist an equilibrium in *G* in which some player *i* obtains an expected payoff of at least *v*?
  - **(Guaranteed social welfare)** Given a game *G*, does there exist an equilibrium in which the sum of agents’ utilities is at least *k*?
  - **(Action inclusion or Exclusion)** Given a game *G* and an action $a_i \in A_i$ for some player *i*, does there exist an equilibrium of *G* in which player *i* plays action a_i with strictly positive (or Zero) probability?

2Nash Problem

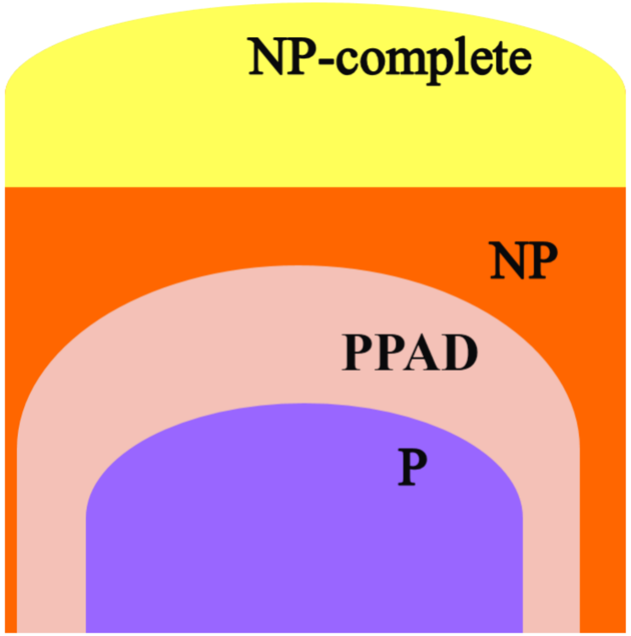
- The 2Nash Problem: given a game and a Nash equilibrium, find another one, or output "no" if none exist.
- Theorem: the 2Nash problem is NP-Complete.
  - Proof: See the blackboard.

TFNP Class

- Due to the fact that Nash always has a solution, we are interested more generally in the class of search problems for which every instance has a solution. We call this class **TFNP** (which stands for *total function non-deterministic polynomial*).
- $NASH \in TFNP \subseteq NP$
- Is Nash TFNP-complete?
  - Probably not, because TFNP probably has no complete problems
  - Intuitively because the class needs to be defined on a more solid basis than an uncheckable universal statement such as “every instance has a solution.”
- The idea: subdivide TFNP according to the method of proof.

PPAD Complexity Class

- “If a directed graph has an unbalanced node (a vertex with different in-degree and out-degree), then it has another one.” This is the *parity argument for directed graphs*, which gives rise to the class **PPAD**.
  - $PPAD \subseteq TFNP$
- Another classes such as PLS, PPP, PPA are defined similarly.
- PPAD is the class of all search problems which always have a solution and whose proof is based on the parity argument for directed graphs.



PPAD Complexity Class

- We are given a graph *G* where the in-degree and the out-degree of each node is at most 1.
  - there are four kinds of nodes: sources, sinks, midnodes, and isolated vertices.
- Our graph *G* is exponential in size, since otherwise we would be able to explore the structure of the graph (in particular, we can identify sources and sinks) efficiently; to be specific, suppose *G* has 2^n vertices, one for every bit string of length *n*.
- The edges of *G* will be represented by two Boolean circuits, of size polynomial in *n*, each with *n* input bits and *n* output bits. The circuits are denoted *P* and *S* (for potential predecessor and potential successor).

PPAD Complexity Class

- There is a directed edge from vertex *u* to vertex *v* if and only if *v* = *S*(*u*) and *u* = *P*(*v*), i.e. given input *u*, *S* outputs *v* and, vice-versa, given input *v*, *P* outputs *u*.
- Also, we assume that the specific vector 00 · · · 0 has no predecessor (the circuit *P* is so wired that $P(0^n) = 0^n$)
- The search problem END OF THE LINE is the following:

  "Given (*S, P*), find a sink or another source."
- END OF THE LINE ∈ $TFNP$
- The class PPAD: The class PPAD contains all search problems in TFNP that reduce to END OF THE LINE.

NASH & the PPAD Class

- Theorem: NASH is PPAD-Complete
  - For games with ≥ 4 players (Daskalakis, Goldberg, Papadimitriou 2005)
  - For games with 3 players (Chen, Deng 2005 & Daskalakis, Papadimitriou 2005)
  - For games with 2 players (Chen, Deng 2006)
- General Proof:
  - $NASH \in PPAD$
  - Reducing END OF THE LINE to NASH
    - NASH→BROUWER
    - BROUWER→ END OF THE LINE
    - See the blackboard and next slides for proof ideas

NASH→BROUWER

- Proof idea: Defining graphical games for each mathematical operation. See the black board for $\times\,\alpha$ operator ($s_{v_2} = \min(\alpha s_{v_1}, 1)$)

Payoffs to v_2 :

	w *plays* 0	w *plays* 1
v_2 *plays* 0	0	1
v_2 *plays* 1	1	0

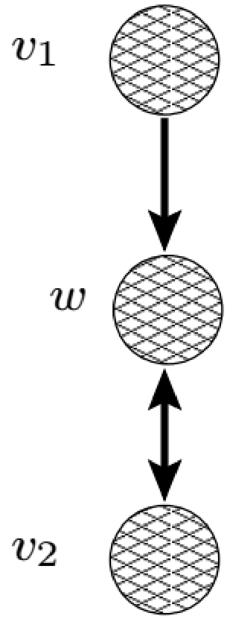
Payoffs to w:

w *plays* 0

	v_2 *plays* 0	v_2 *plays* 1
v_1 *plays* 0	0	0
v_1 *plays* 1	α	α

w *plays* 1

	v_2 *plays* 0	v_2 *plays* 1
v_1 *plays* 0	0	1
v_1 *plays* 1	0	1



$\mathcal{G}_{\times\alpha}$, $\mathcal{G}_{=}$

BROUWER→ END OF THE LINE

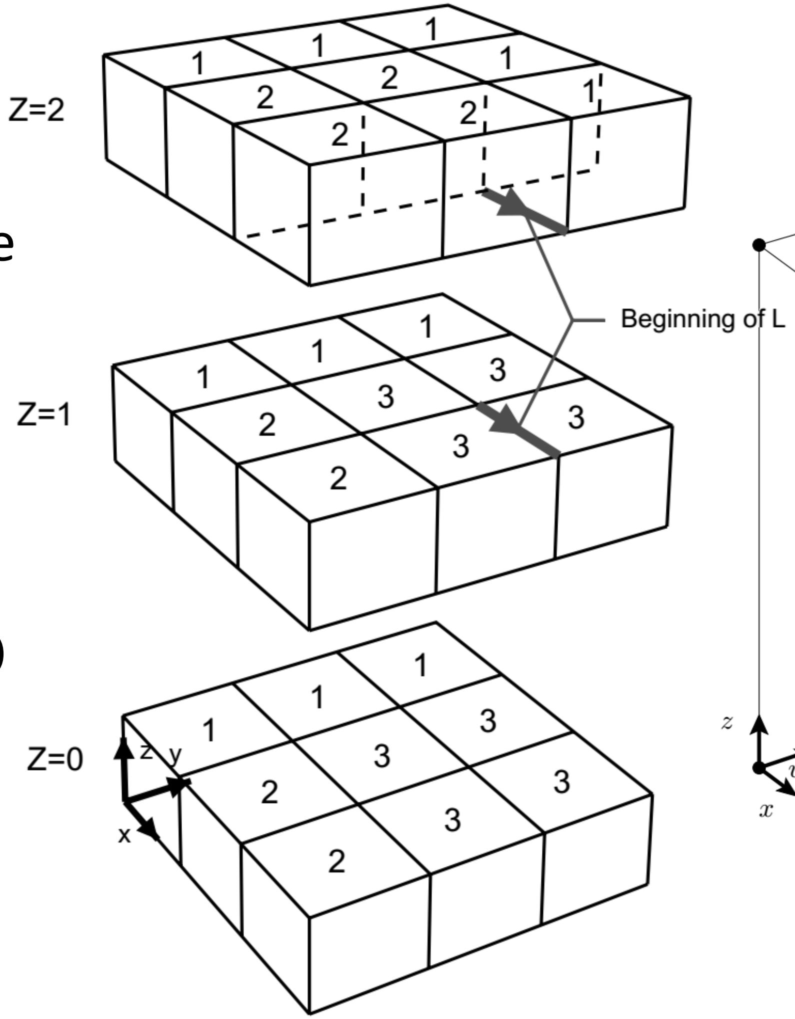
- A cube of 2^{3n} cubletes is defined:

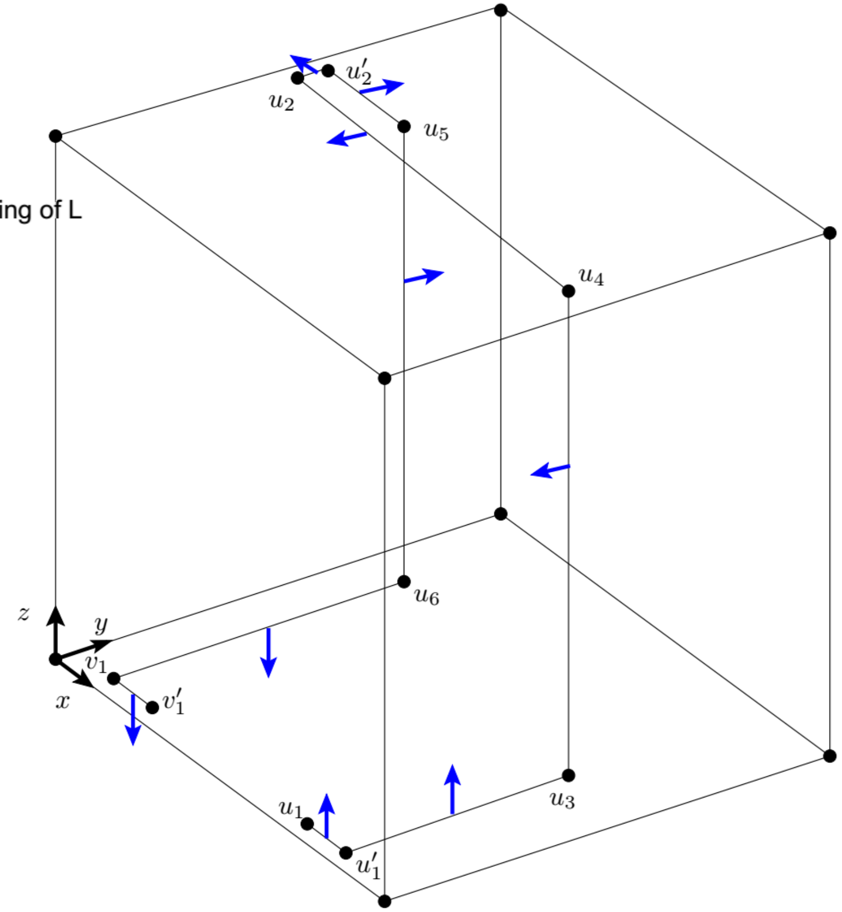
$$K_{ijk} = \{(x,y,z):\ i \cdot 2^{-n} \leq x \leq (i+1) \cdot 2^{-n},\\ j \cdot 2^{-n} \leq y \leq (j+1) \cdot 2^{-n},\\ k \cdot 2^{-n} \leq z \leq (k+1) \cdot 2^{-n}\}$$

- Define c_{ijk} to be the center of the K_{ijk}. Define $\phi(c_{ijk}) = c_{ijk} + \delta_{ijk}$ where δ_{ijk}defines its color which is from one the 3 defined vectors: $(\alpha, 0,0), (0, \alpha, 0), (0,0, \alpha), (-\alpha, -\alpha, -\alpha)$ where α is a little number

BROUWER→ END OF THE LINE

- Proof steps:
  - Embed the input graph in the cube with straight line edges
  - Color the cubelets such that
    - ϕ is defined from the cube to the cube
    - The color of every cubelets is $(-\alpha, -\alpha, -\alpha)$ except the vertices on the edges
    - Panchromatic vertices maps to the source and the sink vertices of the input graph





LCP Formulation (2-Player, General-Sum)

$$\sum_{k \in A_2} u_1(a_1^j, a_2^k) \cdot s_2^k + r_1^j = U_1^* \qquad \forall j \in A_1$$

$$\sum_{j \in A_1} u_2(a_1^j, a_2^k) \cdot s_1^j + r_2^k = U_2^* \qquad \forall k \in A_2$$

$$\sum_{j \in A_1} s_1^j = 1, \quad \sum_{k \in A_2} s_2^k = 1$$

$$s_1^j \geq 0, \quad s_2^k \geq 0 \qquad \forall j \in A_1, \forall k \in A_2$$

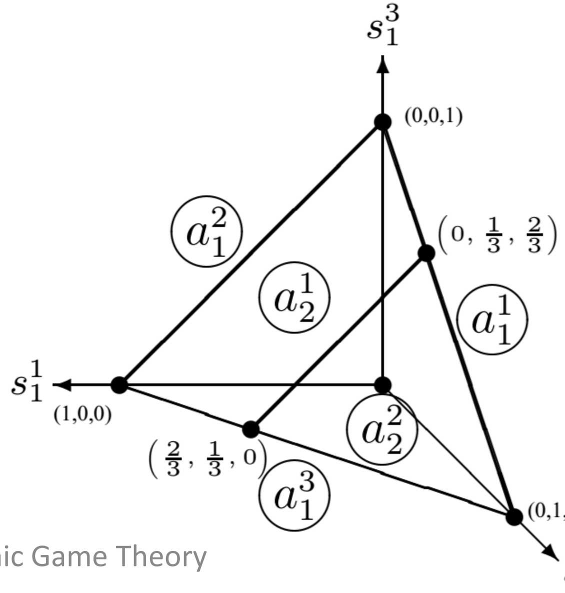
$$r_1^j \geq 0, \quad r_2^k \geq 0 \qquad \forall j \in A_1, \forall k \in A_2$$

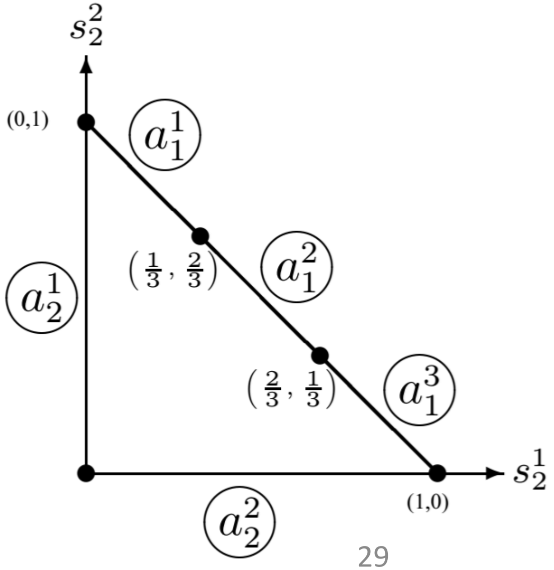
$$r_1^j \cdot s_1^j = 0, \quad r_2^k \cdot s_2^k = 0 \qquad \forall j \in A_1, \forall k \in A_2$$

Lemke-Howson Algorithm

- The best known algorithm for solving the LCP Formulation
- Strategy labels for the player i's mixed strategy s_i ($L(s_i) \subseteq A_1 \cup A_2$):
  - each of player i's actions a_i^j that is *not* in the support of s_i
  - each of player -i's actions a_{-i}^j that *is* a best response by player -i to s_i
- Example:

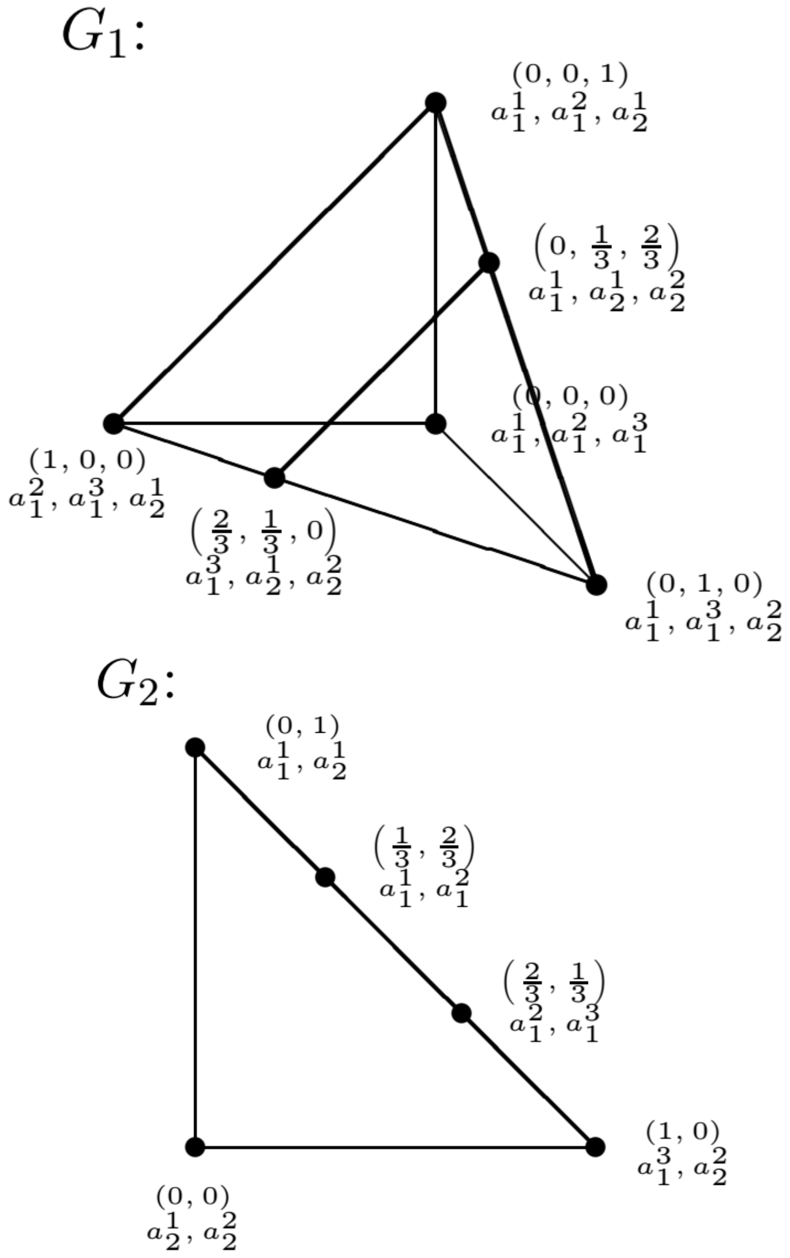
$0, 1$	$6, 0$
$2, 0$	$5, 2$
$3, 4$	$3, 3$





Lemke-Howson Algorithm

- A strategy profile (s_1, s_2) is Nash equilibrium iff $L(s_1) \cup L(s_2) = A_1 \cup A_2$
- The Lemke-Howson algorithm search the cross product of two virtual graphs (G_1 and G_2) to find a Nash equilibrium (completely labeled strategy profile)



Lemke-Howson Algorithm

- In fact, we do not compute the nodes in advance at all.
- At each step, we find the missing label to be added (called the *entering variable*), and add it.
- Find out which label has been lost (it is called the *leaving variable*).
  - Choose the one with the minimum ratio test
  - Ratio test: We deal with equalities in the form of $v = c + qu + T$ where v is a leaving variable, u is the entering variable (q is its coefficient), c is a constant and T is the remaining part of the equality. We define c/q as its ratio test.
- The process repeats until no variable is lost in which case a solution has been obtained.

Lemke-Howson Algorithm

initialize *the two systems of equations at the origin*
arbitrarily pick *one dependent variable from one of the two systems. This variable* enters *the basis.*
repeat
  identify *one of the previous basis variables which must* leave, *according to the minimum ratio test. The result is a new basis.*
  if *this basis is completely labeled* **then**
    return *the basis* // we have found an equilibrium.
  else
    the variable dual to the variable that last left enters *the basis.*

- See the blackboard for an example.
- Theorem: Lemke-Howson algorithm reaches always reaches a Nash-equilibrium.

Computing the Nash Equilibria of n-player, General-sum Games

- There is no known general algorithm for this problem
- Some ideas sometimes work:
  - Using Newton's method:
    - A sequence of LCPs each is an approximation for the main problem and creates the next LCP.
  - Using Constrained Optimization methods:
    - Example: $c_i^j(s) = u_i\left(a_i^j, s_{-i}\right) - u_i(s)$ and $d_i^j(s) = \max\left(c_i^j(s), 0\right)$

$$
\begin{aligned}
\text{minimize} \quad & f(s) = \sum_{i \in N} \sum_{j \in A_i} \left(d_i^j(s)\right)^2 & \\
\text{subject to} \quad & \sum_{j \in A_i} s_i^j = 1 & \forall i \in N \\
& s_i^j \geq 0 & \forall i \in N, \forall j \in A_i
\end{aligned}
$$