

L06 Complexity Classes and Equilibrium Computation

CS 280 Algorithmic Game Theory
Ioannis Panageas

Based in part on C. Daskalakis's slides and T. Roughgarden's notes.

Standard Complexity Classes

- **P:** Decision problems solvable by a deterministic algorithm in polynomial time.

Standard Complexity Classes

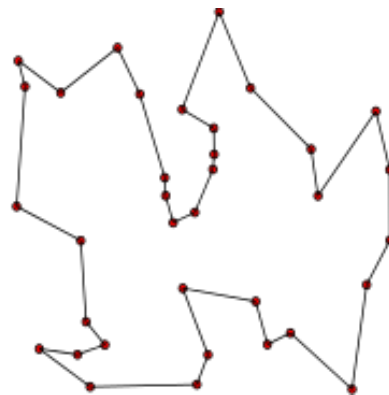
- **P**: Decision problems solvable by a deterministic algorithm in polynomial time.
- **NP**: Decision problems whose yes instances have polynomial-size certificates verifiable in polynomial time.
- **coNP**: Decision problems whose no instances have such certificates.

Problems in P

- Problems that can be solved in **polynomial time**.
 - Decision version of **shortest path** (is the shortest path at most L (yes/no)?)
 - Decision version of finding the **maximum** number of a list (is the maximum at most M (yes/no)?)
 - Is n a **prime** number (yes/no)?

Problems in NP

- **NP:** Yes instances have polynomial-size certificates that can be verified in polynomial time.
- **Travelling salesperson (decision version):** Given a complete weighted graph and bound L , is there a tour of total weight at most L that visits every vertex once and returns to its start?



Equilibrium Computation

- The goal is to **output one mixed Nash equilibrium** of a given game.
- This is a **search problem**, rather than a yes/no question.

For complexity: exact equilibria in rational two-player games, or ϵ -Nash equilibria with rational $\epsilon > 0$ supplied as input.

Equilibrium Computation

- The goal is to **output one mixed Nash equilibrium** of a given game.
- This is a **search problem**, rather than a yes/no question.
- Examples: compute the sum of two integers (function); decide whether the sum is odd (decision).

For complexity: exact equilibria in rational two-player games, or ϵ -Nash equilibria with rational $\epsilon > 0$ supplied as input.

Function Complexity Classes

- **FP:** Functions that can be computed in polynomial time.
- **FNP:** Search problems with polynomially bounded output length and a polynomial-time verifier for each input/output pair.

Function Complexity Classes

- **FP:** Functions that can be computed in polynomial time.
- **FP:** Search problems with polynomially bounded output length and a polynomial-time verifier for each input/output pair.
- **TFNP:** The total problems in FNP: every input has at least one valid output.

Non-constructive arguments

- **Local search:** Every nonempty finite directed acyclic graph has a sink.

Non-constructive arguments

- **Local search:** Every nonempty finite directed acyclic graph has a sink.
- **Pigeonhole principle:** A map from n elements to $n - 1$ elements has a collision.

Non-constructive arguments

- **Local search:** Every nonempty finite directed acyclic graph has a sink.
- **Pigeonhole principle:** A map from n elements to $n - 1$ elements has a collision.
- **Handshaking lemma:** In a finite undirected graph, an odd-degree vertex implies another odd-degree vertex.

Non-constructive arguments

- **Local search:** Every nonempty finite directed acyclic graph has a sink.
- **Pigeonhole principle:** A map from n elements to $n - 1$ elements has a collision.
- **Handshaking lemma:** In a finite undirected graph, an odd-degree vertex implies another odd-degree vertex.
- **End of the line:** A finite directed graph with an unbalanced vertex has another (in-degree differs from out-degree).

Existence arguments and TFNP

Informally, subclasses of TFNP capture these existence arguments:

PLS: Local improvement and a finite potential.

PPP: The pigeonhole principle.

PPA: Parity in an undirected graph.

PPAD: Parity in a directed graph.

Formally, each class uses polynomial-time reductions to a canonical search problem.

Proving a negative result

Question: How can we rule out every polynomial-time algorithm for problems such as SAT or Nash computation?

That would mean ruling out every fixed polynomial bound.

$$O(n), O(n^2), O(n^3), \dots \text{ etc?}$$

Answer: No such unconditional lower bounds are known for these problems. Reductions give conditional hardness results.

The hardest problems in class C

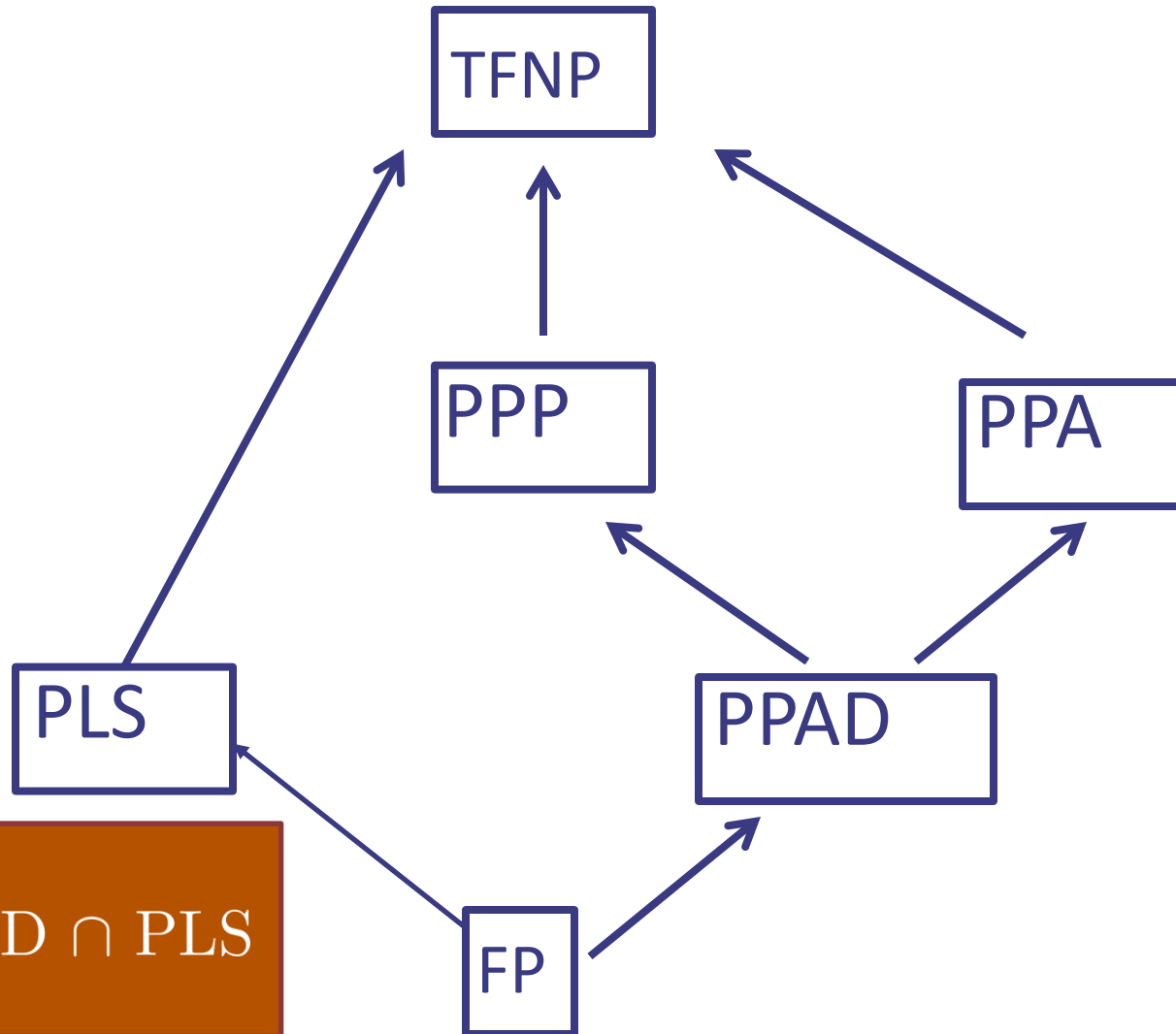
- A problem Q is **C-hard** if every problem P in C reduces to Q in polynomial time.
- An algorithm for Q would solve every problem in C .
- Q is **C-complete** if Q belongs to C and is C-hard.

The hardest problems in class C

- A problem Q is **C-hard** if every problem P in C reduces to Q in polynomial time.
- An algorithm for Q would solve every problem in C.
- Q is **C-complete** if Q belongs to C and is C-hard.
- Examples: **SAT is NP-complete** (Cook, 1971).
- **Nash computation is PPAD-complete**: exact two-player NE, or ϵ -NE with ϵ in the input.
- **Pure NE in congestion games is PLS-complete** (Fabrikant et al., 2004).

AGT and complexity classes

$\text{TFNP} \subseteq \text{FNP}$.
Inclusions need not
be strict.

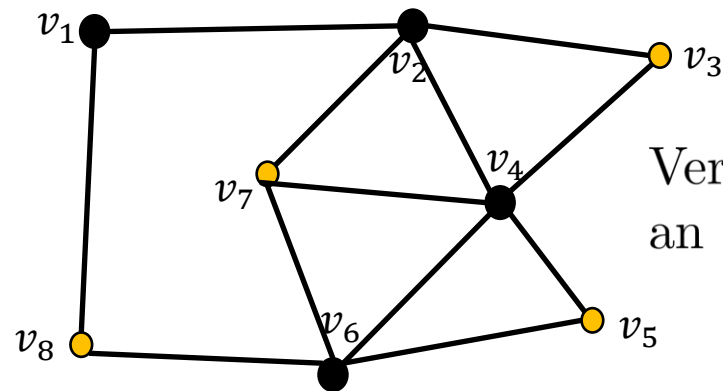


$$\text{CLS} \equiv \text{PPAD} \cap \text{PLS}$$

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 $\geq k$. An independent set is a set $I \subset V$ of vertices such that no two vertices in I are **adjacent**.



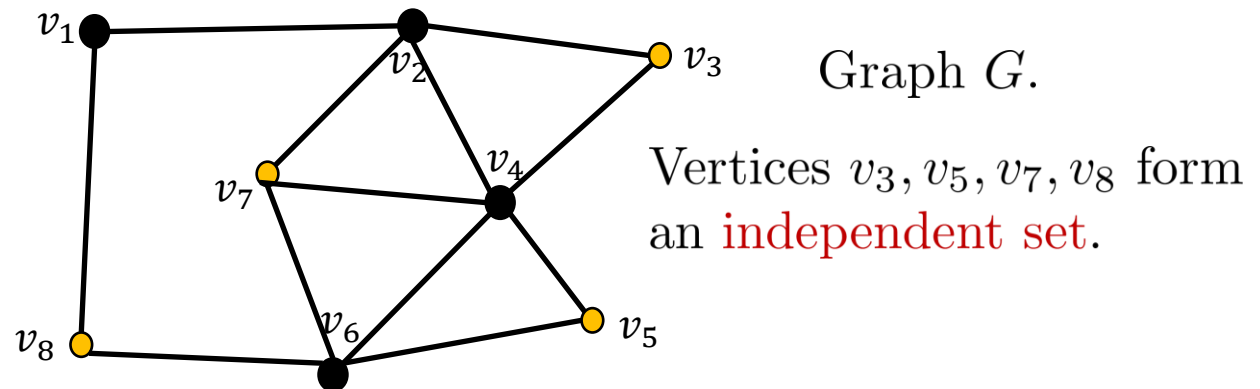
Graph G .

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 $\geq k$. An independent set is a set $I \subset V$ of vertices such that no two vertices in I are **adjacent**.



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

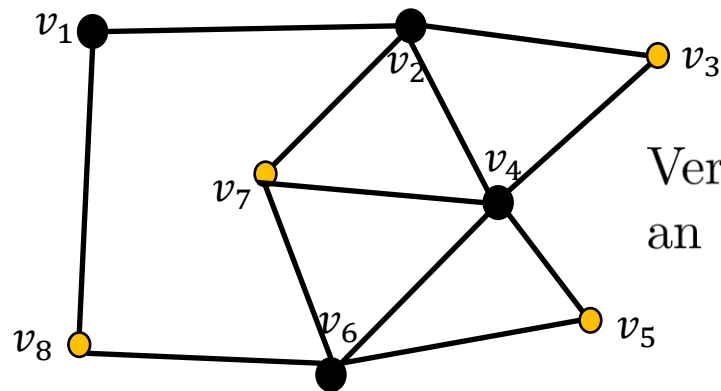
Proof: (1) INDEPENDENT SET **belongs** to **NP** (why?).

(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 $\geq k$. An independent set is a set $I \subset V$ of vertices such that no two vertices in I are **adjacent**.



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 CNF formula is a conjunction of clauses.

Example: “ $E \equiv C_1 \wedge C_2 \wedge C_3$ ” is a CNF formula.

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!*

Construction: create a triangle of literal occurrences for each clause. Join occurrences of complementary literals in different clauses. Set the target size to the number of clauses.

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!*

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

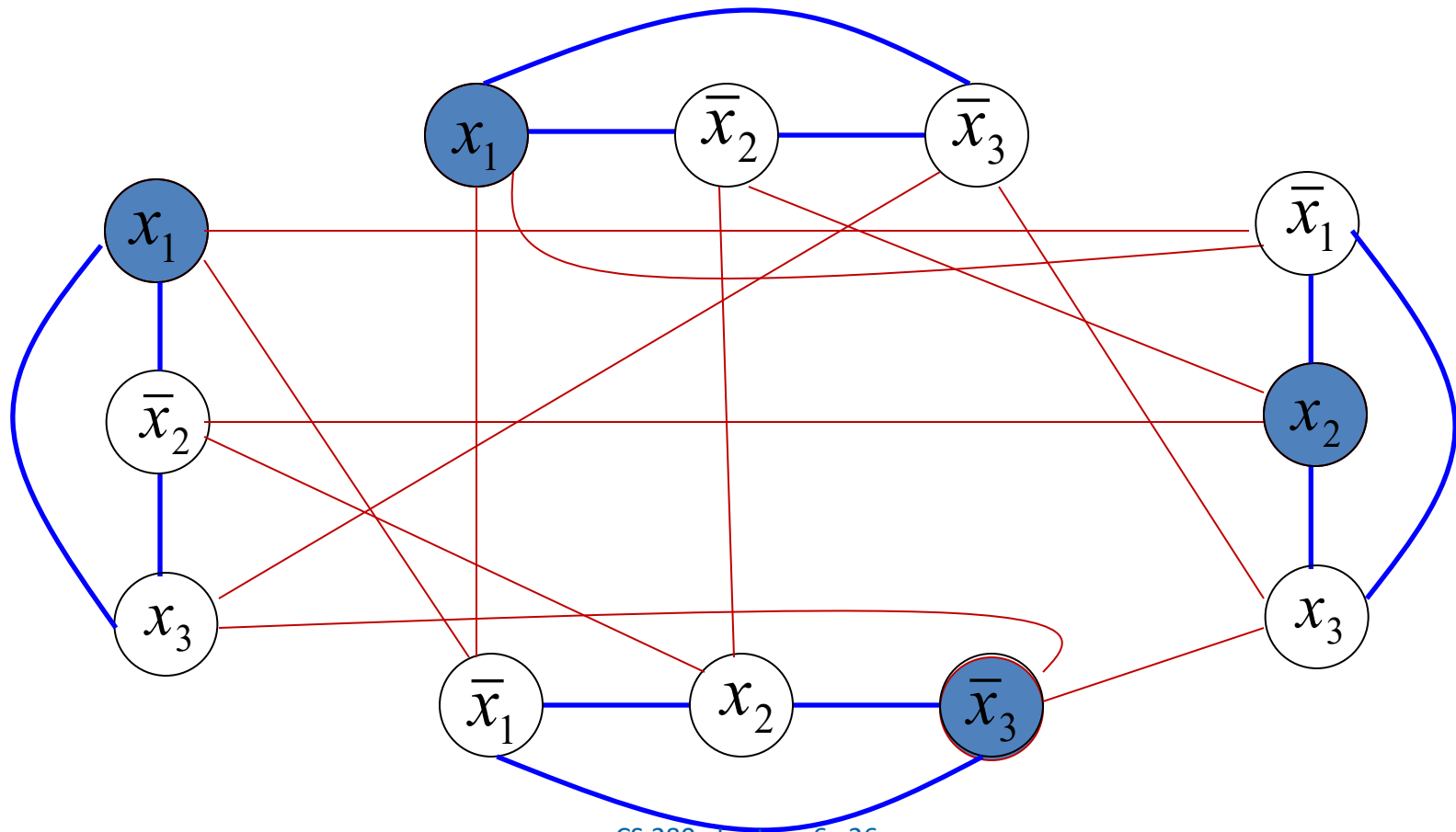
Construction: create a triangle of literal occurrences for each clause. Join occurrences of complementary literals in different clauses. Set the target size to the number of clauses.

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

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**.

**$3\text{-SAT} \leq_p \text{INDEPENDENT SET} \Rightarrow$
 $\text{INDEPENDENT SET is NP-complete!}$**

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**.

**$3\text{-SAT} \leq_p \text{INDEPENDENT SET} \Rightarrow$
 $\text{INDEPENDENT SET is NP-complete!}$**

Question: Could SAT reduce to a total, polynomially verifiable Nash search problem?

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: Could SAT reduce to such a total Nash search problem?

Answer (Megiddo–Papadimitriou): Suppose every valid output of the reduced instance lets us decide the original SAT instance. To certify unsatisfiability, guess a valid search output, verify it, and check that the reduction decodes it as “no.” Totality guarantees this certificate exists for each no instance. Hence $\text{coNP} \subseteq \text{NP}$, so **NP = coNP**.

The class PLS

PLS (Polynomial-time Local Search) describes search for a local optimum. A problem supplies three polynomial-time algorithms. Canonical problem: **LOCAL MAX-CUT**. Given an undirected graph $G = (V, E)$ with nonnegative integer edge weights w_e (in binary), find a cut $(S, V \setminus S)$ such that moving any single vertex v to the other side cannot increase its total weight.

The class PLS

PLS (Polynomial-time Local Search) describes search for a local optimum. A problem supplies three polynomial-time algorithms. Canonical problem: **LOCAL MAX-CUT**. Given an undirected graph $G = (V, E)$ with nonnegative integer edge weights w_e (in binary), find a cut $(S, V \setminus S)$ such that moving any single vertex v to the other side cannot increase its total weight. The decision version of global MAX-CUT is NP-complete. LOCAL MAX-CUT asks only for a locally optimal cut.

The class PLS

PLS (Polynomial-time Local Search): three polynomial-time algorithms. Feasible solutions and objective values have polynomial-length encodings.

1. **Initialization:** output a feasible solution. For LOCAL MAX-CUT, choose any cut.

The class PLS

PLS (Polynomial-time Local Search): three polynomial-time algorithms. Feasible solutions and objective values have polynomial-length encodings.

1. **Initialization:** output a feasible solution. For LOCAL MAX-CUT, choose any cut.
2. **Evaluation:** compute the objective value. For LOCAL MAX-CUT, sum the weights of edges crossing the cut.

The class PLS

PLS (Polynomial-time Local Search): three polynomial-time algorithms. Feasible solutions and objective values have polynomial-length encodings.

1. **Initialization:** output a feasible solution. For LOCAL MAX-CUT, choose any cut.
2. **Evaluation:** compute the objective value. For LOCAL MAX-CUT, sum the weights of edges crossing the cut.
3. **Improvement:** find a better neighboring solution or report local optimality. For LOCAL MAX-CUT, check all $|V|$ single-vertex flips.

Theorem (Local Max-Cut is PLS-complete). *Weighted LOCAL MAX-CUT under single-vertex flips is PLS-complete.*

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *Finding a pure Nash equilibrium in a congestion game with explicitly listed strategies and rational cost tables is PLS-complete.*

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

Describe the **three** polynomial-time algorithms:

- Choose the first listed strategy for each player to obtain a feasible profile.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *Finding a pure Nash equilibrium in a congestion game with explicitly listed strategies and rational cost tables is PLS-complete.*

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

Describe the **three** polynomial-time algorithms:

- Choose the first listed strategy for each player to obtain a feasible profile.
- Given a profile s , compute Rosenthal's cost potential $\Phi(s) = \sum_e \sum_{j=1}^{l_e(s)} c_e(j)$.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *Finding a pure Nash equilibrium in a congestion game with explicitly listed strategies and rational cost tables is PLS-complete.*

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

Describe the **three** polynomial-time algorithms:

- Choose the first listed strategy for each player to obtain a feasible profile.
- Given a profile s , compute Rosenthal's cost potential $\Phi(s) = \sum_e \sum_{j=1}^{l_e(s)} c_e(j)$.
- Check all players and their listed strategies. If player i can **decrease her cost**, return that deviation s'_i . Then $\Phi(s'_i, s_{-i}) < \Phi(s)$. If none exists, report a PNE.

Each step runs in polynomial time in the explicit input size.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *Finding a pure Nash equilibrium in a congestion game with explicitly listed strategies and rational cost tables is PLS-complete.*

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

Given a graph $G = (V, E)$ with nonnegative integer weights 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). *Finding a pure Nash equilibrium in a congestion game with explicitly listed strategies and rational cost tables is PLS-complete.*

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

Given a graph $G = (V, E)$ with nonnegative integer weights we define the following congestion game:

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

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *Finding a pure Nash equilibrium in a congestion game with explicitly listed strategies and rational cost tables is PLS-complete.*

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

Given a graph $G = (V, E)$ with nonnegative integer weights we define the following congestion game:

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

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *Finding a pure Nash equilibrium in a congestion game with explicitly listed strategies and rational cost tables is PLS-complete.*

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

Given a graph $G = (V, E)$ with nonnegative integer edge weights, define:

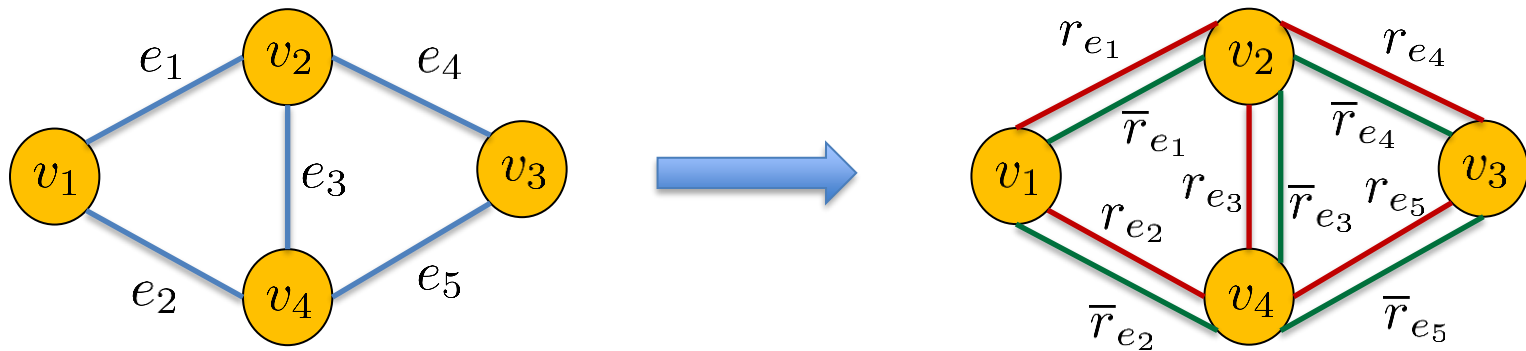
- One player for each vertex $v \in V$.
- Two resources $r_e, \bar{r}_e$ for each edge $e \in E$.
- Player v has two actions, using respectively $s_v = \{r_e : e \text{ is incident to } v\}$ or $\bar{s}_v = \{\bar{r}_e : e \text{ is incident to } v\}$.
- Each resource has cost 0 at load one and w_e at load two. Only the end-points of e can use it.

This transformation is **poly-time**.

The complexity of Pure Nash Eq.

Theorem (PNE in congestion games is PLS-complete). *Finding a pure Nash equilibrium in a congestion game with explicitly listed strategies and rational cost tables is PLS-complete.*

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



Each player chooses either **unbarred resources** (red) or **barred resources** (green).

Let v_1, v_2 use unbarred resources (red), and v_3, v_4 barred resources (green). Their costs are $w_{e_1}, w_{e_1}, w_{e_5}, w_{e_5}$, respectively.

The complexity of Pure Nash Eq.

Proof cont. Action profiles correspond to **labeled cuts** $(S, V \setminus S)$ of G : players in S choose unbarred resources, and players outside S choose barred resources.

The complexity of Pure Nash Eq.

Proof cont. An action profile a defines a labeled cut $(S, V \setminus S)$. An uncut edge contributes w_e to the cost potential; a cut edge contributes zero. Thus

$$w(S, V \setminus S) = \sum_{\substack{e=\{u,v\} \in E \\ u \in S, v \notin S}} w_e = \sum_{e \in E} w_e - \Phi(a).$$

The complexity of Pure Nash Eq.

Proof cont. An action profile a defines a labeled cut $(S, V \setminus S)$. An uncut edge contributes w_e to the cost potential; a cut edge contributes zero. Thus

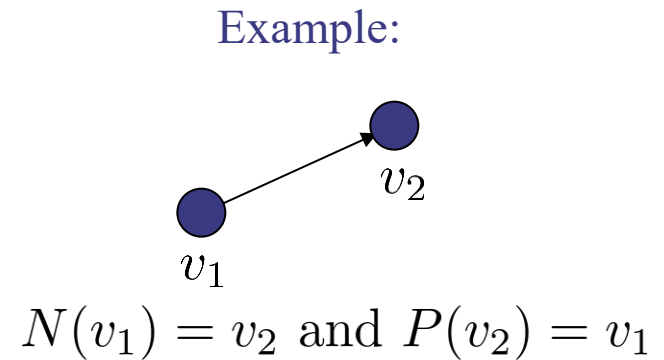
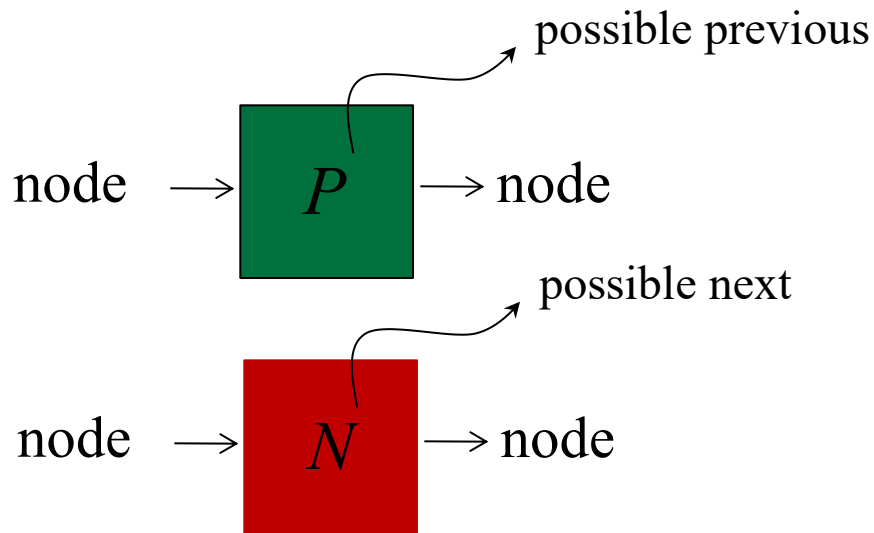
$$w(S, V \setminus S) = \sum_{\substack{e=\{u,v\} \in E \\ u \in S, v \notin S}} w_e = \sum_{e \in E} w_e - \Phi(a).$$

Therefore:

- Larger cut weight means smaller cost potential.
- A single-vertex flip is exactly a unilateral action change.
- Locally maximum cuts correspond to pure Nash equilibria of the constructed game.

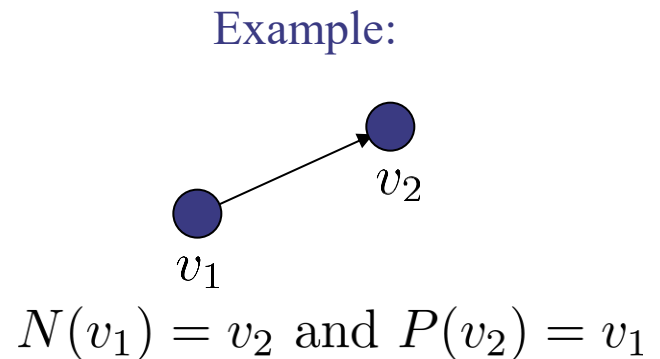
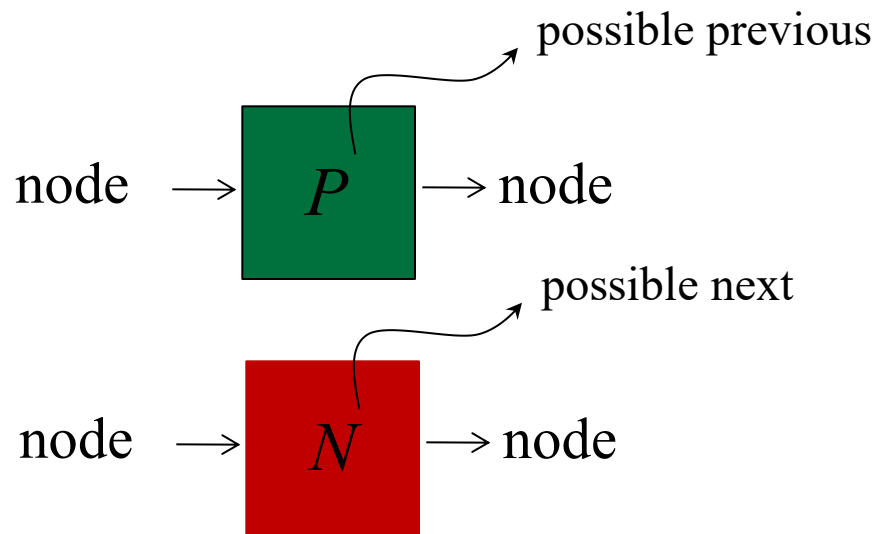
The class PPAD

Two polynomial-size circuits P and N describe a graph on $\{0,1\}^n$ with 2^n vertices. Include $v \rightarrow w$ exactly when $v \neq w$, $N(v) = w$, and $P(w) = v$.



The class PPAD

Two polynomial-size circuits P and N describe a graph on $\{0,1\}^n$ with 2^n vertices. Include $v \rightarrow w$ exactly when $v \neq w$, $N(v) = w$, and $P(w) = v$.

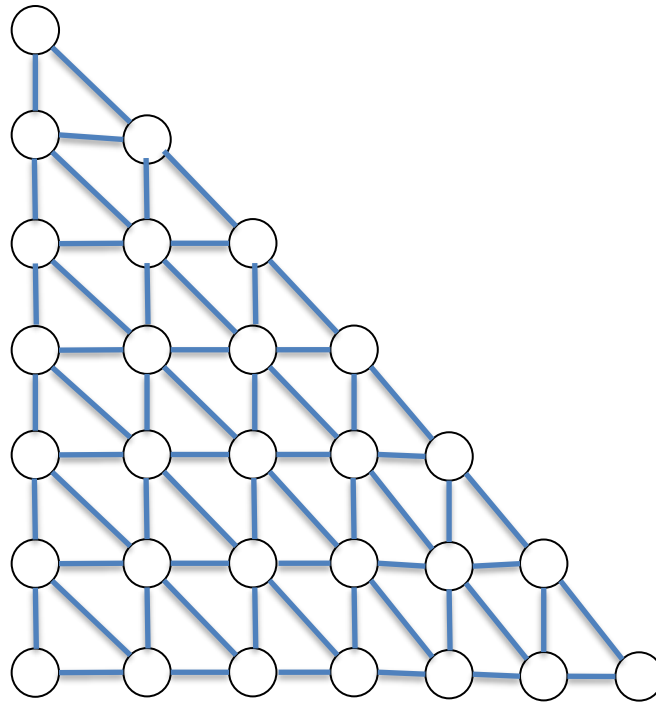


END OF THE LINE: If 0^n is a source or sink, find a different source or sink. Otherwise return 0^n .

Every vertex has in-degree and out-degree at most one. **PPAD** consists of FNP problems reducible to this total search problem.

2D Sperner's Lemma

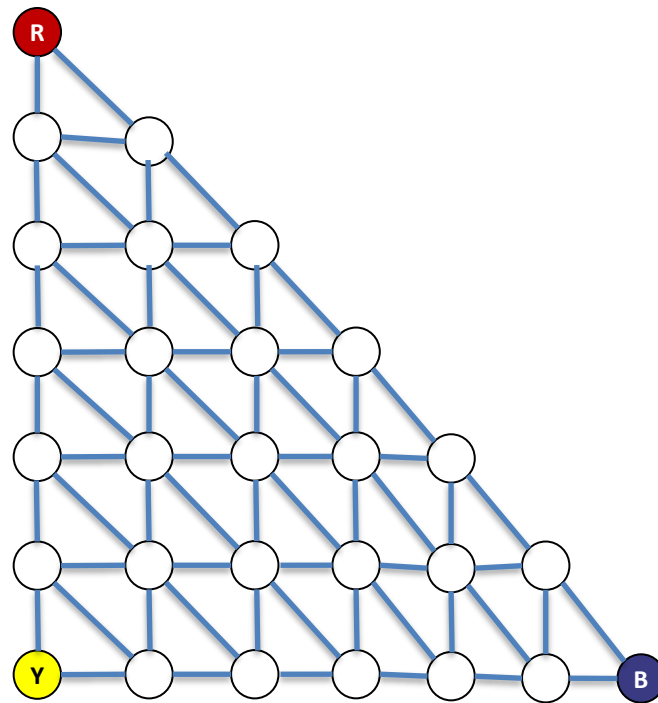
Theorem (Sperner's lemma). *Triangulate a triangle. Give its three corners distinct labels R, Y, B. On each boundary edge use only its endpoint labels; interior vertices may use any label. The number of small triangles with all three labels is **odd**, hence positive.*



R = red
Y = yellow
B = blue

2D Sperner's Lemma

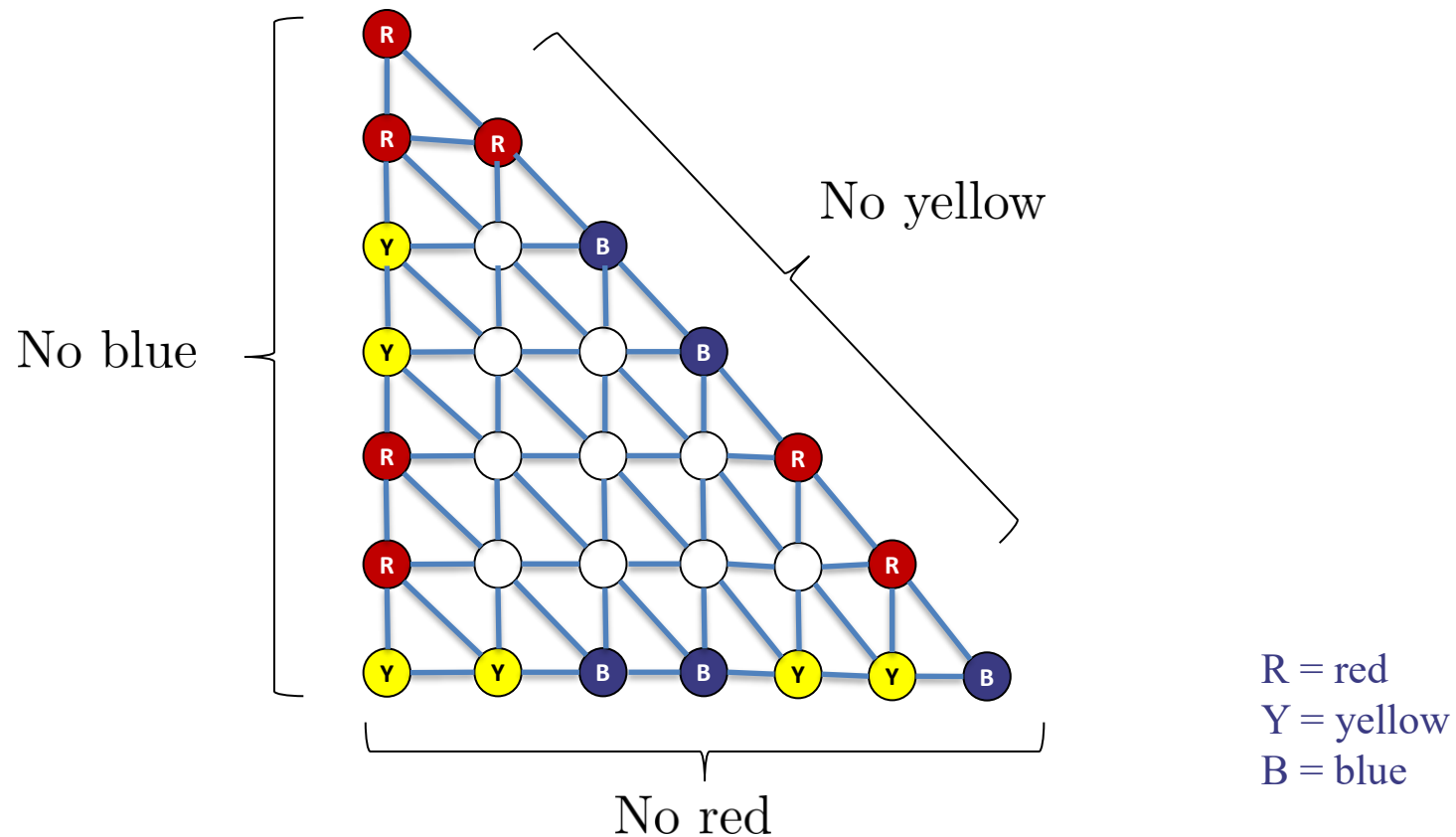
Theorem (Sperner's lemma). *Triangulate a triangle. Give its three corners distinct labels R, Y, B. On each boundary edge use only its endpoint labels; interior vertices may use any label. The number of small triangles with all three labels is **odd**, hence positive.*



R = red
Y = yellow
B = blue

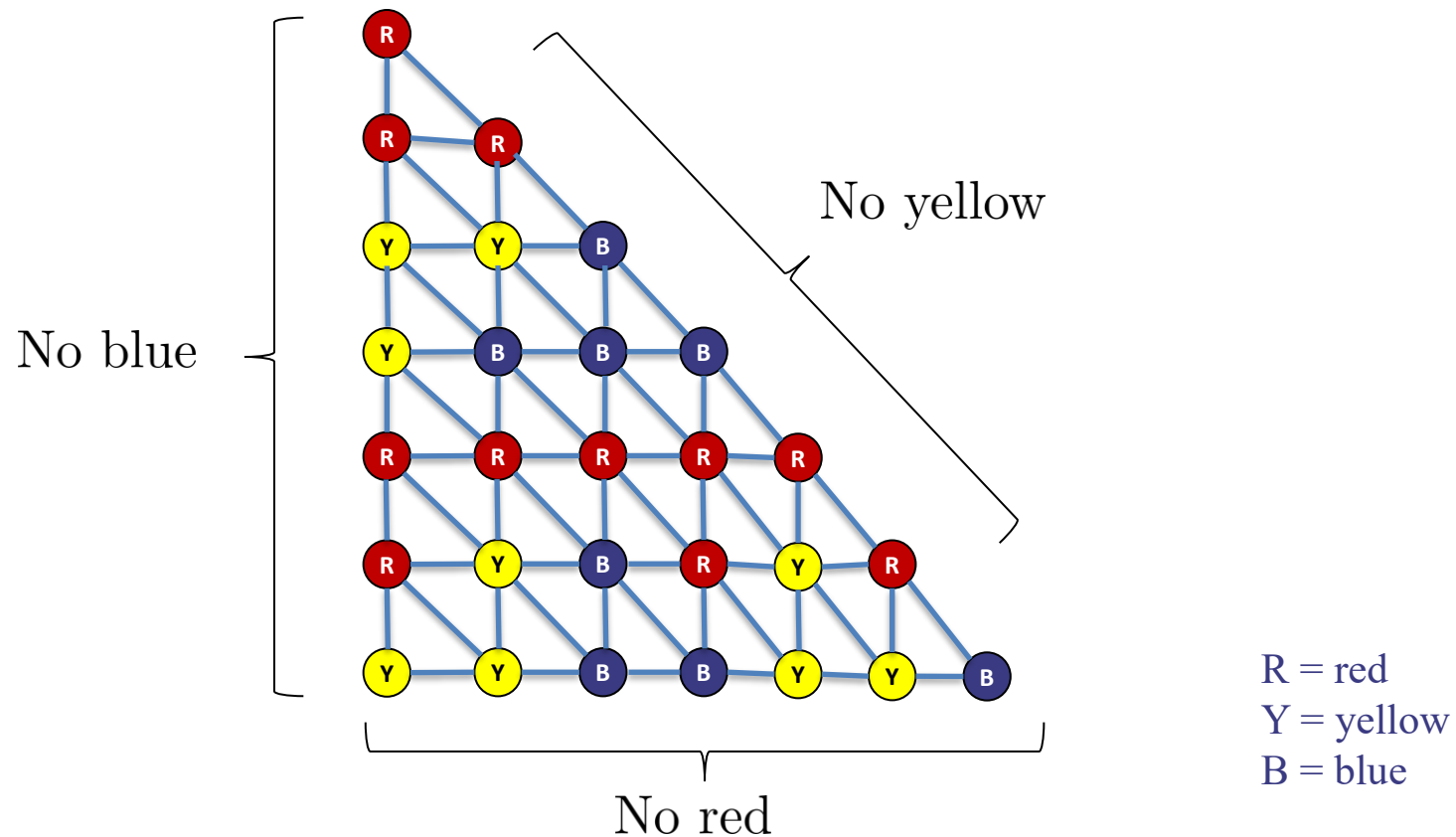
2D Sperner's Lemma

Theorem (Sperner's lemma). *Triangulate a triangle. Give its three corners distinct labels R, Y, B. On each boundary edge use only its endpoint labels; interior vertices may use any label. The number of small triangles with all three labels is **odd**, hence positive.*



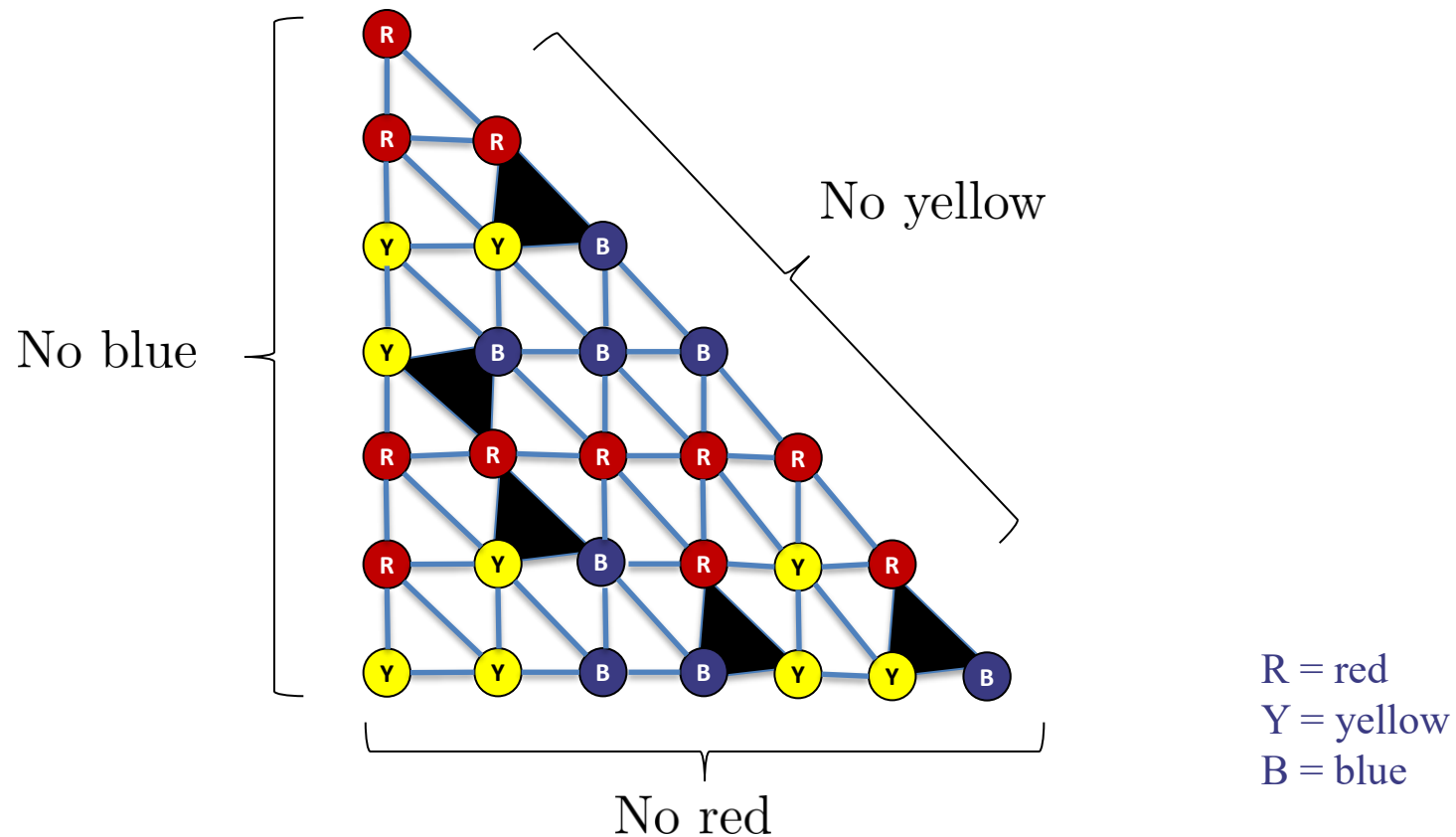
2D Sperner's Lemma

Theorem (Sperner's lemma). *Triangulate a triangle. Give its three corners distinct labels R, Y, B. On each boundary edge use only its endpoint labels; interior vertices may use any label. The number of small triangles with all three labels is **odd**, hence positive.*



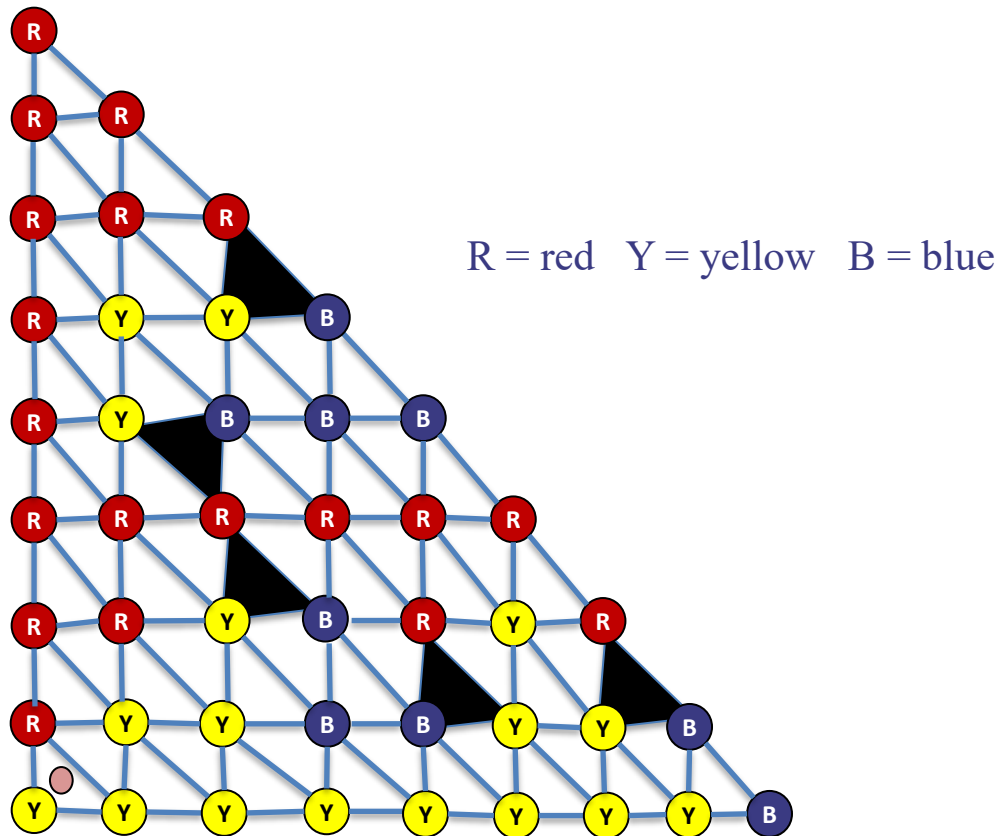
2D Sperner's Lemma

Theorem (Sperner's lemma). *Triangulate a triangle. Give its three corners distinct labels R, Y, B. On each boundary edge use only its endpoint labels; interior vertices may use any label. The number of small triangles with all three labels is **odd**, hence positive.*



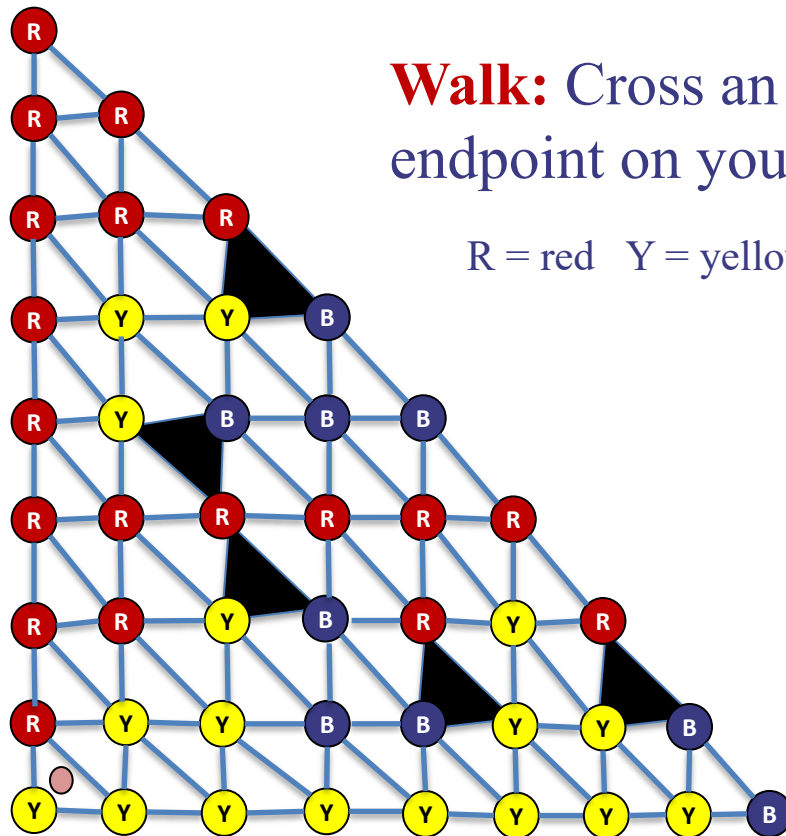
2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.



2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.

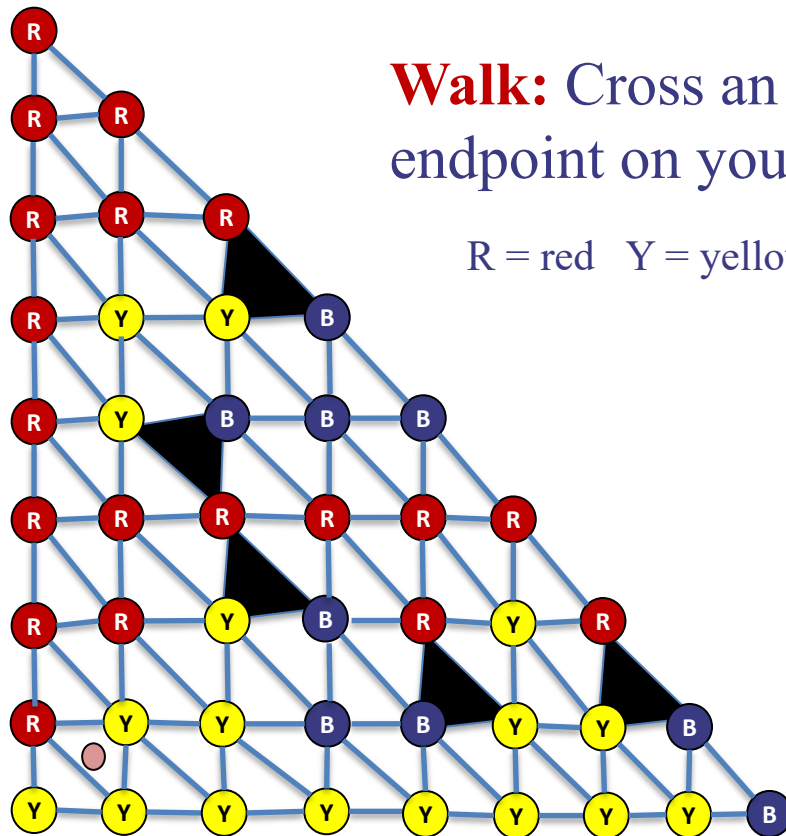


Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.

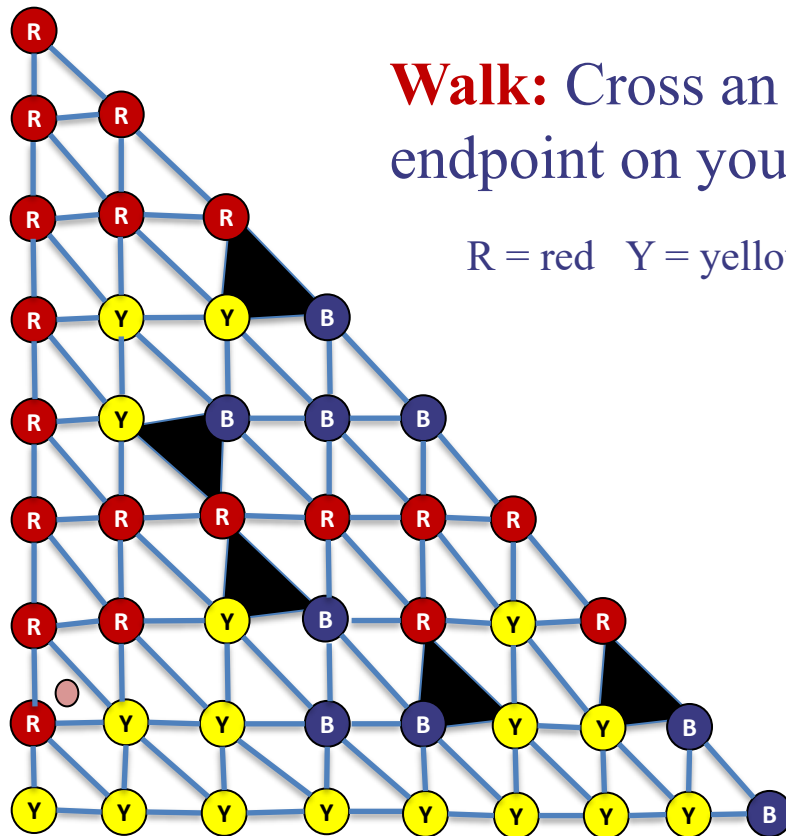


Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.

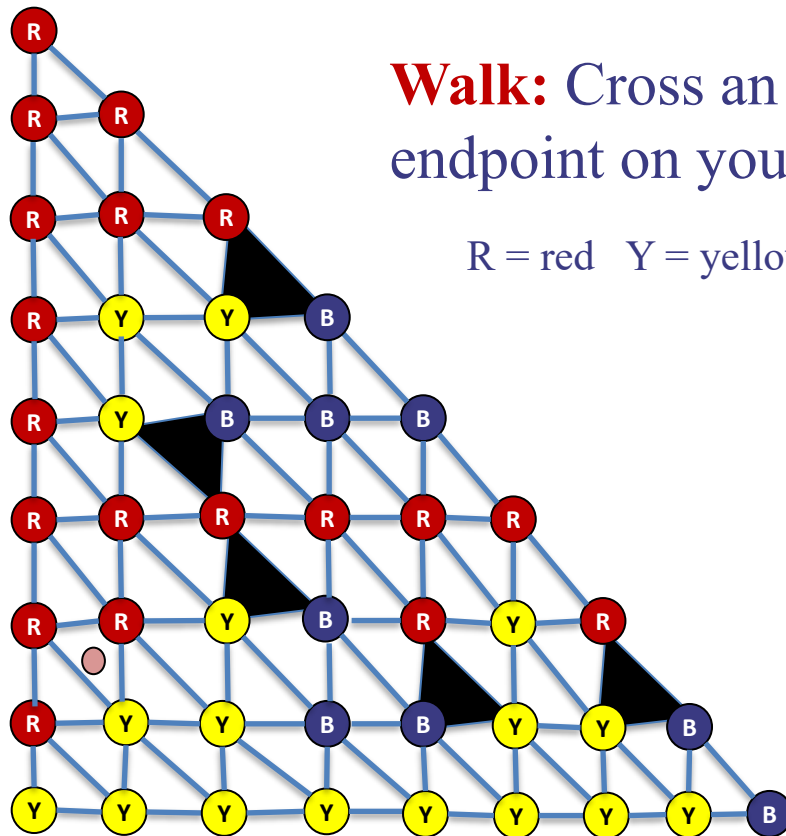


Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.

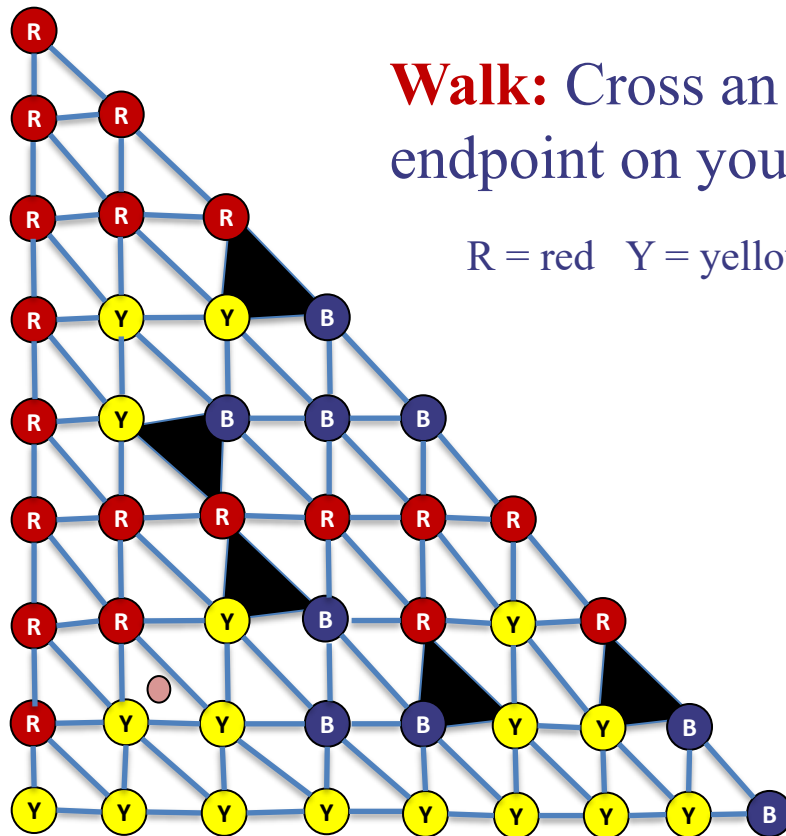


Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.

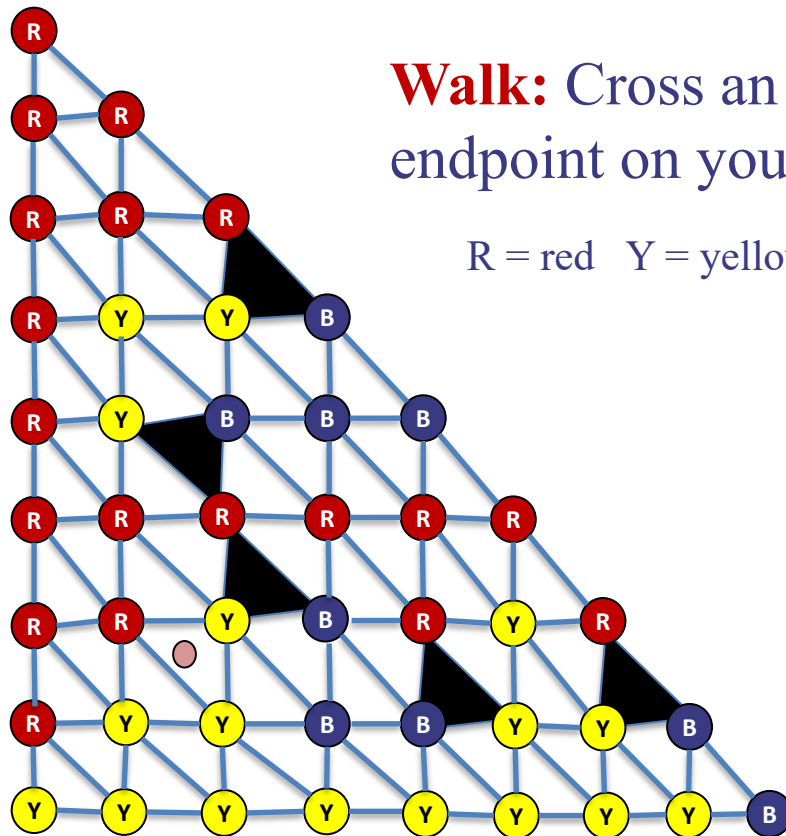


Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.

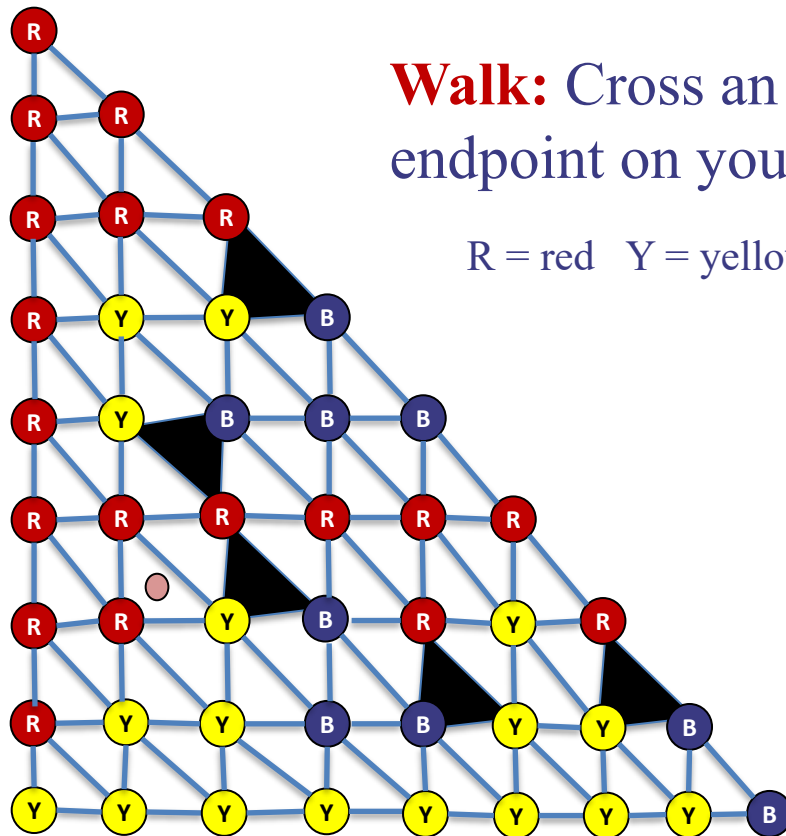


Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.

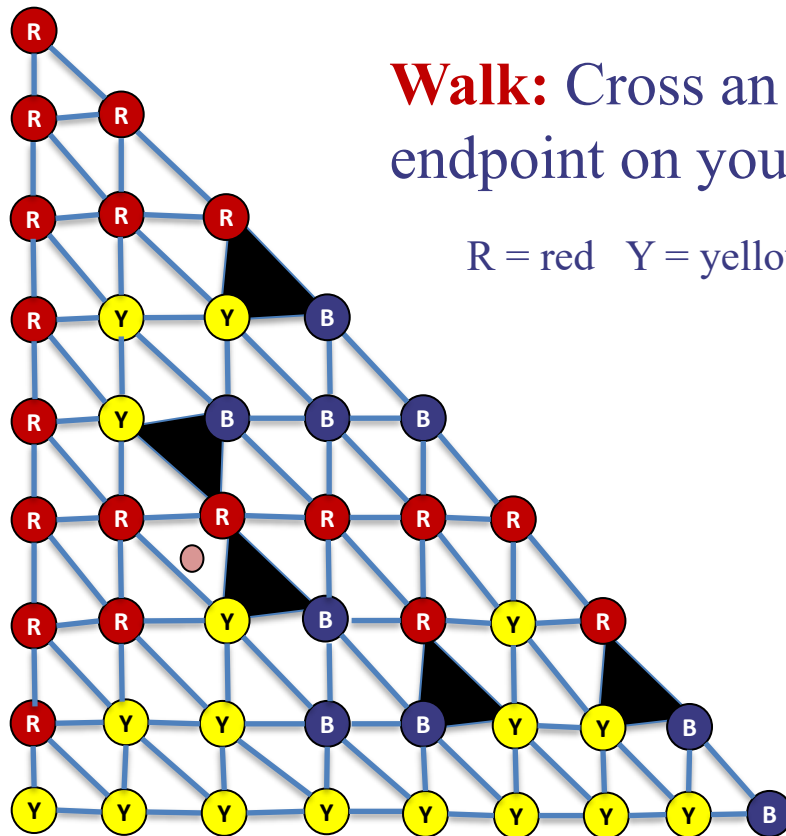


Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.

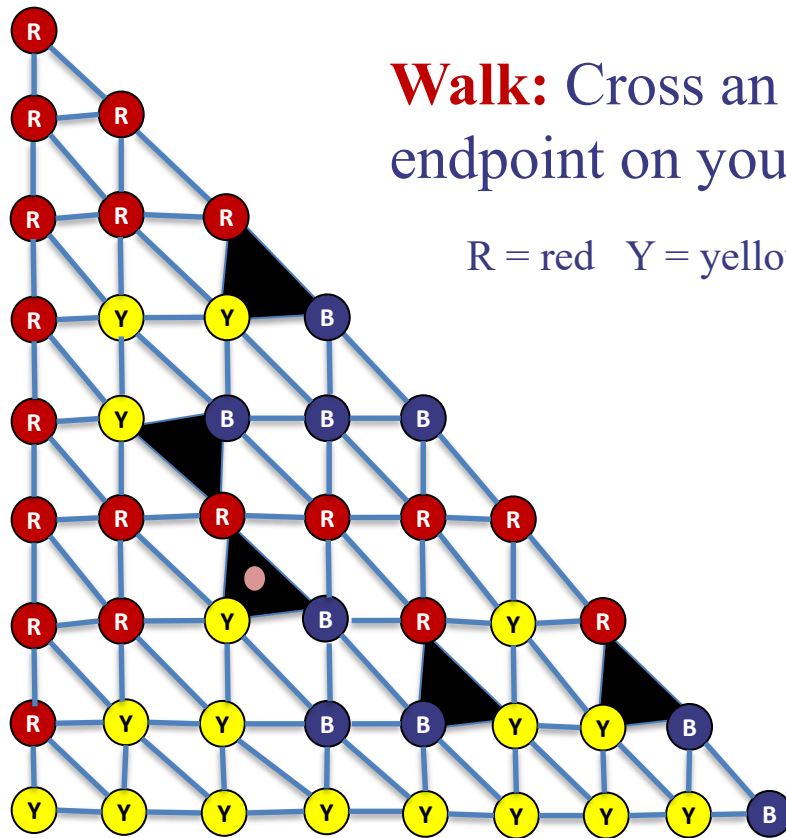


Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof. Extend the boundary without adding trichromatic triangles. Start a directed walk in the bottom-left outer triangle, entering across its R–Y boundary edge.



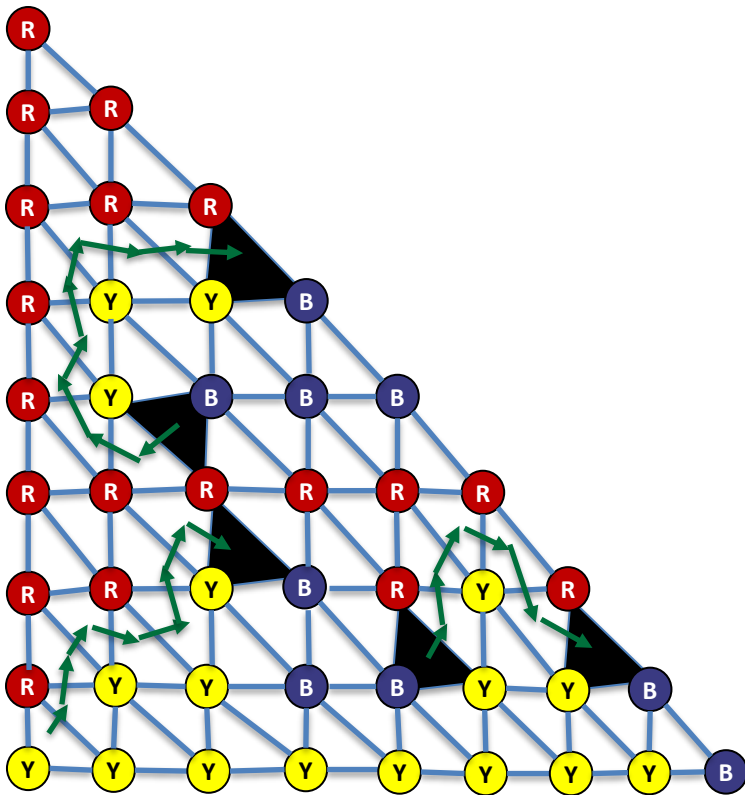
Walk: Cross an R–Y edge, keeping its R endpoint on your left.

R = red Y = yellow B = blue

2D Sperner's Lemma

Proof cont.

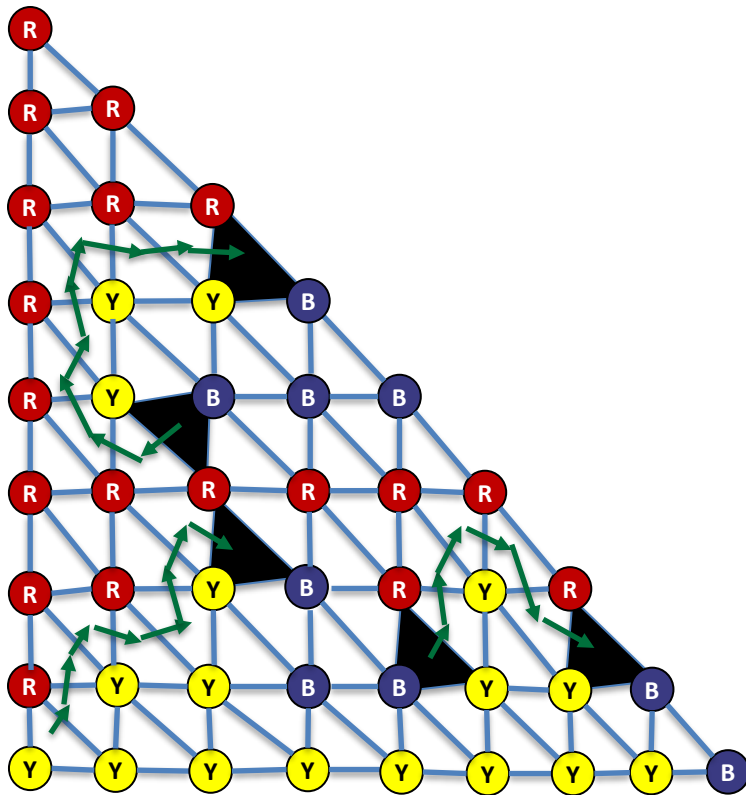
- The walk **cannot leave** through any other outer edge.
- Every visited nonterminal triangle has a unique entrance and exit, so the walk **cannot merge into a cycle**.



2D Sperner's Lemma

Proof cont.

- The walk **cannot leave** through any other outer edge.
- Every visited nonterminal triangle has a unique entrance and exit, so the walk **cannot merge into a cycle**.

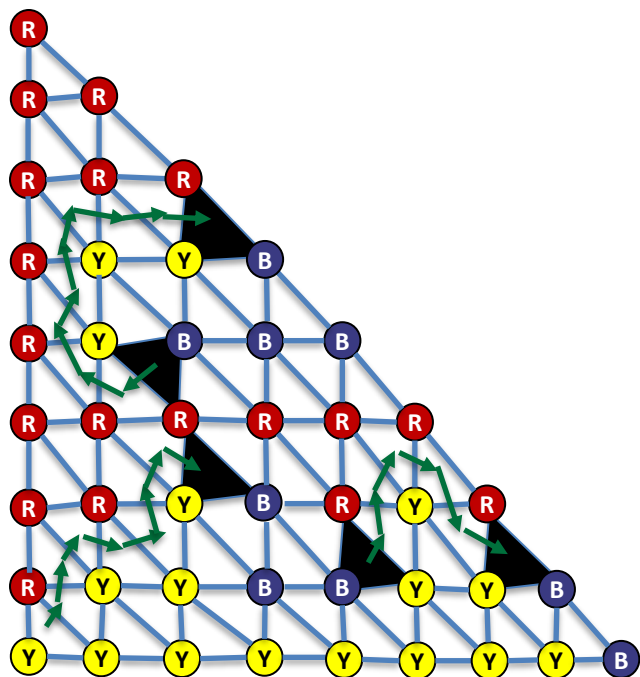


The finite walk ends at an interior triangle.
Its three vertex labels must be R, Y, B.

2D Sperner's Lemma

Proof cont.

- The walk **cannot leave** through any other outer edge.
- Every visited nonterminal triangle has a unique entrance and exit, so the walk **cannot merge into a cycle**.



The finite walk ends at an interior triangle. Its three vertex labels must be R, Y, B.

Sperner's lemma holds in all dimensions.
Succinct SPERNER reduces to END OF THE LINE.

BROUWER

Definition (BROUWER). *Input:* A polynomial-time evaluator for $F : [0, 1]^m \rightarrow [0, 1]^m$ at requested precision, a known rational Lipschitz bound $K \geq 1$, and rational accuracy $\epsilon > 0$ (in binary).

Output: A (rational) point x so that

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

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

BROUWER

Definition (BROUWER). *Input:* A polynomial-time evaluator for $F : [0, 1]^m \rightarrow [0, 1]^m$ at requested precision, a known rational Lipschitz bound $K \geq 1$, and rational accuracy $\epsilon > 0$ (in binary).

Output: A (rational) point x so that

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

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

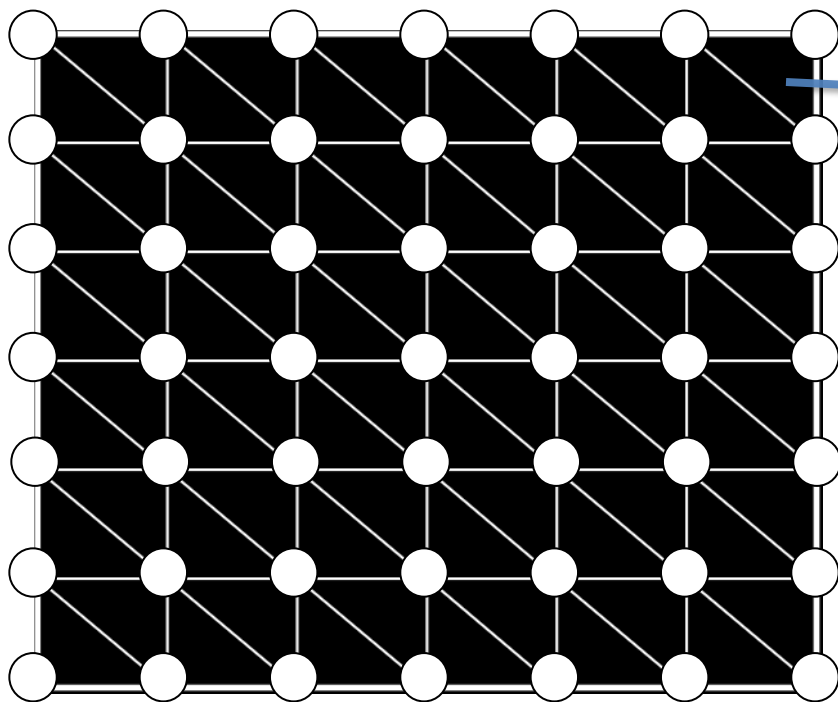
We will show that

BROUWER $\rightarrow$ SPERNER

2D BROUWER reduction to SPERNER

Let $F : [0, 1]^2 \rightarrow [0, 1]^2$ be K -Lipschitz, with $K \geq 1$.
Choose a rational mesh size $\delta \leq \epsilon/K$. Then

$$\|x - y\|_{\infty} \leq \delta \implies \|F(x) - F(y)\|_{\infty} \leq \epsilon.$$

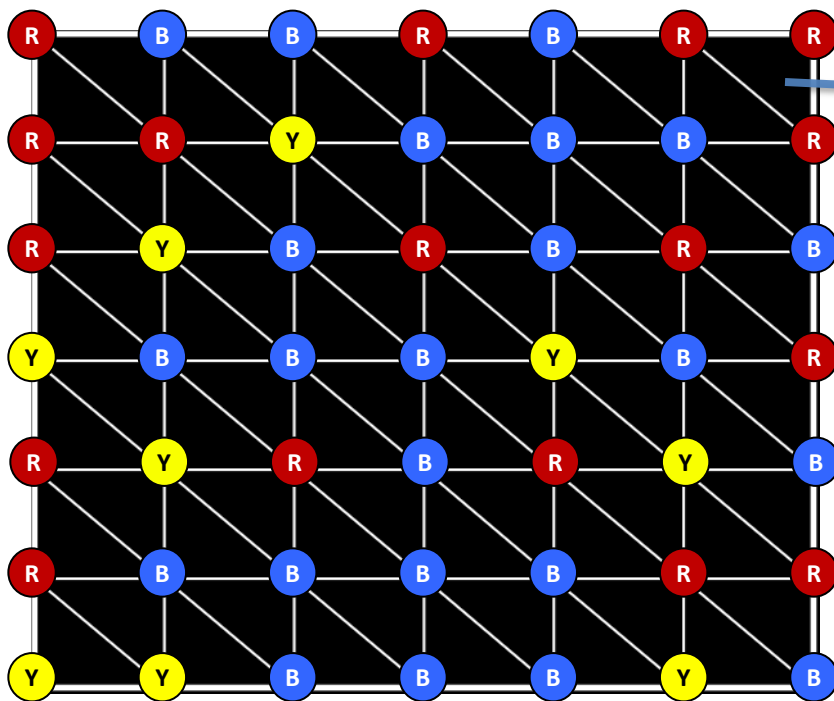


Each small triangle has ℓ_{∞} -diameter at most δ .

2D BROUWER reduction to SPERNER

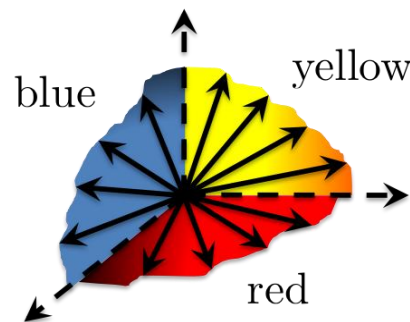
Let $F : [0, 1]^2 \rightarrow [0, 1]^2$ be K -Lipschitz, with $K \geq 1$.
Choose a rational mesh size $\delta \leq \epsilon/K$. Then

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



Each small triangle has ℓ_∞ -diameter at most δ .

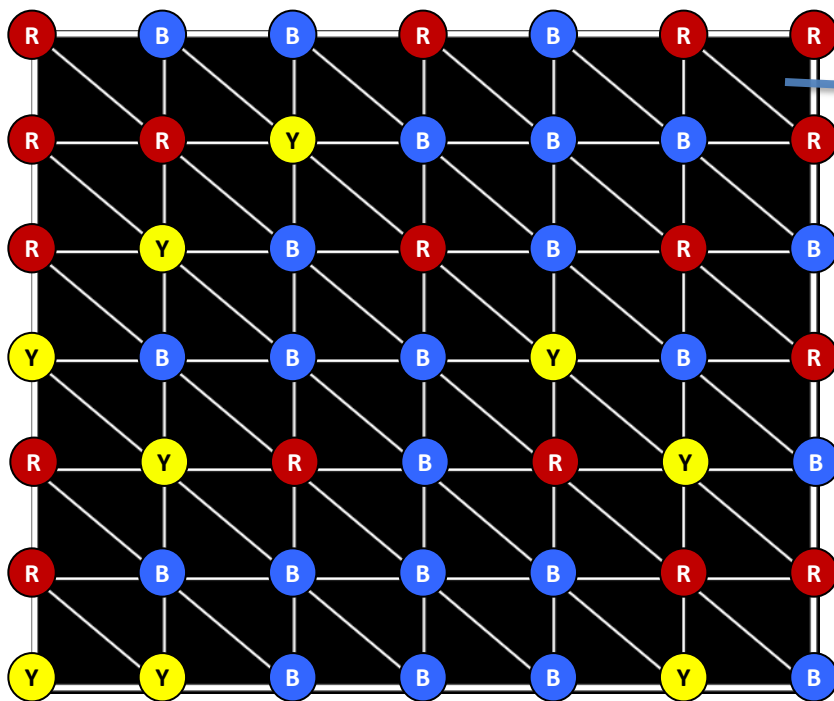
Label vertices by the direction of $F(x) - x$.
If $F(x) = x$, return that fixed point.



2D BROUWER reduction to SPERNER

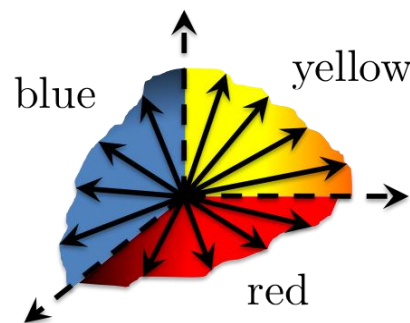
Let $F : [0, 1]^2 \rightarrow [0, 1]^2$ be K -Lipschitz, with $K \geq 1$.
Choose a rational mesh size $\delta \leq \epsilon/K$. Then

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



Each small triangle has ℓ_∞ -diameter at most δ .

Label vertices by the direction of $F(x) - x$.
If $F(x) = x$, return that fixed point.

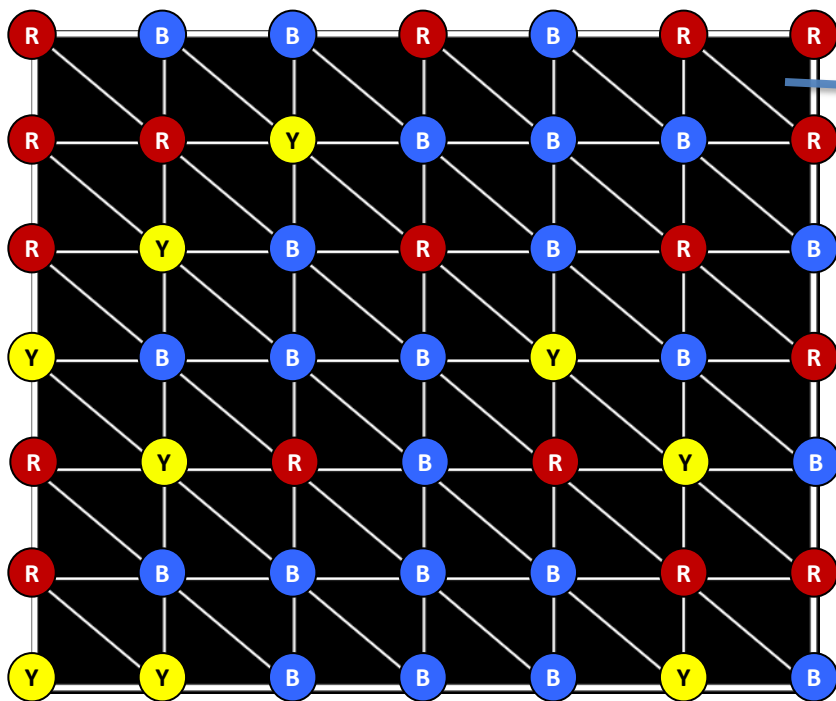


Break ties to preserve the permitted boundary labels.

2D BROUWER reduction to SPERNER

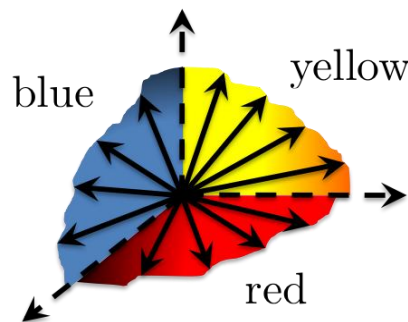
Claim. Let $\delta \leq \epsilon / K$ with $K \geq 1$, and let v^Y be the Y-labeled vertex of a trichromatic small triangle. Then

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



Each small triangle has ℓ_∞ -diameter at most δ .

Label vertices by the direction of $F(x) - x$. If $F(x) = x$, return that fixed point.



Break ties to preserve the permitted boundary labels.

2D BROUWER reduction to SPERNER

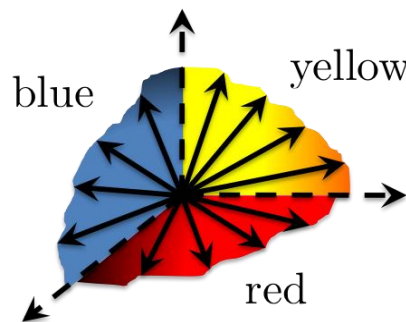
Claim. Let $\delta \leq \epsilon / K$ with $K \geq 1$, and let v^Y be the Y-labeled vertex of a trichromatic small triangle. Then

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

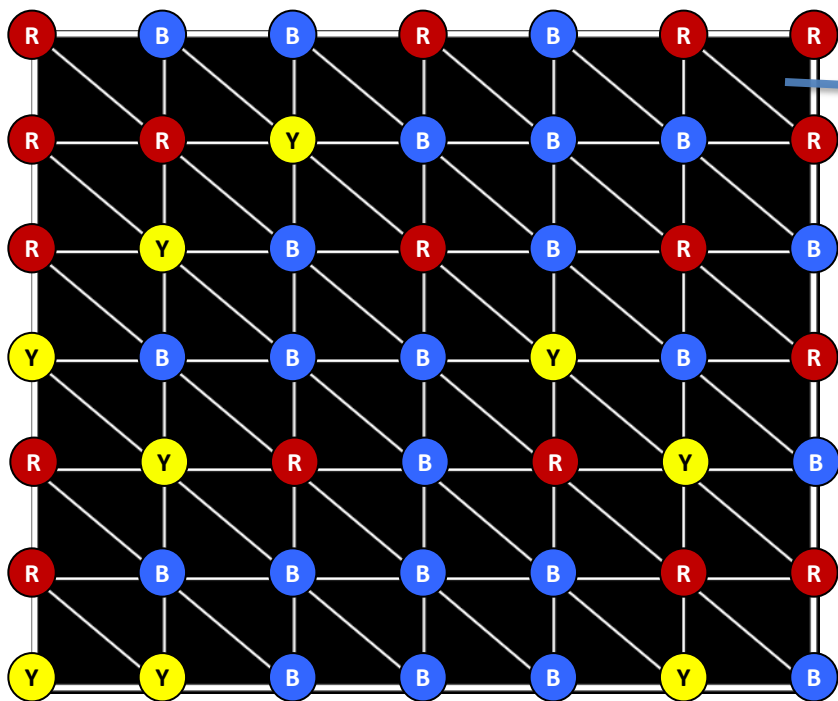
This will be in HW2.

Each small triangle has ℓ_∞ -diameter at most δ .

Label vertices by the direction of $F(x) - x$.
If $F(x) = x$, return that fixed point.



Break ties to preserve the permitted boundary labels.



NASH reduction to BROUWER

An illustration using Nash's continuous map. For a computational reduction, choose the fixed-point accuracy small enough to guarantee the requested Nash accuracy.

NASH reduction to BROUWER

An illustration using Nash's continuous map. For a computational reduction, choose the fixed-point accuracy small enough to guarantee the requested Nash accuracy.

Consider the 2×2 matching-pennies game.

Row / Col	H	T
H	1, -1	-1, 1
T	-1, 1	1, -1

Consider the function f from the proof of Nash.

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

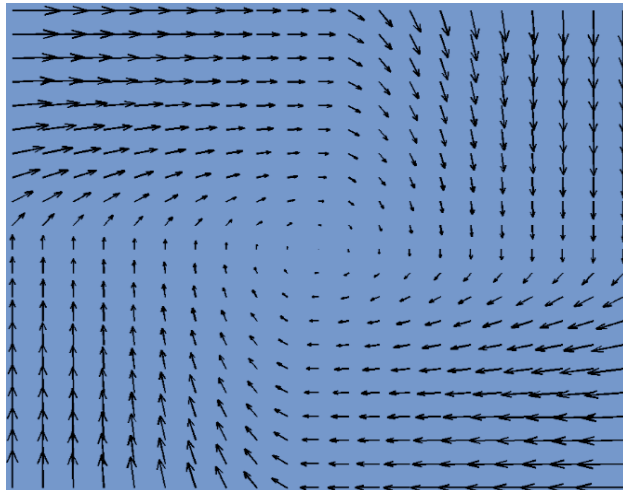
NASH reduction to BROUWER

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

Row / Col	H	T
H	1, -1	-1, 1
T	-1, 1	1, -1

Consider the 2×2 matching-pennies game.

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



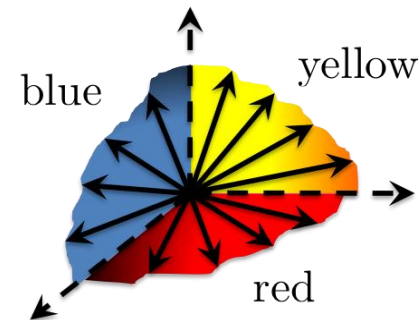
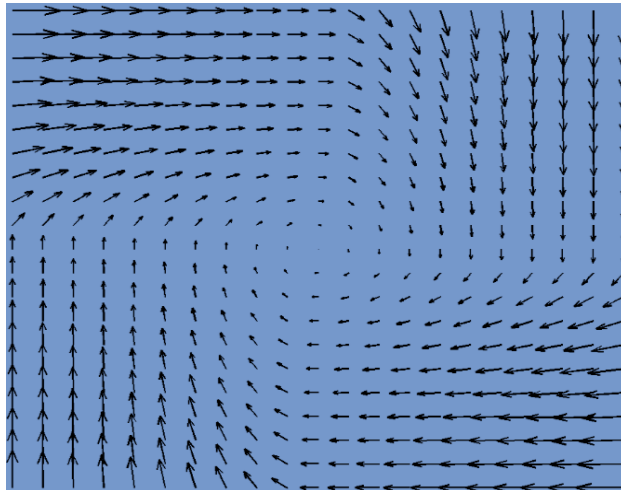
NASH reduction to BROUWER

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

Row / Col	H	T
H	1, -1	-1, 1
T	-1, 1	1, -1

Consider the 2×2 matching-pennies game.

Draw the vector field for $f(x) - x$. Color the points according to



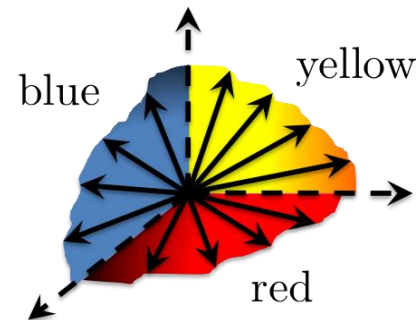
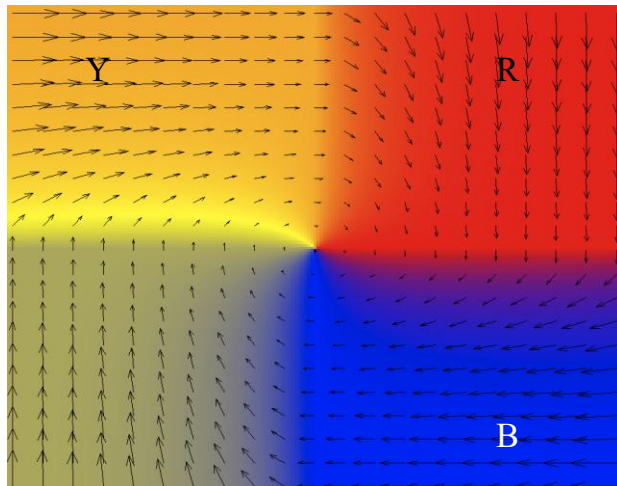
NASH reduction to BROUWER

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

Row / Col	H	T
H	1, -1	-1, 1
T	-1, 1	1, -1

Consider the 2×2 matching-pennies game.

Draw the vector field for $f(x) - x$. Color the points according to



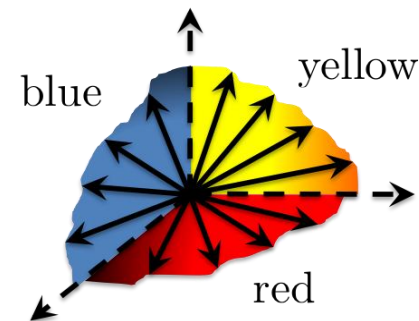
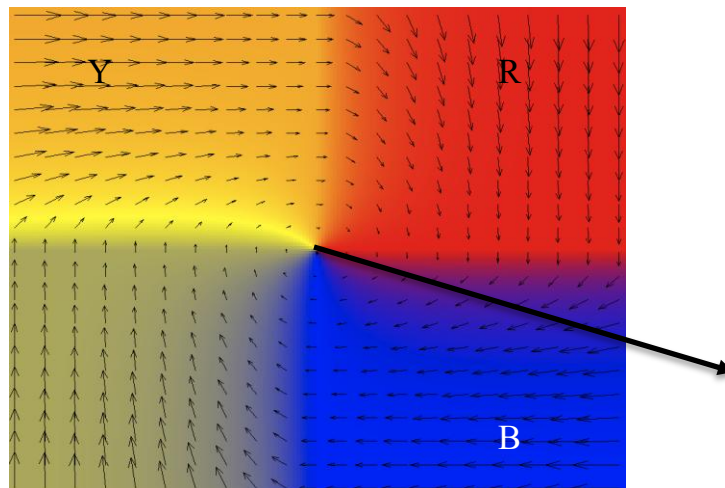
NASH reduction to BROUWER

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

Row / Col	H	T
H	1, -1	-1, 1
T	-1, 1	1, -1

Consider the 2×2 matching-pennies game.

Draw the vector field for $f(x) - x$. Color the points according to

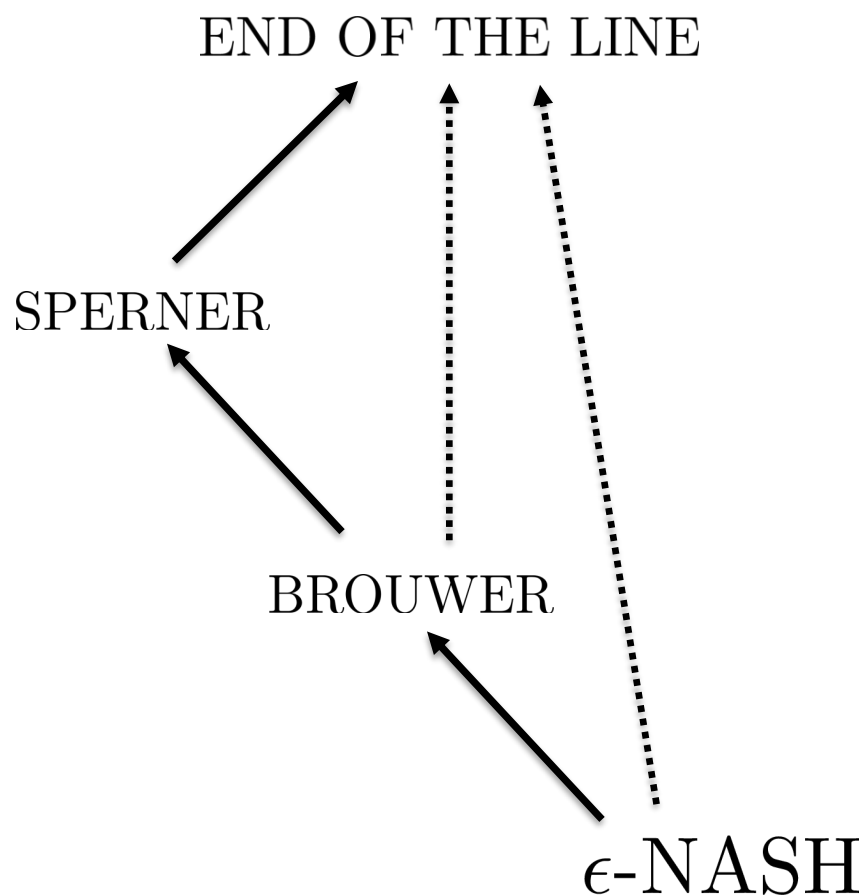


Nash equilibrium: $p = q = \frac{1}{2}$.

Reductions establishing PPAD membership

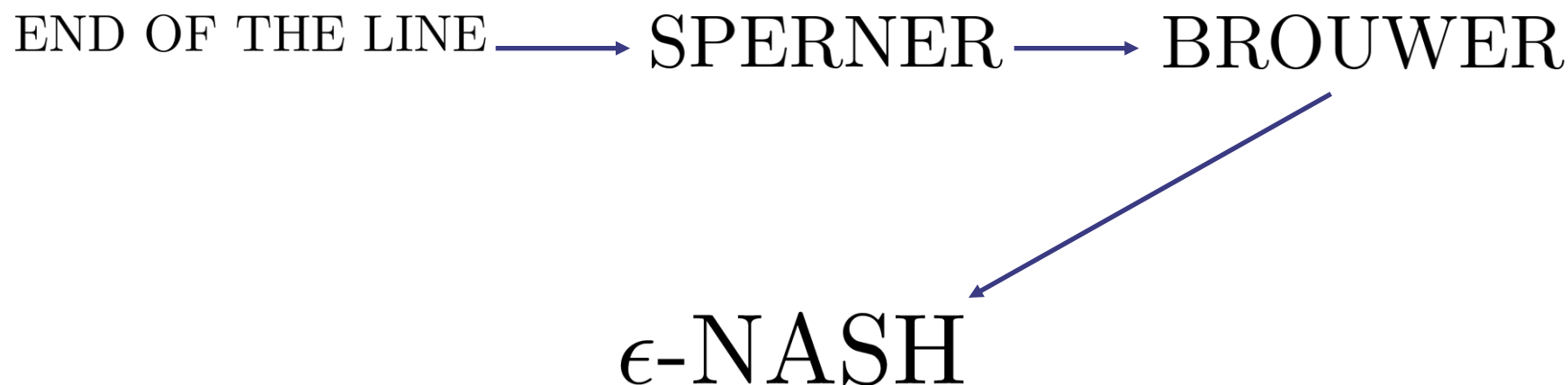
$$\text{PPAD} \subseteq \text{TFNP}$$

↑



Theorem (**Nash computation is PPAD-complete**). *Finding an ϵ -Nash equilibrium (rational $\epsilon > 0$ in the input) is PPAD-complete. Finding an exact equilibrium of a rational two-player game is also PPAD-complete.*

Daskalakis–Goldberg–Papadimitriou; Chen–Deng–Teng. Exact games with three or more players require the FIXP framework.



Polynomial-time equivalences

