COMP 553: Algorithmic Game Theory

Lecture 21

Fall 2014

Yang Cai

How hard is computing a Nash Equilibrium?

NASH, BROUWER and SPERNER

We informally define three computational problems:

- NASH: find a (appx-) Nash equilibrium in a n player game.
- BROUWER: find a (appx-) fixed point x for a continuous function f().
- SPERNER: find a trichromatic triangle (panchromatic simplex) given a legal coloring.

Function NP (FNP)

A *search problem* L is defined by a relation $R_L(x, y)$ such that

$$R_L(x, y)=1 \quad \text{iff} \quad y \text{ is a solution to } x$$

A search problem is called *total* iff for all x there exists y such that $R_L(x, y) =1$.

A search problem L belongs to FNP iff there exists an efficient algorithm $A_L(x, y)$ and a polynomial function $p_L(\cdot)$ such that

(i) if $A_L(x, z)=1$ $\Rightarrow$ $R_L(x, z)=1$

(ii) if $\exists\, y$ s.t. $R_L(x, y)=1$ $\Rightarrow$ $\exists\, z$ with $|z| \leq p_L(|x|)$ such that $A_L(x, z)=1$

Clearly, SPERNER $\in$ FNP.

Reductions between Problems

A search problem $L \in$ FNP, associated with $A_L(x, y)$ and p_L, is ***polynomial-time reducible*** to another problem $L' \in$ FNP, associated with $A_{L'}(x, y)$ and $p_{L'}$, iff there exist efficiently computable functions f, g such that

(i) x is input to L $\Rightarrow$ $f(x)$ is input to L'

(ii)

$A_{L'}(f(x), y)=1 \Rightarrow A_L(x, g(y))=1$

$R_{L'}(f(x), y)=0, \forall y \Rightarrow R_L(x, y)=0, \forall y$

A search problem L is ***FNP-complete*** iff

$\boxed{\text{e.g. SAT}}$

$L \in$ FNP

L' is poly-time reducible to L, for all $L' \in$ FNP

Our Reductions (intuitively)

NASH $\rightsquigarrow$ BROUWER $\rightsquigarrow$ SPERNER $\in$ FNP

both Reductions are polynomial-time

Is then SPERNER FNP-complete?

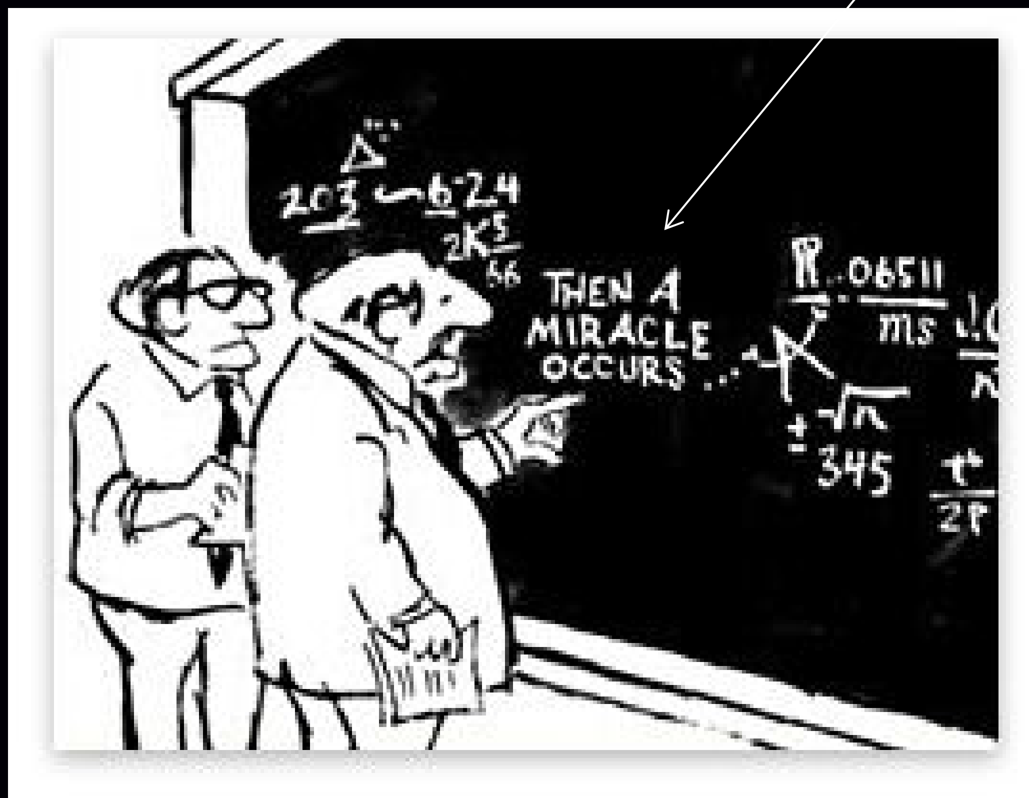
- With our current notion of reduction the answer is no, because SPERNER always has a solution, while a SAT instance may not have a solution;

- To attempt an answer to this question we need to update our notion of reduction. Suppose we try the following: we require that a solution to SPERNER informs us about whether the SAT instance is satisfiable or not, and provides us with a solution to the SAT instance in the ``yes'' case;

  but if such a reduction existed, it could be turned into a non-deterministic algorithm for checking "no" answers to SAT: guess the solution to SPERNER; this will inform you about whether the answer to the SAT instance is "yes" or "no", leading to $NP = co - NP \ldots$

A Complexity Theory of Total Search Problems ?

??



A Complexity Theory of Total Search Problems ?

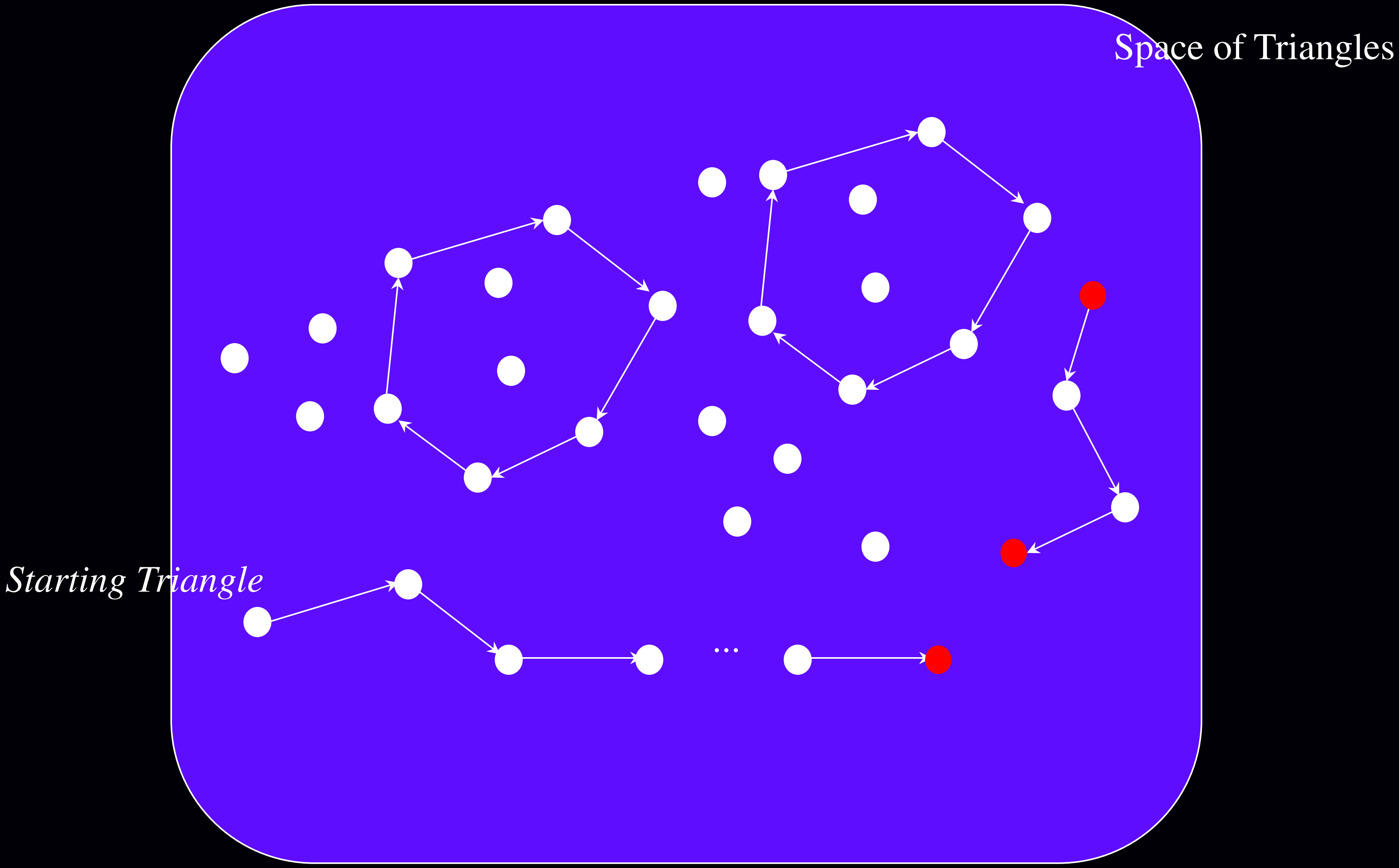
100-feet overview of our methodology:

1. identify the combinatorial argument of existence, responsible for making the problem total;

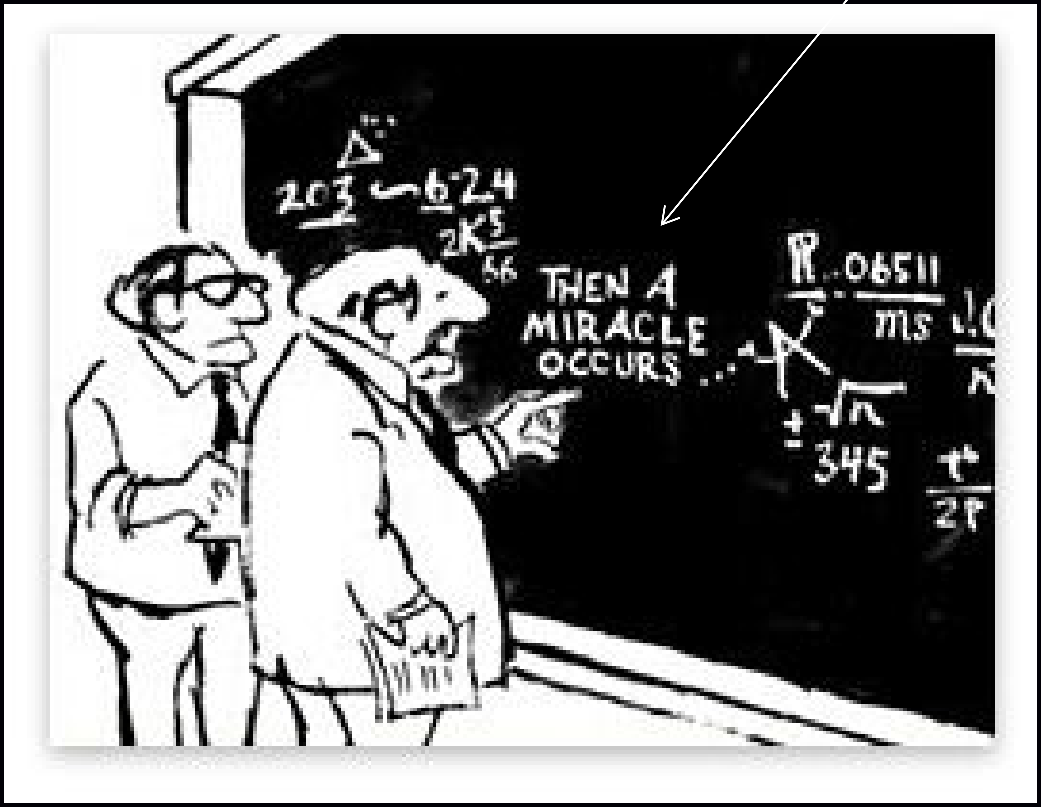
2. define a complexity class inspired by the argument of existence;

3. make sure that the complexity of the problem was captured as tightly as possible (via a completeness result).

Recall Proof of Sperner's Lemma



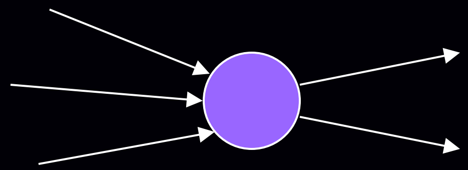
Combinatorial argument of existence?



The Non-Constructive Step

an easy parity lemma:

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



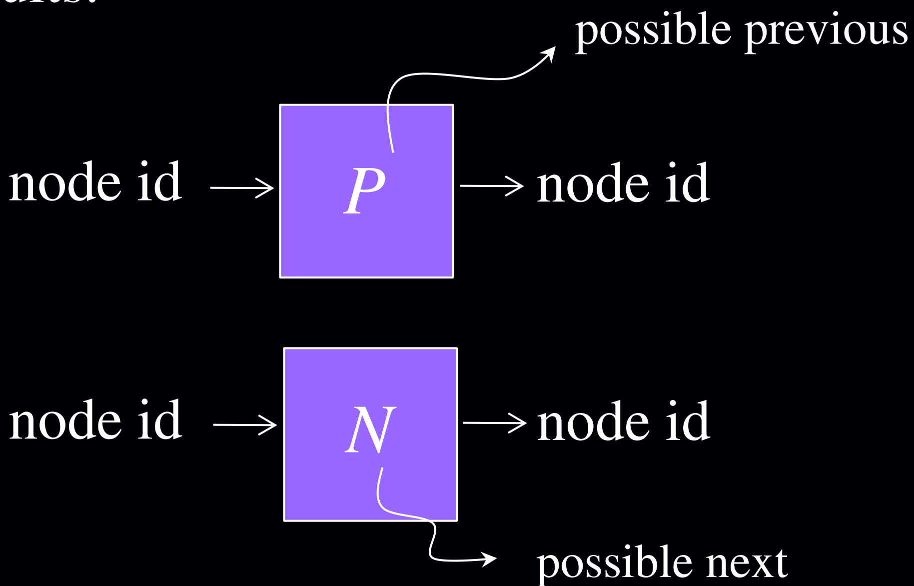
but, why is this non-constructive?

given a directed graph and an unbalanced node, isn't it trivial to find another unbalanced node?

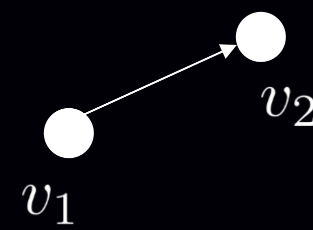
the graph can be exponentially large, but has succinct description…

The PPAD Class [Papadimitriou '94]

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



$$P(v_2) = v_1 \wedge N(v_1) = v_2$$



END OF THE LINE: Given P and N: If 0^n is an unbalanced node, find another unbalanced node. Otherwise say "yes".

PPAD = *{ Search problems in FNP reducible to END OF THE LINE}*

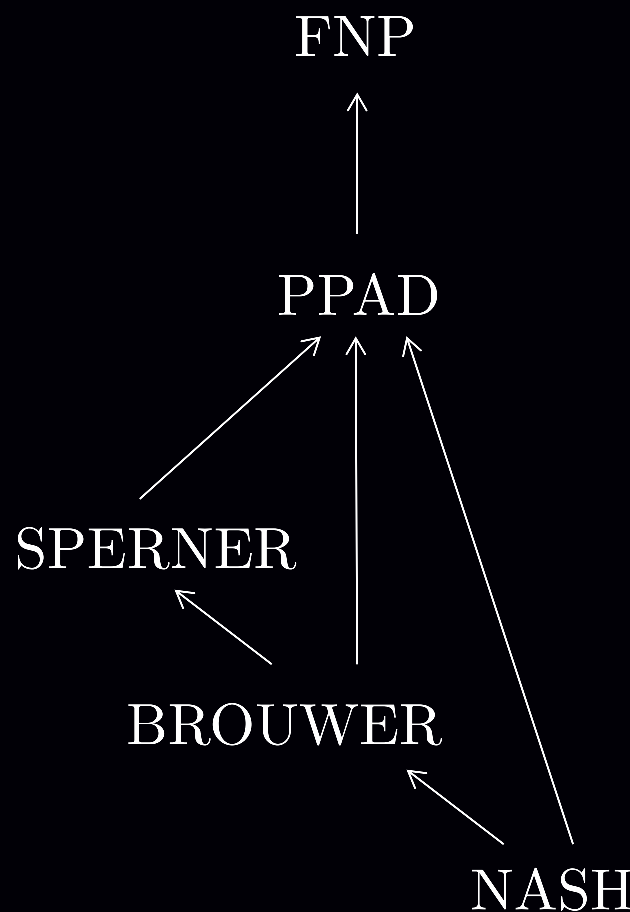
Inclusions

(i) $\mathrm{PPAD} \subseteq \mathrm{FNP}$

(ii) $\mathrm{SPERNER} \in \mathrm{PPAD}$

PROOF (sketch): Sufficient to define appropriate circuits *P* and *N* as we have in our proof.

- Each triangle is associated with a node id.
- If there is a red- yellow door such that red is on your left, then cross this door, you will enter the successor triangle.
- If there is a red- yellow door such that red is on your right, then cross this door, you will enter the predecessor triangle.


FNP
PPAD
SPERNER
BROUWER
NASH

Other arguments of existence, and resulting complexity classes

"If a graph has a node of odd degree, then it must have another."

PPA

"Every directed acyclic graph must have a sink."

PLS

"If a function maps n elements to n-1 elements, then there is a collision."

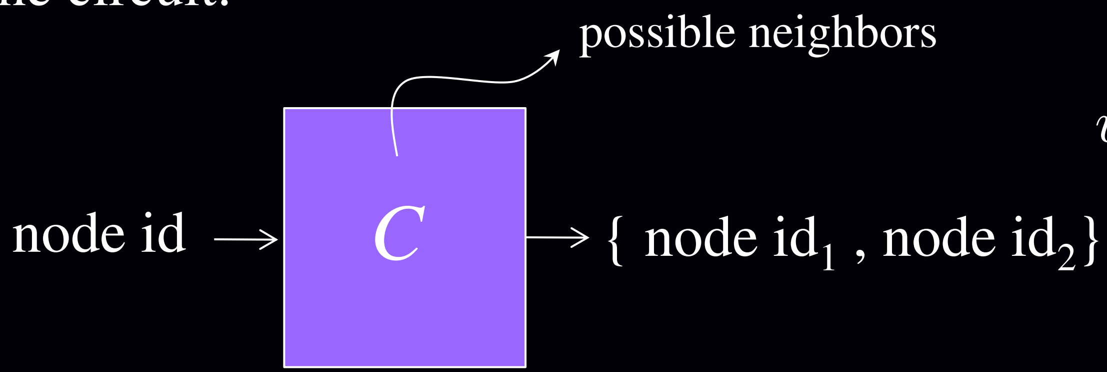
PPP

Formally?

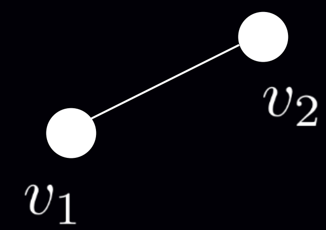
The Class PPA [Papadimitriou ’94]

“If a graph has a node of odd degree, then it must have another.”

Suppose that an exponentially large graph with vertex set $\{0,1\}^n$ is defined by one circuit:



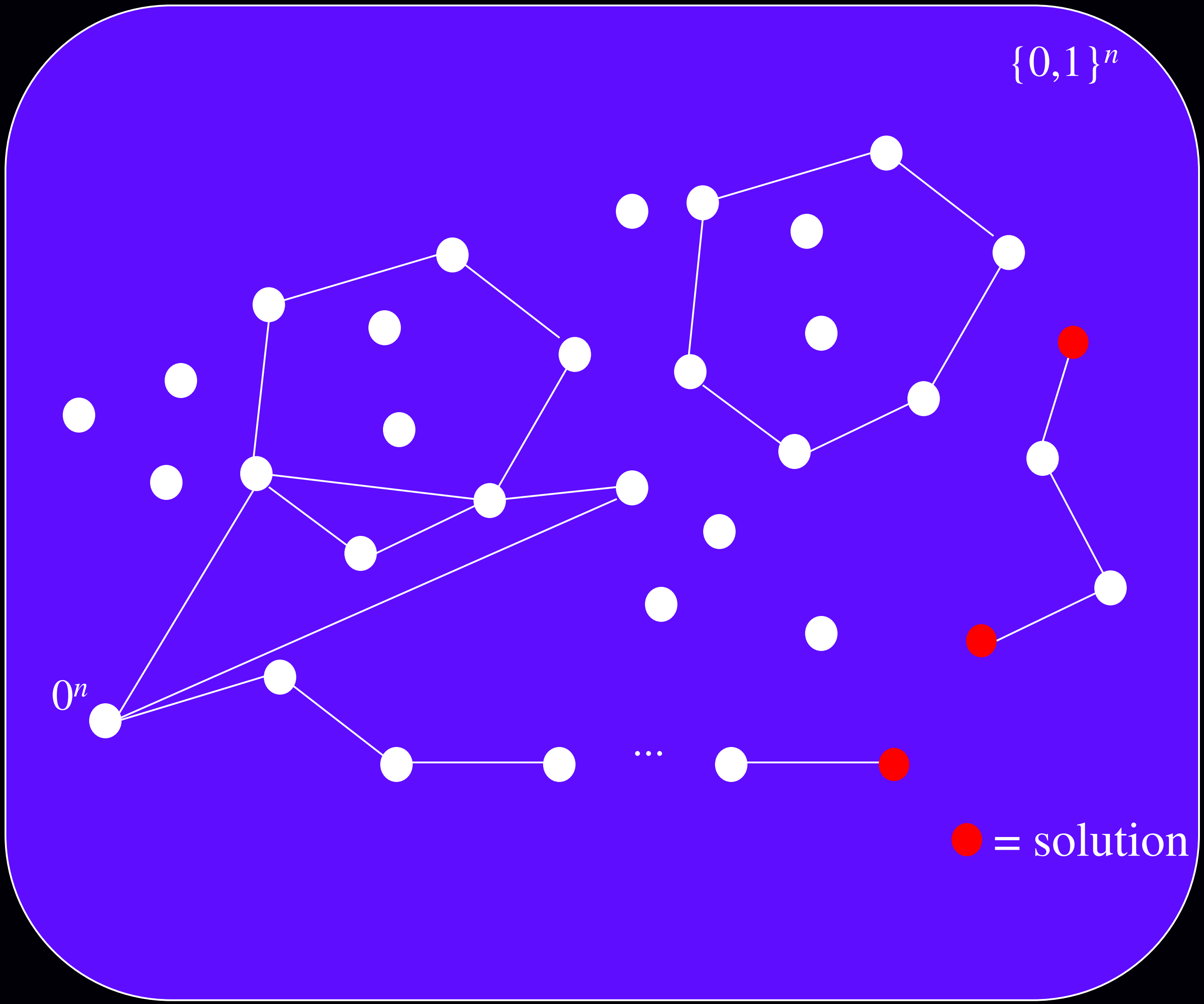
$$v_1 \in C(v_2) \quad \wedge \quad v_2 \in C(v_1)$$



ODD DEGREE NODE: Given C: If 0^n has odd degree, find another node with odd degree. Otherwise say “yes”.

PPA = $\{$ *Search problems in FNP reducible to* ODD DEGREE NODE $\}$

The Undirected Graph



The Class PLS [JPY '89]

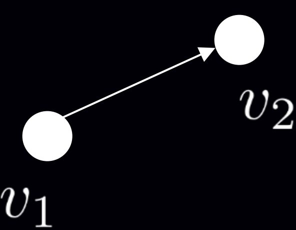
"Every DAG has a sink."

Suppose that a DAG with vertex set $\{0,1\}^n$ is defined by two circuits:

node id $\longrightarrow$ $\boxed{C}$ $\longrightarrow$ {node id$_1$, …, node id$_k$}

node id $\longrightarrow$ $\boxed{F}$ $\longrightarrow$ $\mathbb{R}$

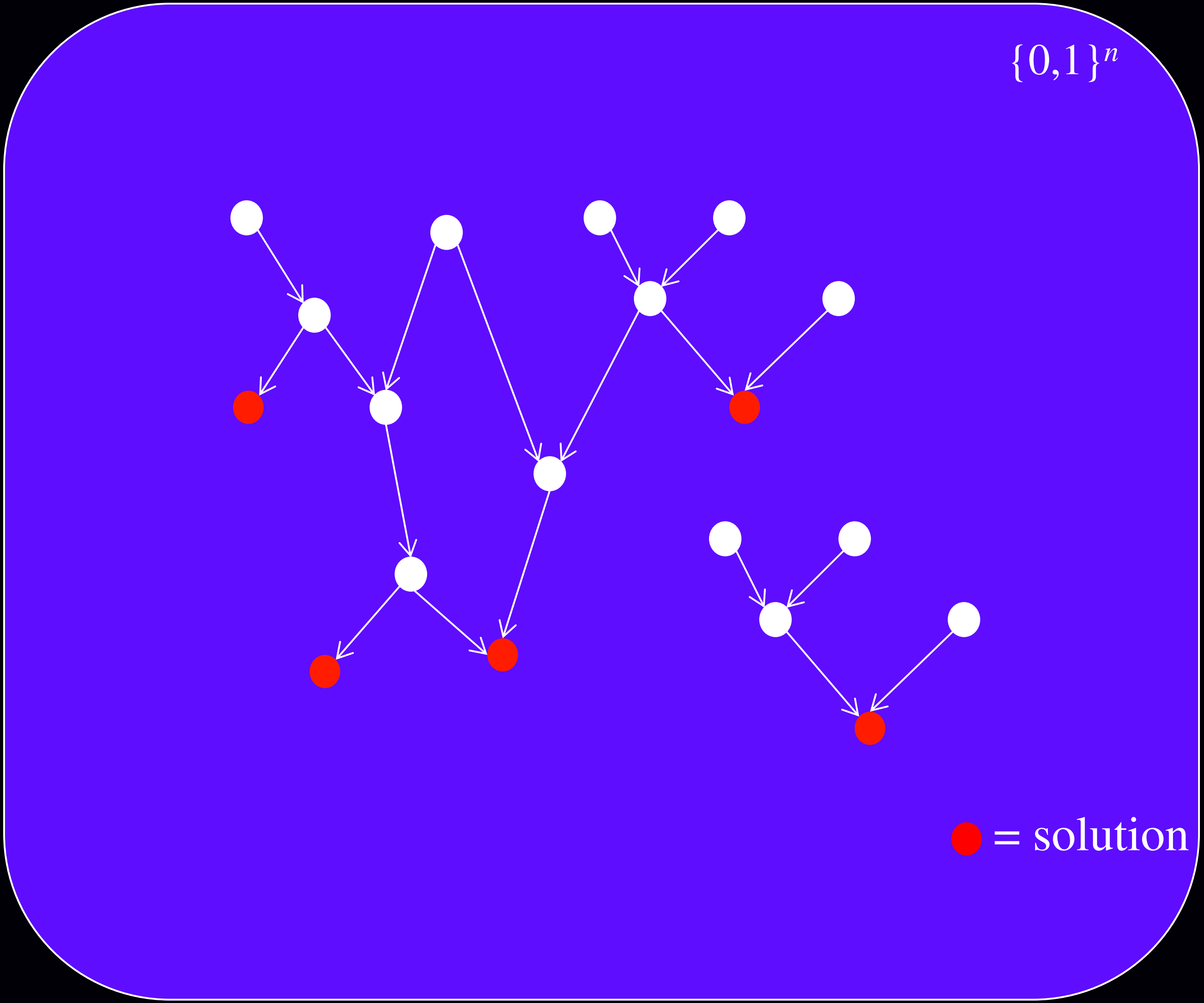
$$v_2 \in C(v_1) \quad \wedge \quad F(v_2) > F_(v_1)$$



FIND SINK: Given *C*, *F*: Find *x* s.t. F(x) ≥ F(y), for all *y* ∈ *C*(*x*).

PLS = { *Search problems in FNP reducible to* FIND SINK }

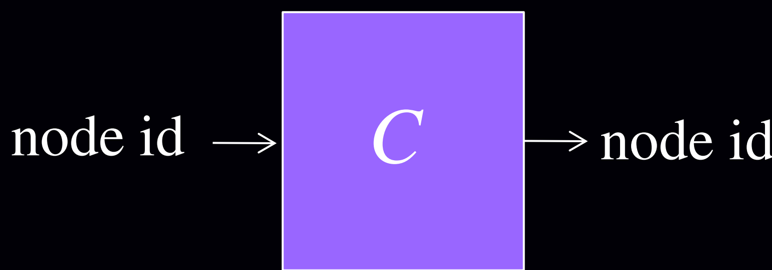
The DAG



The Class PPP [Papadimitriou '94]

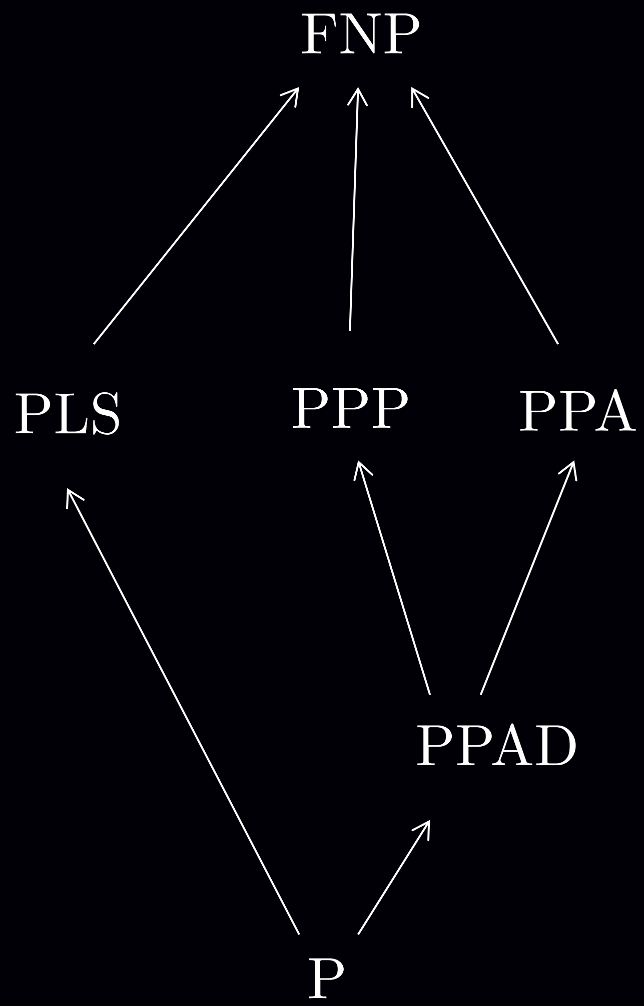
"If a function maps n elements to n-1 elements, then there is a collision."

Suppose that an exponentially large graph with vertex set $\{0,1\}^n$ is defined by one circuit:



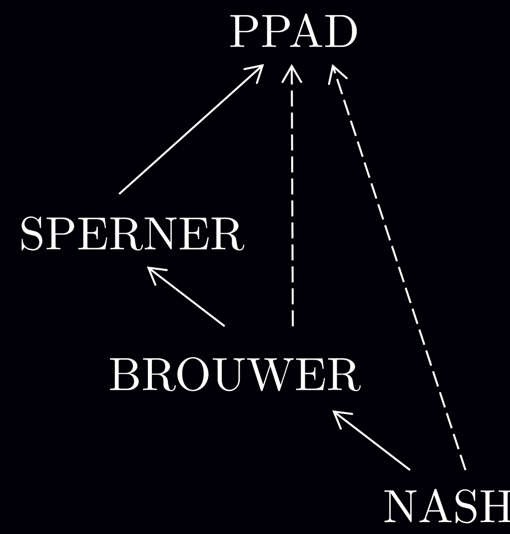
COLLISION: Given C: Find x s.t. $C(x) = 0^n$; or find $x \neq y$ s.t. $C(x)=C(y)$.

PPP = { *Search problems in FNP reducible to* COLLISION }

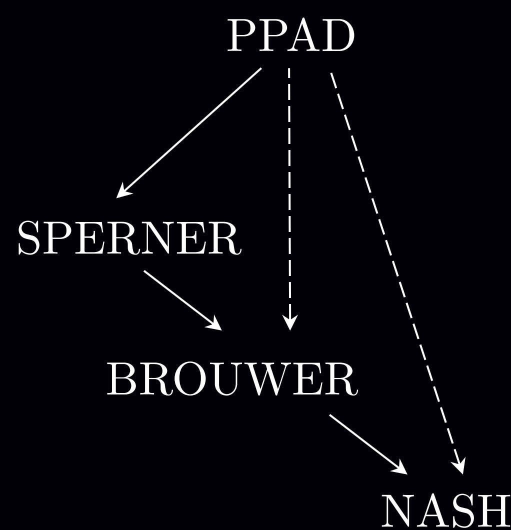

FNP
PLS
PPP
PPA
PPAD
P

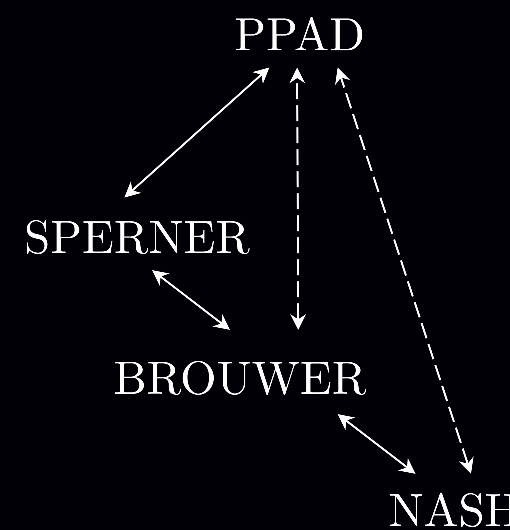
Hardness Results

Inclusions we have already established:



Our next goal:





The Main Result

Theorem[DGP, CD]: Finding a Nash equilibrium of a 2-player game is a PPAD-complete problem.

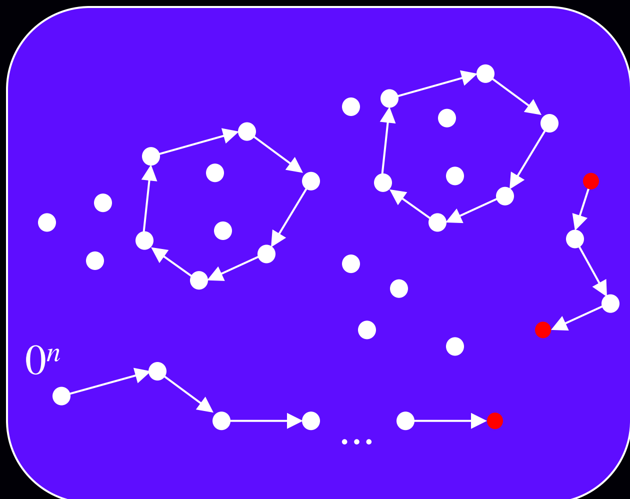
DGP = Daskalakis, Goldberg, Papadimitriou

CD = Chen, Deng

The PLAN

DGP = Daskalakis, Goldberg, Papadimitriou

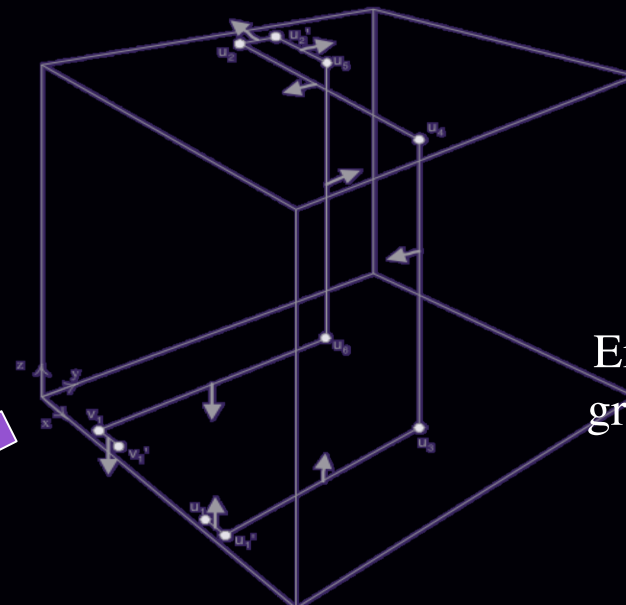
CD = Chen, Deng



Generic PPAD

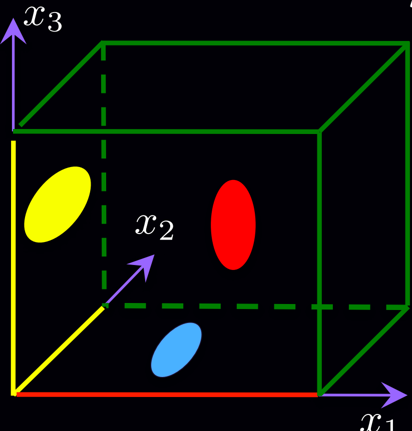
[Pap '94]

[DGP '05]



Embed PPAD graph in $[0,1]^3$

[DGP '05]

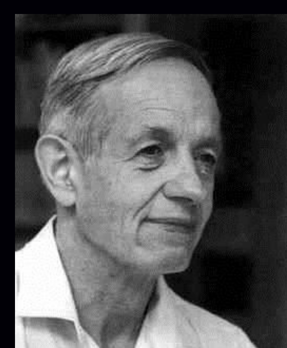


3D-SPERNER

[DGP '05]

p.w. linear BROUWER

[DGP '05]



multi-player NASH

[DGP '05]

4-player NASH

[DP '05] [CD'05]

3-player NASH

[CD'06]

2-player NASH

Algorithms for computing Nash equilibrium

Support Enumeration Algorithms

How better would my life be if I knew the support of the Nash equilibrium?

… and the game is 2-player?

Setting: Let (R, C) be an m by n game, and suppose a friend revealed to us the supports $\mathcal{S}_R$ and $\mathcal{S}_C$ respectively of the Row and Column players' mixed strategies at some equilibrium of the game.

any feasible point (x, y) of the following linear program is an equilibrium!

$$\begin{aligned}
&\max \quad 1 \\
\text{s.t.} \quad & e_i^{\mathrm{T}} R y \geq e_j^{\mathrm{T}} R y, \quad \forall\, i \in \mathcal{S}_R, \ \forall\, j \in [m] \\
& x^{\mathrm{T}} C e_i \geq x^{\mathrm{T}} C e_j, \quad \forall\, i \in \mathcal{S}_C, \ \forall\, j \in [n] \\
& \sum x_i = 1 \text{ and } \sum y_i = 1 \\
& x_i = 0, \ \forall i \notin \mathcal{S}_R \quad \text{and} \quad y_j = 0, \ \forall j \notin \mathcal{S}_C
\end{aligned}$$

Support Enumeration Algorithms

How better would my life be if I knew the support of the Nash equilibrium?

… and the game is 2-player?

Runtime: $2^{m+n} \cdot \text{poly}(|R|, |C|)$

for guessing the support

for solving the LP

Support Enumeration Algorithms

How better would my life be if I knew the support of the Nash equilibrium?

… and the game is separable?

input: the support $\mathcal{S}_v$ of every node v at equilibrium

goal: recover the Nash equilibrium with that support

➔ can do this with Linear Programming too!

the idea of why this is possible is similar to the 2-player case:

- the expected payoff of a node from a given pure strategy is linear in the mixed strategies of the other players;
- hence, once the support is known, the equilibrium conditions correspond to linear equations and inequalities.

Rationality of Equilibria

Important Observation:

The correctness of the support enumeration algorithm implies that in 2-player games and in polymatrix games there always exists an equilibrium in rational numbers, and with description complexity polynomial in the description of the game!

Computation of Approximate Equilibria

Theorem [Lipton, Markakis, Mehta '03]:

For all $\epsilon > 0$ and any 2-player game with at most *n* strategies per player and payoff entries in [0,1], there exists an ϵ-approximate Nash equilibrium in which each player's strategy is uniform on a multiset of their pure strategies of size $O\left(\frac{\log n}{\epsilon^2}\right)$.

Proof idea: (of a stronger claim)

- By Nash's theorem, there exists a Nash equilibrium (*x*, *y*).

- Suppose we take $t = \lceil 16 \log n / \epsilon^2 \rceil$ samples from *x*, viewing it as a distribution.

  $\mathcal{X}$: uniform distribution over the sampled pure strategies

- Similarly, define $\mathcal{Y}$ by taking *t* samples from *y*.

Claim: $(\mathcal{X}, \mathcal{Y})$ is an ϵ-Nash equilibrium with probability at least $1 - \frac{4}{n}$.

Computation of Approximate Equilibria

Suffices to show the following:

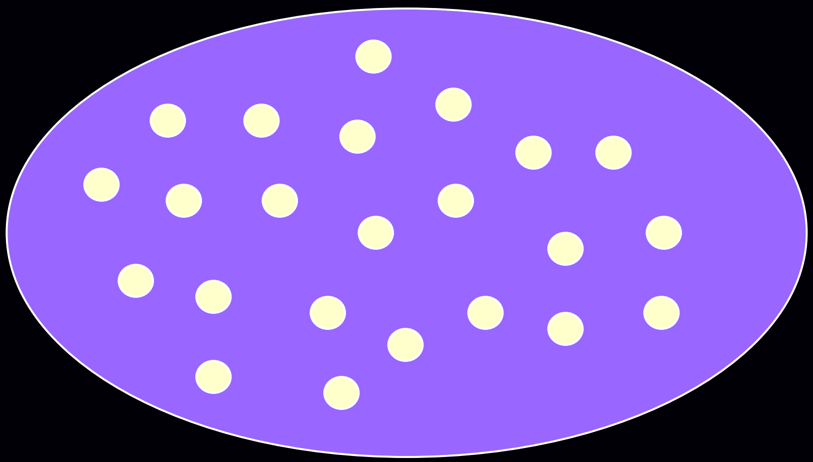
Lemma: With probability at least 1-4/n the following are satisfied:

$$|e_i^{\mathrm{T}} R\mathcal{Y} - e_i^{\mathrm{T}} Ry| \leq \epsilon/2, \text{ for all } i \in [n];$$

$$|\mathcal{X}^{\mathrm{T}} Ce_j - x^{\mathrm{T}} Ce_j| \leq \epsilon/2, \text{ for all } j \in [n].$$

Proof: Chernoff bounds.

Computation of Approximate Equilibria



set S_ϵ : *every point is a pair of mixed strategies that are uniform on a multiset of size* $O\left(\frac{\log n}{\epsilon^2}\right)$.

Random sampling from S_ϵ *takes expected time*

$$n^{O\left(\frac{\log n}{\epsilon^2}\right)}$$

Oblivious Algorithm: set S_ϵ does not depend on the game we are solving.

Theorem [Daskalakis-Papadimitriou ’09] : *Any oblivious algorithm for general games runs in expected time* $\Omega\left(n^{(.8-34\epsilon)\log n}\right)$