Equilibria, Fixed Points and Complexity Classes: Finding Needles in Haystacks

Vasileios-Orestis Papadigenopoulos

School of Electrical and Computer Engineering - N.T.U.A

papadigenopoulos_orestis@yahoo.gr

January 25, 2016
Overview

1 Introduction
   - A Warm Up
   - The Class TFNP
   - Equilibria, Fixed Points and Complexity Classes

2 The Class PLS

3 The Class PPAD
   - Motivating PPAD

4 Irrational Equilibria, Nonlinear Functions, and the Class FIXP
Recall:

**Definition (Class FNP)**

A binary relation $P(x,y)$, where $y$ is at most polynomially longer than $x$, is in FNP if and only if there is a deterministic polynomial time algorithm that can determine whether $P(x,y)$ holds given both $x$ and $y$. 

**Definition (Class FP)**

A binary relation $P(x,y)$ is in FP if and only if there is a deterministic polynomial time algorithm that, given $x$, can find some $y$ such that $P(x,y)$ holds.
Recall:

**Definition (Class FNP)**

A binary relation $P(x,y)$, where $y$ is at most polynomially longer than $x$, is in FNP if and only if there is a deterministic polynomial time algorithm that can determine whether $P(x,y)$ holds given both $x$ and $y$.

**Definition (Class FP)**

A binary relation $P(x,y)$ is in FP if and only if there is a deterministic polynomial time algorithm that, given $x$, can find some $y$ such that $P(x,y)$ holds.
A Warm Up

Recall:

**Definition (Class FNP)**

A binary relation $P(x,y)$, where $y$ is at most polynomially longer than $x$, is in FNP if and only if there is a deterministic polynomial time algorithm that can determine whether $P(x,y)$ holds given both $x$ and $y$.

**Definition (Class FP)**

A binary relation $P(x,y)$ is in FP if and only if there is a deterministic polynomial time algorithm that, given $x$, can find some $y$ such that $P(x,y)$ holds.

$FP = FNP$ if and only if $P = NP$
A Warm Up

So far so good but what can we say about the existence of a solution?
"One way to convey the difficulty of an NP-complete problem to a non-technical audience is to say that solving it is like trying to find a needle in a haystack."

A Warm Up

We would like to "find the needle" or tell whether one exists?
We would like to "find the needle" or tell whether one exists?

There are problems where one of the two tasks (existence testing, object finding) can be performed in polynomial time but the other cannot.
A Warm Up

- We would like to ”find the needle” or tell whether one exists?
- There are problems where one of the two tasks (existence testing, object finding) can be performed in polynomial time but the other cannot.
- Obviously, telling whether a needle exists cannot be harder than finding it.
A Warm Up

- We would like to ”find the needle” or tell whether one exists?
- There are problems where one of the two tasks (existence testing, object finding) can be performed in polynomial time but the other cannot.
- Obviously, telling whether a needle exists cannot be harder than finding it.
- So the major question is whether object finding can be harder than existence testing.
For NP-complete problems the answer is **NO**
A Warm Up

- For NP-complete problems the answer is NO
- Suppose an NP-complete existence question X can be solved in polynomial time. Then all problems in NP can be solved in polynomial time.
For NP-complete problems the answer is NO

Suppose an NP-complete existence question X can be solved in polynomial time. Then all problems in NP can be solved in polynomial time.

The same holds for the problem X’ that asks, given an instance I of X and a string y, whether there is an object of the type desired by X whose description has y as a prefix.
For NP-complete problems the answer is NO.

Suppose an NP-complete existence question $X$ can be solved in polynomial time. Then all problems in NP can be solved in polynomial time.

The same holds for the problem $X'$ that asks, given an instance $I$ of $X$ and a string $y$, whether there is an object of the type desired by $X$ whose description has $y$ as a prefix.

Given this, if the desired object exists, we can construct its description with no more than $p(|I|)$ calls to the polynomial-time algorithm for solving the existence problem $X'$. 
What about non-NP-complete problems in NP for which existence testing can be accomplished in polynomial time but finding a solution cannot?
The class TFNP

- A search problem $\Pi$ has a set of instances, and each instance $I$ has a set $Sol(I)$ of acceptable answers.
The class TFNP

- A search problem Π has a set of instances, and each instance I has a set \( \text{Sol}(I) \) of acceptable answers.
- The search problem is total if \( \text{Sol}(I) \neq \emptyset \) for all instances I.
The class TFNP

- A search problem $\Pi$ has a set of instances, and each instance $I$ has a set $Sol(I)$ of acceptable answers.
- The search problem is total if $Sol(I) \neq \emptyset$ for all instances $I$.
- Given an instance of total search problem, compute any acceptable solution.
Definition (Total Function Nondeterministic Polynomial)

A binary relation \( P(x,y) \) is in TFNP if and only if there is a deterministic polynomial time algorithm that can determine whether \( P(x,y) \) holds given both \( x \) and \( y \), and for every \( x \), there exists a \( y \) such that \( P(x,y) \) holds.
The class TFNP

- FP is a subclass of TFNP.
The class TFNP

- FP is a subclass of TFNP.
- The underlying space of solutions may be:
  - *Finite and Discrete* (As in combinational problems): Solutions are represented as strings.
The class TFNP

- FP is a subclass of TFNP.
- The underlying space of solutions may be:
  - *Finite and Discrete* (As in combinational problems): Solutions are represented as strings.
  - *Infinite and Continuous* (As in most FP-LP problems): Solutions are represented as real-valued vectors of dimension $d$, where $d$ is polynomial in the input size.
The class TFNP

- FP is a subclass of TFNP.
- The underlying space of solutions may be:
  - *Finite and Discrete* (As in combinational problems): Solutions are represented as strings.
  - *Infinite and Continuous* (As in most FP-LP problems): Solutions are represented as real-valued vectors of dimension $d$, where $d$ is polynomial in the input size.
- Note that, in many problems, the solutions are inherently irrational.
Many situations involve the computation of an equilibrium or a stable configuration of some sort in a dynamic environment.
Equilibria, Fixed Points and Complexity Classes

- Many situations involve the computation of an equilibrium or a stable configuration of some sort in a dynamic environment.
- Often the sought objects can be described mathematically as the fixed points of an equation $x = F(x)$. 
Many situations involve the computation of an equilibrium or a stable configuration of some sort in a dynamic environment.

Often the sought objects can be described mathematically as the fixed points of an equation $x = F(x)$.

Examples: Nash equilibria in games, market equilibria, neural network stable configuration etc.

For most of these problems it is not known whether they can be solved in polynomial time.
Many situations involve the computation of an equilibrium or a stable configuration of some sort in a dynamic environment.

Often the sought objects can be described mathematically as the fixed points of an equation $x = F(x)$.

Examples: Nash equilibria in games, market equilibria, neural network stable configuration etc.

For most of these problems it is not known whether they can be solved in polynomial time.

There are certain common computational principles that underlie many of these different types of problems, which are captured by the complexity classes PLS, PPAD, and FIXP.
Nash equilibrium

Definition

Let \((S, u)\) be a game with \(n\) players, where \(S_i\) is the strategy set for player \(i\), \(S = S_1 \times S_2 \times \ldots \times S_n\) in the set of strategy profiles and \(u = (u_1(x), \ldots, u_n(x))\) is the payoff (utility) function for \(x \in S\). Let \(x_i\) be a strategy profile of player \(i\) and \(x_{-i}\) be a strategy profile of all players except for player \(i\). When each player \(i \in \{1, \ldots, n\}\) chooses strategy \(x_i\) resulting in strategy profile \(x = (x_1, \ldots, x_n)\) then player \(i\) obtains payoff \(f_i(x)\). A strategy profile \(x^* \in S\) is a Nash equilibrium (NE) if no unilateral deviation in strategy by any single player is profitable for that player, that is that is \(\forall i, x_i \in S_i : u_i(x_i^*, x_{-i}^*) \geq u_i(x_i, x_{-i}^*)\).
A game can have a pure-strategy or a mixed Nash Equilibrium. (In the latter a pure strategy is chosen stochastically with a fixed probability).
Nash equilibrium

- A game can have a *pure*-strategy or a *mixed* Nash Equilibrium. (In the latter a pure strategy is chosen stochastically with a fixed probability).
- Nash’s theorem suggests that if we allow mixed strategies, then every game with a finite number of players in which each player can choose from finitely many pure strategies has at least one Nash equilibrium.
A game can have a pure-strategy or a mixed Nash Equilibrium. (In the latter a pure strategy is chosen stochastically with a fixed probability).

Nash’s theorem suggests that if we allow mixed strategies, then every game with a finite number of players in which each player can choose from finitely many pure strategies has at least one Nash equilibrium.

Note that the nature of the argument is existential.
A game can have a \textit{pure}-strategy or a \textit{mixed} Nash Equilibrium. (In the latter a pure strategy is chosen stochastically with a fixed probability).

Nash’s theorem suggests that if we allow mixed strategies, then every game with a finite number of players in which each player can choose from finitely many pure strategies \textit{has at least one Nash equilibrium}.

Note that the nature of the argument is existential.

Also note that this does not hold in general for pure strategies.
Consider the following *neural network* model:

- Undirected graph $G = (V, E)$, with a positive or negative weight $w(e), e \in E$ ($w(u, v) = 0$ if we have no edge).
Consider the following *neural network* model:

- Undirected graph $G = (V, E)$, with a positive or negative weight $w(e), e \in E$ ($w(u, v) = 0$ if we have no edge).
- $\forall v \in V$, we have a threshold $t(v), v \in V$. 

A configuration of the network is an assignment of the state $s(v) = +1$ ("on"), or $s(v) = -1$ ("off") to each node $v$. A node $v$ is stable ("happy") if $s(v) = 1$ and $\sum u w(v, u) s(u) + t(v) \geq 0$, or $s(v) = -1$ and $\sum u w(v, u) s(u) + t(v) \leq 0$. i.e. the state of $v$ agrees with the sign of the weighted sum of its neighbors plus the threshold. A configuration is stable if all the nodes are stable.

Problem: Find a stable configuration!
Consider the following *neural network* model:

- Undirected graph $G = (V, E)$, with a positive or negative weight $w(e), e \in E$ ($w(u, v) = 0$ if we have no edge).
- $\forall v \in V$, we have a threshold $t(v), v \in V$.
- A configuration of the network is an assignment of the state $s(v) = +1$ ("on"), or $s(v) = -1$ ("off") to each node $v$. 

A node $v$ is stable ("happy") if $s(v) = 1$ and $\sum_{u} w(v, u) s(u) + t(v) \geq 0$, or $s(v) = -1$ and $\sum_{u} w(v, u) s(u) + t(v) \leq 0$. i.e. the state of $v$ agrees with the sign of the weighted sum of its neighbors plus the threshold.

A configuration is stable if all the nodes are stable.

Problem: Find a stable configuration!
Consider the following neural network model:

- Undirected graph $G = (V, E)$, with a positive or negative weight $w(e), e \in E$ ($w(u, v) = 0$ if we have no edge).
- $\forall v \in V$, we have a threshold $t(v), v \in V$.
- A configuration of the network is an assignment of the state $s(v) = +1$ ("on"), or $s(v) = -1$ ("off") to each node $v$.
- A node $v$ is stable ("happy") if $s(v) = 1$ and $\sum_u w(v, u)s(u) + t(v) \geq 0$, or $s(v) = -1$ and $\sum_u w(v, u)s(u) + t(v) \leq 0$. i.e. the state of $v$ agrees with the sign of the weighted sum of its neighbors plus the threshold.
Discrete, Pure Equilibria and the Class PLS

Consider the following *neural network* model:

- Undirected graph $G = (V, E)$, with a positive or negative weight $w(e), e \in E$ ($w(u, v) = 0$ if we have no edge).
- $\forall v \in V$, we have a threshold $t(v), v \in V$.
- A configuration of the network is an assignment of the state $s(v) = +1$ ("on"), or $s(v) = -1$ ("off") to each node $v$.
- A node $v$ is stable ("happy") if $s(v) = 1$ and $\sum_u w(v, u)s(u) + t(v) \geq 0$, or $s(v) = -1$ and $\sum_u w(v, u)s(u) + t(v) \leq 0$. i.e. the state of $v$ agrees with the sign of the weighted sum of its neighbors plus the threshold.
- A configuration is *stable* if all the nodes are stable.
Discrete, Pure Equilibria and the Class PLS

Consider the following *neural network* model:

- Undirected graph $G = (V, E)$, with a positive or negative weight $w(e), e \in E$ ($w(u, v) = 0$ if we have no edge).
- $\forall v \in V$, we have a threshold $t(v), v \in V$.
- A configuration of the network is an assignment of the state $s(v) = +1$ ("on"), or $s(v) = -1$ ("off") to each node $v$.
- A node $v$ is stable ("happy") if $s(v) = 1$ and $\sum_u w(v, u)s(u) + t(v) \geq 0$, or $s(v) = -1$ and $\sum_u w(v, u)s(u) + t(v) \leq 0$. i.e. the state of $v$ agrees with the sign of the weighted sum of its neighbors plus the threshold.
- A configuration is *stable* if all the nodes are stable.
- **Problem**: Find a stable configuration!
Finding a stable configuration...

- How do we know whether one exists?
Finding a stable configuration...

- How do we know whether one exists?
- For directed networks there may not exist any.
Finding a stable configuration...

- How do we know whether one exists?
- For directed networks there may not exist any.
- For every undirected network there exists at least one stable configuration!!!
Finding a stable configuration...

- How do we know whether one exists?
- For directed networks there may not exist any.
- For every undirected network there exists at least one stable configuration!!!

- A dynamic process where in each step one node that is unstable switches its state, is guaranteed to eventually converge in a finite number of steps to a stable configuration, no matter which unstable node is switched in each step (one node at a time).
For every undirected network there exists at least one stable configuration...
For every undirected network there exists at least one stable configuration...

- Why? [J.J.Hopfield] Potential Function for a configuration $s$:
  $$p(s) = \sum_{(v,u) \in E} w(v, u)s(v)s(u) + \sum_{v \in V} t(v)s(v).$$
For every undirected network there exists at least one stable configuration...

- Why? [J.J.Hopfield] Potential Function for a configuration $s$:
  \[ p(s) = \sum_{(v,u) \in E} w(v, u)s(v)s(u) + \sum_{v \in V} t(v)s(v). \]
- If $v$ is an unstable node in configuration $s$, then switching its state results in a configuration $s'$ with strictly higher value.
- \[ p(s') = p(s) + 2|\sum_u w(v, u)s(u) + t(v)| > p(s) \]
Discrete, Pure Equilibria and the Class PLS

For every undirected network there exists at least one stable configuration...

- Why? [J.J.Hopfield] Potential Function for a configuration $s$: 
  \[ p(s) = \sum_{(v,u) \in E} w(v, u)s(v)s(u) + \sum_{v \in V} t(v)s(v). \]
- If $v$ is an unstable node in configuration $s$, then switching its state results in a configuration $s'$ with strictly higher value.
- \[ p(s') = p(s) + 2|\sum_u w(v, u)s(u) + t(v)| > p(s) \]
- Finite number of configurations $2^{|V|}$. 
Discrete, Pure Equilibria and the Class PLS

For every undirected network there exists at least one stable configuration...

- Why? [J.J.Hopfield] Potential Function for a configuration $s$:
  \[ p(s) = \sum_{(v,u) \in E} w(v, u)s(v)s(u) + \sum_{v \in V} t(v)s(v). \]
- If $v$ is an unstable node in configuration $s$, then switching its state results in a configuration $s'$ with strictly higher value.
  \[ p(s') = p(s) + 2|\sum_u w(v, u)s(u) + t(v)| > p(s) \]
- Finite number of configurations $2^{|V|}$.
- The process has to converge to a stable configuration in a finite number of steps.
The Stable Configuration problem is a search problem.
The Stable Configuration problem is a search problem.

Since a stable solution is guaranteed to exist it belongs to the class TFNP.
The Stable Configuration problem is a search problem.

Since a stable solution is guaranteed to exist it belongs to the class TFNP.

Attention: Computing a configuration with max value (w.r.t a metric) is a NP-hard problem. We just search for any solution (easier problem).
The Stable Configuration problem is a search problem.

Since a stable solution is guaranteed to exist it belongs to the class TFNP.

Attention: Computing a configuration with max value (w.r.t a metric) is a NP-hard problem. We just search for any solution (easier problem).

In a Combinational Optimization problem, every instance I has a finite set $S(I)$ of candidate solutions, and each configuration $s \in S(I)$ has a rational values $p_I(s)$ to be minimized/maximized.
The Stable Configuration problem is a search problem.

Since a stable solution is guaranteed to exist it belongs to the class TFNP.

Attention: Computing a configuration with max value (w.r.t a metric) is a NP-hard problem. We just search for any solution (easier problem).

In a Combinational Optimization problem, every instance \( I \) has a finite set \( S(I) \) of candidate solutions, and each configuration \( s \in S(I) \) has a rational values \( p_I(s) \) to be minimized/maximized.

In Local Search, each solution has in addition an associated neighborhood \( N_I(s) \subseteq S(I) \). A solution is locally optimal if doesn’t have strictly better neighbor.
The class PLS (Polynomial Local Search) captures the difficulty of those types of problems.

**Definition**

A problem \( \Pi \) is in PLS if solutions (represented by strings) are polynomially bounded on the input size, and there are polynomial-time algorithms for the following:

1. Given string \( I \), test whether \( I \) is an instance of \( \Pi \), and if so compute an initial solution in \( S(I) \).
2. Given \( I, s \), test whether \( s \in S(I) \), and if so compute its value \( p_I(s) \).
3. Given \( I, s \), test whether \( s \) is a local optimum, and if not, compute a better neighbor \( s' \in N_I(s) \).
The class PLS (Polynomial Local Search) captures the difficulty of those types of problems.

**Definition**

A problem $\Pi$ is in PLS if solutions (represented by strings) are polynomially bounded on the input size, and there are polynomial-time algorithms for the following:

- Given string $I$, test whether $I$ is an instance of $\Pi$, and if so compute an initial solution in $S(I)$.
The class PLS (Polynomial Local Search) captures the difficulty of those types of problems.

**Definition**

A problem $\Pi$ is in PLS if solutions (represented by strings) are polynomially bounded on the input size, and there are polynomial-time algorithms for the following:

- Given string $I$, test whether $I$ is an instance of $\Pi$, and if so compute an initial solution in $S(I)$
- Given $I,s$, test whether $s \in S(I)$, and if so compute its value $p_I(s)$
Discrete, Pure Equilibria and the Class PLS

The class PLS (Polynomial Local Search) captures the difficulty of those types of problems.

**Definition**

A problem $\Pi$ is in PLS if solutions (represented by strings) are polynomially bounded on the input size, and there are polynomial-time algorithms for the following:

- Given string $I$, test whether $I$ is an instance of $\Pi$, and if so compute an initial solution in $S(I)$
- Given $I,s$, test whether $s \in S(I)$, and if so compute its value $p_I(s)$
- Given $I,s$, test whether $s$ is a local optimum, and if not, compute a better neighbor $s \in N_I(s)$.
Discrete, Pure Equilibria and the Class PLS

Comments.

- Even each step takes polynomial time, the number of steps can be potentially exponential.
Discrete, Pure Equilibria and the Class PLS

Comments.

- Even each step takes polynomial time, the number of steps can be potentially exponential.
- A PLS-Reduction from problem A to problem B is a polynomial-time reduction between discrete search problems (finite solution spaces).
Discrete, Pure Equilibria and the Class PLS

Comments.

- Even each step takes polynomial time, the number of steps can be potentially exponential.
- A PLS-Reduction from problem A to problem B is a polynomial-time reduction between discrete search problems (finite solution spaces).
- Tight PLS-reduction, allows us to conclude that the corresponding standard local search algorithm is exponential.
Comments.

- Even each step takes polynomial time, the number of steps can be potentially exponential.
- A PLS-Reduction from problem A to problem B is a polynomial-time reduction between discrete search problems (finite solution spaces).
- Tight PLS-reduction, allows us to conclude that the corresponding standard local search algorithm is exponential.
- A number of combinational optimization problems are shown to be PLS-Complete, such as: GRAPH PARTITIONING, TSP, MAX CUT, MAX SAT etc.
Discrete, Pure Equilibria and the Class PLS

Comments.

- Even each step takes polynomial time, the number of steps can be potentially exponential.
- A PLS-Reduction from problem A to problem B is a polynomial-time reduction between discrete search problems (finite solution spaces).
- Tight PLS-reduction, allows us to conclude that the corresponding standard local search algorithm is exponential.
- A number of combinational optimization problems are shown to be PLS-Complete, such as: GRAPH PARTITIONING, TSP, MAX CUT, MAX SAT etc.
- $P \subseteq PLS \subseteq TFNP$. 
Discrete, Pure Equilibria and the Class PLS

Comments.

- Even each step takes polynomial time, the number of steps can be potentially exponential.
- A PLS-Reduction from problem A to problem B is a polynomial-time reduction between discrete search problems (finite solution spaces).
- Tight PLS-reduction, allows us to conclude that the corresponding standard local search algorithm is exponential.
- A number of combinational optimization problems are shown to be PLS-Complete, such as: GRAPH PARTITIONING, TSP, MAX CUT, MAX SAT etc.
- \( P \subseteq PLS \subseteq TFNP \).
- The matter of interest is the inherent complexity of a search problem (by any algorithm). For example (the number of steps executing Simplex can be exponential. However, \( LINEARPROGRAMMING \in P \)).
Also:

- Notice that a solution to the Stable Configuration problem is in fact a Nash equilibrium to a finite game of $|V|$ players where the two states of each player (+1, -1) corresponds to its pure strategies.
Also:

- Notice that a solution to the Stable Configuration problem is in fact a Nash equilibrium to a finite game of $|V|$ players where the two states of each player (+1, -1) corresponds to its pure strategies.

- Not surprisingly, finding pure Nash equilibria for games where they are guaranteed to exist is also in PLS.
Also:

- Notice that a solution to the Stable Configuration problem is in fact a Nash equilibrium to a finite game of $|V|$ players where the two states of each player ($+1$, $-1$) corresponds to its pure strategies.
- Not surprisingly, finding pure Nash equilibria for games where they are guaranteed to exist is also in PLS.
- The class of *Congestion Games* is PLS-Complete.
Also:

- Notice that a solution to the Stable Configuration problem is in fact a Nash equilibrium to a finite game of $|V|$ players where the two states of each player (+1, -1) corresponds to its pure strategies.

- Not surprisingly, finding pure Nash equilibria for games where they are guaranteed to exist is also in PLS.

- The class of Congestion Games is PLS-Complete.

- Dynamic games played iteratively over time (chess, backgammon) are in PLS but not known to be in P.
Rational Equilibria and the class PPAD

Definition

A search problem $\Pi$ is in PPAD if each instance $I$ has a set $S(I)$ of candidate solutions which are (strings) polynomially bounded in the input size $|I|$, and there are polynomial-time algorithms for the following tasks:
Rational Equilibria and the class PPAD

Definition

A search problem $\Pi$ is in PPAD if each instance $I$ has a set $S(I)$ of candidate solutions which are (strings) polynomially bounded in the input size $|I|$, and there are polynomial-time algorithms for the following tasks:

- Test whether a given string $I$ is an instance of $\Pi$ and if so, compute an initial candidate solution $s_0$ in $S(I)$. 
- Given $I$, test whether $s \in S(I)$ and if so, compute a successor $\text{succ}_I(s) \in S(I)$ and a predecessor $\text{pred}_I(s) \in S(I)$, such that $\text{pred}_I(s_0) = s_0$, $\text{succ}_I(s_0) \neq s_0$ and $\text{pred}_I(\text{succ}_I(s_0)) = s_0$. 

The $\text{pred}$ and $\text{succ}$ functions induce a directed graph $G = (S(I), E)$, where:

$$E = \{(u, v) \mid u = v, \text{succ}_I(u) = v, \text{pred}_I(v) = u\}$$
Rational Equilibria and the class PPAD

Definition

A search problem $\Pi$ is in PPAD if each instance $I$ has a set $S(I)$ of candidate solutions which are (strings) polynomially bounded in the input size $|I|$, and there are polynomial-time algorithms for the following tasks:

- Test whether a given string $I$ is an instance of $\Pi$ and if so, compute an initial candidate solution $s_0$ in $S(I)$.
- Given $I$, $s$ test whether $s \in S(I)$ and if so, compute a successor $\text{succ}_I(s) \in S(I)$ and a predecessor $\text{pred}_I(s) \in S(I)$, such that $\text{pred}_I(s_0) = s_0$, $\text{succ}_I(s_0) \neq s_0$ and $\text{pred}_I(\text{succ}_I(s_0)) = s_0$. 

The pred and succ functions induce a directed graph $G = (S(I), E)$, where:

$$E = \{ (u, v) | u = v, \text{succ}_I(u) = v, \text{pred}_I(v) = u \}.$$
Rational Equilibria and the class PPAD

Definition

A search problem $\Pi$ is in PPAD if each instance $I$ has a set $S(I)$ of candidate solutions which are (strings) polynomially bounded in the input size $|I|$, and there are polynomial-time algorithms for the following tasks:

- Test whether a given string $I$ is an instance of $\Pi$ and if so, compute an initial candidate solution $s_0$ in $S(I)$.
- Given $I, s$ test whether $s \in S(I)$ and if so, compute a successor $\text{succ}_I(s) \in S(I)$ and a predecessor $\text{pred}_I(s) \in S(I)$, such that $\text{pred}_I(s_0) = s_0$, $\text{succ}_I(s_0) \neq s_0$ and $\text{pred}_I(\text{succ}_I(s_0)) = s_0$.

The pred and succ functions induce a directed graph $G = (S(I), E)$, where: $E = \{(u, v) | u = v, \text{succ}_I(u) = v, \text{pred}_I(v) = u\}$
Rational Equilibria and the class PPAD
Brouwer’s Fixed Point Theorem

Theorem

Let $f : D \rightarrow D$ be a continuous function from a convex and compact subset $D$ of the Euclidean space to itself. Then there exists a $x \in D$ s.t. $x = f(x)$. This $x$ is called a fixed point.
Brouwer’s Fixed Point Theorem

Theorem

Let \( f : D \rightarrow D \) be a continuous function from a convex and compact subset \( D \) of the Euclidean space to itself. Then there exists a \( x \in D \) s.t. \( x = f(x) \). This \( x \) is called a fixed point.

Example

\( D \) is the 2-dimensional disk:
Brouwer’s Fixed Point Theorem

Rotation:

fixed point
Brouwer’s Fixed Point Theorem

Shrink and move within the boundaries of D:
Brouwer’s Fixed Point Theorem

Shrink, distort and move within the boundaries of D:
Brouwer’s FPT $\rightarrow$ Nash equilibrium

Here is a simple game:

<table>
<thead>
<tr>
<th>Kick</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dive</td>
<td>1, -1</td>
<td>-1, 1</td>
</tr>
<tr>
<td>Left</td>
<td>-1, 1</td>
<td>1, -1</td>
</tr>
<tr>
<td>Right</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that for a 2-player zero-sum game the existence of an equilibrium was shown by von Neumann. Existence of Eq in 2-player zero-sum games is equivalent to Strong LP Duality.
Brouwer’s FPT → Nash equilibrium

<table>
<thead>
<tr>
<th>Kick</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>1, -1</td>
<td>-1, 1</td>
</tr>
<tr>
<td>Right</td>
<td>-1, 1</td>
<td>1, -1</td>
</tr>
</tbody>
</table>

$f: [0,1]^2 \rightarrow [0,1]^2$, continuous such that fixed points $\equiv$ Nash eq.
Brouwer’s FPT $\rightarrow$ Nash equilibrium

Penalty Shot Game

<table>
<thead>
<tr>
<th>Kick</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dive</td>
<td>1, -1</td>
<td>-1, 1</td>
</tr>
<tr>
<td>Left</td>
<td>-1, 1</td>
<td>1, -1</td>
</tr>
</tbody>
</table>
Brouwer's FPT → Nash equilibrium

<table>
<thead>
<tr>
<th>Kick Dive</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>1, -1</td>
<td>-1, 1</td>
</tr>
<tr>
<td>Right</td>
<td>-1, 1</td>
<td>1, -1</td>
</tr>
</tbody>
</table>

Penalty Shot Game
**Brouwer’s FPT → Nash equilibrium**

<table>
<thead>
<tr>
<th>Kick Dive</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>1, -1</td>
<td>-1, 1</td>
</tr>
<tr>
<td>Right</td>
<td>-1, 1</td>
<td>1, -1</td>
</tr>
</tbody>
</table>

**Penalty Shot Game**

![Diagram showing the relationship between FPT and Nash equilibrium.](image-url)
Brouwer’s FPT $\rightarrow$ Nash equilibrium
Sperner’s Lemma (2-D)

Suppose we have a grid made of triangles.
Sperner’s Lemma (2-D)

Every node can have any color from red, yellow or blue except for the boundaries:
Sperner’s Lemma (2-D)

Every node can have any color from red, yellow or blue except for the boundaries:
Sperner’s Lemma (2-D)

[Sperner 1928]: If the boundary is legally colored (and regardless how the internal nodes are colored), there exists a tri-chromatic triangle. In fact, an odd number of them.
Sperner’s Lemma $\rightarrow$ Brouwer’s FPT

Given $f : [0, 1]^2 \rightarrow [0, 1]^2$: 
Given $f : [0, 1]^2 \to [0, 1]^2$:
1. For all $\epsilon$, existence of approximate fixed point $|f(x) - x| < \epsilon$, can be shown via Sperner’s lemma.
Given $f : [0, 1]^2 \rightarrow [0, 1]^2$:
1. For all $\epsilon$, existence of approximate fixed point $|f(x) - x| < \epsilon$, can be shown via Sperners lemma.
2. Then use compactness.
Sperner’s Lemma → Brouwer’s FPT

Triangulate $[0, 1]^2$ with the diameter of every triangle small enough w.r.t $\epsilon$. Also, add one extra layer of nodes outside the boundaries.
Sperner’s Lemma $\rightarrow$ Brouwer’s FPT

Color points according to the direction of $f(x) - x$. 
Sperner’s Lemma $\rightarrow$ Brouwer’s FPT

Color points according to the direction of $f(x) - x$. 
Computational Version of SPERNER

INPUT:

- Grid of size $2^n$.
- Suppose boundary has standard coloring, and colors of internal vertices are given by a circuit.
INPUT:
- Grid of size $2^n$.
- Suppose boundary has standard coloring, and colors of internal vertices are given by a circuit.

OUTPUT: A tri-chromatic triangle
Computational Version of NASH

INPUT:

- A Game defined by:
  - The number of players $n$
  - An enumeration of the strategy set $S$ of every player $p$, $p = 1, \ldots, n$
  - The utility function $u_p : S \rightarrow \mathbb{R}$ of every player.

OUTPUT: An $\epsilon$-Nash equilibrium of the game, i.e. the expected payoff of every player is within additive $\epsilon$ from the optimal expected payoff given the others' strategies.

2-player Games: 2-player games always have a rational equilibrium of polynomial description complexity in the size of the game.

Approximation: Already in 1951, Nash provides a 3-player game whose unique equilibrium is irrational.
Computational Version of NASH

INPUT:

- A Game defined by:
  - The number of players $n$
Computational Version of NASH

INPUT:
- A Game defined by:
  - The number of players $n$
  - An enumeration of the strategy set $S_p$ of every player $p = 1 \ldots n$
Computational Version of NASH

INPUT:

- A Game defined by:
  - The number of players \( n \)
  - An enumeration of the strategy set \( S_p \) of every player \( p = 1 \ldots n \)
  - The utility function \( u_p : S \rightarrow R \) of every player.
Computational Version of NASH

INPUT:
- A Game defined by:
  - The number of players $n$
  - An enumeration of the strategy set $S_p$ of every player $p = 1 \ldots n$
  - The utility function $u_p : S \rightarrow R$ of every player.
- An approximation requirement $\epsilon$
Computational Version of NASH

INPUT:
- A Game defined by:
  - The number of players $n$
  - An enumeration of the strategy set $S_p$ of every player $p = 1 \ldots n$
  - The utility function $u_p : S \rightarrow R$ of every player.
- An approximation requirement $\epsilon$

OUTPUT: An $\epsilon$-Nash equilibrium of the game, i.e. the expected payoff of every player is within additive $\epsilon$ from the optimal expected payoff given the others’ strategies.
Computational Version of NASH

INPUT:
- A Game defined by:
  - The number of players $n$
  - An enumeration of the strategy set $S_p$ of every player $p = 1 \ldots n$
  - The utility function $u_p : S \rightarrow R$ of every player.
- An approximation requirement $\epsilon$

OUTPUT: An $\epsilon$-Nash equilibrium of the game, i.e. the expected payoff of every player is within additive $\epsilon$ from the optimal expected payoff given the others’ strategies.

- 2-player Games: 2-player games always have a rational equilibrium of polynomial description complexity in the size of the game.
Computational Version of NASH

INPUT:
- A Game defined by:
  - The number of players $n$
  - An enumeration of the strategy set $S_p$ of every player $p = 1 \ldots n$
  - The utility function $u_p : S \rightarrow R$ of every player.
- An approximation requirement $\epsilon$

OUTPUT: An $\epsilon$-Nash equilibrium of the game, i.e., the expected payoff of every player is within additive $\epsilon$ from the optimal expected payoff given the others’ strategies.

- **2-player Games**: 2-player games always have a rational equilibrium of polynomial description complexity in the size of the game.
- **Approximation**: Already in 1951, Nash provides a 3-player game whose unique equilibrium is irrational.
The Class PPAD

Motivating PPAD

Is NP-Completeness Useful?

Remember: $TFNP = \{ L \in FNP | L \text{ is total} \}$.
Remember: \( TFNP = \{ L \in FNP | L \text{ is total} \} \).

\( \text{SPERNER, NASH, BROUWER} \in TFNP \)
Is NP-Completeness Useful?

- Remember: $TFNP = \{L \in FNP | L$ is total\}.
- SPERNER, NASH, BROUWER $\in TFNP$
- A search problem $L \in FNP$, associated with $A_L$ and $p_L$, is poly-time (Karp) reducible to another problem $L' \in FNP$, associated with $A_L'$ and $p_L'$, iff there exist efficiently computable functions $f, g$ such that:

\[
\forall x, y: A_L'(f(x), y) = 1 \rightarrow A_L(x, g(y)) = 1
\]

\[
\forall x: A_L'(f(x), y) = 0, \forall y: A_L(x, y) = 0
\]
Is NP-Completeness Useful?

- Remember: \( \text{TFNP} = \{ L \in \text{FNP} | L \text{ is total} \} \).
- \( \text{SPERNER, NASH, BROUWER} \in \text{TFNP} \)
- A search problem \( L \in \text{FNP} \), associated with \( A_L \) and \( p_L \), is poly-time (Karp) reducible to another problem \( L' \in \text{FNP} \), associated with \( A'_L \) and \( p'_L \), iff there exist efficiently computable functions \( f, g \) such that:
  - \( f : \{0,1\}^* \rightarrow \{0,1\}^* \) maps inputs \( x \) to \( L \) into inputs \( f(x) \) to \( L' \).
Is NP-Completeness Useful?

- Remember: \( \text{TFNP} = \{ L \in \text{FNP} | L \text{ is total} \} \).
- \( \text{SPERNER, NASH, BROUWER} \in \text{TFNP} \)
- A search problem \( L \in \text{FNP} \), associated with \( A_L \) and \( p_L \), is poly-time (Karp) reducible to another problem \( L' \in \text{FNP} \), associated with \( A'_L \) and \( p'_L \), iff there exist efficiently computable functions \( f, g \) such that:
  - \( f : \{0,1\}^* \rightarrow \{0,1\}^* \) maps inputs \( x \) to \( L \) into inputs \( f(x) \) to \( L' \).
  - \( \forall x, y : A_{L'}(f(x), y) = 1 \rightarrow A_L(x, g(y)) = 1 \)
  - \( \forall x : A_{L'}(f(x), y) = 0, \forall y \rightarrow A_L(x, y) = 0, \forall y \)
Is NP-Completeness Useful?

- Remember: $\text{TFNP} = \{ L \in \text{FNP} | \text{L is total} \}$.  
- SPERNER, NASH, BROUWER $\in \text{TFNP}$
- A search problem $L \in \text{FNP}$, associated with $A_L$ and $p_L$, is poly-time (Karp) reducible to another problem $L' \in \text{FNP}$, associated with $A'_L$ and $p'_L$, iff there exist efficiently computable functions $f$, $g$ such that:
  - $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ maps inputs $x$ to $L$ into inputs $f(x)$ to $L'$.
  - $\forall x, y : A_L'(f(x), y) = 1 \rightarrow A_L(x, g(y)) = 1$
  - $\forall x : A_L'(f(x), y) = 0, \forall y \rightarrow A_L(x, y) = 0, \forall y$
- We cannot possibly reduce SAT to any of the problems we are interested in (there is no Karp reduction from SAT to SPERNER).
Proof of Sperners Lemma

[Sperner 1928]: If the boundary is legally colored (and regardless how the internal nodes are colored), there exists a tri-chromatic triangle. In fact, an odd number of them.

Proof: ...
Proof of Sperners Lemma
Proof of Sperner's Lemma

Create an artificial triangle and define a walk:
Proof of Sperners Lemma

Transition Rule: If $\exists$ red - yellow door cross it keeping yellow on your left hand
Proof of Sperner's Lemma
Proof of Sperners Lemma
Proof of Sperner's Lemma
A Directed Parity Argument

Vertices of Graph = Triangles
all vertices have in-degree, out-degree ≤ 1

Artificial Trichromatic

degree 1 vertices: trichromatic triangles
degree 2 vertices: no blue, non-trichromatic
degree 0 vertices: all other triangles
The Non-Constructive Step

- A directed graph with an unbalanced node (a node with indegree \(\neq\) outdegree) must have another.
A directed graph with an unbalanced node (a node with indegree ≠ outdegree) must have another.

Given a directed graph and an unbalanced node, isn’t it trivial to find another unbalanced node?
The Non-Constructive Step

- A directed graph with an unbalanced node (a node with indegree \(\neq\) outdegree) must have another.
- Given a directed graph and an unbalanced node, isn’t it trivial to find another unbalanced node?
- The graph can be exponentially large, but has succinct description...
The END OF THE LINE Problem

Suppose that an exponentially large graph with vertex set \( \{0, 1\}^n \) is defined by two circuits:

- **P** circuit:
  - Node id \( \rightarrow \) \( P \) \( \rightarrow \) node id
  - \( P(v_2) = v_1 \) \( \land \) \( N(v_1) = v_2 \)

- **N** circuit:
  - Node id \( \rightarrow \) \( N \) \( \rightarrow \) node id
  - \( v_1 \) \( \rightarrow \) \( v_2 \)

possible previous

possible next
The END OF THE LINE Problem

- Suppose that an exponentially large graph with vertex set \( \{0, 1\}^n \) is defined by two circuits:

  
  \[ P(v_2) = v_1 \land N(v_1) = v_2 \]

- **END OF THE LINE**: Given \( P \) and \( N \): If \( 0^n \) is an unbalanced node, find another unbalanced node. Otherwise output \( 0^n \).
The Class PPAD

The graph created using the two circuits should look like this:
The Class PPAD

The graph created using the two circuits should look like this:

PPAD = \{ Search problems in FNP reducible to END OF THE LINE \}
The Class PPAD

The graph created using the two circuits should look like this:

PPAD \equiv \{ \text{Search problems in FNP reducible to END OF THE LINE} \}

PPAD stands for Polynomial Parity Arguments on Directed graphs
PPAD-Completeness

So far, we have seen the following (easy) reductions:
[Daskalakis-Goldberg-Papadimitriou06]: The opposite order of reductions is also true:
Therefore NASH, SPERNER and BROUWER are **PPAD-Complete**.
PPAD-like Classes

Other complexity classes, similar to PPAD:

- PPA, in which the underlying graph is undirected.
Other complexity classes, similar to PPAD:

- PPA, in which the underlying graph is undirected.
- PPADS, in which the graph is directed and the answer set consists only of the sinks of the paths.
The Class PPAD

Motivating PPAD

Vasileios-Orestis Papadigenopoulos (NTUA)

Equilibria, Fixed Points and Complexity Classes

January 25, 2016

61 / 69
In many search problems, the solutions (resp. the corresponding fixed points) that we want to compute may be irrational.
In many search problems, the solutions (resp. the corresponding fixed points) that we want to compute may be irrational.

For instance, in games with 3 or more players, Nash equilibria are generally irrational.
In many search problems, the solutions (resp. the corresponding fixed points) that we want to compute may be irrational.

For instance, in games with 3 or more players, Nash equilibria are generally irrational.

In the usual (discrete) Turing model of computation and complexity, we have to state carefully and precisely what is the (finite) information about the solution that we want to compute.
In many search problems, the solutions (resp. the corresponding fixed points) that we want to compute may be irrational.

For instance, in games with 3 or more players, Nash equilibria are generally irrational.

In the usual (discrete) Turing model of computation and complexity, we have to state carefully and precisely what is the (finite) information about the solution that we want to compute.

The nature of the desired information can actually affect the complexity of the problem, i.e., some things may be easier to compute than others.
Some types of questions one may ask:

- **Decision problem**: Given game $\Gamma$ and rational $r$, is the value of the game $\geq r$?
Some types of questions one may ask:

- **Decision problem**: Given game $\Gamma$ and rational $r$, is the value of the game $\geq r$?

- **Partial computation**: Given $\Gamma$, integer $k$, compute the $k$ most significant bits of the value.
Some types of questions one may ask:

- **Decision problem**: Given game $\Gamma$ and rational $r$, is the value of the game $\geq r$?

- **Partial computation**: Given $\Gamma$, integer $k$, compute the $k$ most significant bits of the value.

- **Approximation**: Given $\Gamma$, rational $\epsilon > 0$, compute an $\epsilon$-approximation to the value.
Definition (Square Root Sum Problem)

Given positive integers $d_1, d_2, \ldots, d_n$ and $k$, decide whether:

$$\sum_{i=1}^{n} \sqrt{d_i} \leq k$$
Definition (Square Root Sum Problem)
Given positive integers $d_1, d_2, \ldots, d_n$ and $k$, decide whether:

$$\sum_{i=1}^{n} \sqrt{d_i} \leq k$$

- The Square Root Sum Problem is solvable in PSPACE.
Definition (Square Root Sum Problem)

Given positive integers $d_1, d_2, \ldots, d_n$ and $k$, decide whether:

$$\sum_{i=1}^{n} \sqrt{d_i} \leq k$$

- The Square Root Sum Problem is solvable in PSPACE.
- We don’t know whether it is in NP.
Irrational Equilibria, Nonlinear Functions, and the Class FIXP

Definition (Square Root Sum Problem)

Given positive integers \(d_1, d_2, \ldots, d_n\) and \(k\), decide whether:

\[
\sum_{i=1}^{n} \sqrt{d_i} \leq k
\]

The Square Root Sum Problem is solvable in PSPACE.

We don’t know whether it is in NP.

The Sqrt-Sum problem can be reduced to the decision version of many problems: Shapley problem, concurrent reachability games, branching processes, RMCs, and Nash Equilibrium (> 3 players).
Definition (Square Root Sum Problem)

Given positive integers $d_1, d_2, \ldots, d_n$ and $k$, decide whether:

$$\sum_{i=1}^{n} \sqrt{d_i} \leq k$$

- The Square Root Sum Problem is solvable in PSPACE.
- We don’t know whether it is in NP.
- The Sqrt-Sum problem can be reduced to the decision version of many problems: Shapley problem, concurrent reachability games, branching processes, RMCs, and Nash Equilibrium ($> 3$ players).
- Also, for several problems, the approximation of desired objects is at least as hard.
Nash’s function from an n-dimensional simplex to itself looks like this:

\[ F_{\Gamma}(x)_{(i,j)} = \frac{x_{i,j} + \max\{0, g_{i,j}(x)\}}{1 + \sum_{l=1}^{|S_i|} \max\{0, g_{i,l}(x)\}} \]

where \( g_{i,j}(x) \) is the (positive or negative) gain in payoff of player i if he switches to pure strategy j while the other players continue to play according to \( x \).
• Nash’s function from an n-dimensional simplex to itself looks like this:

\[ F_{\Gamma}(x)_{i,j} = \frac{x_{i,j} + \max\{0, g_{i,j}(x)\}}{1 + \sum_{l=1}^{\left|S_i\right|} \max\{0, g_{i,l}(x)\}} \]

where \( g_{i,j}(x) \) is the (positive or negative) gain in payoff of player i if he switches to pure strategy j while the other players continue to play according to x.

• Search problems in FIXP are fixed-point problems of functions that use the usual algebraic operations and max,min, like Nash’s function.
Definition

FIXP is the class of search problems Π, such that there is a polynomial-time algorithm which, given an instance I, constructs an algebraic-circuit (straight-line program) $C_I$ over the basis $(+,*,-,/,\text{max},\text{min})$, with rational constants, that defines a continuous function $F_I$ from a domain to itself, with the property that $\text{Sol}_\Pi(I)$ is the set of fixed points of $F_I$. 
References

Mihalis Yannakakis
Equilibria, Fixed Points and Complexity Classes

David S. Johnson
The NP-Completeness Column: Finding Needles in Haystacks

K. Daskalakis
Algorithmic Lower Bounds: Fun with Hardness Proofs (Guest Lecture)
Thank You