Computing Pure Strategy Nash Equilibria in Compact Symmetric Games

Christopher Thomas Ryan,
Albert Xin Jiang, Kevin Leyton-Brown

University of British Columbia, Vancouver, Canada

Computing Pure Strategy Nash Equilibria (PSNE)

Computing Pure Strategy Nash Equilibria (PSNE)

- Computational questions: How hard is it to decide if a game has a PSNE? How hard is it to find one? etc.

Computing Pure Strategy Nash Equilibria (PSNE)

- Computational questions: How hard is it to decide if a game has a PSNE? How hard is it to find one? etc.
- Answer: depends on the input.
  - Polynomial time when input is in normal form.
    - size exponential in the number of players

Computing Pure Strategy Nash Equilibria (PSNE)

- Computational questions: How hard is it to decide if a game has a PSNE? How hard is it to find one? etc.
- Answer: depends on the input.
  - Polynomial time when input is in normal form.
    - size exponential in the number of players
  - Potentially difficult (NP-complete, PLS-complete) when input is "compact".
    - Congestion games [Fabrikant, Papadimitriou & Talwar, 2004; Ieong et al., 2005]
    - Graphical games [Gottlob, Greco & Scarcello 2005]
    - Action graph games [Jiang & Leyton-Brown, 2007; Daskalakis, Schoenebeck, Valiant & Valiant 2009]

Symmetric Games

- We focus on
  - Symmetric games: all players are identical and indistinguishable.
  - Fixed number of actions m, varying number of players n.
  - Utilities are integers.

Symmetric Games

- We focus on
  - Symmetric games: all players are identical and indistinguishable.
  - Fixed number of actions m, varying number of players n.
  - Utilities are integers.
- Define configuration:

$$\mathbf{x} = (x_a : a \in A)$$

where x_a is the number of players playing action a.

Symmetric Games

- We focus on
  - Symmetric games: all players are identical and indistinguishable.
  - Fixed number of actions m, varying number of players n.
  - Utilities are integers.
- Define configuration:

$$\mathbf{x} = (x_a : a \in A)$$

  where x_a is the number of players playing action a.
- Sufficient to specify utility function $u_a(\mathbf{x})$ for each action a and each configuration $\mathbf{x}$.
  - There are $\binom{n+m-1}{m-1} = \Theta(n^{m-1})$ distinct configurations.

Symmetric Games

- We focus on
  - Symmetric games: all players are identical and indistinguishable.
  - Fixed number of actions m, varying number of players n.
  - Utilities are integers.
- Define configuration:

$$\mathbf{x} = (x_a : a \in A)$$

  where x_a is the number of players playing action a.
- Sufficient to specify utility function $u_a(\mathbf{x})$ for each action a and each configuration $\mathbf{x}$.
  - There are $\binom{n+m-1}{m-1} = \Theta(n^{m-1})$ distinct configurations.
  - In previous studies [e.g. Brandt, Fischer & Holzer, 2009; Roughgarden & Papadimitriou, 2005], utility values are given explicitly.

Symmetric Games

- We focus on
  - Symmetric games: all players are identical and indistinguishable.
  - Fixed number of actions m, varying number of players n.
  - Utilities are integers.
- Define configuration:

$$\mathbf{x} = (x_a : a \in A)$$

where x_a is the number of players playing action a.
- Sufficient to specify utility function $u_a(\mathbf{x})$ for each action a and each configuration $\mathbf{x}$.
  - There are $\binom{n+m-1}{m-1} = \Theta(n^{m-1})$ distinct configurations.
  - In previous studies [e.g. Brandt, Fischer & Holzer, 2009; Roughgarden & Papadimitriou, 2005], utility values are given explicitly.
  - Compute PSNE in poly time by enumerating configurations

More compact representations of u_a

- We focus on compact representations of u_a: those requiring only $poly(\log n)$ bits.

More compact representations of u_a

- We focus on compact representations of u_a: those requiring only $poly(\log n)$ bits.
- Sanity check:
  - Specifying input: need only $m \log n$ bits.
  - Specifying output: can map utilities to $\left\{1, 2, \ldots, \binom{n+m-1}{m-1}\right\}$ while preserving PSNE, thus need only $O(\log n)$ bits.

More compact representations of u_a

- We focus on compact representations of u_a: those requiring only $poly(\log n)$ bits.
- Sanity check:
  - Specifying input: need only $m \log n$ bits.
  - Specifying output: can map utilities to $\left\{1, 2, \ldots, \binom{n+m-1}{m-1}\right\}$ while preserving PSNE, thus need only $O(\log n)$ bits.

- Computing PSNE: with such a compact representation, is it even in NP?

More compact representations of u_a

- We focus on compact representations of u_a: those requiring only $poly(\log n)$ bits.
- Sanity check:
  - Specifying input: need only $m \log n$ bits.
  - Specifying output: can map utilities to $\left\{1, 2, \ldots, \binom{n+m-1}{m-1}\right\}$ while preserving PSNE, thus need only $O(\log n)$ bits.

- Computing PSNE: with such a compact representation, is it even in NP?
  - To check if $\mathbf{x}$ is in N, the set of of PSNE configurations, only need to check for each pair of actions a and a', whether there is a profitable deviation from playing a to playing a'.
  - Checking whether $\mathbf{x} \in N$ is in P (thus computing PSNE in NP) if the utility functions can be evaluated in poly time.

Circuit Symmetric Games

- How hard can it get?
- Represent each u_a by a Boolean circuit
  - general method for representing utility functions; complexity for other circuit-based models studied in e.g. [Schoenebeck & Vadhan, 2006]
- Compact when number of gates is $poly(\log n)$

Circuit Symmetric Games

- How hard can it get?
- Represent each u_a by a Boolean circuit
  - general method for representing utility functions; complexity for other circuit-based models studied in e.g. [Schoenebeck & Vadhan, 2006]
- Compact when number of gates is $poly(\log n)$

Theorem (Circuit symmetric games)

- *When utilities are represented by Boolean circuits, and $m \geq 3$, deciding if a PSNE exists is NP-complete.*
- *When $m = 2$, there exists at least one PSNE and a sample PSNE can be found in poly time.*

- existence of PSNE for the $m = 2$ case was proved by [Cheng, Reeves, Vorobeychik & Wellman 2004]; also follows from the fact that such a game is a potential game.

Piecewise-linear symmetric games

- We can do better by considering a natural subclass: piecewise-linear functions.

Piecewise-linear symmetric games

- We can do better by considering a natural subclass: piecewise-linear functions.

Theorem (Informal version)

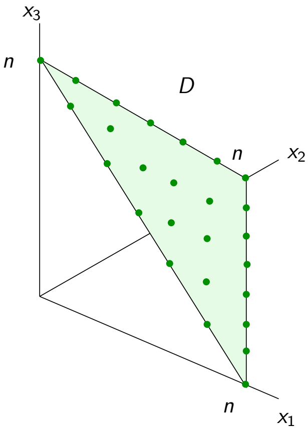
When utilities are expressed as piecewise-linear functions, there exist polynomial time algorithms to decide if a PSNE exists and find a sample equilibrium.

PWL symmetric game

PWL symmetric game

- Domain of utility functions: configurations

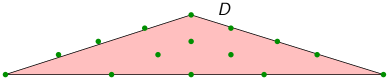
$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, \mathbf{x} \geq \mathbf{0} \right\}$$



PWL symmetric game

- Domain of utility functions: configurations

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, \mathbf{x} \geq \mathbf{0} \right\}$$



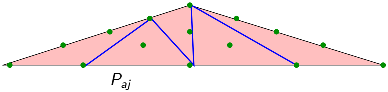
PWL symmetric game

- Domain of utility functions: configurations

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, \mathbf{x} \geq \mathbf{0} \right\}$$

- Piecewise linear utilities: For each $a \in A$:

$$D = \biguplus_{P_{a,j} \in \mathbf{P}_a} (P_{a,j} \cap \mathbb{Z}^m)$$



PWL symmetric game

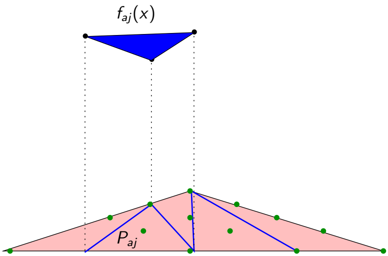
- Domain of utility functions: configurations

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, \mathbf{x} \geq \mathbf{0} \right\}$$

- Piecewise linear utilities: For each $a \in A$:

$$D = \biguplus_{P_{a,j} \in \mathbf{P}_a} (P_{a,j} \cap \mathbb{Z}^m)$$

- Over each cell $P_{a,j} \cap \mathbb{Z}^m$ there is an affine function $f_{a,j}(\mathbf{x}) = \boldsymbol{\alpha}_{a,j} \cdot \mathbf{x} + \beta_{a,j}$.



PWL symmetric game

- Domain of utility functions: configurations

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, \mathbf{x} \geq \mathbf{0} \right\}$$

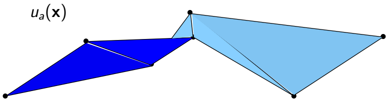
- Piecewise linear utilities: For each $a \in A$:

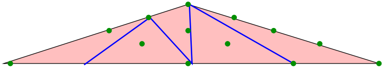
$$D = \biguplus_{P_{a,j} \in \mathbf{P}_a} (P_{a,j} \cap \mathbb{Z}^m)$$

- Over each cell $P_{a,j} \cap \mathbb{Z}^m$ there is an affine function $f_{a,j}(\mathbf{x}) = \boldsymbol{\alpha}_{a,j} \cdot \mathbf{x} + \beta_{a,j}$.
- Piecing them together:

$$u_a(\mathbf{x}) = f_{a,j}(\mathbf{x}) \text{ for } \mathbf{x} \in P_{a,j} \cap \mathbb{Z}^m$$

- Compact when number of pieces $|\mathbf{P}_a|$ is *poly*$(\log n)$.

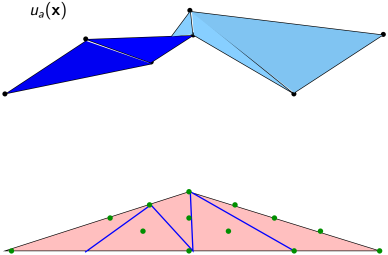




Theorem (Formal version)

Consider a symmetric game with PWL utilities given by the following input:

- *the binary encoding of the number n of players;*
- *for each $a \in A$, the utility function $u_a(\mathbf{x})$ represented as the binary encoding of the inequality description of each P_{aj} and affine functions f_{aj}.*



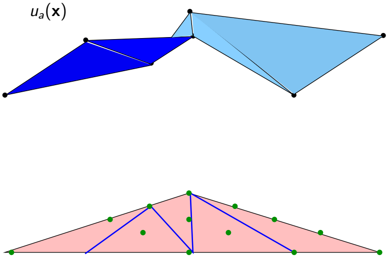
Theorem (Formal version)

Consider a symmetric game with PWL utilities given by the following input:

- *the binary encoding of the number n of players;*
- *for each $a \in A$, the utility function $u_a(\mathbf{x})$ represented as the binary encoding of the inequality description of each P_{aj} and affine functions f_{aj}.*

Then, when the number of actions m is fixed, and even when the number of pieces are poly$(\log n)$, there exists

1. *a polynomial-time algorithm to compute the number of PSNE*
2. *a polynomial-time algorithm to find a sample PSNE*
3. *a polynomial-space, polynomial-delay enumeration algorithm to enumerate all PSNE.*

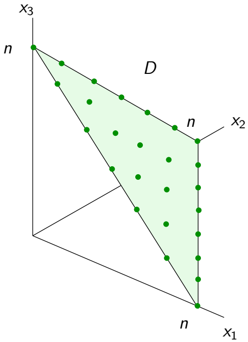


Tool of analysis

- Encode the set of PSNE by a rational generating function.
- Leverage theory from encoding sets of polytopal lattice points.
  - previously applied in combinatorics, optimization, compiler design [e.g. De Loera et al. 2007]

Tool of analysis

- Encode the set of PSNE by a rational generating function.
- Leverage theory from encoding sets of polytopal lattice points.
  - previously applied in combinatorics, optimization, compiler design [e.g. De Loera et al. 2007]



Generating function encoding

- Given $S \subseteq \mathbb{Z}^n$ we represent the points as a generating function:
$$g(S, w) = \sum_{a \in S} w_1^{a_1} w_2^{a_2} \cdots w_n^{a_n}$$

Generating function encoding

- Given $S \subseteq \mathbb{Z}^n$ we represent the points as a generating function:
$$g(S, w) = \sum_{a \in S} w_1^{a_1} w_2^{a_2} \cdots w_n^{a_n}$$
- w_i are complex variables
- Point $(2, -3)$ is encoded as monomial $w_1^2 w_2^{-3}$.

Generating function encoding

- Given $S \subseteq \mathbb{Z}^n$ we represent the points as a generating function:

$$g(S, w) = \sum_{a \in S} w_1^{a_1} w_2^{a_2} \cdots w_n^{a_n}$$

- w_i are complex variables
- Point $(2, -3)$ is encoded as monomial $w_1^2 w_2^{-3}$.

Example

- $S = \{0, 1, \ldots, 1000\}$

Generating function encoding

- Given $S \subseteq \mathbb{Z}^n$ we represent the points as a generating function:
$$g(S, w) = \sum_{a \in S} w_1^{a_1} w_2^{a_2} \cdots w_n^{a_n}$$
- w_i are complex variables
- Point $(2, -3)$ is encoded as monomial $w_1^2 w_2^{-3}$.

Example

- $S = \{0, 1, \ldots, 1000\}$
- $g(S, w) = 1 + w + w^2 + \cdots + w^{1000}$

Generating function encoding

- Given $S \subseteq \mathbb{Z}^n$ we represent the points as a generating function:

$$g(S, w) = \sum_{a \in S} w_1^{a_1} w_2^{a_2} \cdots w_n^{a_n}$$

- w_i are complex variables
- Point $(2, -3)$ is encoded as monomial $w_1^2 w_2^{-3}$.

Example

- $S = \{0, 1, \dots, 1000\}$
- $g(S, w) = 1 + w + w^2 + \cdots + w^{1000}$
- $g(S, w) = \frac{1}{1-w} - \frac{w^{1001}}{1-w}$

Barvinok's result (1994)

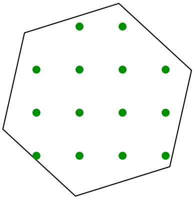
Theorem

Let P be a rational convex polytope, i.e. $P = \{x \in \mathbb{R}^m : Ax \le b\}$. There is a polynomial time algorithm which computes a short rational generating function:

$$g(P \cap \mathbb{Z}^m; w) = \sum_{j \in J} \gamma_j \frac{w^{c_j}}{(1 - w^{d_{j1}})(1 - w^{d_{j2}}) \dots (1 - w^{d_{jm}})},$$

of the lattice points inside P when the dimension m is fixed. The number of terms in the sum is polynomially bounded and $\gamma_j \in \{-1, 1\}$.

A Tale of Two Representations



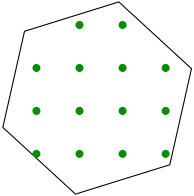
Lattice points: S

A Tale of Two Representations

Inequality representation:

$\{x : Ax \leq b,\ x \in \mathbb{Z}^n\}$

Data: A, b



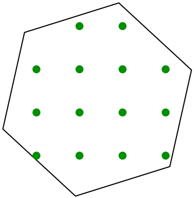
Lattice points: S

A Tale of Two Representations

Inequality representation:

$$\{x : Ax \leq b,\ x \in \mathbb{Z}^n\}$$

Data: A, b



Lattice points: S

Gen. Function Representation:

$$\sum_{j \in J} \gamma_j \frac{w^{c_j}}{\prod_{k=1}^{n}(1 - w^{d_{jk}})}$$

Data: c_j, d_{jk}

Accessing the points in a generating function encoding

Accessing the points in a generating function encoding

- Count the number of integer points in S in polynomial time.
  [Barvinok, 1994]

Accessing the points in a generating function encoding

- Count the number of integer points in S in polynomial time. [Barvinok, 1994]

Example

- $S = \{0, 1, \ldots, 1000\}$

Accessing the points in a generating function encoding

- Count the number of integer points in S in polynomial time. [Barvinok, 1994]

Example

- $S = \{0, 1, \ldots, 1000\}$
- $g(S, w) = 1 + w + w^2 + \cdots + w^{1000}$.
  Count: substitute $w = 1$, get $g(S, 1) = 1001$.

Accessing the points in a generating function encoding

- Count the number of integer points in S in polynomial time. [Barvinok, 1994]

Example

- $S = \{0, 1, \ldots, 1000\}$
- $g(S, w) = 1 + w + w^2 + \cdots + w^{1000}$.
  Count: substitute $w = 1$, get $g(S, 1) = 1001$.
- $g(S, w) = \frac{1}{1-w} - \frac{w^{1001}}{1-w}$.
  Count: take limit as $w \to 1$, get $\lim_{w\to 1} g(S, w) = 1001$.

Accessing the points in a generating function encoding

- Count the number of integer points in S in polynomial time. [Barvinok, 1994]

Example

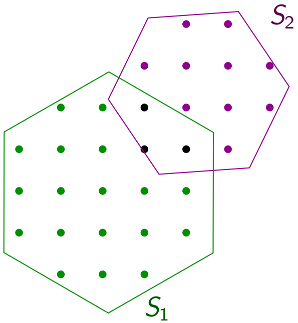
- $S = \{0, 1, \ldots, 1000\}$
- $g(S, w) = 1 + w + w^2 + \cdots + w^{1000}$.
  Count: substitute $w = 1$, get $g(S, 1) = 1001$.
- $g(S, w) = \frac{1}{1-w} - \frac{w^{1001}}{1-w}$.
  Count: take limit as $w \to 1$, get $\lim_{w\to 1} g(S, w) = 1001$.

- Enumerate the elements of S: There exists a polynomial-delay enumeration algorithm which outputs the elements of S. [De Loera et al. 2007]

More ways to encode (Barvinok-Woods, 2003)

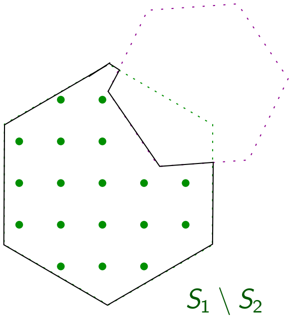
More ways to encode (Barvinok-Woods, 2003)

Boolean combinations:



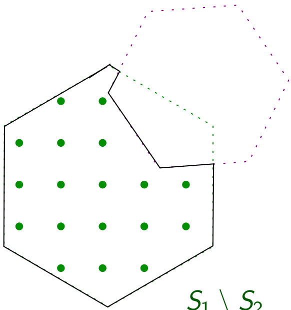
More ways to encode (Barvinok-Woods, 2003)

Boolean combinations:

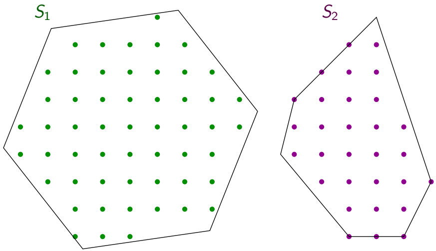


More ways to encode (Barvinok-Woods, 2003)

Boolean combinations:



Disjoint unions:



$g(S_1 \cup S_2, w) = g(S_1, w) + g(S_2, w)$

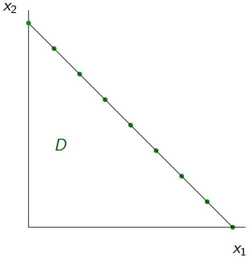
Key insight into proof: Express PSNE via polytopes

- Want to encode N, the set of PSNE configurations

$$\mathbf{x} \in N \iff \forall a \in A : (x_a = 0) \ \text{ OR } \ (\forall a' \in A,\ u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x}+\mathbf{e}_{a'}-\mathbf{e}_a))$$

- D is the set of configurations and candidate equilibria:

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, \mathbf{x} \geq \mathbf{0} \right\}$$



Key insight into proof: Express PSNE via polytopes

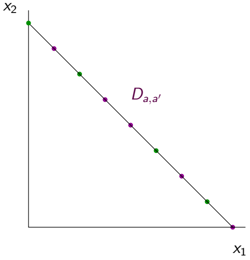
- Want to encode N, the set of PSNE configurations

$$\mathbf{x} \in N \iff \forall a \in A : (x_a = 0) \text{ OR } (\forall a' \in A,\ u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x}+\mathbf{e}_{a'}-\mathbf{e}_a))$$

- D is the set of configurations and candidate equilibria:

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, \mathbf{x} \geq \mathbf{0} \right\}$$

- $D_{a,a'}$ those configurations where it is profitable for a player playing action a to deviate.



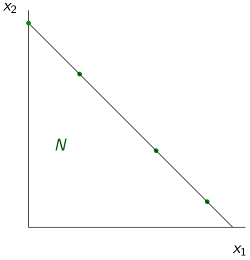
Key insight into proof: Express PSNE via polytopes

- Want to encode N, the set of PSNE configurations

$$\mathbf{x} \in N \iff \forall a \in A : (x_a = 0) \ \text{ OR } \ (\forall a' \in A,\ u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x}+\mathbf{e}_{a'}-\mathbf{e}_a))$$

- D is the set of configurations and candidate equilibria:

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, \mathbf{x} \geq \mathbf{0} \right\}$$



- $D_{a,a'}$ those configurations where it is profitable for a player playing action a to deviate.

$$N = D \setminus \bigcup_{a,a' \in A} D_{a,a'}$$

Expressing $D_{a,a'}$

$$D_{a,a'} = \biguplus_{P_{a,j} \in \mathbf{P}_a} \biguplus_{P_{a',j'} \in \mathbf{P}_{a'}} \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{a,j}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a',j'} \\ f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1 \end{array} \right\}$$

Expressing $D_{a,a'}$

$$D_{a,a'} = \biguplus_{P_{a,j} \in \mathbf{P}_a} \biguplus_{P_{a',j'} \in \mathbf{P}_{a'}} \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{a,j}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a',j'} \\ f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1 \end{array} \right\}$$

- Polynomial number of disjoint unions
- Once the pieces $P_{a,j}$ and $P_{a',j'}$ fixed, can formulate profitable deviation as a set of linear constraints

Expressing $D_{a,a'}$

$$D_{a,a'} = \biguplus_{P_{a,j} \in \mathbf{P}_a} \biguplus_{P_{a',j'} \in \mathbf{P}_{a'}} \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{a,j}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a',j'} \\ f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1 \end{array} \right\}$$

- Polynomial number of disjoint unions
- Once the pieces $P_{a,j}$ and $P_{a',j'}$ fixed, can formulate profitable deviation as a set of linear constraints
  - $x_a \geq 1$: at least one player chose a

Expressing $D_{a,a'}$

$$D_{a,a'} = \biguplus_{P_{a,j} \in \mathbf{P}_a} \biguplus_{P_{a',j'} \in \mathbf{P}_{a'}} \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{a,j}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a',j'} \\ f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1 \end{array} \right\}$$

- Polynomial number of disjoint unions
- Once the pieces $P_{a,j}$ and $P_{a',j'}$ fixed, can formulate profitable deviation as a set of linear constraints
  - $x_a \geq 1$: at least one player chose a
  - $\mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a$: result of deviating from a to a'

Expressing $D_{a,a'}$

$$D_{a,a'} = \biguplus_{P_{a,j} \in \mathbf{P}_a} \biguplus_{P_{a',j'} \in \mathbf{P}_{a'}} \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{a,j}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a',j'} \\ f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1 \end{array} \right\}$$

- Polynomial number of disjoint unions
- Once the pieces $P_{a,j}$ and $P_{a',j'}$ fixed, can formulate profitable deviation as a set of linear constraints
  - $x_a \geq 1$: at least one player chose a
  - $\mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a$: result of deviating from a to a'
  - $f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1$: since utilities are integers, equivalent to $f_{a,j}(\mathbf{x}) < f_{a',j'}(\mathbf{x}')$

Expressing $D_{a,a'}$

$$D_{a,a'} = \biguplus_{P_{a,j} \in \mathbf{P}_a} \; \biguplus_{P_{a',j'} \in \mathbf{P}_{a'}} \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{a,j}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a',j'} \\ f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1 \end{array} \right\}$$

- Polynomial number of disjoint unions
- Once the pieces $P_{a,j}$ and $P_{a',j'}$ fixed, can formulate profitable deviation as a set of linear constraints
  - $x_a \geq 1$: at least one player chose a
  - $\mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a$: result of deviating from a to a'
  - $f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1$: since utilities are integers, equivalent to $f_{a,j}(\mathbf{x}) < f_{a',j'}(\mathbf{x}')$
- Therefore N can be expressed as a short rational generating function

Expressing $D_{a,a'}$

$$D_{a,a'} = \biguplus_{P_{a,j} \in \mathbf{P}_a} \biguplus_{P_{a',j'} \in \mathbf{P}_{a'}} \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{a,j}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a',j'} \\ f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1 \end{array} \right\}$$

- Polynomial number of disjoint unions
- Once the pieces $P_{a,j}$ and $P_{a',j'}$ fixed, can formulate profitable deviation as a set of linear constraints
  - $x_a \geq 1$: at least one player chose a
  - $\mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a$: result of deviating from a to a'
  - $f_{a,j}(\mathbf{x}) \leq f_{a',j'}(\mathbf{x}') - 1$: since utilities are integers, equivalent to $f_{a,j}(\mathbf{x}) < f_{a',j'}(\mathbf{x}')$
- Therefore N can be expressed as a short rational generating function
- Can check existence of PSNE via counting operation; find a sample PSNE via enumeration operation.

Other results

- Find a PSNE that approximately optimizes the sum of the utilities (FPTAS).
- Encode the PSNEs of a parameterized family of symmetric games with utility pieces:

  $$f_{a,j}(\mathbf{x}, \mathbf{p}) = \boldsymbol{\alpha}_{a,j} \cdot \mathbf{x} + \boldsymbol{\beta}_{a,j} \cdot \mathbf{p},$$

  where **p** is a fixed dimensional integer vector of parameters inside a polytope.

Other results

- Find a PSNE that approximately optimizes the sum of the utilities (FPTAS).
- Encode the PSNEs of a parameterized family of symmetric games with utility pieces:

  $$f_{a,j}(\mathbf{x}, \mathbf{p}) = \boldsymbol{\alpha}_{a,j} \cdot \mathbf{x} + \boldsymbol{\beta}_{a,j} \cdot \mathbf{p},$$

  where **p** is a fixed dimensional integer vector of parameters inside a polytope.
  - Answer questions about PSNEs of the family of games without solving each game
  - e.g. finding parameter **p** that optimizes some objective.

Conclusion

- computing PSNE for symmetric games with fixed number of actions, focusing on compact representations of utility: $poly(\log n)$ bits
- circuit symmetric games: NP-complete when at least 3 actions
- symmetric games with piecewise-linear utility: polynomial-time algorithms
  - encode set of PSNE as a rational generating function

Thanks!