Algorithms for solving two-player normal form games

Tuomas Sandholm
Carnegie Mellon University
Computer Science Department

Recall: Nash equilibrium

- Let A and B be $|M|$ x $|N|$ matrices.
- Mixed strategies: Probability distributions over M and N
- If player 1 plays x, and player 2 plays y, the payoffs are $x^T Ay$ and $x^T By$
- Given y, player 1's best response maximizes $x^T Ay$
- Given x, player 2's best response maximizes $x^T By$
- (x,y) is a Nash equilibrium if x and y are best responses to each other

Finding Nash equilibria

- Zero-sum games
  - Solvable in poly-time using linear programming
- General-sum games
  - PPAD-complete
  - Several algorithms with exponential worst-case running time
    - Lemke-Howson [1964] – linear complementarity problem
    - Porter-Nudelman-Shoham [AAAI-04] = support enumeration
    - Sandholm-Gilpin-Conitzer [2005] - MIP Nash = mixed integer programming approach

Zero-sum games

- Among all best responses, there is always at least one pure strategy
- Thus, player 1's optimization problem is:

$$\begin{aligned} \text{maximize} \quad & \min_{j \in N} \sum_{i \in M} a_{ij} x_i \\ \text{such that} \quad & \sum_{i \in M} x_i = 1 \\ & x_i \geq 0 \text{ for all } i \in M \end{aligned}$$

- This is equivalent to:

$$\begin{aligned} \text{maximize} \quad & z \\ \text{such that} \quad & z - \sum_{i \in M} a_{ij} x_i \leq 0 \text{ for all } j \in N \\ & \sum_{i \in M} x_i = 1 \\ & x_i \geq 0 \text{ for all } i \in M \end{aligned}$$

- By LP duality, player 2's optimal strategy is given by the dual variables

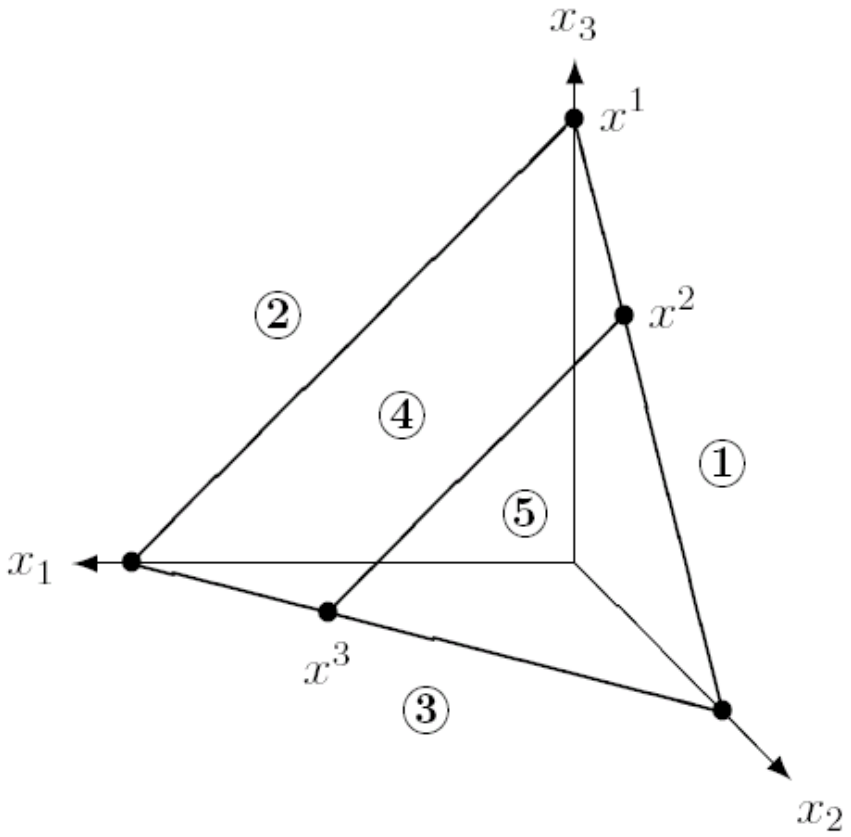
General-sum games: Lemke-Howson algorithm

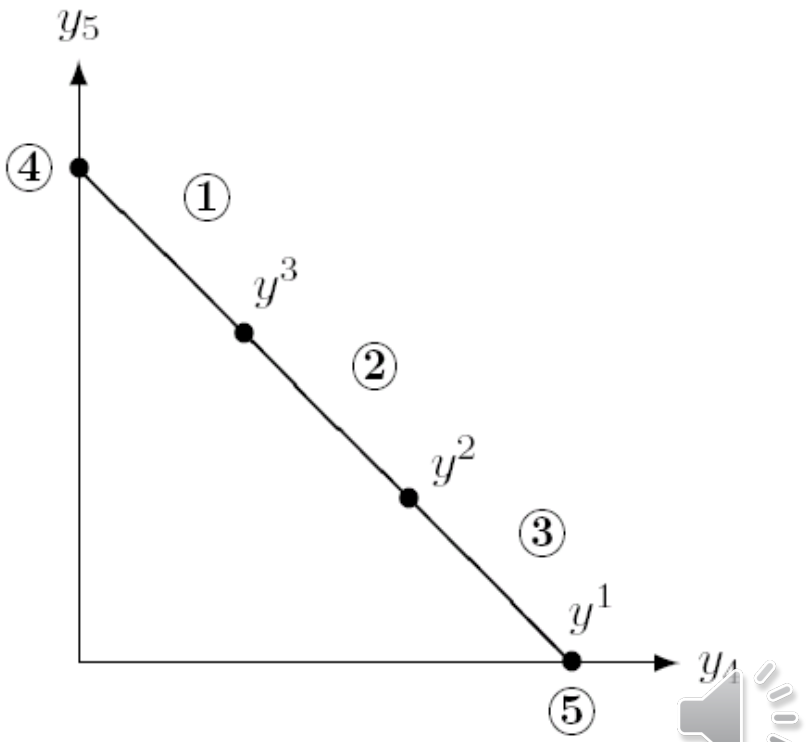
- = pivoting algorithm similar to simplex algorithm
- We say each mixed strategy is "labeled" with the player's *unplayed* pure strategies and the pure best responses of the other player
- A Nash equilibrium is a completely labeled pair (i.e., the union of their labels is the set of pure strategies)

Lemke-Howson Illustration

Example of label definitions

$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$

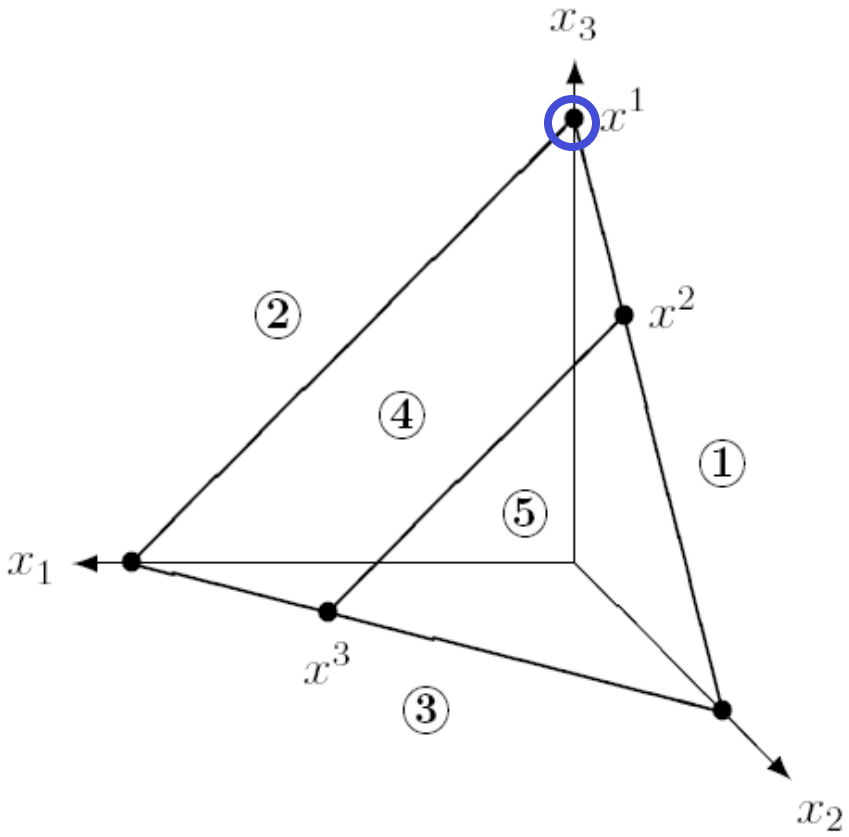


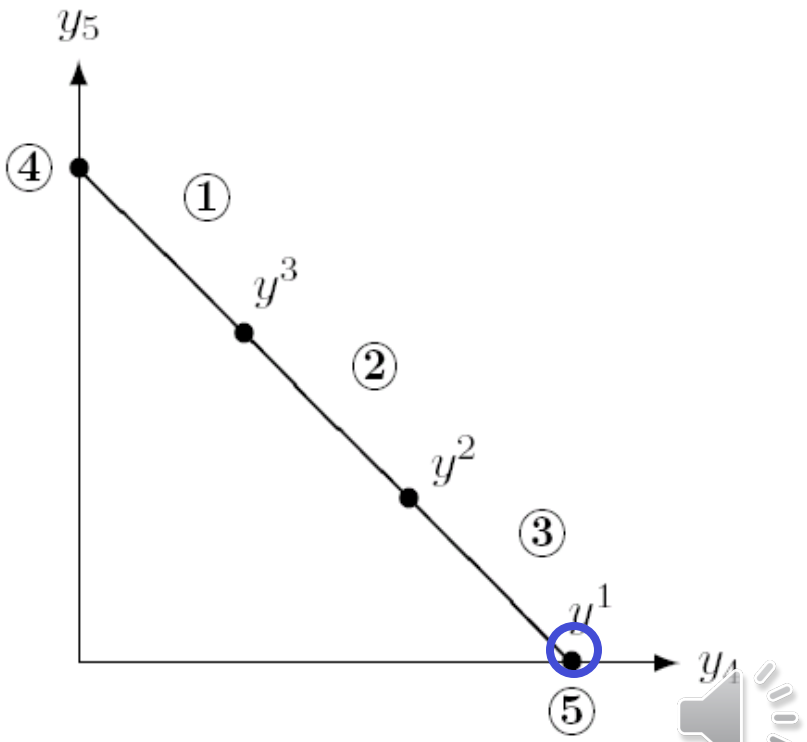


Lemke-Howson Illustration

Equilibrium 1

$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$

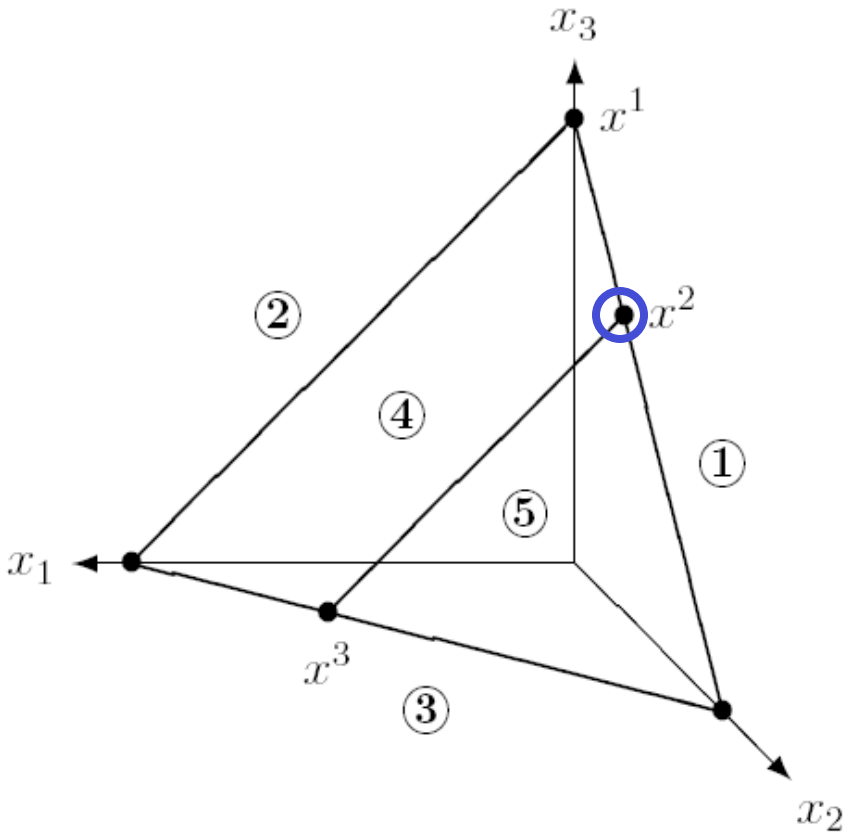


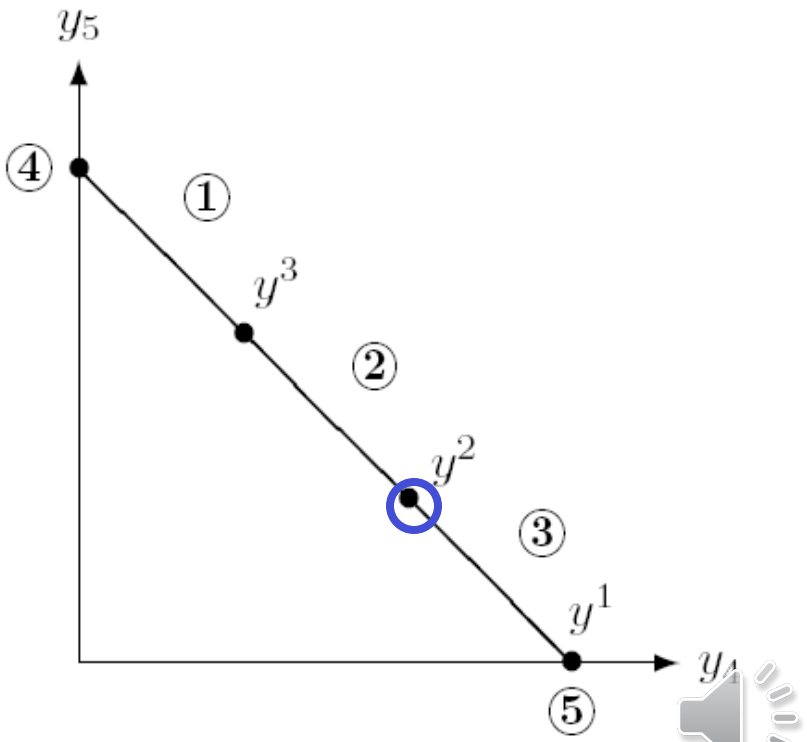


Lemke-Howson Illustration

Equilibrium 2

$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$

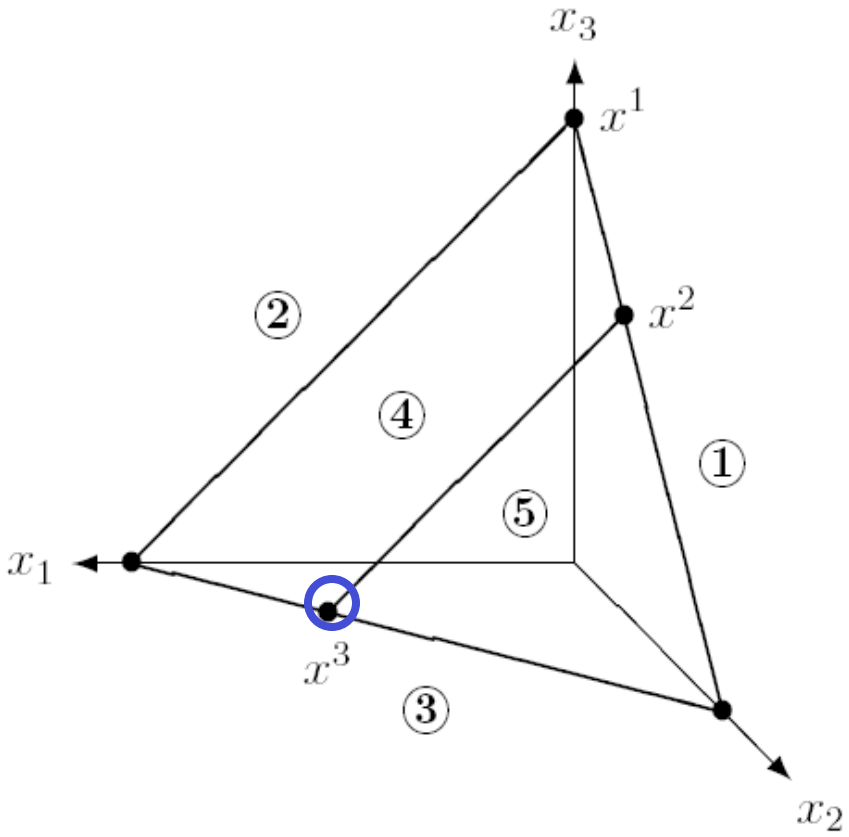


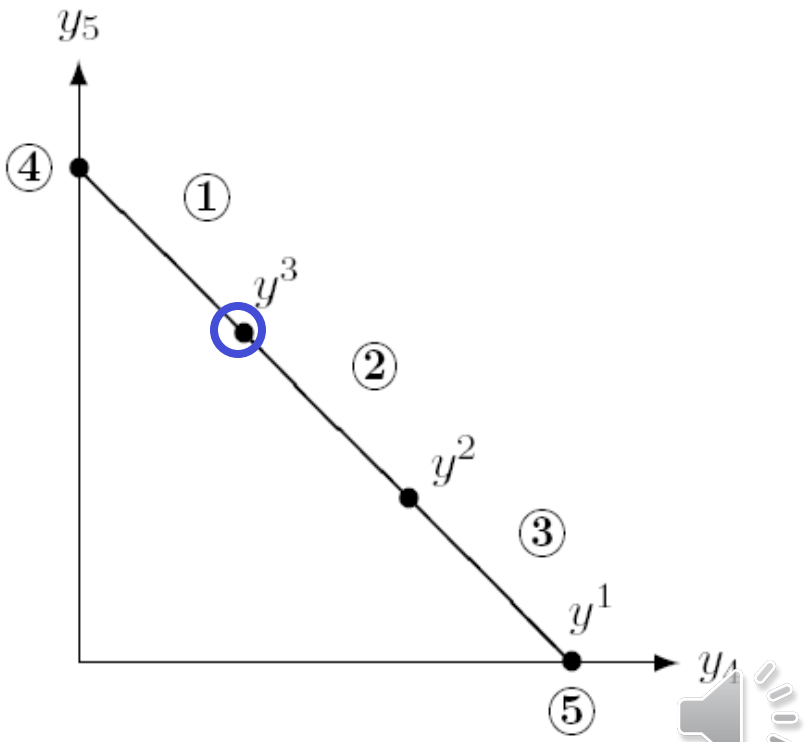


Lemke-Howson Illustration

Equilibrium 3

$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$

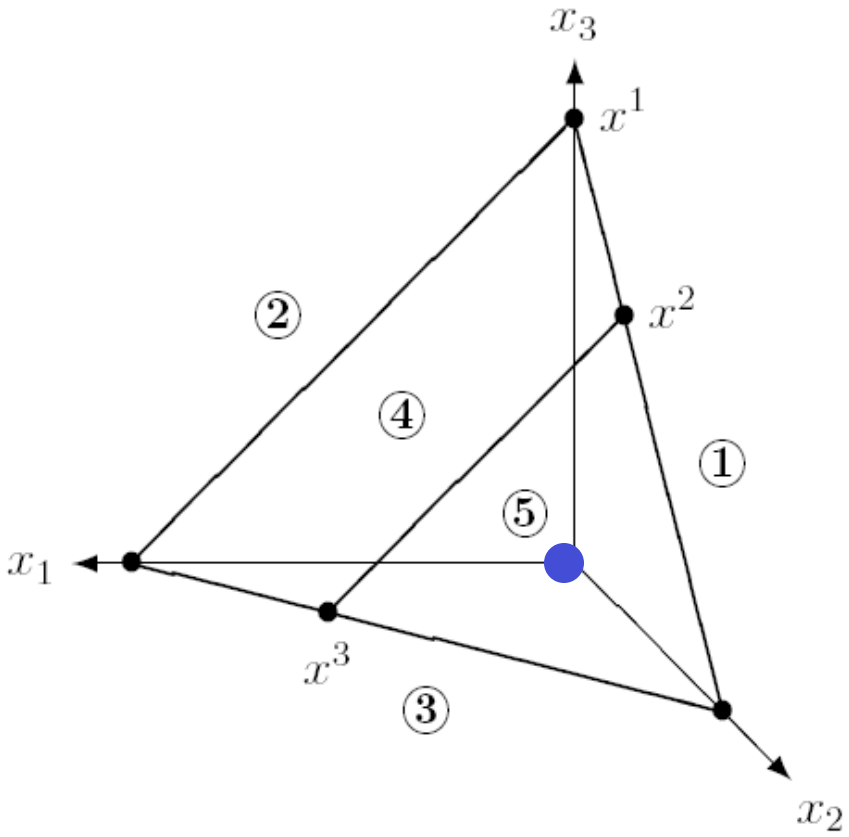


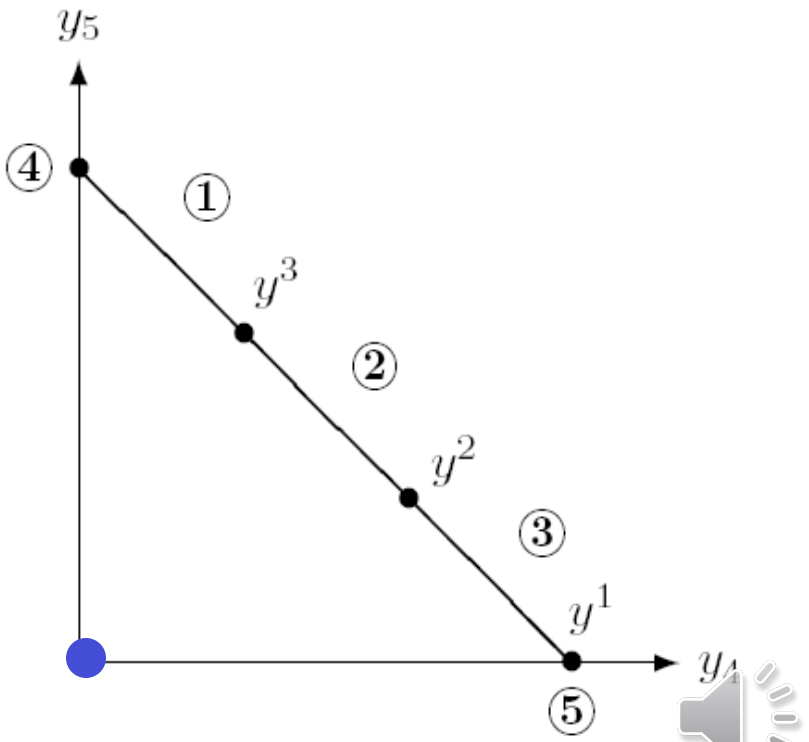


Lemke-Howson Illustration

Run of the algorithm

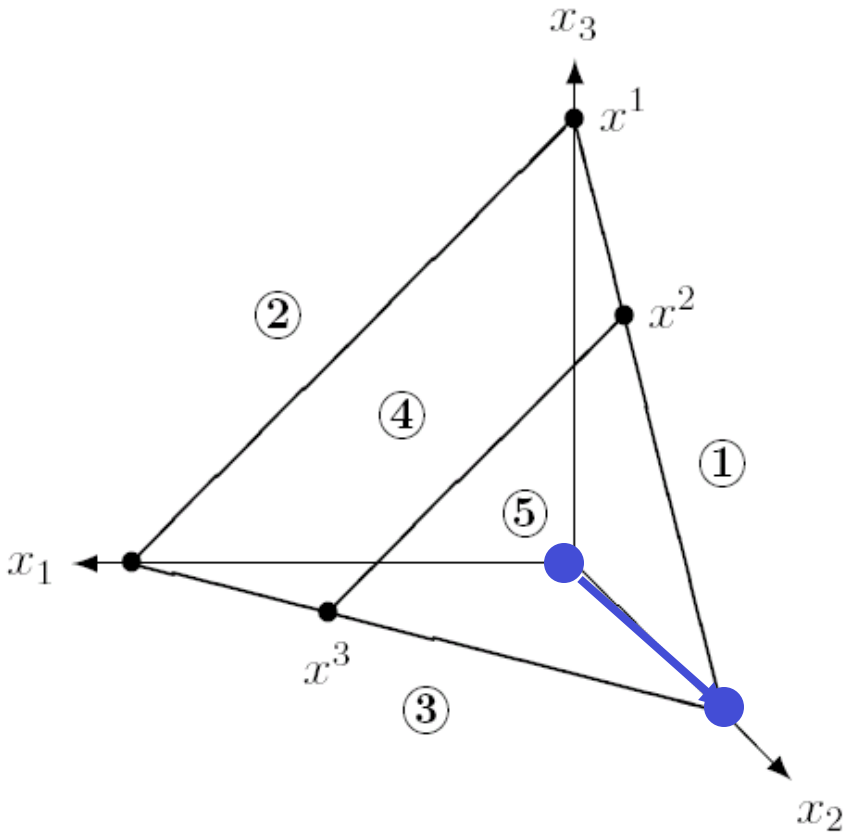
$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$

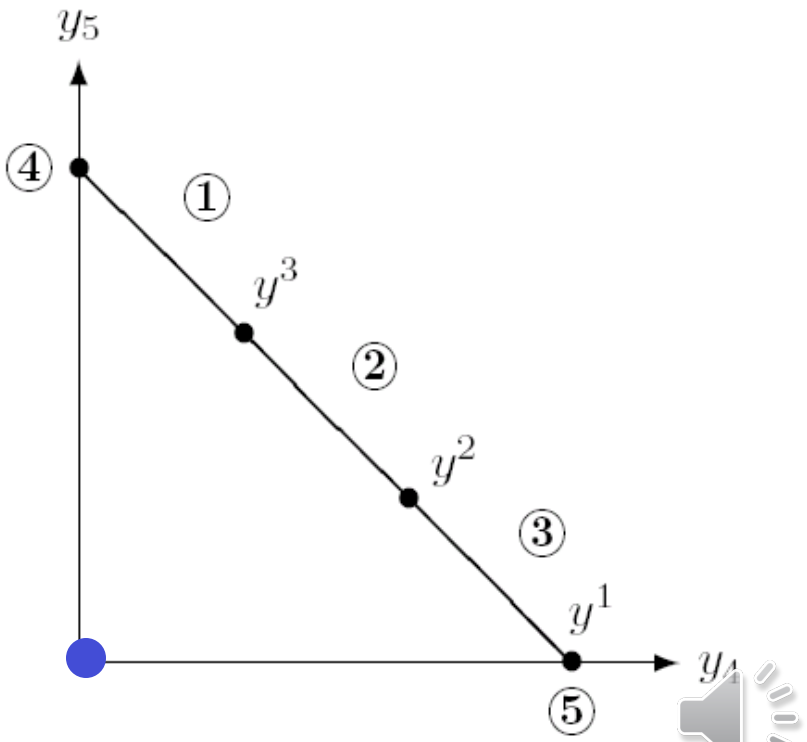




Lemke-Howson Illustration

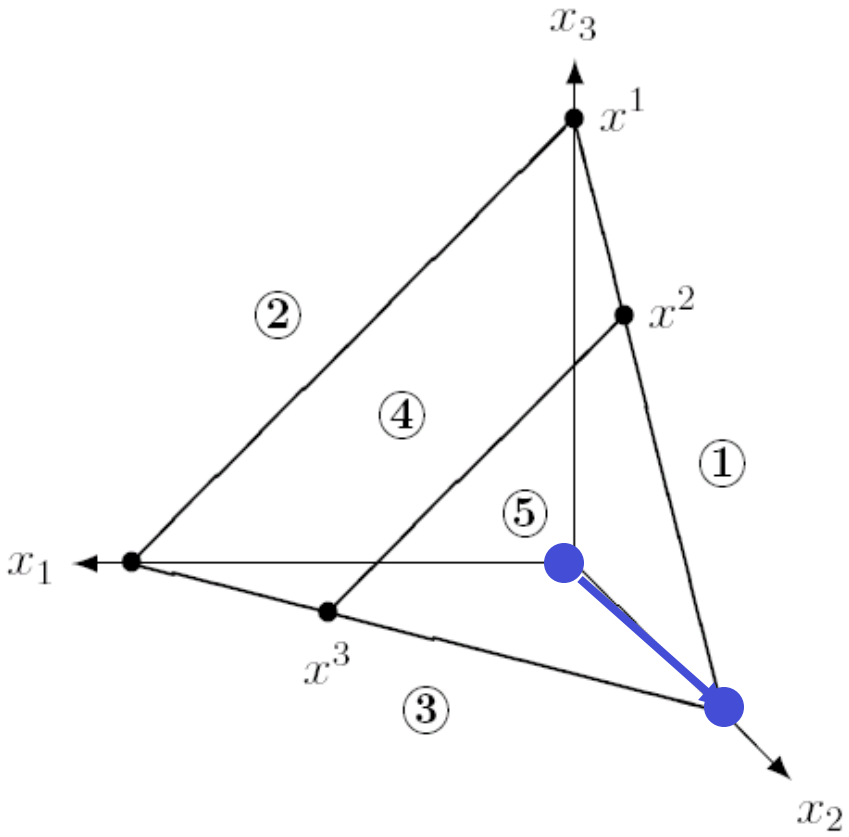
$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$

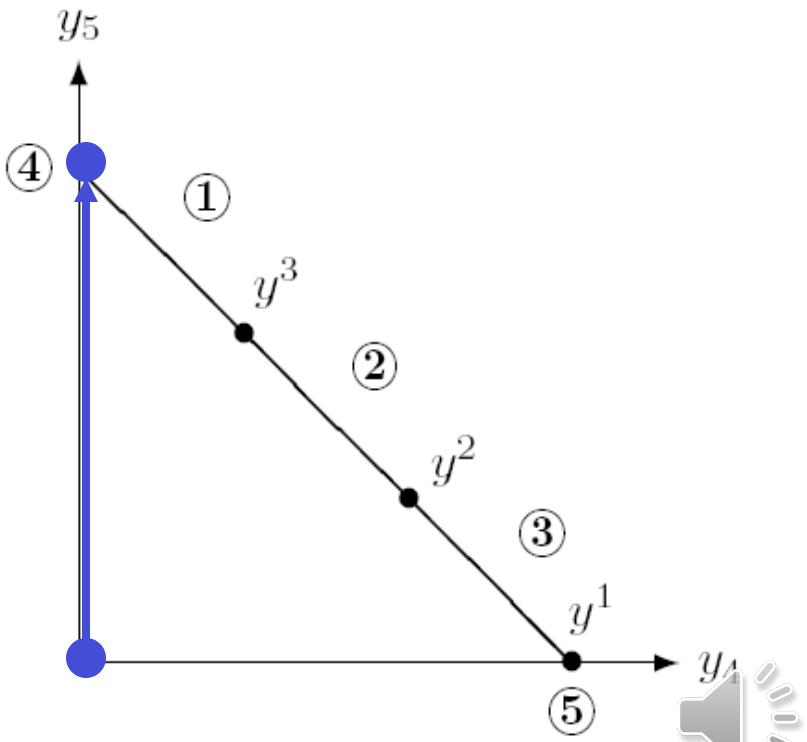




Lemke-Howson Illustration

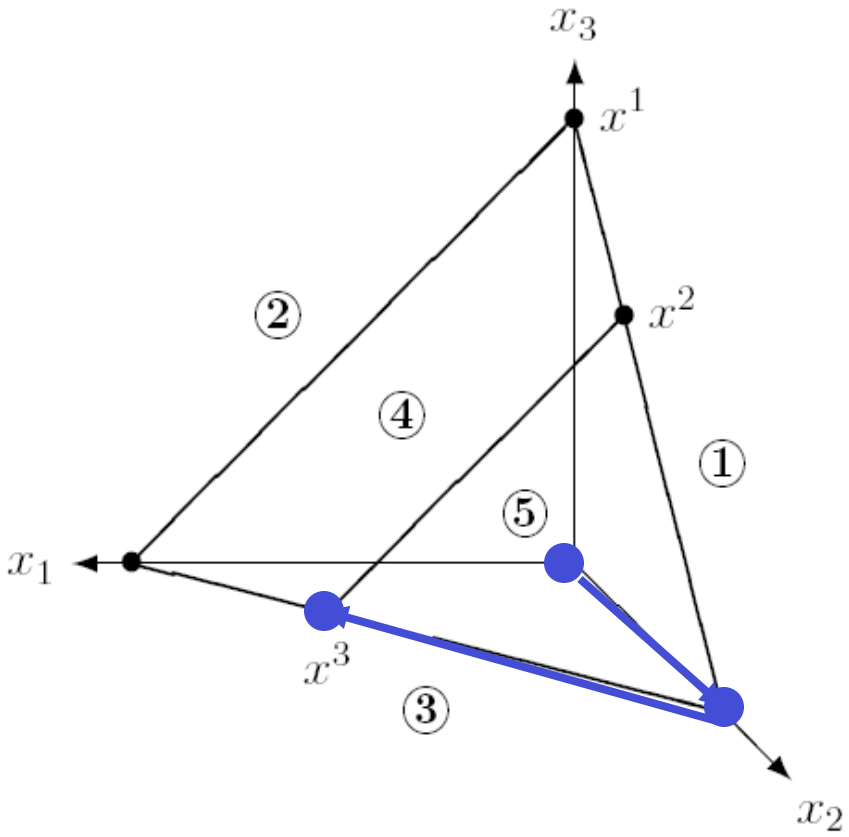
$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$

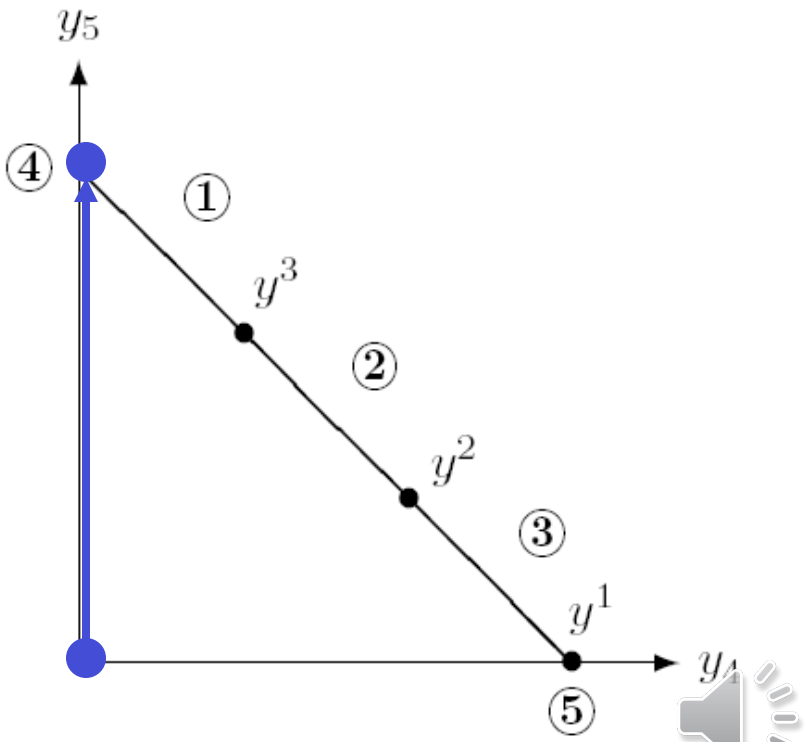




Lemke-Howson Illustration

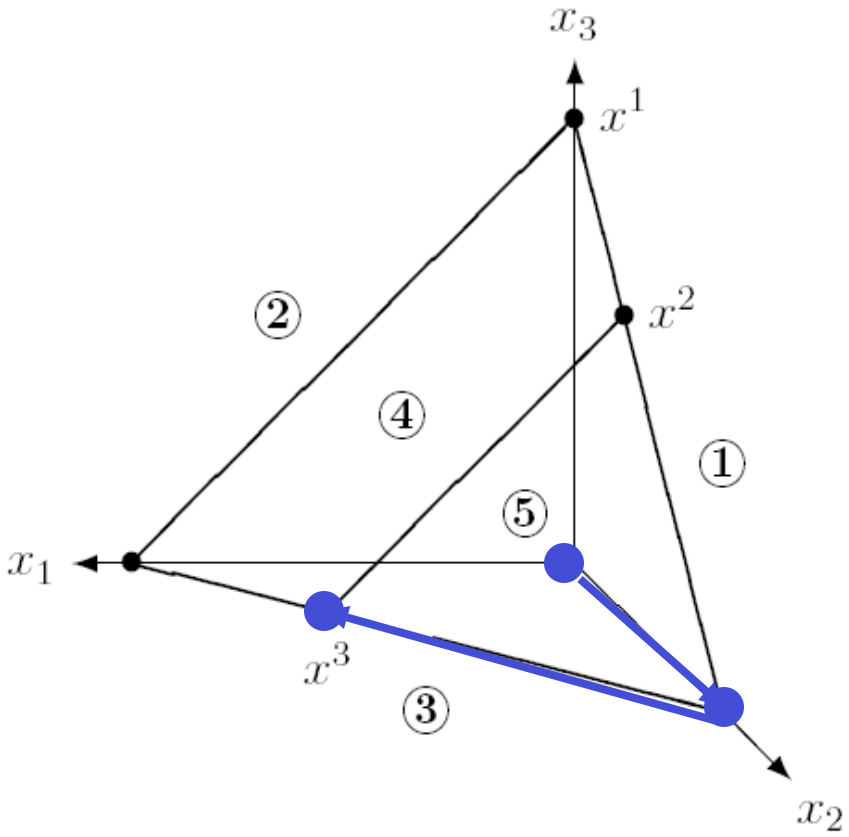
$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$

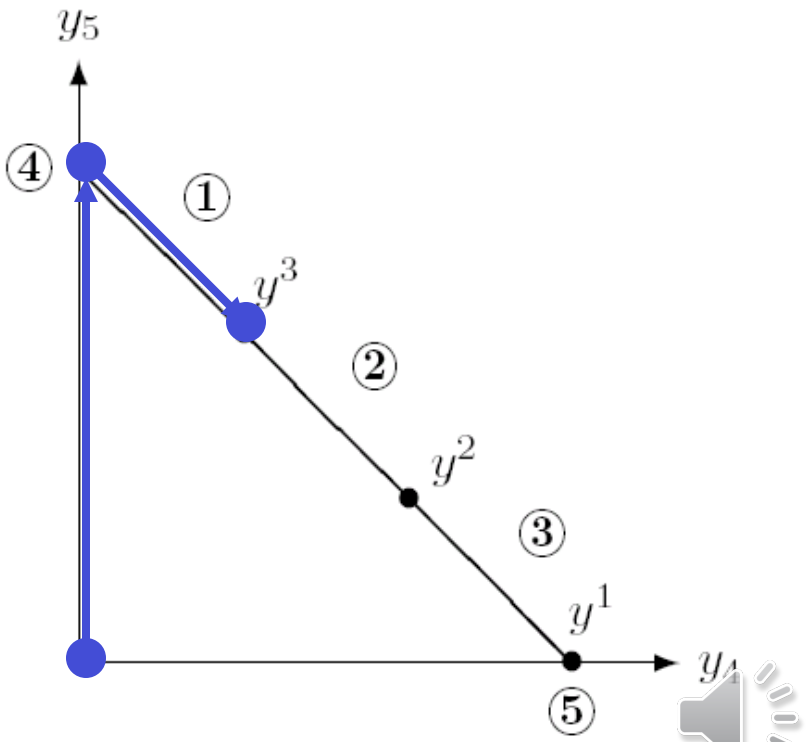




Lemke-Howson Illustration

$$A = \begin{bmatrix} 0 & 6 \\ 2 & 5 \\ 3 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 4 & 3 \end{bmatrix}$$





Lemke-Howson

- There exist instances where the algorithm takes exponentially many steps [Savani & von Stengel FOCS-04]

Simple Search Methods for Finding a Nash Equilibrium

Ryan Porter, Eugene Nudelman & Yoav Shoham

[AAAI-04, extended version in GEB]

A subroutine that we'll need when searching over supports

Feasibility Program (Checks whether there is a NE with given supports)

Input: $S = (S_1, \ldots, S_n)$, a support profile
Output: NE p, if there exists both a strategy profile $p = (p_1, \ldots, p_n)$ and a value profile $v = (v_1, \ldots, v_n)$ s.t.:

$$\forall i \in N, a_i \in S_i : \quad \sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) = v_i$$

$$\forall i \in N, a_i \notin S_i : \quad \sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) \leq v_i$$

$$\forall i \in N : \quad \sum_{a_i \in S_i} p_i(a_i) = 1$$

$$\forall i \in N, a_i \in S_i : \quad p_i(a_i) \geq 0$$

$$\forall i \in N, a_i \notin S_i : \quad p_i(a_i) = 0$$

Solvable by LP

Features of PNS = support enumeration algorithm

- Separately instantiate supports
  - for each pair of supports, test whether there is a NE with those supports (using Feasibility Problem solved as an LP)
  - To save time, don't run the Feasibility Problem on supports that include conditionally dominated actions
    - a_i is *conditionally dominated*, given $R_{-i} \subseteq A_{-i}$

$$\exists a'_i \in A_i \; \forall a_{-i} \in R_{-i} : \quad u_i(a_i, a_{-i}) < u_i(a'_i, a_{-i})$$

- Prefer balanced (= equal-sized for both players) supports
  - Motivated by an old theorem: any nondegenerate game has a NE with balanced supports
- Prefer small supports
  - Motivated by existing theoretical results for particular distributions (e.g., [MB02])

Pseudocode of two-player PNS algorithm

---

for all support size profiles $x = (x_1, x_2)$, sorted in increasing order of, first, $|x_1 - x_2|$ and, second, $(x_1 + x_2)$ **do**
  for all $S_1 \subseteq A_1$ s.t. $|S_1| = x_1$ **do**
    $A_2' \leftarrow \{a_2 \in A_2$ not cond. dominated, given $S_1\}$
    if $\nexists a_1 \in S_1$ cond. dominated, given A_2' **then**
      for all $S_2 \subseteq A_2'$ s.t. $|S_2| = x_2$ **do**
        if $\nexists a_1 \in S_1$ cond. dominated, given S_2 **then**
          if Feasibility Program 1 is satisfiable for $S = (S_1, S_2)$ **then**
            Return the found NE p

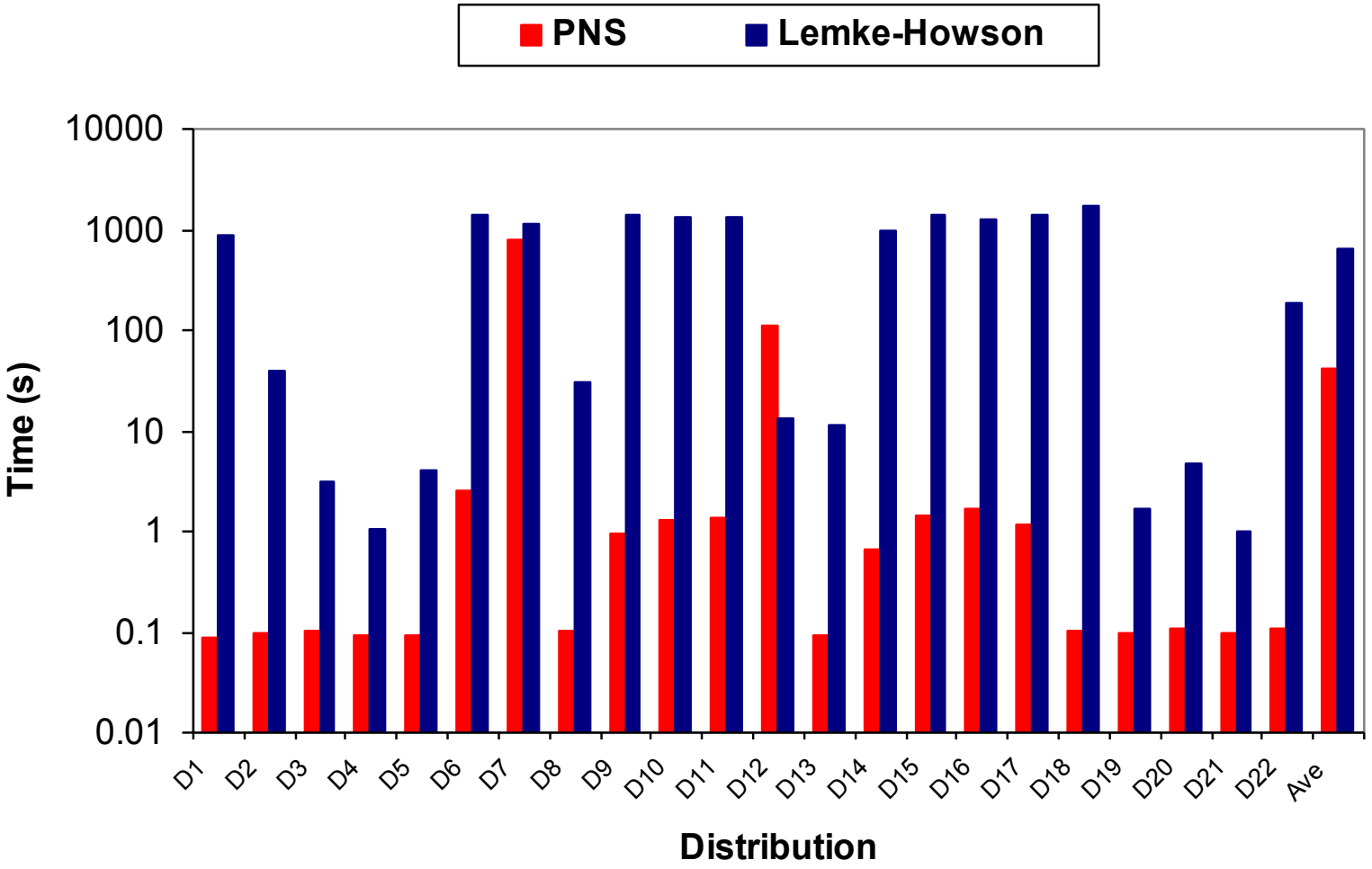
PNS: Experimental Setup

- Most previous empirical tests only on "random" games:
  - Each payoff drawn independently from uniform distribution
- GAMUT distributions [NWSL04]
  - Based on extensive literature search
  - Generates games from a wide variety of distributions
  - Available at `http://gamut.stanford.edu`

D1	Bertrand Oligopoly	D2	Bidirectional LEG, Complete Graph
D3	Bidirectional LEG, Random Graph	D4	Bidirectional LEG, Star Graph
D5	Covariance Game: $\rho = 0.9$	D6	Covariance Game: $\rho = 0$
D7	Covariance Game: Random $\rho 2$ [-1/(N-1),1]	D8	Dispersion Game
D9	Graphical Game, Random Graph	D10	Graphical Game, Road Graph
D11	Graphical Game, Star Graph	D12	Location Game
D13	Minimum Effort Game	D14	Polymatrix Game, Random Graph
D15	Polymatrix Game, Road Graph	D16	Polymatrix Game, Small-World Graph
D17	Random Game	D18	Traveler's Dilemma
D19	Uniform LEG, Complete Graph	D20	Uniform LEG, Random Graph
D21	Uniform LEG, Star Graph	D22	War Of Attrition

PNS: Experimental results on 2-player games

- Tested on 100 2-player, 300-action games for each of 22 distributions
- Capped all runs at 1800s



Mixed-Integer Programming Methods for Finding Nash Equilibria

Tuomas Sandholm, Andrew Gilpin, Vincent Conitzer

[AAAI-05 & more recent results]

Motivation of MIP Nash

- Regret of pure strategy s_i is difference in utility between playing optimally (given other player's mixed strategy) and playing s_i.
- Observation: In any equilibrium, every pure strategy either is not played or has zero regret.
- Conversely, any strategy profile where every pure strategy is either not played or has zero regret is an equilibrium.

MIP Nash formulation

- For every pure strategy s_i:
  - There is a 0-1 variable b_{s_i} such that
    - If $b_{s_i} = 1$, s_i is played with 0 probability
    - If $b_{s_i} = 0$, s_i is played with positive probability, and must have 0 regret
  - There is a [0,1] variable p_{s_i} indicating the probability placed on s_i
  - There is a variable u_{s_i} indicating the utility from playing s_i
  - There is a variable r_{s_i} indicating the regret from playing s_i
- For each player i:
  - There is a variable u_i indicating the utility player i receives
  - There is a constant that captures the diff between her max and min utility:

$$U_i = \max_{s_i^h, s_i^l \in S_i, s_{1-i}^h, s_{1-i}^l \in S_{1-i}} u_i(s_i^h, s_{1-i}^h) - u_i(s_i^l, s_{1-i}^l)$$

MIP Nash formulation: Only equilibria are feasible

find $p_{s_i}, u_i, u_{s_i}, r_{s_i}, b_{s_i}$ **such that**

$$(\forall i) \sum_{s_i \in S_i} p_{s_i} = 1 \quad (1)$$

$$(\forall i)(\forall s_i \in S_i) \quad u_{s_i} = \sum_{s_{1-i} \in S_{1-i}} p_{s_{1-i}} \, \pi_i(s_i, s_{1-i}) \quad (2)$$

$$(\forall i)(\forall s_i \in S_i) \quad u_i \geq u_{s_i} \quad (3)$$

$$(\forall i)(\forall s_i \in S_i) \quad r_{s_i} = u_i - u_{s_i} \quad (4)$$

$$(\forall i)(\forall s_i \in S_i) \quad p_{s_i} \leq 1 - b_{s_i} \quad (5)$$

$$(\forall i)(\forall s_i \in S_i) \quad r_{s_i} \leq U_i b_{s_i} \quad (6)$$

domains: $p_{s_i} \geq 0, u_i \geq 0, u_{s_i} \geq 0, r_{s_i} \geq 0, b_{s_i} \in \{0, 1\}.$

MIP Nash formulation: Only equilibria are feasible

- Has the advantage of being able to specify objective function
  - **Can be used to find optimal equilibria (for any linear objective)**

MIP Nash formulation

- Other three formulations explicitly make use of regret minimization:
  - Formulation 2. Penalize regret on strategies that are played with positive probability
  - Formulation 3. Penalize probability placed on strategies with positive regret
  - Formulation 4. Penalize either the regret of, or the probability placed on, a strategy

MIP Nash: Comparing formulations

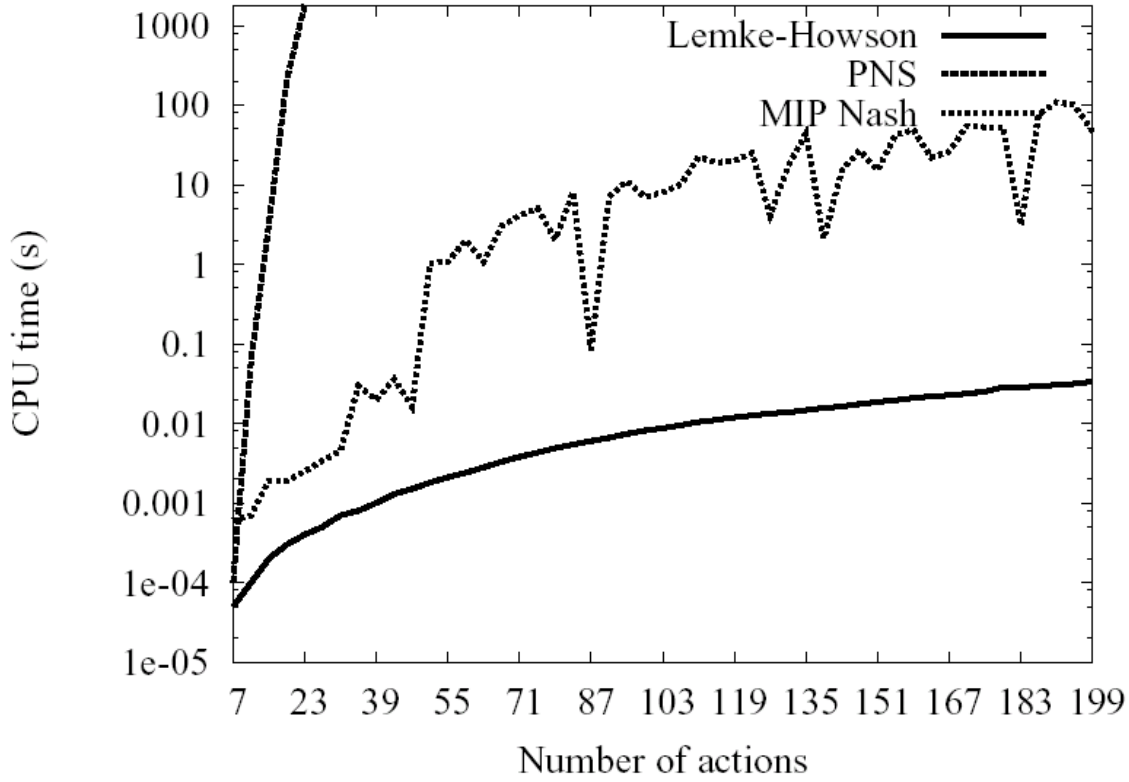
These results are from a newer, extended version of the paper.

	Form. 1	Form. 2	Form. 3	Form. 4
BertrandOligopoly	0	0	0	0
BidirectionalLEG_CG	0	0	0	0
BidirectionalLEG_RG	0	0	0	0
BidirectionalLEG_SG	0	0	0	0
CovariantGame_Pos	0	0	0	0
CovariantGame_Rand	864	864	844.56	864
CovariantGame_Zero	300.97	432	172.01	432
DispersionGame	0	0	0	0
GraphicalGame_RG	289.41	576	270.89	576
GraphicalGame_Road	440.34	648	293.69	648
GraphicalGame_SG	428.82	720	216.46	720
GraphicalGame_SW	433.31	864	279.02	864
LocationGame	0	0	0	0
MinimumEffortGame	0	0	0	0
PolymatrixGame_CG	72	72	72	72
PolymatrixGame_RG	36.04	72	32.84	72
PolymatrixGame_Road	237.21	360	217.93	360
PolymatrixGame_SW	95	216	101.53	216
RandomGame	515.27	1008	411.3	1008
TravelersDilemma	0	0	0	0
UniformLEG_CG	0	0	0	0
UniformLEG_RG	0	0	0	0
UniformLEG_SG	0	0	0	0
WarOfAttrition	0	0	0	0
OVERALL:	3712.37	5832	**2912.23**	5832

Table 1: Average time (in seconds) to find an equilibrium using the different MIP formulations, in 150×150 games from the GAMUT distributions (25 instances of each). If an instance reached the 1800 second limit, that time was counted toward the average. Zero entries indicate that a pure strategy equilibrium existed in each of the 25 instances.

Games with medium-sized supports

- Since PNS performs support enumeration, it should perform poorly on games with medium-sized support
- There is a family of games such that there is a single equilibrium, and the support size is about half
  - And, none of the strategies are dominated (no cascades either)



MIP Nash: Computing optimal equilibria

- MIP Nash is best at finding *optimal* equilibria
- Lemke-Howson and PNS are good at finding sample equilibria
  - M-Enum is an algorithm similar to Lemke-Howson for enumerating all equilibria
- M-Enum and PNS can be modified to find optimal equilibria by finding all equilibria, and choosing the best one
  - In addition to taking exponential time, there may be exponentially many equilibria

actions	*M-Enum*	*PNS*	*MIP Nash*
10	2.21 (0%)	26.45 (3.7%)	0.001 (0%)
25	429.14 (66.7%)	600 (100%)	6.97 (0%)
50	425.07 (66.7%)	600 (100%)	27.2 (2.1%)

Table 3: *Average time (in seconds), over all GAMUT distributions (6 instances of each), for finding a welfare-maximizing equilibrium. The percentage of timeouts (limit here was 600s) is in parentheses.*

Fastest (by and large) algorithm for finding a Nash equilibrium in 2-player normal form games

[Gatti, Rocco & Sandholm, UAI-12]

Algorithm 1 Lemke–Howson with random restart (rrLH)

```
1:  cutoff = cutoff_0
2:  randomly choose one path non–visited till cutoff
3:  repeat
4:      apply complementary pivoting
5:      if the path is longer than cutoff then
6:          if all the paths are visited till cutoff then
7:              cutoff = cutoff + cutoff_0
8:          go to Step 2
9:  until a completely complementary solution is found
10: return current solution
```

Algorithms for solving other types of games

Structured games

- Graphical games
  - Payoff to *i* only depends on a subset of the other agents
  - Poly-time algorithm for undirected trees (Kearns, Littman, Singh 2001)
  - Graphs (Ortiz & Kearns 2003)
  - Directed graphs (Vickery & Koller 2002)
- Action-graph games (Bhat & Leyton-Brown 2004)
  - Each agent's action set is a subset of the vertices of a graph
  - Payoff to *i* only depends on number of agents who take *neighboring* actions

>2 players

- Finding a Nash equilibrium
  - Problem is no longer a linear complementarity problem
    - So Lemke-Howson does not apply
  - Simplicial subdivision method
    - Path-following method derived from Scarf's algorithm
    - Exponential in worst-case
  - Govindan-Wilson method
    - Continuation-based method
    - Can take advantage of structure in games
  - Method like MIP Nash, where the indifference equations are approximated with piecewise linear [Ganzfried & Sandholm CMU-CS-10-105]
  - Non globally convergent methods (*i.e.* incomplete)
    - Non-linear complementarity problem
    - Minimizing a function
    - Slow in practice