Computational aspects of finding Nash Equilibria for 2-player games

Fall 2020

Nash equilibrium

Consider a 2-player game $\Gamma = (A_1, A_2, u_1, u_2)$.
Let $X = \Delta(A_1)$ and $Y = \Delta(A_2)$.

($\Delta(A)$ is the set of probability distributions over A)

Nash equilibrium

Consider a 2-player game $\Gamma = (A_1, A_2, u_1, u_2)$.
Let $X = \Delta(A_1)$ and $Y = \Delta(A_2)$.

($\Delta(A)$ is the set of probability distributions over A)

A Nash equilibrium is a mixed strategy profile $\sigma = (x, y) \in X \times Y$ such that, for every $x' \in X$, $y' \in Y$, it holds

$$U_1(x, y) \geq U_1(x', y) \text{ and } U_2(x, y) \geq U_2(x, y')$$

Linear algebra notation

Consider a 2-player game $\Gamma = (A_1, A_2, u_1, u_2)$

Linear algebra notation

Consider a 2-player game $\Gamma = (A_1, A_2, u_1, u_2)$
Let $m = |A_1|$ and $n = |A_2|$.

Linear algebra notation

Consider a 2-player game $\Gamma = (A_1, A_2, u_1, u_2)$
Let $m = |A_1|$ and $n = |A_2|$.

$x \in X$ is a probability distribution:

$$x = (x_1, \ldots, x_m),\ x_i \geq 0 \text{ and } x_1 + \cdots + x_m = 1$$

Linear algebra notation

Consider a 2-player game $\Gamma = (A_1, A_2, u_1, u_2)$
Let $m = |A_1|$ and $n = |A_2|$.

$x \in X$ is a probability distribution:

$$x = (x_1, \ldots, x_m),\ x_i \geq 0 \text{ and } x_1 + \cdots + x_m = 1$$

$y \in Y$ is a probability distribution:

$$y = (y_1, \ldots, y_n),\ y_j \geq 0 \text{ and } y_1 + \cdots + y_n = 1$$

Linear algebra notation

Consider a 2-player game $\Gamma = (A_1, A_2, u_1, u_2)$
Let $m = |A_1|$ and $n = |A_2|$.

$x \in X$ is a probability distribution:

$$x = (x_1, \ldots, x_m),\ x_i \geq 0 \text{ and } x_1 + \cdots + x_m = 1$$

$y \in Y$ is a probability distribution:

$$y = (y_1, \ldots, y_n),\ y_j \geq 0 \text{ and } y_1 + \cdots + y_n = 1$$

Utilities can be described by a $m \times n$ matrix R, for the row player, and C, for the column player. Then,

Linear algebra notation

Consider a 2-player game $\Gamma = (A_1, A_2, u_1, u_2)$
Let $m = |A_1|$ and $n = |A_2|$.

$x \in X$ is a probability distribution:

$$x = (x_1, \dots, x_m),\ x_i \geq 0 \text{ and } x_1 + \cdots + x_m = 1$$

$y \in Y$ is a probability distribution:

$$y = (y_1, \dots, y_n),\ y_j \geq 0 \text{ and } y_1 + \cdots + y_n = 1$$

Utilities can be described by a $m \times n$ matrix R, for the row player, and C, for the column player. Then,

$$U_1(x, y) = x^T R\, y \text{ and } U_2(x, y) = x^T C\, y$$

Computing a best response

For a given $x \in X$, we have to solve:

Computing a best response

For a given $x \in X$, we have to solve:

$$\begin{array}{rl} \max & x^T C\, y \\ \text{Subject to:} & y_1 + \cdots + y_n = 1,\ y_j \geq 0. \end{array}$$

Computing a best response

For a given $x \in X$, we have to solve:

$$\begin{aligned} \max \quad & x^T C\, y \\ \text{Subject to:}\ & y_1 + \cdots + y_n = 1,\ y_j \geq 0. \end{aligned}$$

For a given y, we have to solve:

Computing a best response

For a given $x \in X$, we have to solve:

$$\begin{array}{rl} \max & x^T C\, y \\ \text{Subject to:} & y_1 + \cdots + y_n = 1,\ y_j \geq 0. \end{array}$$

For a given y, we have to solve:

$$\begin{array}{rl} \max & x^T R\, y \\ \text{Subject to:} & x_1 + \cdots + x_m = 1,\ x_i \geq 0 \end{array}$$

Computing a best response

For a given $x \in X$, we have to solve:

$$\begin{aligned} \max \quad & x^T C\, y \\ \text{Subject to:} \quad & y_1 + \cdots + y_n = 1,\ y_j \geq 0. \end{aligned}$$

For a given y, we have to solve:

$$\begin{aligned} \max \quad & x^T R\, y \\ \text{Subject to:} \quad & x_1 + \cdots + x_m = 1,\ x_i \geq 0 \end{aligned}$$

Those are linear programming problems, so

Computing a best response

For a given $x \in X$, we have to solve:

$$\begin{array}{rl} \max & x^T C\, y \\ \text{Subject to:} & y_1 + \cdots + y_n = 1,\ y_j \geq 0. \end{array}$$

For a given y, we have to solve:

$$\begin{array}{rl} \max & x^T R\, y \\ \text{Subject to:} & x_1 + \cdots + x_m = 1,\ x_i \geq 0 \end{array}$$

Those are linear programming problems, so
A best response can be computed in polynomial time for 2-player games with rational utilities.

Zero-sum games

- A zero-sum game is a 2-player game such that, for each pure strategy profile (a, b), $u_1(a, b) + u_2(a, b) = 0$.

Zero-sum games

- A zero-sum game is a 2-player game such that, for each pure strategy profile (a, b), $u_1(a, b) + u_2(a, b) = 0$.
- That is Let $u = u_1$, we have $u_2 = -u$.

Zero-sum games

- A zero-sum game is a 2-player game such that, for each pure strategy profile (a, b), $u_1(a, b) + u_2(a, b) = 0$.
- That is Let $u = u_1$, we have $u_2 = -u$.
- Player 1 is interested in maximizing u and player 2 in minimizing u.

Zero-sum games

- A zero-sum game is a 2-player game such that, for each pure strategy profile (a, b), $u_1(a, b) + u_2(a, b) = 0$.
- That is Let $u = u_1$, we have $u_2 = -u$.
- Player 1 is interested in maximizing u and player 2 in minimizing u.
- In terms of matrices we have $C = -R$.

ZS: Nash conditions

- (x^*, y^*) is a NE

ZS: Nash conditions

- (x^*, y^*) is a NE

$$(x^*)^T R\, y^* \geq x^T R\, y^*, \text{ for all } x \in X,$$
$$(x^*)^T C\, y^* \geq (x^*)^T C\, y, \text{ for all } y \in Y.$$

ZS: Nash conditions

- (x^*, y^*) is a NE

$$(x^*)^T R\, y^* \geq x^T R\, y^*, \text{ for all } x \in X,$$
$$(x^*)^T C\, y^* \geq (x^*)^T C\, y, \text{ for all } y \in Y.$$

- As $C = -R$ the second equation becomes

ZS: Nash conditions

- (x^*, y^*) is a NE

$$(x^*)^T R\, y^* \geq x^T R\, y^*, \text{ for all } x \in X,$$
$$(x^*)^T C\, y^* \geq (x^*)^T C\, y, \text{ for all } y \in Y.$$

- As $C = -R$ the second equation becomes

$$(x^*)^T R\, y^* \leq (x^*)^T R\, y, \forall y \in Y.$$

ZS: Nash conditions

- (x^*, y^*) is a NE

$$(x^*)^T R\, y^* \geq x^T R\, y^*, \text{ for all } x \in X,$$
$$(x^*)^T C\, y^* \geq (x^*)^T C\, y, \text{ for all } y \in Y.$$

- As $C = -R$ the second equation becomes

$$(x^*)^T R\, y^* \leq (x^*)^T R\, y, \forall y \in Y.$$

- Combining both,

$$x^T R\, y^* \leq (x^*)^T R\, y^* \leq (x^*)^T R\, y, \forall x \in X, \forall y \in Y.$$

ZS: Nash conditions

- (x^*, y^*) is a NE

$$(x^*)^T R\, y^* \geq x^T R\, y^*, \text{ for all } x \in X,$$
$$(x^*)^T C\, y^* \geq (x^*)^T C\, y, \text{ for all } y \in Y.$$

- As $C = -R$ the second equation becomes

$$(x^*)^T R\, y^* \leq (x^*)^T R\, y, \forall y \in Y.$$

- Combining both,

$$x^T R\, y^* \leq (x^*)^T R\, y^* \leq (x^*)^T R\, y, \forall x \in X, \forall y \in Y.$$

i.e., (x^*, y^*) is a saddle point
of the function $x^T R\, y$ defined over $X \times Y$.

Minimax inequality

Minimax inequality

Theorem

For any function $\Phi : X \times Y :\to \mathbb{R}$, *we have*

$$\sup_{x \in X} \inf_{y \in Y} \Phi(x, y) \leq \inf_{y \in Y} \sup_{x \in X} \Phi(x, y).$$

Minimax inequality

Theorem

For any function $\Phi : X \times Y :\rightarrow \mathbb{R}$*, we have*

$$\sup_{x \in X} \inf_{y \in Y} \Phi(x, y) \leq \inf_{y \in Y} \sup_{x \in X} \Phi(x, y).$$

Proof.

Minimax inequality

Theorem

For any function $\Phi : X \times Y :\to \mathbb{R}$, *we have*

$$\sup_{x \in X} \inf_{y \in Y} \Phi(x, y) \leq \inf_{y \in Y} \sup_{x \in X} \Phi(x, y).$$

Proof.

$\forall x' \in X$, $\Phi(x', y) \leq \sup_{x \in X} \Phi(x, y)$. Then,

Minimax inequality

Theorem

For any function $\Phi : X \times Y :\rightarrow \mathbb{R}$, *we have*

$$\sup_{x \in X} \inf_{y \in Y} \Phi(x,y) \leq \inf_{y \in Y} \sup_{x \in X} \Phi(x,y).$$

Proof.

$\forall x' \in X$, $\Phi(x', y) \leq \sup_{x \in X} \Phi(x,y)$. Then,

$$\forall x' \in X, \inf_{y \in Y} \Phi(x', y) \leq \inf_{y \in Y} \sup_{x \in X} \Phi(x,y).$$

Minimax inequality

Theorem

For any function $\Phi : X \times Y :\to \mathbb{R}$, *we have*

$$\sup_{x \in X} \inf_{y \in Y} \Phi(x, y) \leq \inf_{y \in Y} \sup_{x \in X} \Phi(x, y).$$

Proof.

$\forall x' \in X$, $\Phi(x', y) \leq \sup_{x \in X} \Phi(x, y)$. Then,

$$\forall x' \in X, \inf_{y \in Y} \Phi(x', y) \leq \inf_{y \in Y} \sup_{x \in X} \Phi(x, y).$$

Taking the supremum over $x' \in X$ on the left hand-side,

$$\sup_{x \in X} \inf_{y \in Y} \Phi(x, y) \leq \inf_{y \in Y} \sup_{x \in X} \Phi(x, y).$$

ZS: Nash conditions

We have

ZS: Nash conditions

We have

- $x^T R\, y^* \leq (x^*)^T R\, y^* \leq (x^*)^T R\, y, \forall x \in X, \forall y \in Y.$

ZS: Nash conditions

We have

- $x^T R\, y^* \leq (x^*)^T R\, y^* \leq (x^*)^T R\, y,\ \forall x \in X,\ \forall y \in Y.$
- Thus

$$\sup_{x \in X} x^T R\, y^* \leq (x^*)^T R\, y^* \leq \inf_{y \in Y} (x^*)^T R\, y$$

ZS: Nash conditions

We have

- $x^T R\, y^* \leq (x^*)^T R\, y^* \leq (x^*)^T R\, y,\ \forall x \in X,\ \forall y \in Y.$
- Thus

$$\sup_{x \in X} x^T R\, y^* \leq (x^*)^T R\, y^* \leq \inf_{y \in Y} (x^*)^T R\, y$$

$$\inf_{y \in Y} \sup_{x \in X} x^T R\, y \leq \sup_{x \in X} x^T R\, y^* \leq (x^*)^T R\, y^* \leq \inf_{y \in Y} (x^*)^T R\, y \leq \sup_{x \in X} \inf_{y \in Y} x^T R\, y$$

ZS: Nash conditions

We have

- $x^T R\, y^* \leq (x^*)^T R\, y^* \leq (x^*)^T R\, y,\ \forall x \in X,\ \forall y \in Y.$
- Thus
$$\sup_{x \in X} x^T R\, y^* \leq (x^*)^T R\, y^* \leq \inf_{y \in Y} (x^*)^T R\, y$$

$$\inf_{y \in Y} \sup_{x \in X} x^T R\, y \leq \sup_{x \in X} x^T R\, y^* \leq (x^*)^T R\, y^* \leq \inf_{y \in Y} (x^*)^T R\, y \leq \sup_{x \in X} \inf_{y \in Y} x^T R\, y$$

- Using the minimax inequality, we get

ZS: Nash conditions

We have

- $x^T R\, y^* \leq (x^*)^T R\, y^* \leq (x^*)^T R\, y,\ \forall x \in X,\ \forall y \in Y.$
- Thus

$$\sup_{x \in X} x^T R\, y^* \leq (x^*)^T R\, y^* \leq \inf_{y \in Y} (x^*)^T R\, y$$

$$\inf_{y \in Y} \sup_{x \in X} x^T R\, y \leq \sup_{x \in X} x^T R\, y^* \leq (x^*)^T R\, y^* \leq \inf_{y \in Y} (x^*)^T R\, y \leq \sup_{x \in X} \inf_{y \in Y} x^T R\, y$$

- Using the minimax inequality, we get

$$\inf_{y \in Y} \sup_{x \in X} x^T R\, y = (x^*)^T R\, y^* = \sup_{x \in X} \inf_{y \in Y} x^T R\, y$$

ZS: Nash conditions

We have

- $x^T R y^* \leq (x^*)^T R y^* \leq (x^*)^T R y, \forall x \in X, \forall y \in Y.$
- Thus

$$\sup_{x \in X} x^T R y^* \leq (x^*)^T R y^* \leq \inf_{y \in Y} (x^*)^T R y$$

$$\inf_{y \in Y} \sup_{x \in X} x^T R y \leq \sup_{x \in X} x^T R y^* \leq (x^*)^T R y^* \leq \inf_{y \in Y} (x^*)^T R y \leq \sup_{x \in X} \inf_{y \in Y} x^T R y$$

- Using the minimax inequality, we get

$$\inf_{y \in Y} \sup_{x \in X} x^T R y = (x^*)^T R y^* = \sup_{x \in X} \inf_{y \in Y} x^T R y$$

We refer to $\inf_{y \in Y} \sup_{x \in X} x^T R y$ as the value of the game.

Best response condition and Bimatrix Games

For a fixed $y \in Y$, let u_r the value of the best response of player 1 to y:

$$u_r = \max_{x \in X} x^T R y = \max_{x \in X} \sum_{i=1}^{m} \sum_{j=1}^{n} x_i r_{ij} y_j$$

Let $[Ry]_i = \sum_{j=1}^{n} r_{ij} y_j$

Theorem (Nash)

For a fixed $y \in Y$,

$$u_r = \max_{k=1,\ldots,m} \{[Ry]_k\},$$

and if x is a BR to y, then for all $x_i > 0$, $[Ry]_i = u_r$

Best response condition

Proof.

Let x be a BR to y.

$$u_r = x^T Ry = \sum_{i=1}^{m} x_i [Ry]_i \leq \sum_{i=1}^{m} x_i (\max_{k=1,\ldots,m} \{[Ry]_k\})$$

Best response condition

Proof.

Let x be a BR to y.

$$u_r = x^T R y = \sum_{i=1}^{m} x_i [Ry]_i \leq \sum_{i=1}^{m} x_i (\max_{k=1,\ldots,m} \{[Ry]_k\})$$

Hence,

$$u_r \leq \max_{k=1,\ldots,m} \{[Ry]_k\}$$

Best response condition

Proof.

Let x be a BR to y.

$$u_r = x^T Ry = \sum_{i=1}^{m} x_i [Ry]_i \leq \sum_{i=1}^{m} x_i (\max_{k=1,\dots,m} \{[Ry]_k\})$$

Hence,

$$u_r \leq \max_{k=1,\dots,m} \{[Ry]_k\}$$

If $[Ry]_i = \max_{k=1,\dots,m}\{[Ry]_k\}$, $x'_i = 1$ and $x'_j = 0$ for all $j \neq i$, then $u_r \geq u_1(x', y) = \max_{k=1,\dots,m}\{[Ry]_k\}$.

(x' is a support of x and a BR to y) $\square$

Best response condition

Moreover, if x is a BR to y,

$$x_i > 0 \Rightarrow [Ry]_i = \max_{k=1,\dots,m} \{[Ry]_k\}$$

Assume that $\exists j, x_j > 0$ and

$$[Ry]_j < \max_{k=1,\dots,m} \{[Ry]_k\}. \text{Then,}$$

$$u_r = \sum_{x_i>0} x_i[Ry]_i < \sum_{x_i>0} x_i(\max_{k=1,\dots,m} \{[Ry]_k\}) = \max_{k=1,\dots,m} \{[Ry]_k\} \sum_{x_i>0} x_i = u_r$$

Contradiction!

ZS: algorithm for finding NE

- For a fixed y, we have

ZS: algorithm for finding NE

- For a fixed y, we have

$$\max_{x \in X} x^T R y = \max_{i=1,\ldots,m} \{[Ry]_i\},$$

ZS: algorithm for finding NE

- For a fixed y, we have

$$\max_{x \in X} x^T R y = \max_{i=1,\ldots,m} \{[Ry]_i\},$$

therefore

$$\min_{y \in Y} \max_{x \in X} x^T R y = \min_{y \in Y} \max\{[Ry]_1, \ldots [Ry]_m\}$$

ZS: algorithm for finding NE

- For a fixed y, we have

$$\max_{x \in X} x^T R y = \max_{i=1,\dots,m} \{[Ry]_i\},$$

therefore

$$\min_{y \in Y} \max_{x \in X} x^T R y = \min_{y \in Y} \max\{[Ry]_1, \dots [Ry]_m\}$$

- So, both the value of the game and a Nash equilibrium strategy for player 2 can be obtained by solving the linear programming problem:

ZS: algorithm for finding NE

- For a fixed y, we have

$$\max_{x \in X} x^T R y = \max_{i=1,\ldots,m} \{[Ry]_i\},$$

therefore

$$\min_{y \in Y} \max_{x \in X} x^T R y = \min_{y \in Y} \max\{[Ry]_1, \ldots [Ry]_m\}$$

- So, both the value of the game and a Nash equilibrium strategy for player 2 can be obtained by solving the linear programming problem:

$$\min v$$
$$v \mathbf{1}_n \geq Ry, y \in Y.$$

ZS: algorithm for finding NE

- Similarly, we have

$$\max_{x \in X} \min_{y \in Y} x^T R y = \max_{x \in X} \min\{[R^T x]_1, \ldots [R^T]_n\}$$

ZS: algorithm for finding NE

- Similarly, we have

$$\max_{x \in X} \min_{y \in Y} x^T R y = \max_{x \in X} \min\{[R^T x]_1, \ldots [R^T]_n\}$$

- So, a Nash equilibrium strategy for player 1 can be obtained by solving the linear programming problem:

ZS: algorithm for finding NE

- Similarly, we have

$$\max_{x \in X} \min_{y \in Y} x^T R y = \max_{x \in X} \min\{[R^T x]_1, \ldots [R^T]_n\}$$

- So, a Nash equilibrium strategy for player 1 can be obtained by solving the linear programming problem:

$$\max w$$

$$w \mathbf{1}_m \leq R^T x, x \in X.$$

ZS: algorithm for finding NE

- Similarly, we have

$$\max_{x \in X} \min_{y \in Y} x^T R y = \max_{x \in X} \min\{[R^T x]_1, \ldots [R^T]_n\}$$

- So, a Nash equilibrium strategy for player 1 can be obtained by solving the linear programming problem:

$$\max w$$

$$w \mathbf{1}_m \leq R^T x, x \in X.$$

- LP can be solved efficiently, thus there is a polynomial time algorithm for computing NE for zero-sum games.

PPAD

(Papadimitriou 94)
Polynomial Parity Argument on Directed Graphs

PPAD

(Papadimitriou 94)
Polynomial Parity Argument on Directed Graphs

- The class of all problems with guaranteed solution by use of the following graph-theoretic lemma

PPAD

(Papadimitriou 94)
Polynomial Parity Argument on Directed Graphs

- The class of all problems with guaranteed solution by use of the following graph-theoretic lemma

  A directed graph with an unbalanced node (node with in-degree $\neq$ outdegree) must have another.

PPAD

(Papadimitriou 94)
Polynomial Parity Argument on Directed Graphs

- The class of all problems with guaranteed solution by use of the following graph-theoretic lemma

  A directed graph with an unbalanced node (node with in-degree $\neq$ outdegree) must have another.

- Such problems are defined by a directed graph G represented implicitly and an unbalanced node u of G

PPAD

(Papadimitriou 94)
Polynomial Parity Argument on Directed Graphs

- The class of all problems with guaranteed solution by use of the following graph-theoretic lemma

  A directed graph with an unbalanced node (node with in-degree $\neq$ outdegree) must have another.

- Such problems are defined by a directed graph G represented implicitly and an unbalanced node u of G
  and the objective is finding another unbalanced node.

PPAD

(Papadimitriou 94)

Polynomial Parity Argument on Directed Graphs

- The class of all problems with guaranteed solution by use of the following graph-theoretic lemma

  A directed graph with an unbalanced node (node with in-degree $\neq$ outdegree) must have another.

- Such problems are defined by a directed graph G represented implicitly and an unbalanced node u of G
  and the objective is finding another unbalanced node.
- Usually G is huge but implicitly defined as the graphs defining solutions in local search algorithms.

PPAD

- The class PPAD contains interesting computational problems not known to be in P.

PPAD

- The class PPAD contains interesting computational problems not known to be in P.
  PPAD has complete problems.

PPAD

- The class PPAD contains interesting computational problems not known to be in P.
  PPAD has complete problems.
- But not a clear complexity cut.

P = NP implies P = PPAD

(Proof: PPAD is essentially a subset of NP, since a solution, such as a Nash equilibrium, can be certified quickly if found)

A PPAD-complete problem

End-of-Line

A PPAD-complete problem

End-of-Line

Given an implicit representation of a directed graph G with vertices of degree at most 2 and a vertex $v \in G$, where v has in degree 0. Find a node $v' \neq v$, such that v' has out degree 0.

A PPAD-complete problem

End-of-Line

Given an implicit representation of a directed graph G with vertices of degree at most 2 and a vertex $v \in G$, where v has in degree 0. Find a node $v' \neq v$, such that v' has out degree 0.

- Since every node has degree at most 2, it is a collection of paths and cycles.

A PPAD-complete problem

End-of-Line
Given an implicit representation of a directed graph G with vertices of degree at most 2 and a vertex $v \in G$, where v has in degree 0. Find a node $v' \neq v$, such that v' has out degree 0.

- Since every node has degree at most 2, it is a collection of paths and cycles.
- We know that
  Every directed graph with in/outdegree ≤ 1 nodes and a source, has a sink.

A PPAD-complete problem

End-of-Line
Given an implicit representation of a directed graph G with vertices of degree at most 2 and a vertex $v \in G$, where v has in degree 0. Find a node $v' \neq v$, such that v' has out degree 0.

- Since every node has degree at most 2, it is a collection of paths and cycles.
- We know that
  Every directed graph with in/outdegree ≤ 1 nodes and a source, has a sink.
- Which guarantees that
  the End-of-Line problem has always a solution.

End-of-Line: graph representation

- G is given implicitly by a circuit C
- C provides a predecessor and successor pair for each given vertex in G, i.e. $C(u) = (v, w)$.
- A special label indicates that a node has no predecessor/successor.

The complexity of finding a NE

Theorem (Daskalakis, Goldberg, Papadimitriou '06)

Finding a Nash equilibrium is PPAD-complete

The complexity of finding a NE

Theorem (Daskalakis, Goldberg, Papadimitriou '06)

Finding a Nash equilibrium is PPAD-complete

Theorem (Chen, Deng '06)

Finding a Nash equilibrium is PPAD-complete even in 2-player games.

The complexity of finding a NE

Theorem (Daskalakis, Goldberg, Papadimitriou '06)

Finding a Nash equilibrium is PPAD-complete

Theorem (Chen, Deng '06)

Finding a Nash equilibrium is PPAD-complete even in 2-player games.

- C. Daskalakis, P-W. Goldberg, C.H. Papadimitriou: The complexity of computing a Nash equilibrium. SIAM J. Comput. 39(1): 195-259 (2009) first version STOC 2006
- X. Chen, X. Deng, S-H. Teng: Settling the complexity of computing two-player Nash equilibria. J. ACM 56(3) (2009) first version FOCS 2006

1. Linear Algebra formulation
2. Zero-sum games
3. The complexity of finding a NE
4. An exact algorithm to compute NE
5. Other algorithms

NE characterization

Theorem

In a strategic game in which each player has finitely many actions a mixed strategy profile σ^* is a NE iff, for each player i,

- *the expected payoff, given σ_{-i}, to every action in the support of σ_i^* is the same*
- *the expected payoff, given σ_{-i}, to every action not in the support of σ_i^* is at most the expected payoff on an action in the support of σ_i^*.*

NE conditions given support

Let $A \subseteq \{1, \ldots m\}$ and $B \subseteq \{1, \ldots n\}$.
The conditions for having a NE on this particular support can be written as follows:

$$\max \lambda_1 + \lambda_2$$

Subject to:

$$\begin{aligned}
[R\,y]_i &= \lambda_1, \text{ for } i \in A \\
[R\,y]_i &\leq \lambda_1, \text{ for } i \notin A \\
[x^T\,C]_j &= \lambda_2, \text{ for } j \in B \\
[x^T\,C]_j &\leq \lambda_2, \text{ for } j \notin B
\end{aligned}$$

Iterating over all supports

- For every possible combination of supports $A \subseteq \{1, \ldots, m\}$ and $B \subseteq \{1, \ldots, n\}$.
  Solve the set of simultaneous equations using linear programming.

Iterating over all supports

- For every possible combination of supports $A \subseteq \{1, \ldots, m\}$ and $B \subseteq \{1, \ldots, n\}$.
  Solve the set of simultaneous equations using linear programming.
- This is an exact exponential time algorithm as the number of supports can be exponential.

Iterating over all supports

- For every possible combination of supports $A \subseteq \{1, \ldots, m\}$ and $B \subseteq \{1, \ldots, n\}$.
  Solve the set of simultaneous equations using linear programming.
- This is an exact exponential time algorithm as the number of supports can be exponential.
- The same algorithm can be applied to a multiplayer game. We would be able to compute a NE on rationals if such a NE exists.

Other algorithms

- Lemke-Howson (1964) algorithm defines a polytope based on best response conditions and membership to the support and uses ideas similar to Simplex with a ad-hoc pivoting rule.
- Lemke-Howson requires exponential time [R. Savani, B. von Stengel, 2004]).