L09 Complexity of Computing NE

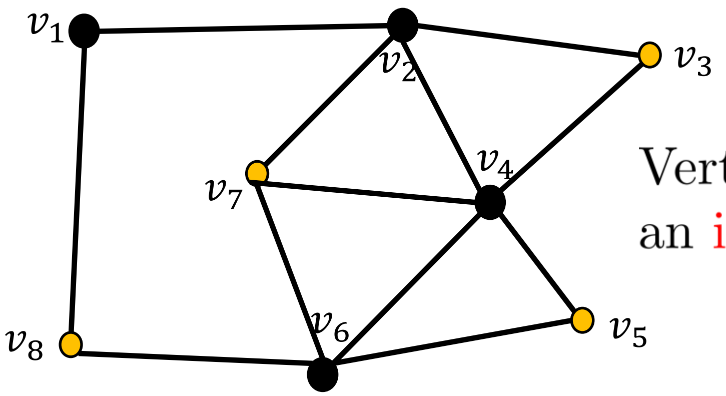
CS 295 Introduction to Algorithmic Game Theory
Ioannis Panageas

Inspired and some figures
by C. Daskalakis slides and T. Roughgarden notes

Warm-up: Reductions in NP

Example: **INDEPENDENT SET (IS) Problem**

Given a simple undirected graph $G(V, E)$ and k, is there an independent set in G of size $\geq k$. Independent set is called a set $I \subset V$ of vertices such that pairwise the vertices in I are not connected.



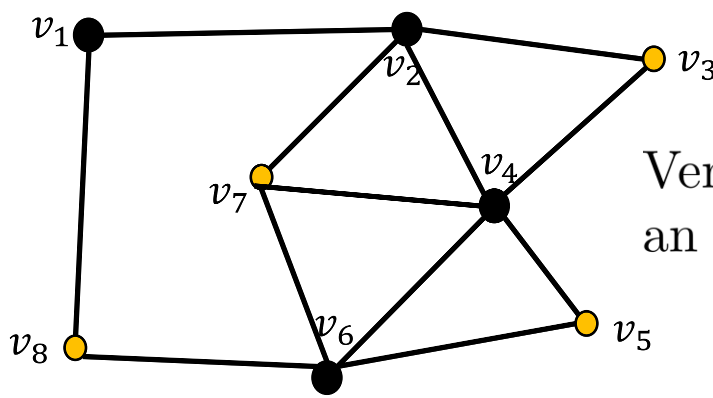
Graph G.

Vertices v_3, v_5, v_7, v_8 form an independent set.

Warm-up: Reductions in NP

Example: **INDEPENDENT SET (IS) Problem**

Given a simple undirected graph $G(V, E)$ and k, is there an independent set in G of size $\geq k$. Independent set is called a set $I \subset V$ of vertices such that pairwise the vertices in I are not connected.



Graph G.

Vertices v_3, v_5, v_7, v_8 form an independent set.

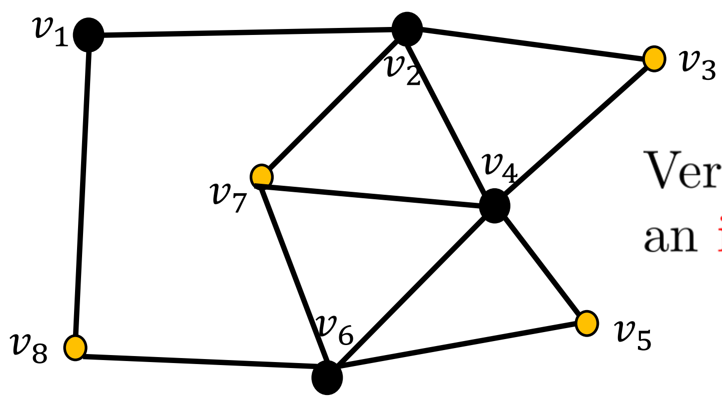
Claim: INDEPENDENT SET is **NP-complete**.

Proof: (1) INDEPENDENT SET **belongs** to NP (why?).
(2) Reduce 3-SAT to INDEPENDENT SET. Since 3-SAT is NP-hard, INDEPENDENT SET is NP-hard.

Warm-up: Reductions in NP

Example: **INDEPENDENT SET (IS) Problem**

Given a simple undirected graph $G(V, E)$ and k, is there an independent set in G of size $\geq k$. Independent set is called a set $I \subset V$ of vertices such that pairwise the vertices in I are not connected.



Graph G.

Vertices v_3, v_5, v_7, v_8 form an independent set.

(1), (2) imply IND. SET is NP-complete!

Claim: INDEPENDENT SET is **NP-complete**.

Proof: (1) INDEPENDENT SET **belongs** to NP (why?).
(2) Reduce 3-SAT to INDEPENDENT SET. Since 3-SAT is NP-hard, INDEPENDENT SET is NP-hard.

3-SAT reduction to IS

Problem: 3-SAT

Given a Boolean expression E, such that E is a conjunction of clauses, where each clause is a disjunction of exactly 3 literals, is E satisfiable?

3-SAT reduction to IS

Problem: 3-SAT

Given a Boolean expression E, such that E is a conjunction of clauses, where each clause is a disjunction of exactly 3 literals, is E satisfiable?

A **literal** is a Boolean expression consisting of just a single Boolean variable, or the negation of a Boolean variable.

- Example: "$\neg x_1$" and "x_2" are literals.

A **clause** is a Boolean expression of the form "$\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$", i.e. a **disjunction** of some literals $\ell_1, \ell_2, \dots, \ell_k$. In 3-SAT $k = 3$.

- Example: "$C_1 \equiv x_1 \vee \neg x_2 \vee x_3$" is a clause.

A Boolean expression is a conjunction of clauses.

Example: "$E \equiv C_1 \wedge C_2 \wedge C_3$" is a clause.

3-SAT reduction to IS

Satisfiability: Can you assign True, False to the variables so that the expression is True?

Theorem (3-SAT is NP-complete). *The 3-SAT problem is NP-complete!*

3-SAT reduction to IS

Satisfiability: Can you assign True, False to the variables so that the expression is True?

Theorem (3-SAT is NP-complete)**.** *The 3-SAT problem is NP-complete!*

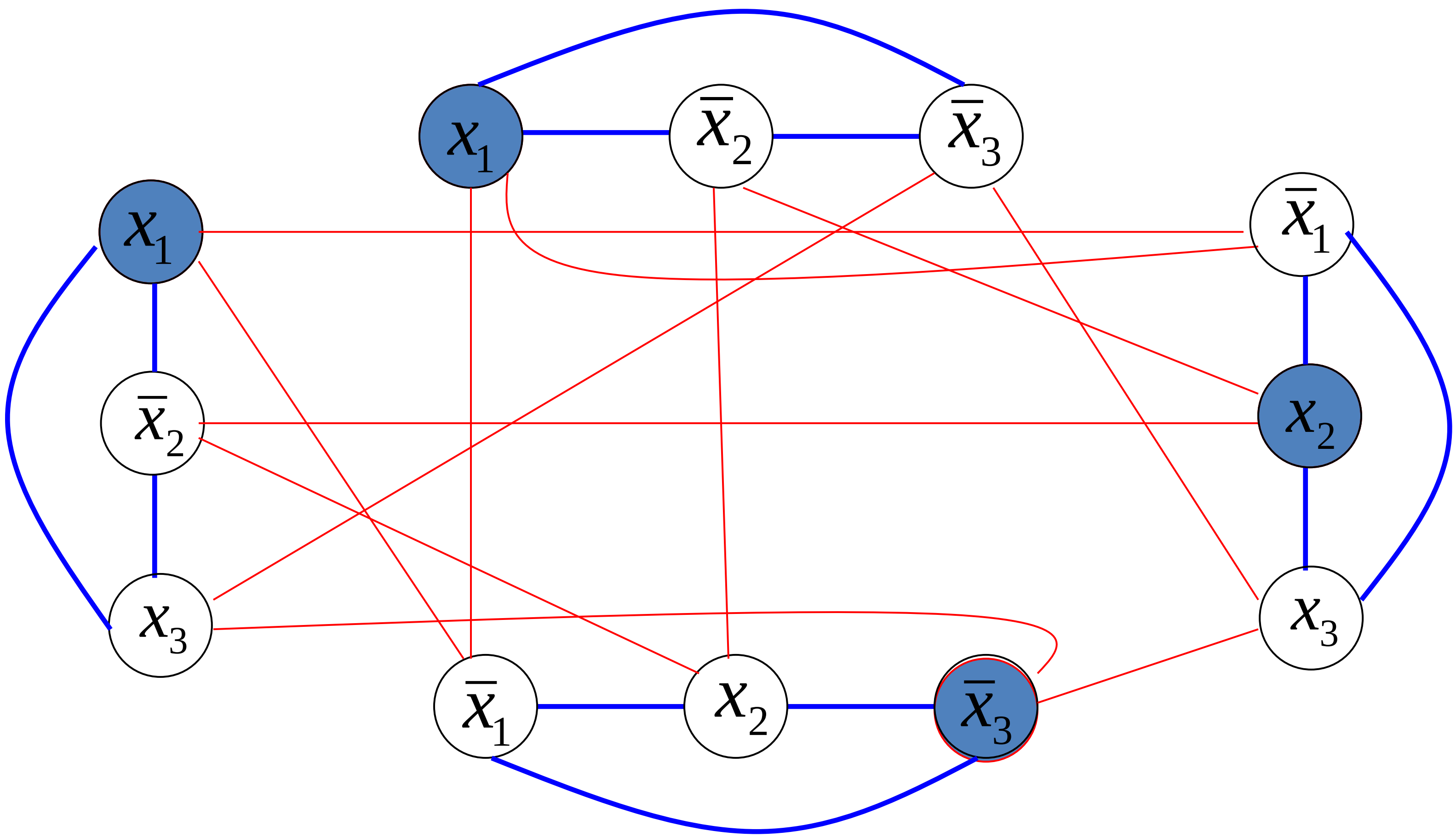
$$E = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

3-SAT reduction to IS

Satisfiability: Can you assign True, False to the variables so that the expression is True?

Theorem (3-SAT is NP-complete). *The 3-SAT problem is NP-complete!*

$$E = \overbrace{(x_1 \vee \bar{x}_2 \vee \bar{x}_3)}^{C_1} \wedge \overbrace{(\bar{x}_1 \vee x_2 \vee x_3)}^{C_2} \wedge \overbrace{(\bar{x}_1 \vee x_2 \vee \bar{x}_3)}^{C_3} \wedge \overbrace{(x_1 \vee \bar{x}_2 \vee x_3)}^{C_4}$$



3-SAT reduction to IS

Claim: Expression E with k clauses is satisfiable if and only if the induced graph G has an IS of size k.

Therefore, given a **graph *G* and a *k***, if we can identify in **poly-time** if there exists an **Independent Set of size at least k**, then we can solve **in poly-time 3-SAT**.

3-SAT reduction to IS

Claim: Expression E with k clauses is satisfiable if and only if the induced graph G has an IS of size k.

Therefore, given a **graph *G* and a *k***, if we can identify in **poly-time** if there exists an **Independent Set of size at least k**, then we can solve **in poly-time 3-SAT**.

3-SAT $\leq_p$ INDEPENDENT SET $\Rightarrow$ INDEPENDENT SET is NP-complete!

3-SAT reduction to IS

Claim: Expression E with k clauses is satisfiable if and only if the induced graph G has an IS of size k.

Therefore, given a **graph *G* and a *k***, if we can identify in **poly-time** if there exists an **Independent Set of size at least k**, then we can solve **in poly-time 3-SAT**.

3-SAT $\leq_p$ INDEPENDENT SET $\Rightarrow$ INDEPENDENT SET is NP-complete!

Question: Can the problem of computing a Nash Equilibrium be NP-complete?

3-SAT reduction to IS

Claim: Expression E with k clauses is satisfiable if and only if the induced graph G has an IS of size k.

Therefore, given a **graph G and a k**, if we can identify in **poly-time** if there exists an **Independent Set of size at least k**, then we can solve **in poly-time 3-SAT**.

3-SAT $\leq_p$ INDEPENDENT SET $\Rightarrow$ INDEPENDENT SET is NP-complete!

Question: Can the problem of computing a Nash Equilibrium be NP-complete?

Answer: (Megiddo) Suppose we have a reduction from SAT to NASH, s.t any solution to the instance of NASH tells us whether or not the SAT instance has a solution. Then we could turn this into a nondeterministic algorithm for verifying that an instance of SAT has no solution: Just guess a solution of the NASH instance, and check that it indeed implies that the SAT instance has no solution. NP = co-NP (unlikely).

The class PLS

PLS (Polynomial-time Local Search) is a complexity class intended to exemplify local search problems. An abstract local search problem is specified by three polynomial-time algorithms.

Canonical Problem: LOCAL MAX-CUT

Given an undirected graph $G = (V, E)$ with non-negative weights w_e on edges, find a cut $(S, \overline{S})$ that maximizes the total weight of cut edges. You are allowed to do only local moves that improve the objective, i.e., moving one vertex v from one side of the cut to the other that improves the total weight of cut edges.

The class PLS

PLS (Polynomial-time Local Search) is a complexity class intended to exemplify local search problems. An abstract local search problem is specified by three polynomial-time algorithms.

Canonical Problem: LOCAL MAX-CUT

Given an undirected graph $G = (V, E)$ with non-negative weights w_e on edges, find a cut $(S, \overline{S})$ that maximizes the total weight of cut edges. You are allowed to do only local moves that improve the objective, i.e., moving one vertex v from one side of the cut to the other that improves the total weight of cut edges.

Remark: (classic) MAX-CUT is NP-Complete.

The class PLS

PLS (Polynomial-time Local Search) is a complexity class intended to exemplify local search problems. An abstract local search problem is specified by three polynomial-time algorithms.

1. The first algorithm takes as input an instance and outputs an arbitrary feasible solution (for LOCAL MAX-CUT this is an arbitrary cut).

The class PLS

PLS (Polynomial-time Local Search) is a complexity class intended to exemplify local search problems. An abstract local search problem is specified by three polynomial-time algorithms.

1. The first algorithm takes as input an instance and outputs an arbitrary feasible solution (for LOCAL MAX-CUT this is an arbitrary cut).

2. The second algorithm takes as input an instance and a feasible solution, and returns the objective function value of the solution (for LOCAL MAX-CUT it is the sum of the total weight of the edges crossing the cut).

The class PLS

PLS (Polynomial-time Local Search) is a complexity class intended to exemplify local search problems. An abstract local search problem is specified by three polynomial-time algorithms.

1. The first algorithm takes as input an instance and outputs an arbitrary feasible solution (for LOCAL MAX-CUT this is an arbitrary cut).

2. The second algorithm takes as input an instance and a feasible solution, and returns the objective function value of the solution (for LOCAL MAX-CUT it is the sum of the total weight of the edges crossing the cut).

3. The third algorithm takes as input an instance and a feasible solution and either reports "locally optimal" or produces a better solution (for LOCAL MAX-CUT it checks all possible $|V|$ moves. If one improves the objective choose that move).

Theorem (Local Max-cut is PLS-complete). *The LOCAL MAX-CUT problem is PLS-complete.*

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete)**.** *The problem of computing Pure Nash Equilibria in Congestion Games is PLS-complete.*

Proof. We show first that PNE CONGESTION GAMES $\in$ PLS.

Describe the three algorithms:

- First algorithm takes as input a congestion game and returns an arbitrary strategy profile (e.g., all agents choose first path).

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *The problem of computing Pure Nash Equilibria in Congestion Games is PLS-complete.*

Proof. We show first that PNE CONGESTION GAMES $\in$ PLS.

Describe the three algorithms:

- First algorithm takes as input a congestion game and returns an arbitrary strategy profile (e.g., all agents choose first path).
- Second algorithm takes a congestion game and a strategy profile s, and returns the value of the potential function $\Phi(s) = \sum_e \sum_{j=1}^{l_e(s)} c_e(j)$.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete)**.** *The problem of computing Pure Nash Equilibria in Congestion Games is PLS-complete.*

Proof. We show first that PNE CONGESTION GAMES $\in$ PLS.

Describe the three algorithms:

- First algorithm takes as input a congestion game and returns an arbitrary strategy profile (e.g., all agents choose first path).
- Second algorithm takes a congestion game and a strategy profile s, and returns the value of the potential function $\Phi(s) = \sum_e \sum_{j=1}^{l_e(s)} c_e(j)$.
- The third algorithm checks if the given strategy profile s is a PNE; if not, we find an agent i that deviates from s_i to another pure s'_i and decreases her utility. Then $\Phi(s'_i, s_{-i}) < \Phi(s_i, s_{-i})$. This can be done polynomial time in the description of the game.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete)**.** *The problem of computing Pure Nash Equilibria in Congestion Games is PLS-complete.*

Proof cont. We now reduce LOCAL MAX-CUT to PNE CONGESTION GAMES.

Given a weighted graph $G(V, E)$ we define the following congestion game:

- Agents are the vertices V.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete)**.** *The problem of computing Pure Nash Equilibria in Congestion Games is PLS-complete.*

Proof cont. We now reduce LOCAL MAX-CUT to PNE CONGESTION GAMES.

Given a weighted graph $G(V, E)$ we define the following congestion game:

- Agents are the vertices V.
- For each edge $e \in E$ we have two resources r_e, $\overline{r}_e$ ($2|E|$ resources in total).

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete)**.** *The problem of computing Pure Nash Equilibria in Congestion Games is PLS-complete.*

Proof cont. We now reduce LOCAL MAX-CUT to PNE CONGESTION GAMES.

Given a weighted graph $G(V, E)$ we define the following congestion game:

- Agents are the vertices V.
- For each edge $e \in E$ we have two resources r_e, $\overline{r}_e$ ($2|E|$ resources in total).
- Each player v has two strategies,
  $s_v = \{r_e : e \text{ is incident to } v\}$ and $\overline{s}_v = \{\overline{r}_e : e \text{ is incident to } v\}$.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *The problem of computing Pure Nash Equilibria in Congestion Games is PLS-complete.*

Proof cont. We now reduce LOCAL MAX-CUT to PNE CONGESTION GAMES.

Given a weighted graph $G(V, E)$ we define the following congestion game:

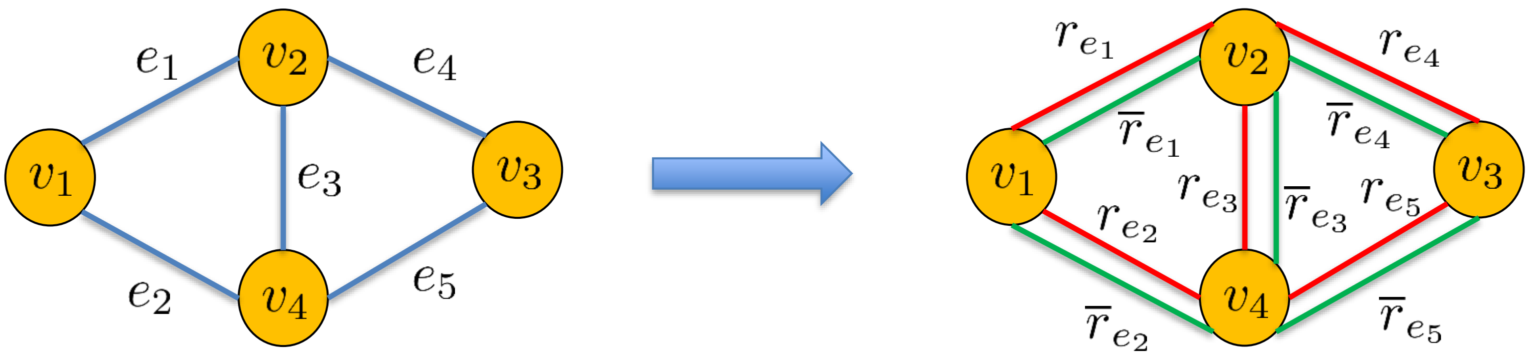
- Agents are the vertices V.
- For each edge $e \in E$ we have two resources r_e, $\overline{r}_e$ ($2|E|$ resources in total).
- Each player v has two strategies, $s_v = \{r_e : e \text{ is incident to } v\}$ and $\overline{s}_v = \{\overline{r}_e : e \text{ is incident to } v\}$.
- The cost $c_{r_e}/c_{\overline{r}_e}$ of a resource r_e or $\overline{r}_e$ is 0 if one agent uses it and w_e if two players use it.

This transformation is poly-time.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete)**.** *The problem of computing Pure Nash Equilibria in Congestion Games is PLS-complete.*

Proof cont. We now reduce LOCAL MAX-CUT to PNE CONGESTION GAMES.



Each agent has two strategies, red and green.

Say agents v_1, v_2 choose red and v_3, v_4 choose green. Cost of v_1, v_2 is w_{e_1} and of v_3, v_4 is w_{e_5}.

The complexity of Pure Nash Eq.

Proof cont. Observe that there is a bijection between strategy profiles of this congestion game and cuts of the given graph G.

The complexity of Pure Nash Eq.

Proof cont. Observe that there is a bijection between strategy profiles of this congestion game and cuts of the given graph G. Given a cut $(S, \overline{S})$ (agents in S choose red and agents in $\overline{S}$ choose green strategy), we have that

$$w(S, \overline{S}) = \sum_{e=(u,v):u \in S, v \in \overline{S}} w_e = \sum_{e \in E} w_e - \Phi(s, \overline{s}).$$

The complexity of Pure Nash Eq.

Proof cont. Observe that there is a bijection between strategy profiles of this congestion game and cuts of the given graph G. Given a cut $(S, \overline{S})$ (agents in S choose red and agents in $\overline{S}$ choose green strategy), we have that

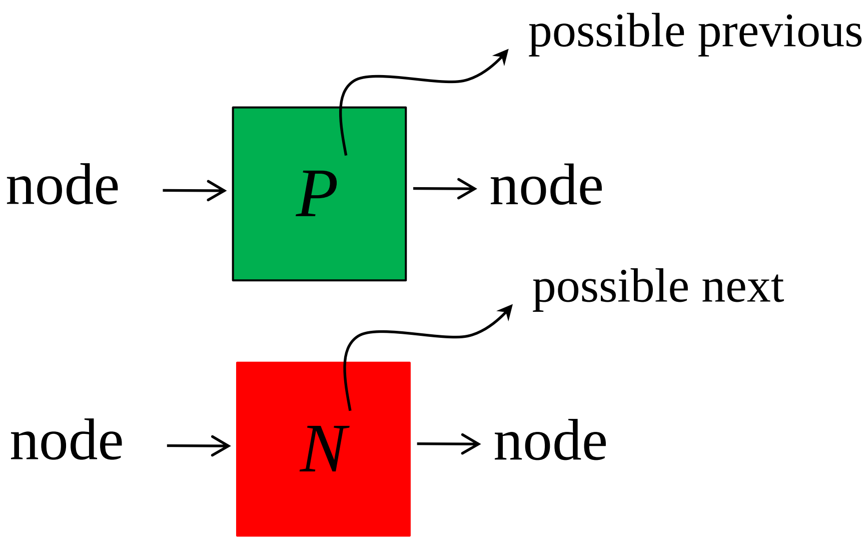
$$w(S, \overline{S}) = \sum_{e=(u,v):u \in S, v \in \overline{S}} w_e = \sum_{e \in E} w_e - \Phi(s, \overline{s}).$$

Therefore:

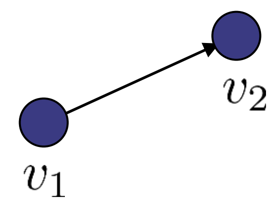
- Cuts with larger weight correspond to strategy profiles with smaller potential.
- Local maxima of cuts of G correspond to local minima of the potential function.

The class PPAD

Suppose that an exponentially large graph with vertex set $\{0,1\}^n$ (i.e, 2^n vertices) is defined by two circuits:



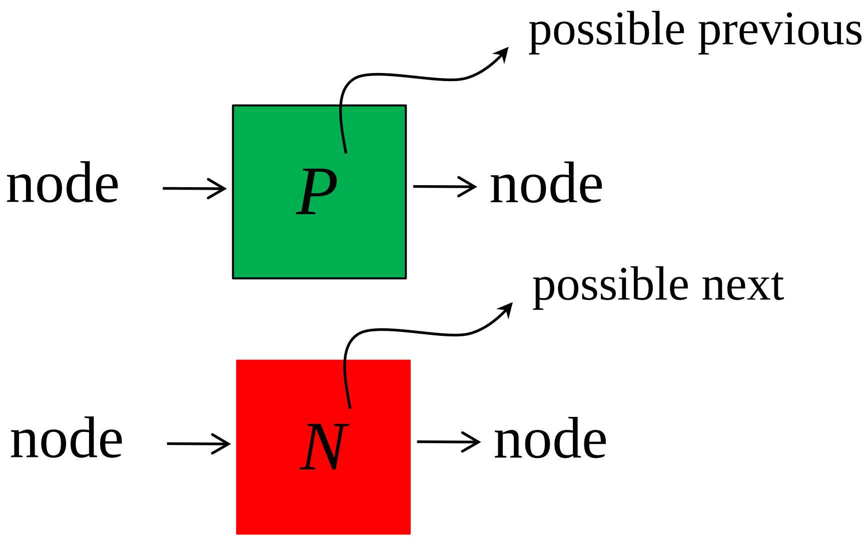
Example:



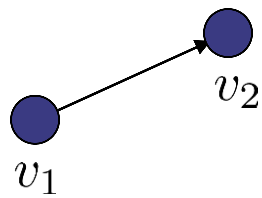
$$N(v_1) = v_2 \text{ and } P(v_2) = v_1$$

The class PPAD

Suppose that an exponentially large graph with vertex set $\{0,1\}^n$ (i.e, 2^n vertices) is defined by two circuits:



Example:



$N(v_1) = v_2$ and $P(v_2) = v_1$

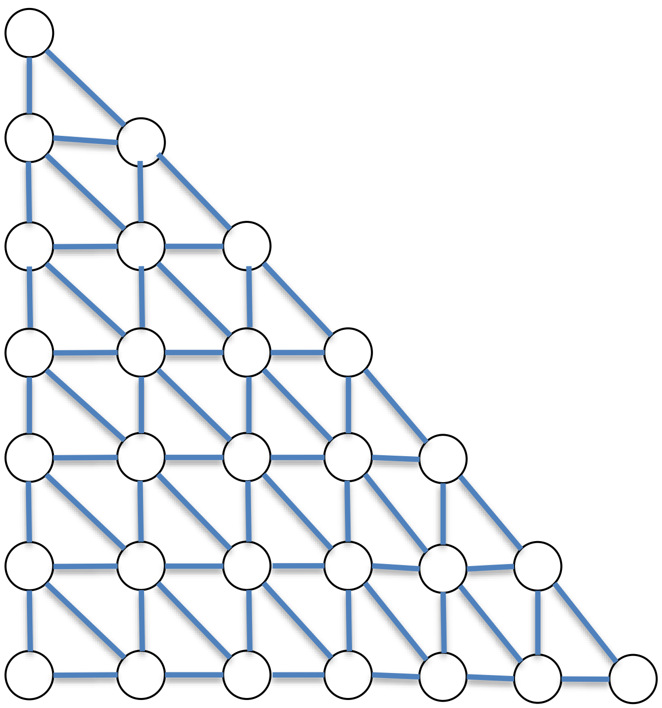
Canonical Problem:

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

PPAD (Papadimitriou 94'): All problems in FNP reducible to END OF THE LINE.

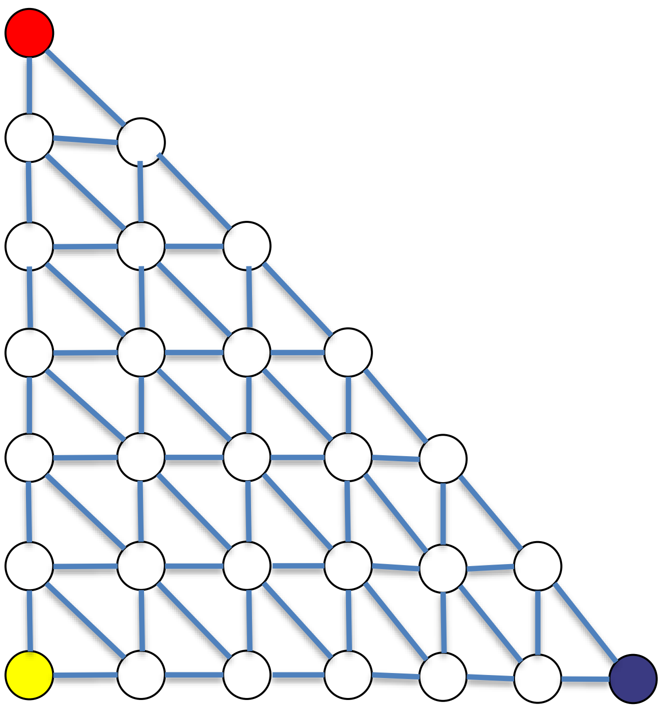
2D Sperner's Lemma

Theorem (A trichromatic triangle always exist)**.** *Consider triangulation of 2d simplex Δ and a proper 3-coloring, that assign each vertex a different color and inside vertices on each edge of Δ use only the two colors of the respective endpoints. Then there always exists a trichromatic triangle (odd in number!).*



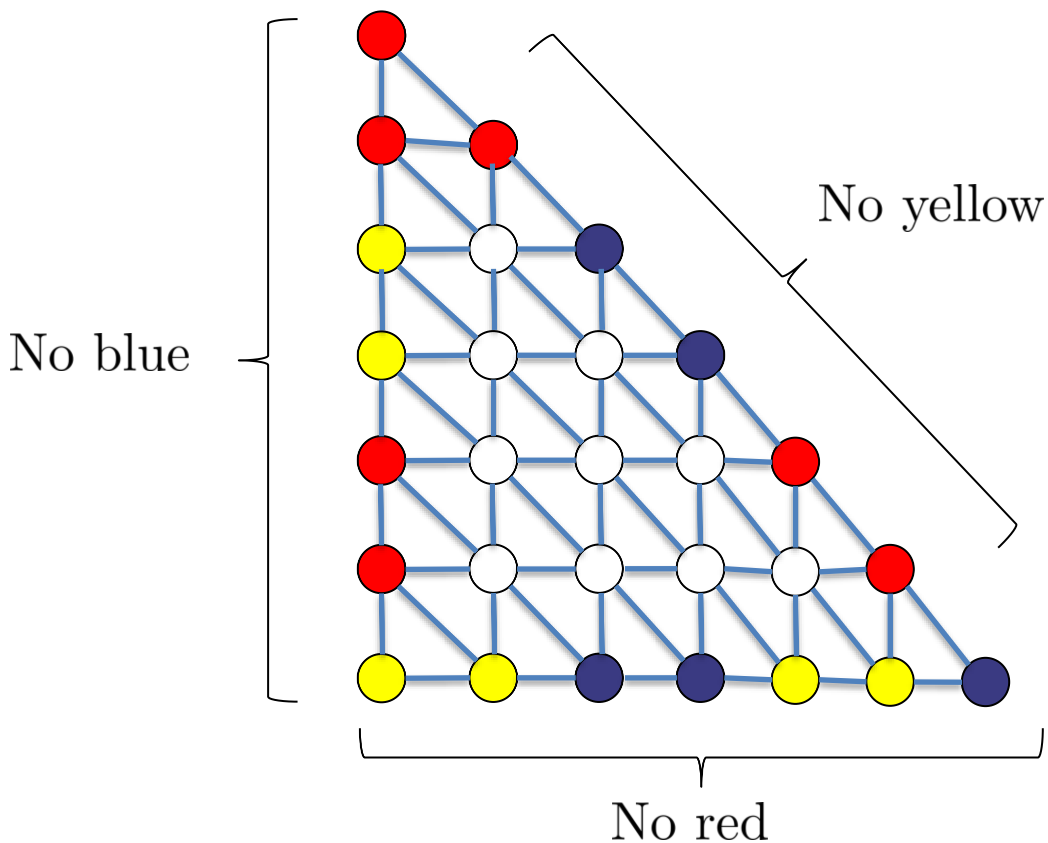
2D Sperner's Lemma

Theorem (A trichromatic triangle always exist). *Consider triangulation of 2d simplex* Δ *and a proper 3-coloring, that assign each vertex a different color and inside vertices on each edge of* Δ *use only the two colors of the respective endpoints. Then there always exists a trichromatic triangle (odd in number!).*



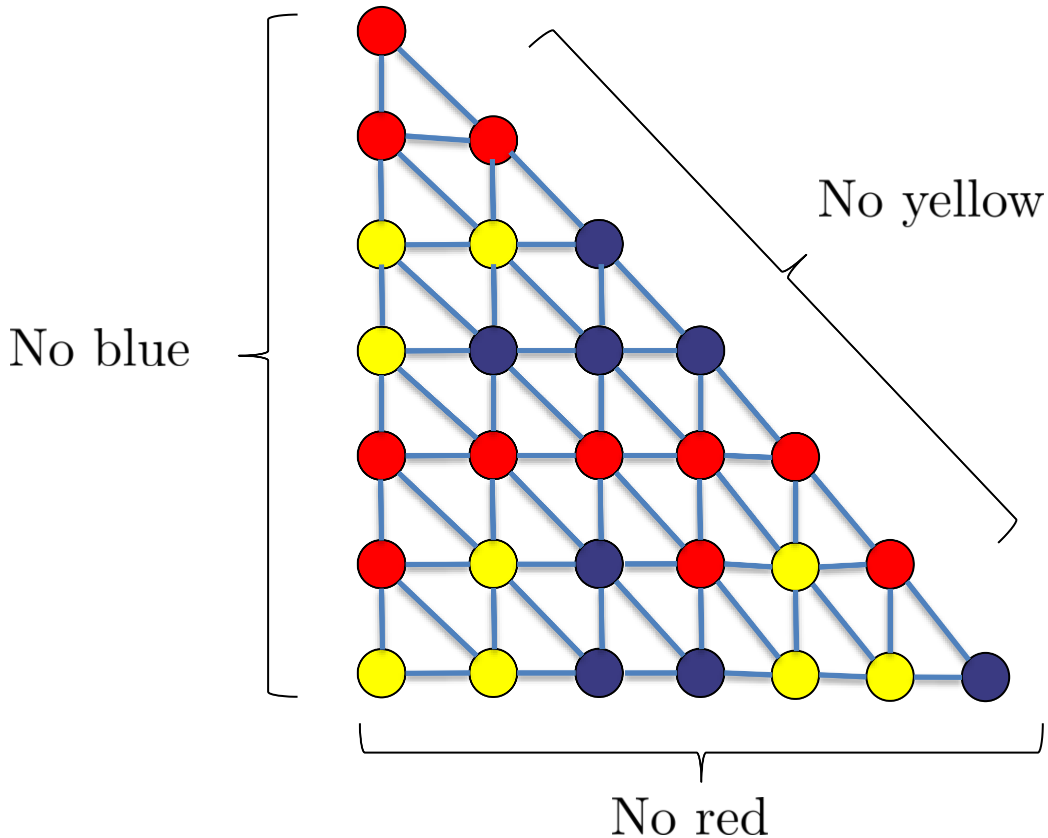
2D Sperner's Lemma

Theorem (A trichromatic triangle always exist). *Consider triangulation of 2d simplex* Δ *and a proper 3-coloring, that assign each vertex a different color and inside vertices on each edge of* Δ *use only the two colors of the respective endpoints. Then there always exists a trichromatic triangle (odd in number!).*



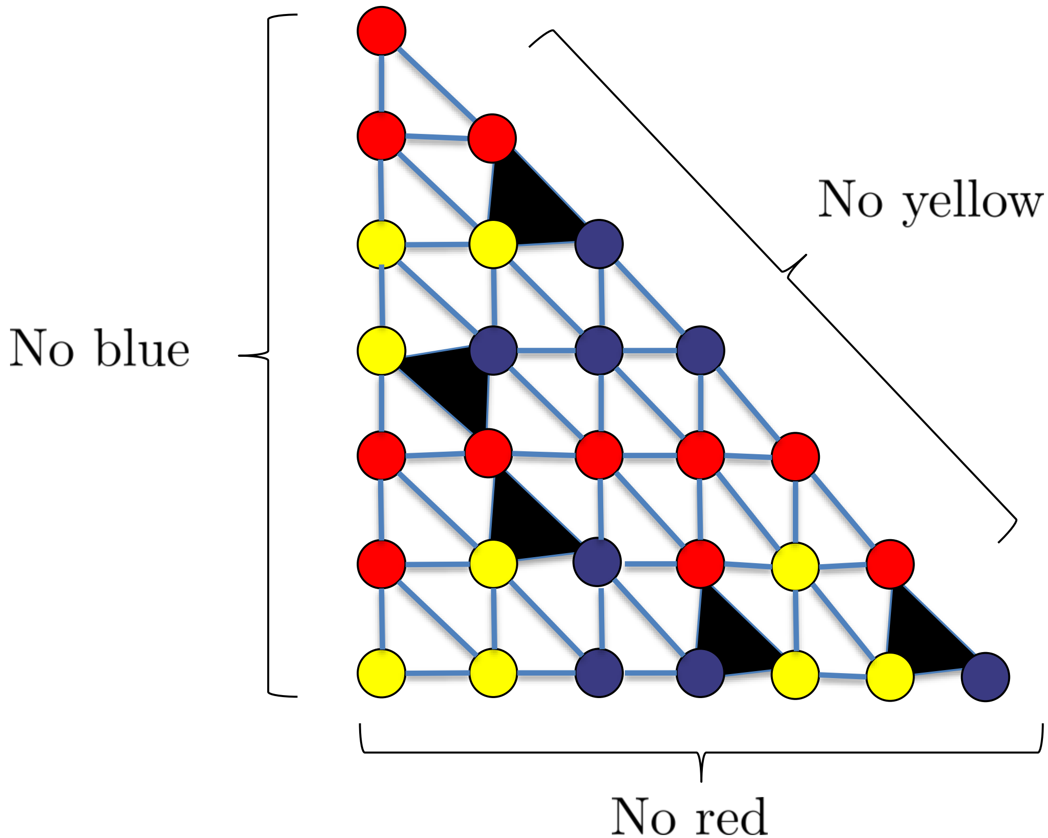
2D Sperner's Lemma

Theorem (A trichromatic triangle always exist). *Consider triangulation of 2d simplex Δ and a proper 3-coloring, that assign each vertex a different color and inside vertices on each edge of Δ use only the two colors of the respective endpoints. Then there always exists a trichromatic triangle (odd in number!).*



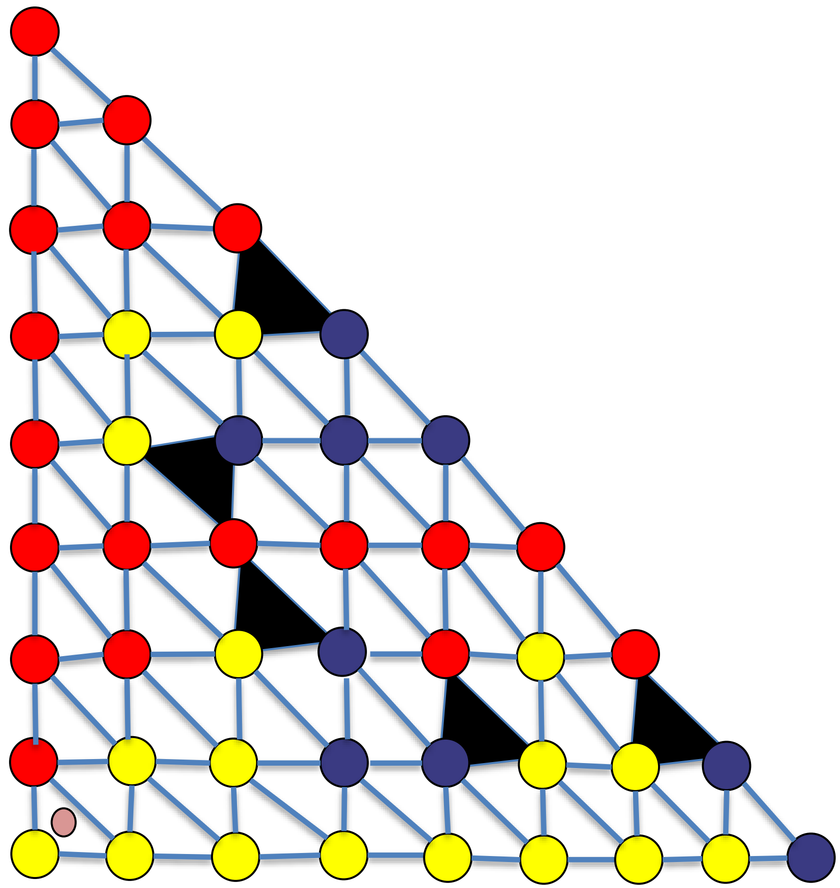
2D Sperner's Lemma

Theorem (A trichromatic triangle always exist). *Consider triangulation of 2d simplex* Δ *and a proper 3-coloring, that assign each vertex a different color and inside vertices on each edge of* Δ *use only the two colors of the respective endpoints. Then there always exists a trichromatic triangle (odd in number!).*



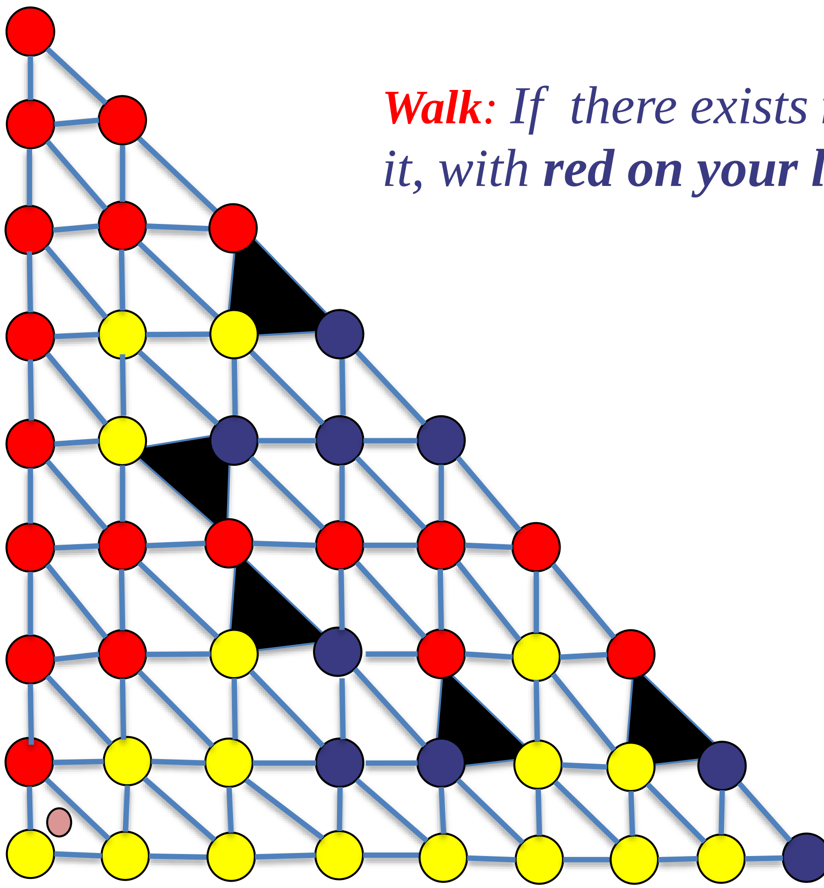
2D Sperner's Lemma

Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.



2D Sperner's Lemma

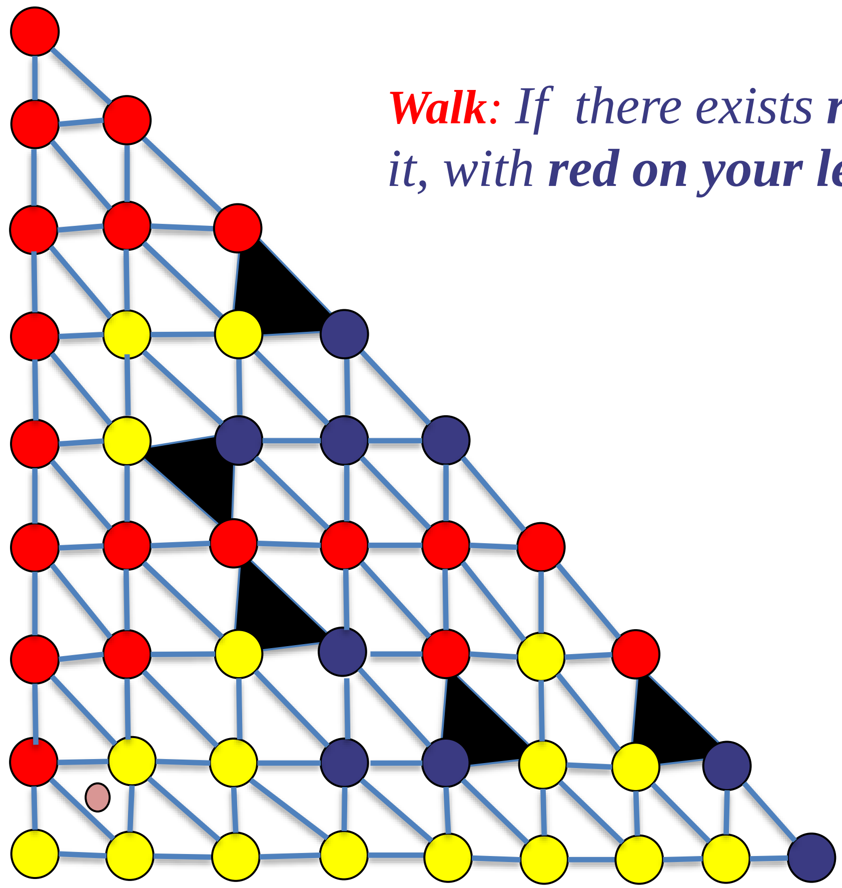
Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.



***Walk**: If there exists **red-yellow** edge cross it, with **red on your left**.*

2D Sperner's Lemma

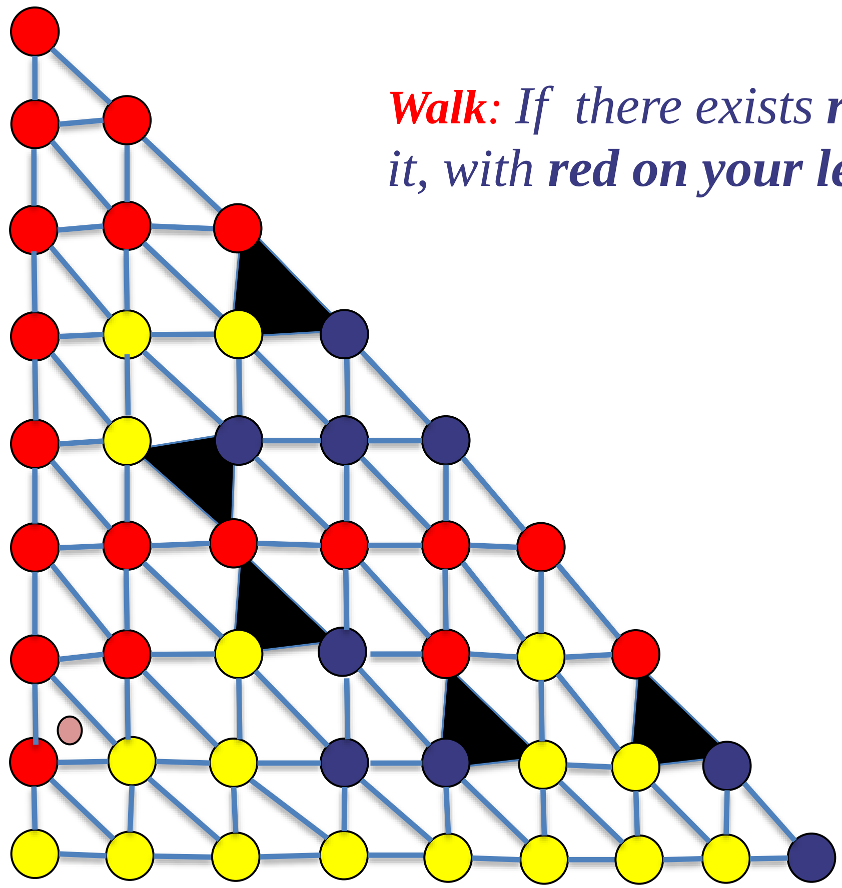
Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.



Walk: *If there exists* ***red-yellow*** *edge cross it, with* ***red on your left****.*

2D Sperner's Lemma

Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.

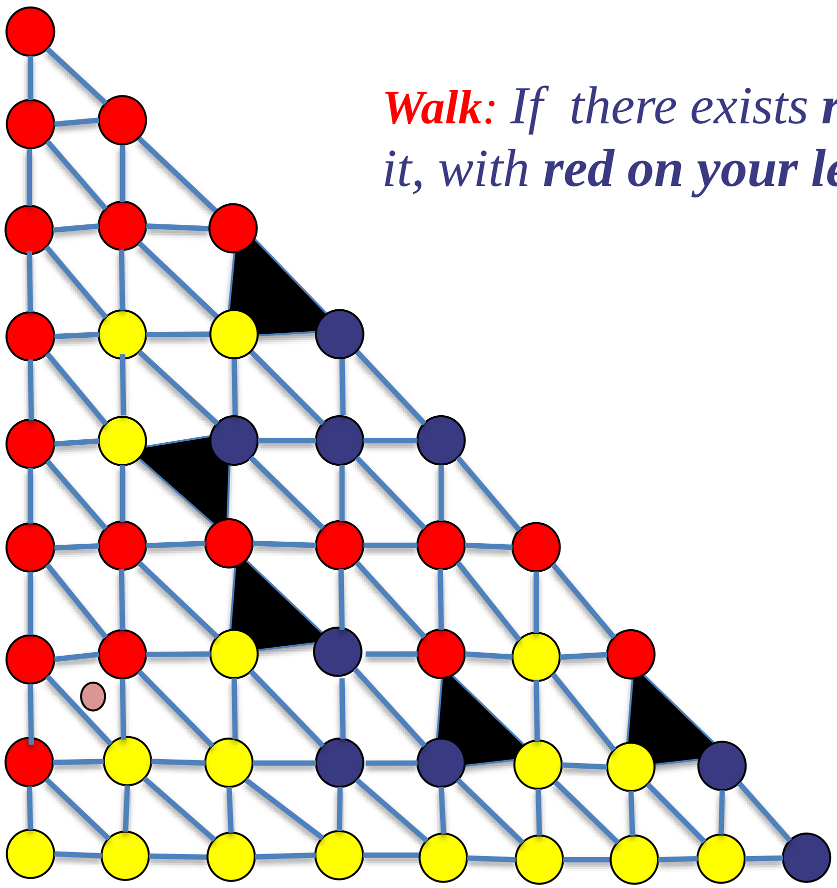


***Walk**: If there exists **red-yellow** edge cross it, with **red on your left**.*

2D Sperner's Lemma

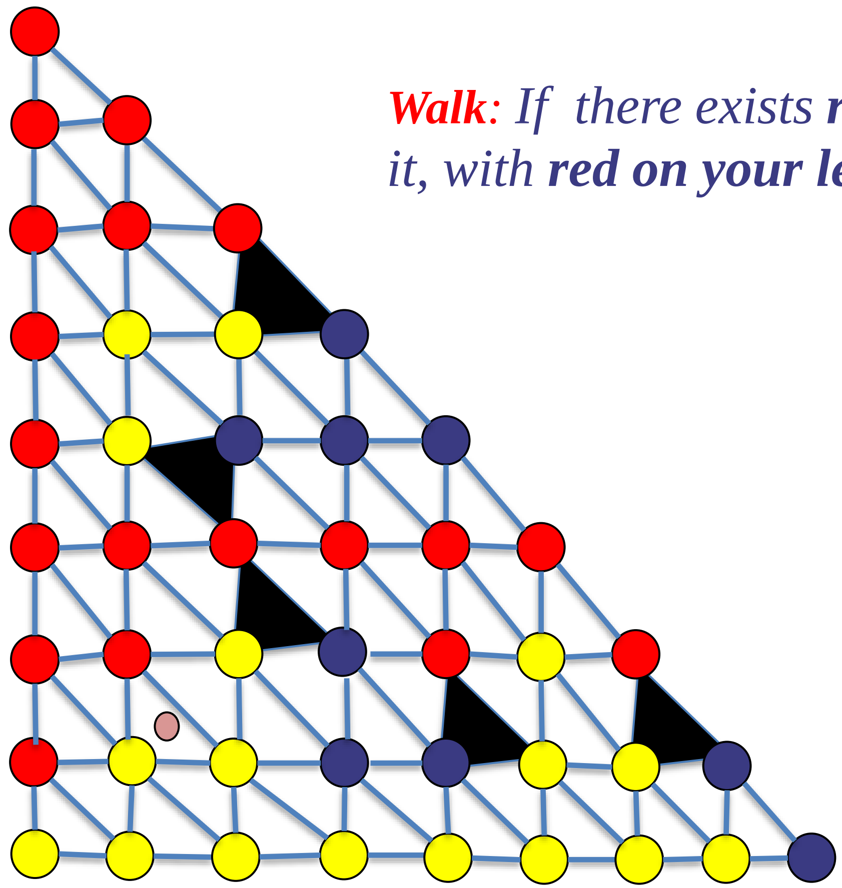
Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.

Walk: *If there exists **red-yellow** edge cross it, with **red on your left**.*



2D Sperner's Lemma

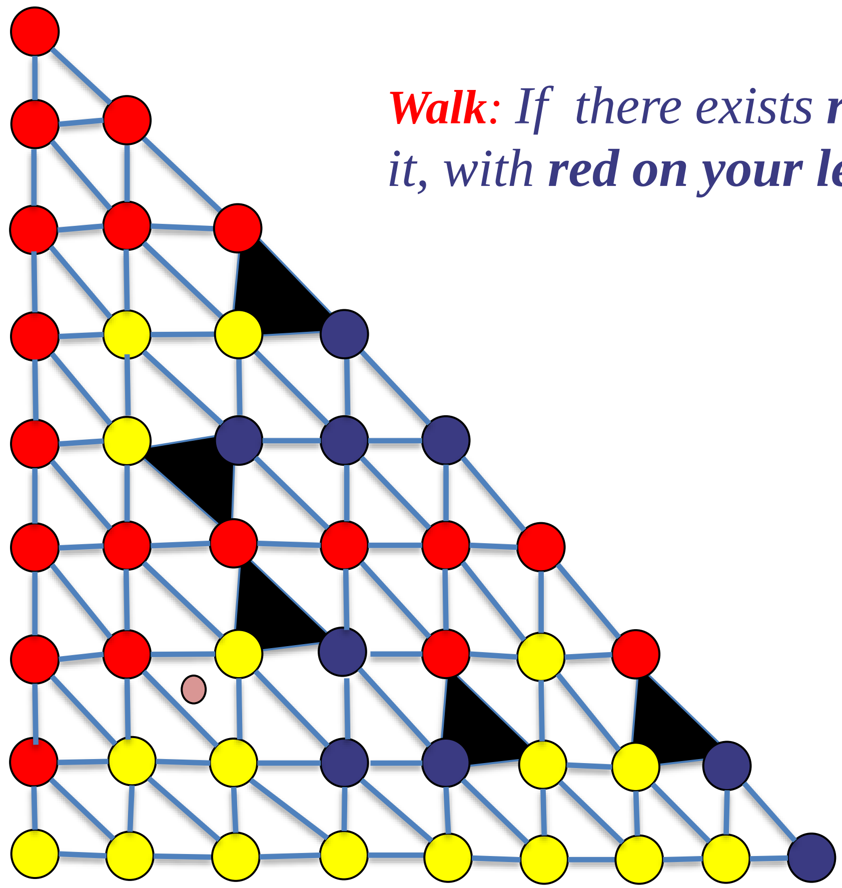
Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.



***Walk**: If there exists **red-yellow** edge cross it, with **red on your left**.*

2D Sperner's Lemma

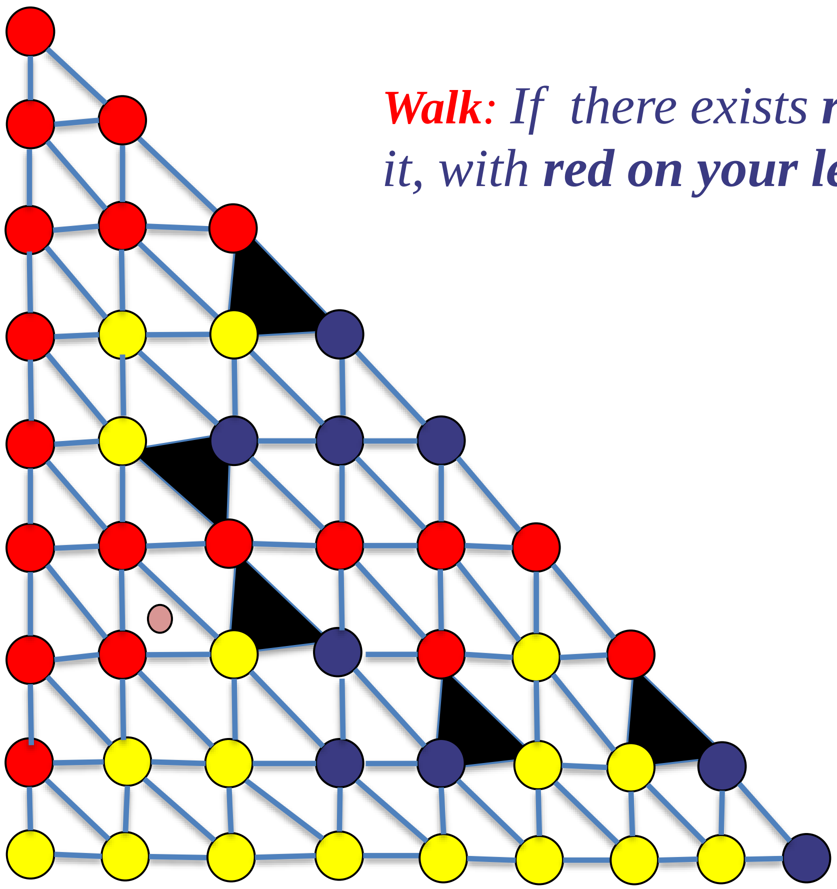
Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.



***Walk**: If there exists **red-yellow** edge cross it, with **red on your left**.*

2D Sperner's Lemma

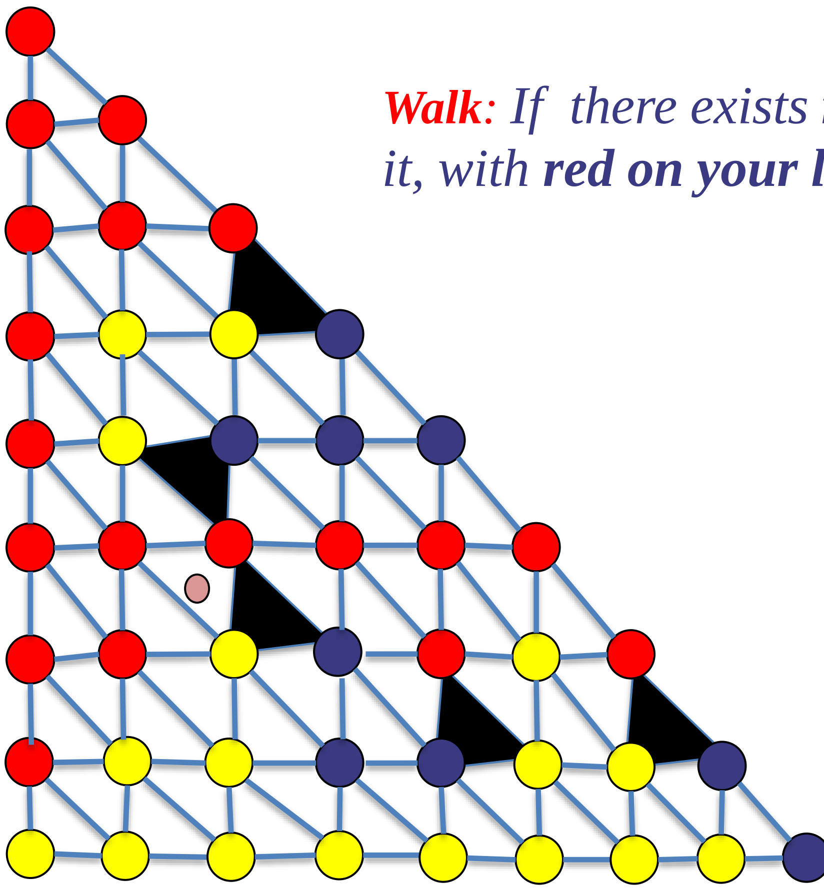
Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.



***Walk**: If there exists **red-yellow** edge cross it, with **red on your left**.*

2D Sperner's Lemma

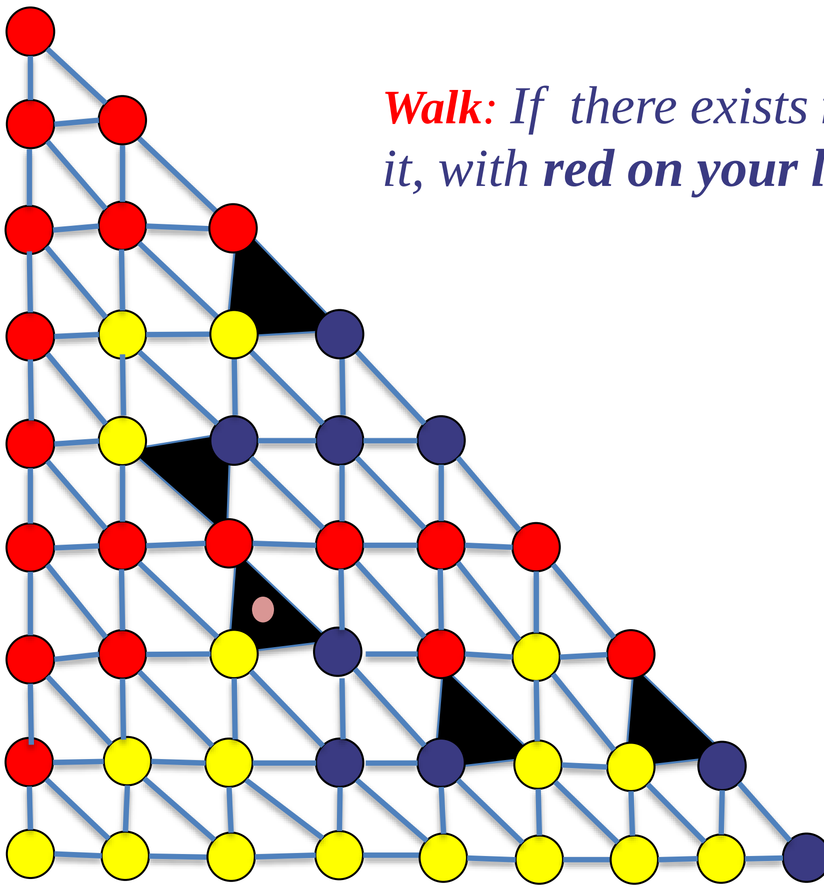
Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.



***Walk**: If there exists **red-yellow** edge cross it, with **red on your left**.*

2D Sperner's Lemma

Proof. We introduce an outer boundary for conveniece that does not create new trichromatic triangles. Next we define a directed walk starting from the bottom-left triangle.

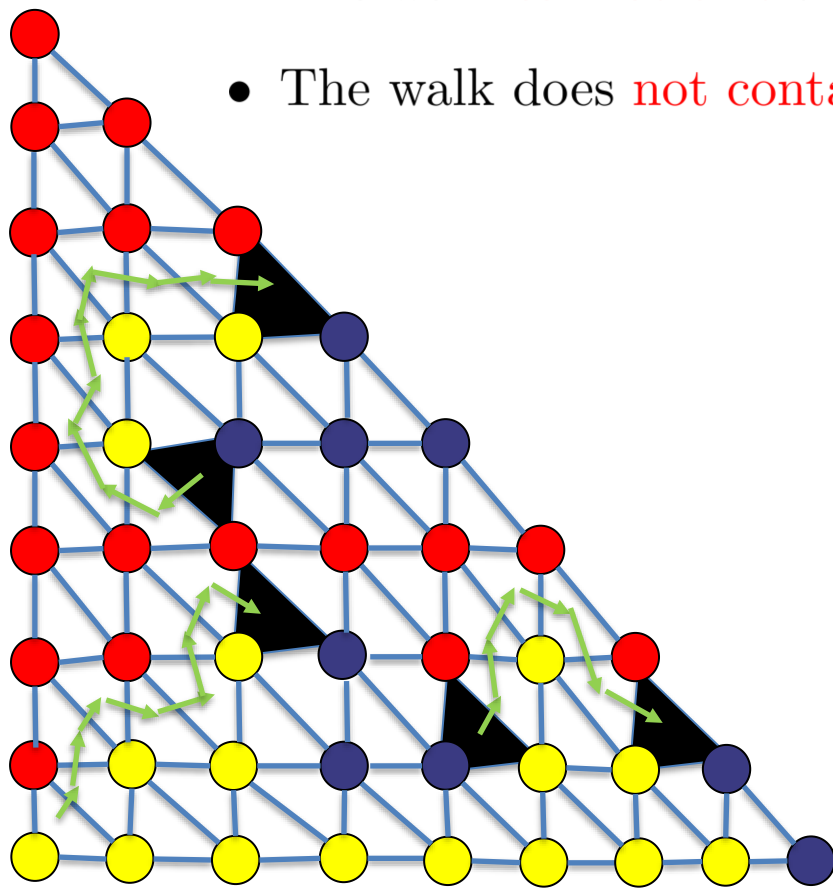


Walk: *If there exists* ***red-yellow*** *edge cross it, with* ***red on your left***.

2D Sperner's Lemma

Proof cont.

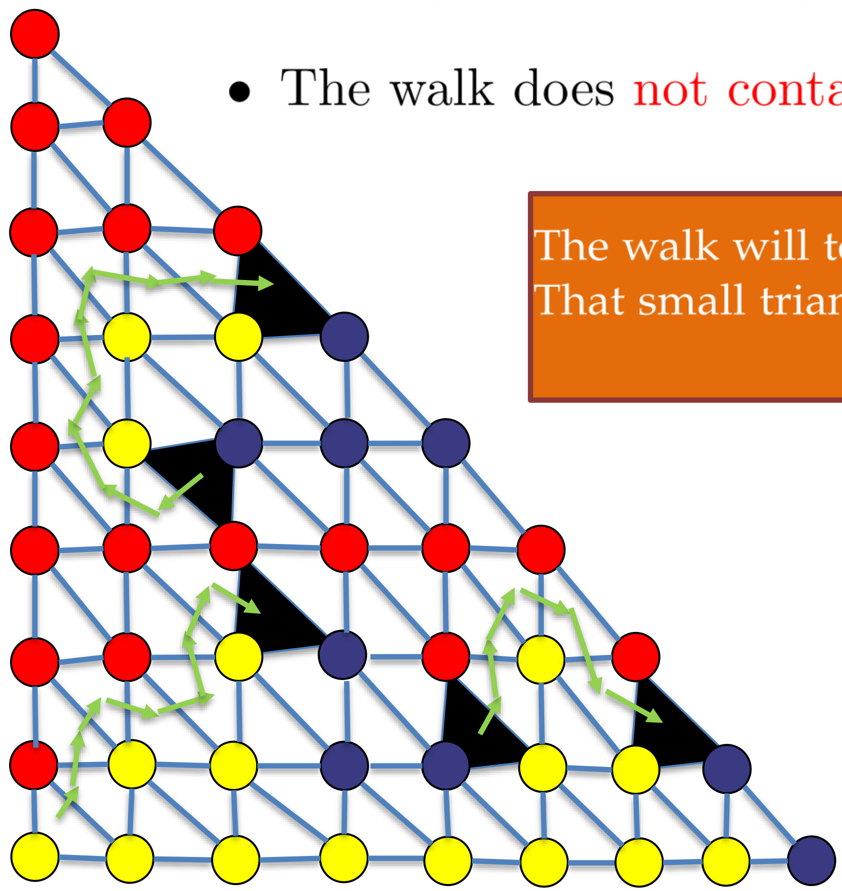
- The walk cannot exit the outer triangle (why?).
- The walk does not contain ρ shapes (why?).



2D Sperner's Lemma

Proof cont.

- The walk cannot exit the outer triangle (why?).
- The walk does not contain ρ shapes (why?).

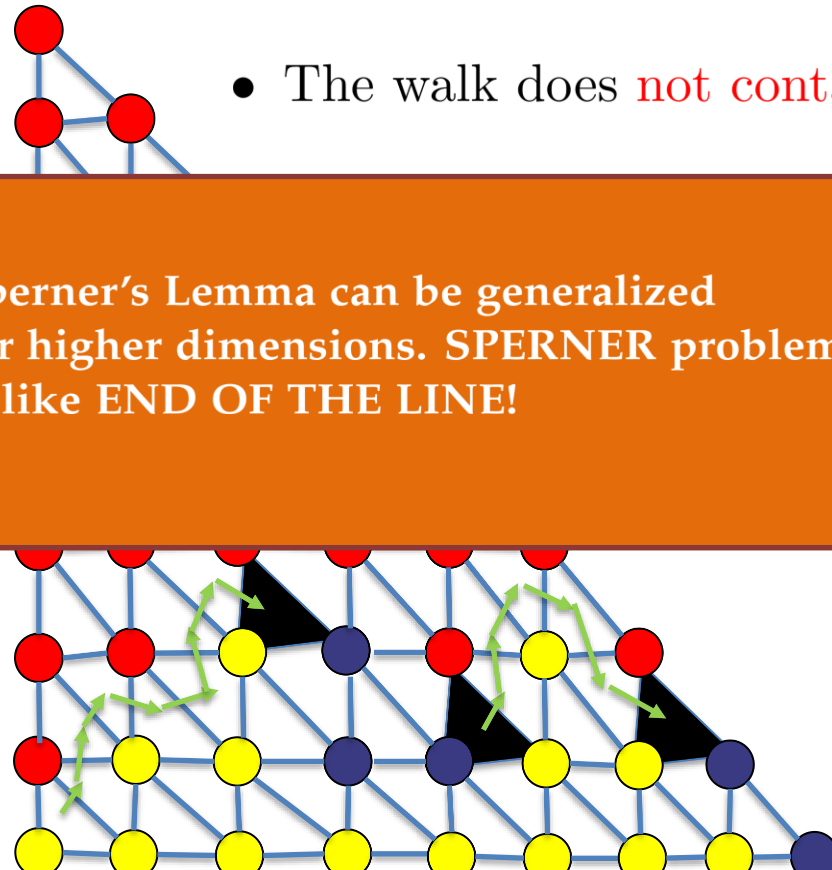


The walk will terminate incide somewhere!
That small triangle should be trichromatic!

2D Sperner's Lemma

Proof cont.

- The walk cannot exit the outer triangle (why?).
- The walk does not contain ρ shapes (why?).



...te incide somewhere!
...ould be trichromatic!

Sperner's Lemma can be generalized for higher dimensions. SPERNER problem is like END OF THE LINE!

BROUWER

Definition (BROUWER). *The problem BROUWER is defined below:*

Input: *A poly-time algorithm* Π_F *for the evaluation of a function* $F : [0,1]^m \to [0,1]^m$, *a constant K such that F is K-Lipschitz and accuracy* ϵ.

Output: A (rational) point x so that

$$\|F(x) - x\|_\infty \le \epsilon,$$

i.e., x is an approximate fixed point.

BROUWER

Definition (BROUWER). *The problem BROUWER is defined below:*

Input: *A poly-time algorithm Π_F for the evaluation of a function $F : [0,1]^m \to [0,1]^m$, a constant K such that F is K-Lipschitz and accuracy ϵ.*

Output: A (rational) point x so that

$$\|F(x) - x\|_\infty \leq \epsilon,$$

i.e., x is an approximate fixed point.

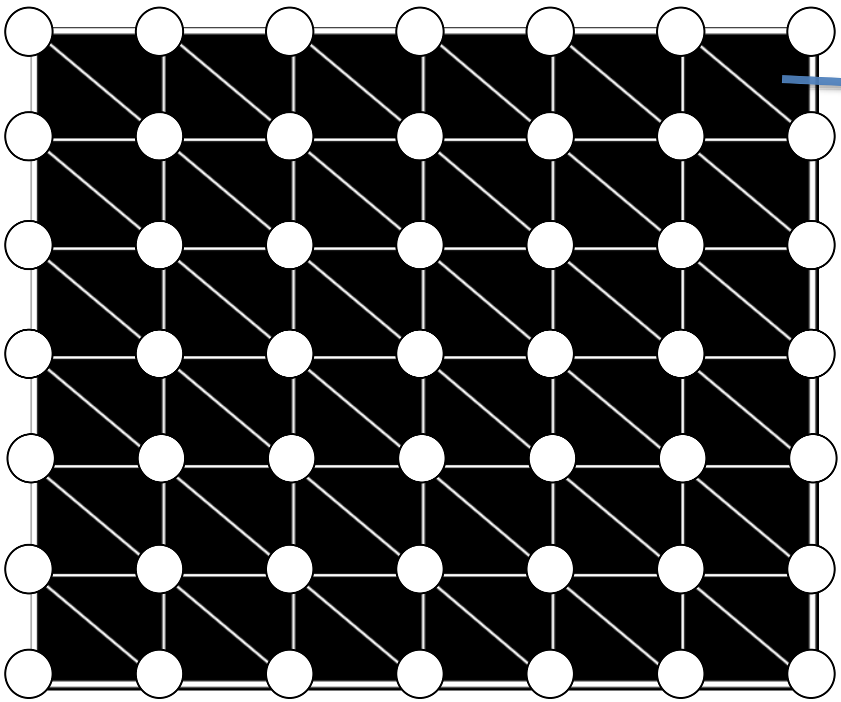
We will show that

BROUWER $\to$ SPERNER

2D BROUWER reduction to SPERNER

Let $F : [0,1]^2 \to [0,1]^2$. By uniform continuity there exists a $\delta(\epsilon)$ so that

$$\|x - y\|_\infty \leq \delta \Rightarrow \|F(x) - F(y)\|_\infty \leq \epsilon.$$

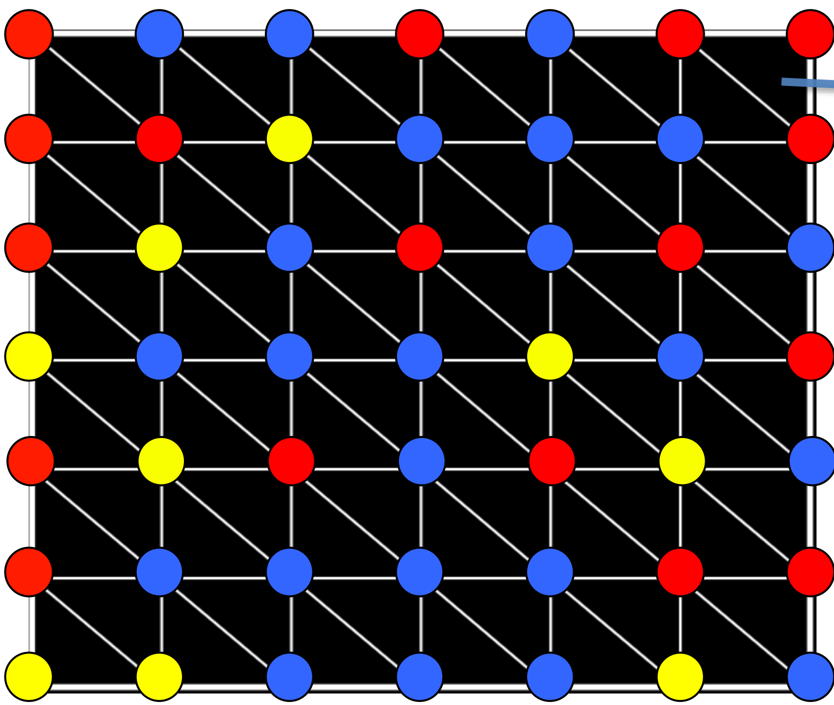


Diameter of each cell is at most $\delta(\epsilon)$

2D BROUWER reduction to SPERNER

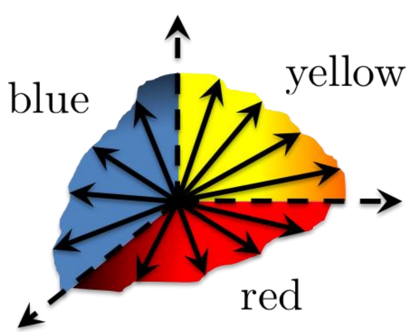
Let $F : [0,1]^2 \to [0,1]^2$. By uniform continuity there exists a $\delta(\epsilon)$ so that

$$\|x - y\|_\infty \leq \delta \Rightarrow \|F(x) - F(y)\|_\infty \leq \epsilon.$$



Diameter of each cell is at most $\delta(\epsilon)$

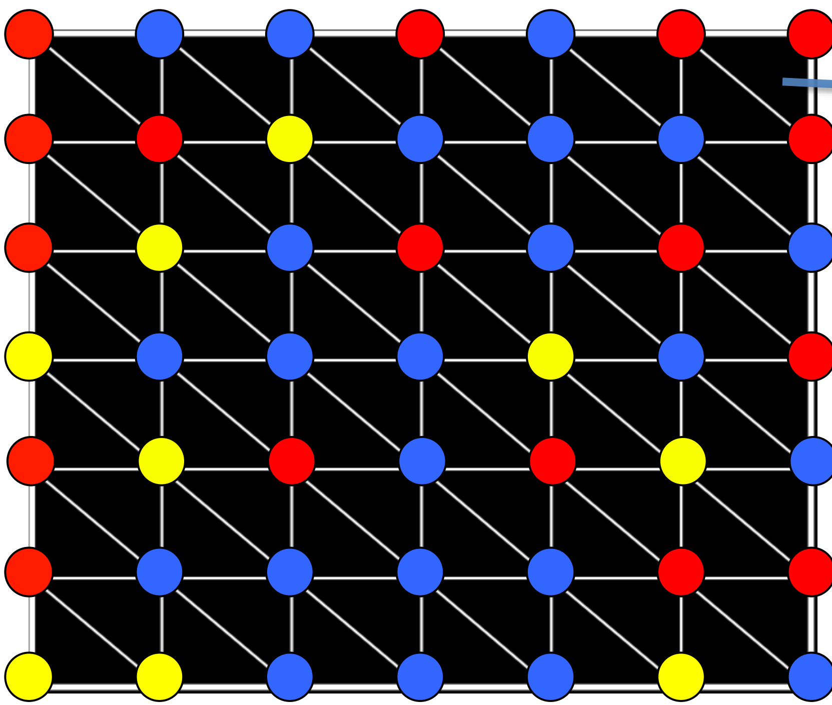
Color the nodes of the triangulation according to the direction of $f(x) - x$.



2D BROUWER reduction to SPERNER

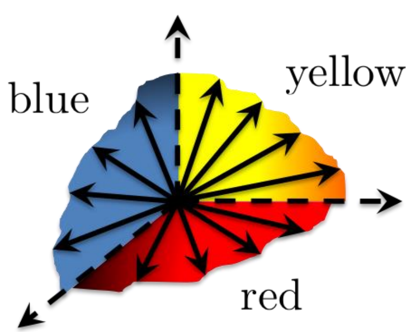
Let $F : [0,1]^2 \to [0,1]^2$. By uniform continuity there exists a $\delta(\epsilon)$ so that

$$\|x - y\|_\infty \leq \delta \Rightarrow \|F(x) - F(y)\|_\infty \leq \epsilon.$$



Diameter of each cell is at most $\delta(\epsilon)$

Color the nodes of the triangulation according to the direction of $f(x) - x$.

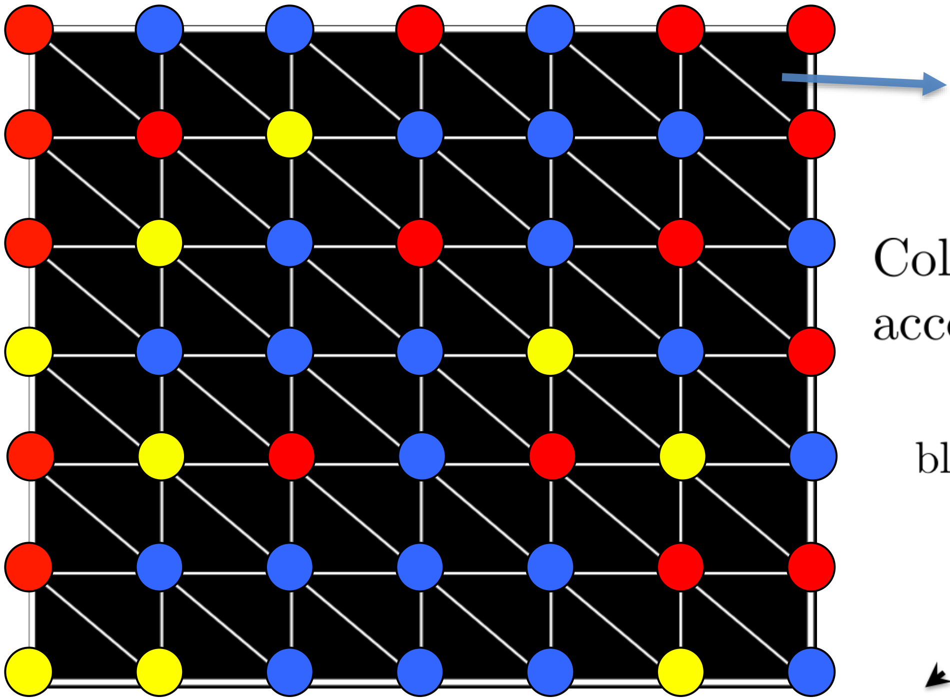


Tie-break at the boundary angles, so that the resulting coloring respects the boundary conditions!

2D BROUWER reduction to SPERNER

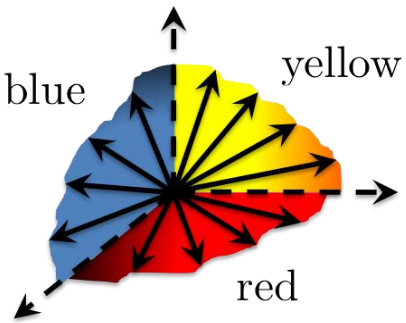
Claim. *Choose* $\delta = \min(\delta(\epsilon), \epsilon)$ *and let* v^y *be the yellow vertex of a trichromatic triangle. It holds that*

$$\|F(v^y) - v^y\|_\infty \leq 2\epsilon.$$



Diameter of each cell is at most $\delta(\epsilon)$

Color the nodes of the triangulation according to the direction of $f(x) - x$.



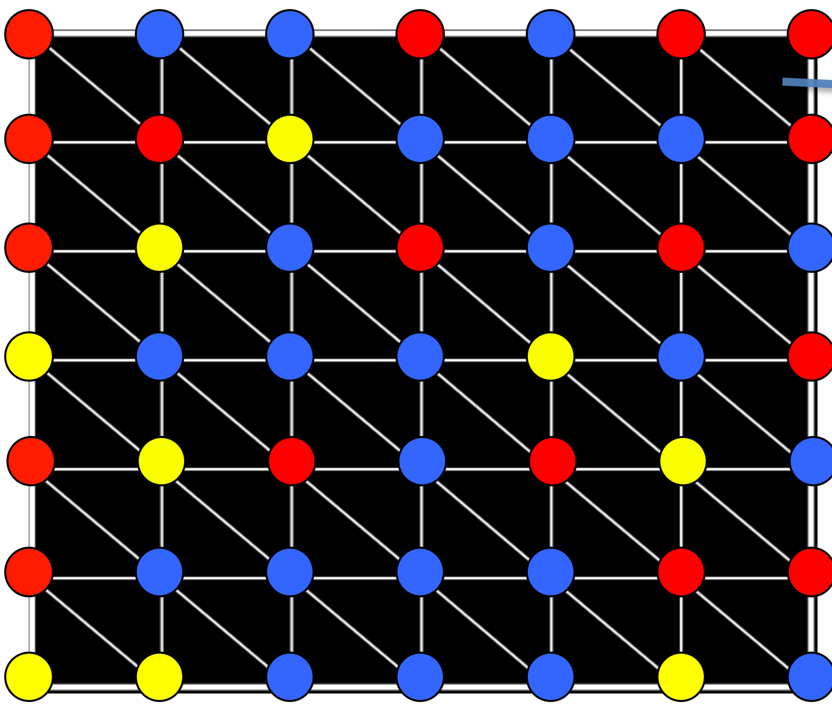
Tie-break at the boundary angles, so that the resulting coloring respects the boundary conditions!

2D BROUWER reduction to SPERNER

Claim. *Choo* ... *ellow*
vertex of a tri ...

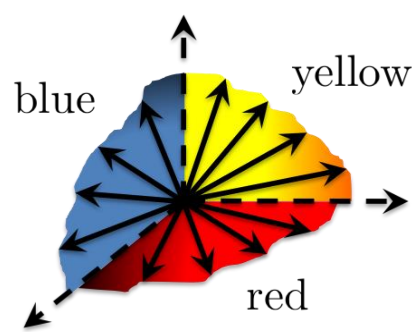
$\|f(v^*) - v^*\|_\infty \leq 2\epsilon.$

This will be in HW2.



Diameter of each cell is at most $\delta(\epsilon)$

Color the nodes of the triangulation according to the direction of $f(x) - x$.



Tie-break at the boundary angles, so that the resulting coloring respects the boundary conditions!

NASH reduction to BROUWER

We will not see the proof, just an idea.

NASH reduction to BROUWER

We will not see the proof, just an idea.

Consider the 2×2 mathcing pennies.

	H	T
H	**1, -1**	**-1,1**
T	**-1, 1**	**1,-1**

Consider the function f from the proof of Nash.

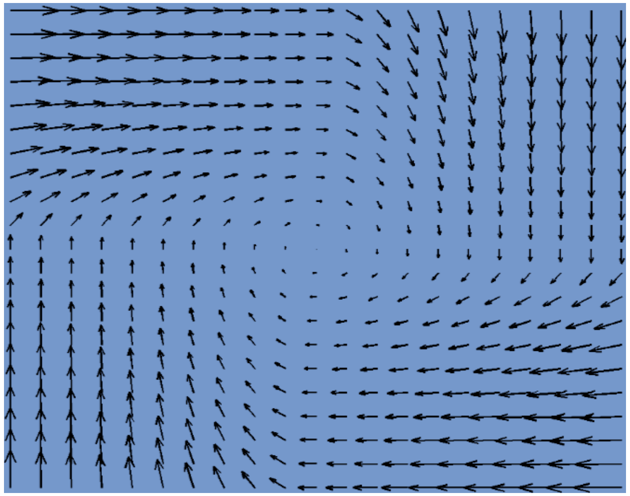
$$f_{is_i}(x) = \frac{x_i(s_i) + \max\{u_i(s_i; x_{-i}) - u_i(x), 0\}}{1 + \sum_{s' \in S_i} \max\{u_i(s'; x_{-i}) - u_i(x), 0\}}$$

NASH reduction to BROUWER

$$f_{is_i}(x) = \frac{x_i(s_i) + \max\{u_i(s_i; x_{-i}) - u_i(x), 0\}}{1 + \sum_{s' \in S_i} \max\{u_i(s'; x_{-i}) - u_i(x), 0\}}$$

1, -1	**-1,1**
-1, 1	**1,-1**

Draw the vector field for $f(x) - x$.

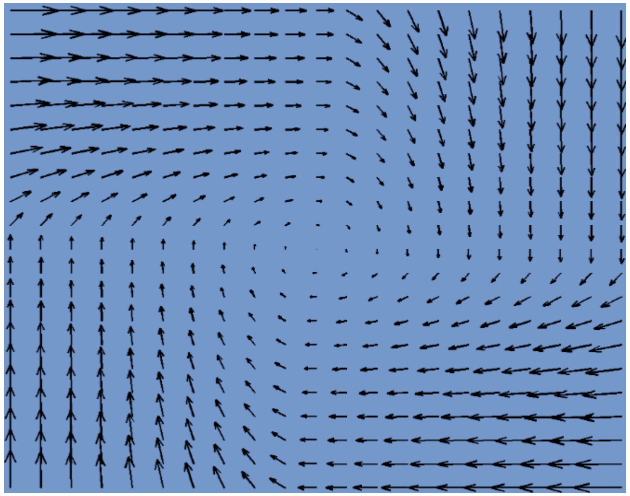


NASH reduction to BROUWER

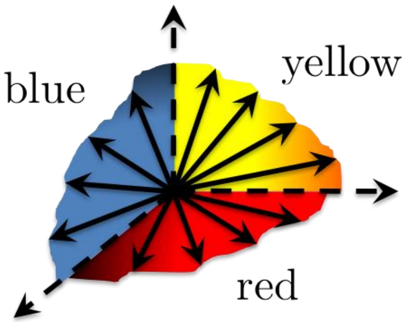
$$f_{is_i}(x) = \frac{x_i(s_i) + \max\{u_i(s_i; x_{-i}) - u_i(x), 0\}}{1 + \sum_{s' \in S_i} \max\{u_i(s'; x_{-i}) - u_i(x), 0\}}$$

1, -1	-1,1
-1, 1	1,-1

Draw the vector field for $f(x) - x$.



Color the points according to

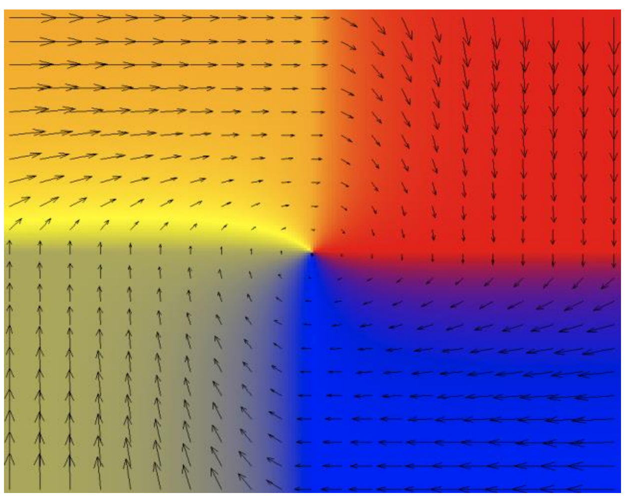


NASH reduction to BROUWER

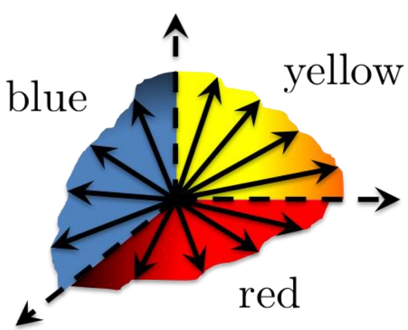
$$f_{is_i}(x) = \frac{x_i(s_i) + \max\{u_i(s_i; x_{-i}) - u_i(x), 0\}}{1 + \sum_{s' \in S_i} \max\{u_i(s'; x_{-i}) - u_i(x), 0\}}$$

1, -1	**-1,1**
-1, 1	**1,-1**

Draw the vector field for $f(x) - x$.



Color the points according to

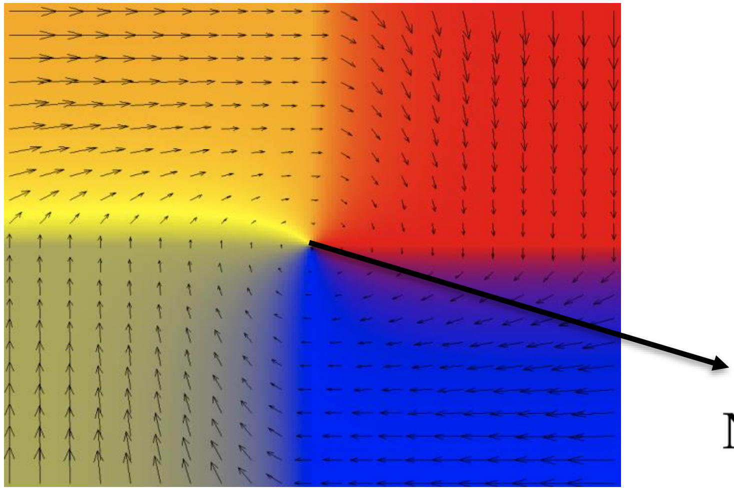


NASH reduction to BROUWER

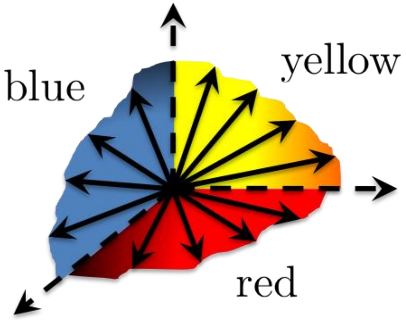
$$f_{is_i}(x) = \frac{x_i(s_i) + \max\{u_i(s_i; x_{-i}) - u_i(x), 0\}}{1 + \sum_{s' \in S_i} \max\{u_i(s'; x_{-i}) - u_i(x), 0\}}$$

1, -1	**-1,1**
-1, 1	**1,-1**

Draw the vector field for $f(x) - x$.

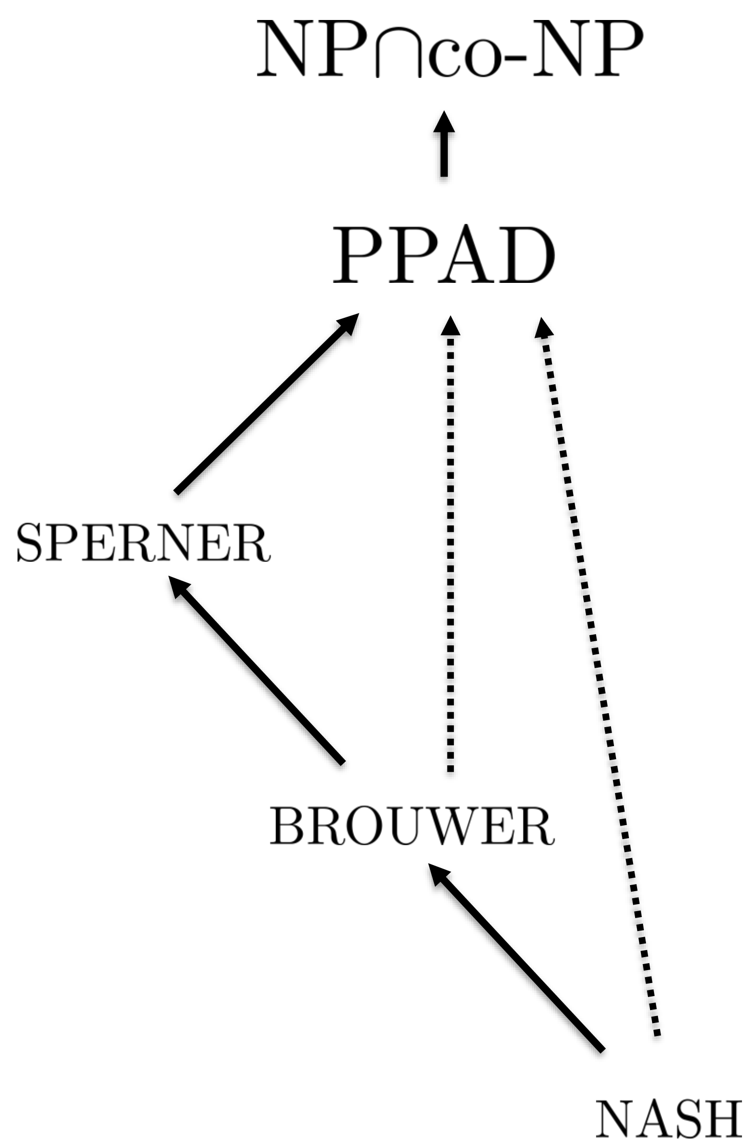


Color the points according to

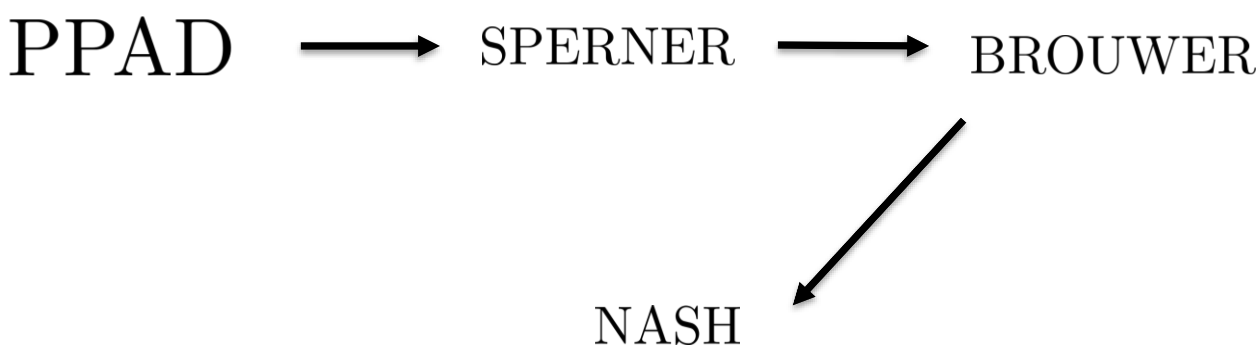


Nash equilibrium $(\frac{1}{2}, \frac{1}{2})$!

Inclusions we showed



Theorem ((NASH is PPAD-complete) Daskalakis, Goldberg, Papadimitriou).
NASH is PPAD-complete.



Inclusions: The full picture