

# Some Warm-Up Problems

**Decision:** has a *yes* or *no* answer (binary)

**Q:** Is the word “algorithm” a permutation of “logarithm”?

**A:** *yes*, they both can be perturbed to used to spell “aghilmort”

---

**Construction:** find a single “how-to” instance

**Q:** What is a permutation of “MISSISSIPPI” to “IIMPSSSSS”?

**A:** [11, 8, 5, 2, 1, 10, 9, 7, 6, 4, 3]

---

**Counting:** how many “how-to” instances exist

**Q:** How many permutations of “MISSISSIPPI” exist?

**A:**  $11! = 39,916,800$  (ignoring repeats)

---

**Enumeration:** find every “how-to” instance

**Q:** What are all the unique permutations of “MISSISSIPPI”?

**A:** Group exercise, hint:  $|S| = 11! / (4! \cdot 4! \cdot 2!) = 34,650$

# Getting Warmer...

**Q:** Is the word “algorithm” a permutation of “logarithm”?

**A:** yes,  $\text{sort}(\text{“algorithm”}) = \text{sort}(\text{“logarithm”}) = \text{“aghilmort”}$

a l g o r i t h m

l o g a r i t h m

**IDEA:** Use *lexicographic sorting* to obtain a *minimum canonical isomorph*

a g h i l m o r t

**QUESTION:** How can we apply this idea to an unlabeled graph with edges?

# Determining a Canonical Graph Isomorph Using Lexicographic Sorting

Captain Chris Augeri<sup>1\*</sup>

Dr. Barry Mullins<sup>1</sup>, Dr. Rusty Baldwin<sup>1</sup>,  
Dr. Dursun Bulutoglu<sup>2</sup>, and Lieutenant Colonel Leemon Baird<sup>3</sup>

Department of Electrical and Computer Engineering<sup>1</sup>  
Department of Mathematics and Statistics<sup>2</sup>  
Air Force Institute of Technology (AFIT), Dayton, OH

Department of Computer Science<sup>3</sup>  
U. S. Air Force Academy (USAFA), Colorado Springs, CO

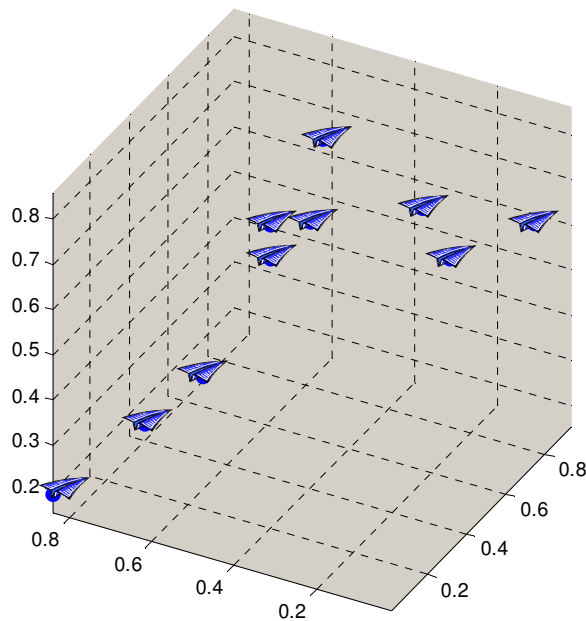
*presented to the*

Discrete Math Seminar, Department of Mathematics  
Wright State University, Dayton, OH

Wednesday, 25 April 2007

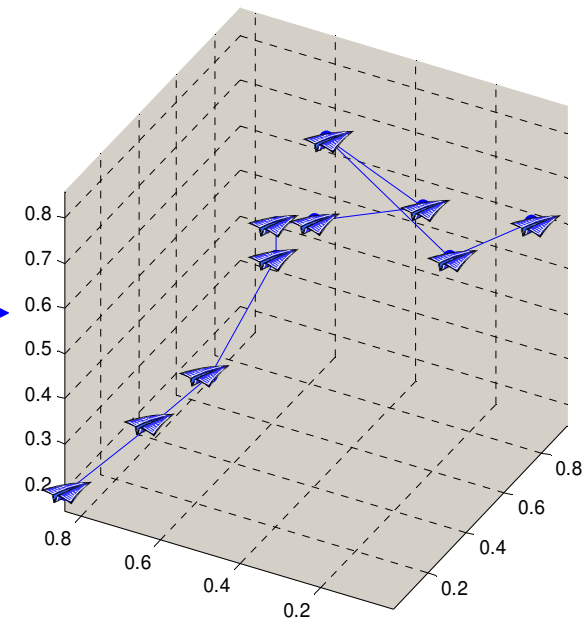
# Linearizing $k$ -Dimensional Data

- **Dimensions:** geographic coordinates, time, values, ...
- **Applications:** queries: speed & relevance, logistics, network security
- **Approaches**
  - **Static:** space-filling curves,  $k$ -D trees
  - **Dynamic:** SVD, multi-dimensional scaling (MDS)
  - **Canonical:** equivalent to determining isomorphism!



**UAV Swarm**

*Vertex  
Ordering*



**A Possible Linearization**

# Linearizing Points from $k$ -D $\rightarrow$ 2-D $\rightarrow$ 1-D

$$p_i = (d_{1_i}, d_{2_i}, \dots, d_{k_i})$$

$$p_j = (d_{1_j}, d_{2_j}, \dots, d_{k_j}), 1 \leq i, j \leq n$$

**$k$ -D**

$$\Delta(p_i, p_j) = \sqrt{(d_{1_i} - d_{1_j})^2 + (d_{2_i} - d_{2_j})^2 + \dots + (d_{k_i} - d_{k_j})^2}$$

**2-D**

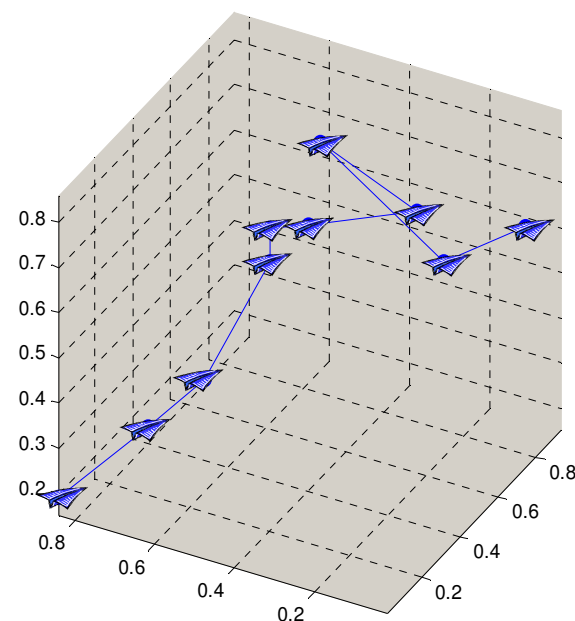
$$\begin{bmatrix} 0 & \Delta(p_1, p_2) & \dots & \Delta(p_1, p_n) \\ \Delta(p_1, p_2) & 0 & \dots & \Delta(p_2, p_n) \\ \vdots & \vdots & \ddots & \vdots \\ \Delta(p_1, p_n) & \Delta(p_2, p_n) & \dots & 0 \end{bmatrix}$$

**1-D**

$$\phi[p_1, p_2, \dots, p_n]$$

**A Possible Linearization**

We assume that  $n$  points are mapped from  $k$ -D  $\rightarrow$  2-D by some *arbitrary* distance metric, e.g., Euclidean distance.



# Overview

- Determining, Deciding, & Difficulty
- A Master Template for Deciding
- Permutations, Triangles, & Chains
- Finding a Canonical Isomorph
- IsoCanon
  - Algorithm & Example
  - Experiments & Analysis
- A New Paradigm
- Logarithmic Coloring
- Coming Full Circle
- Next Talk?

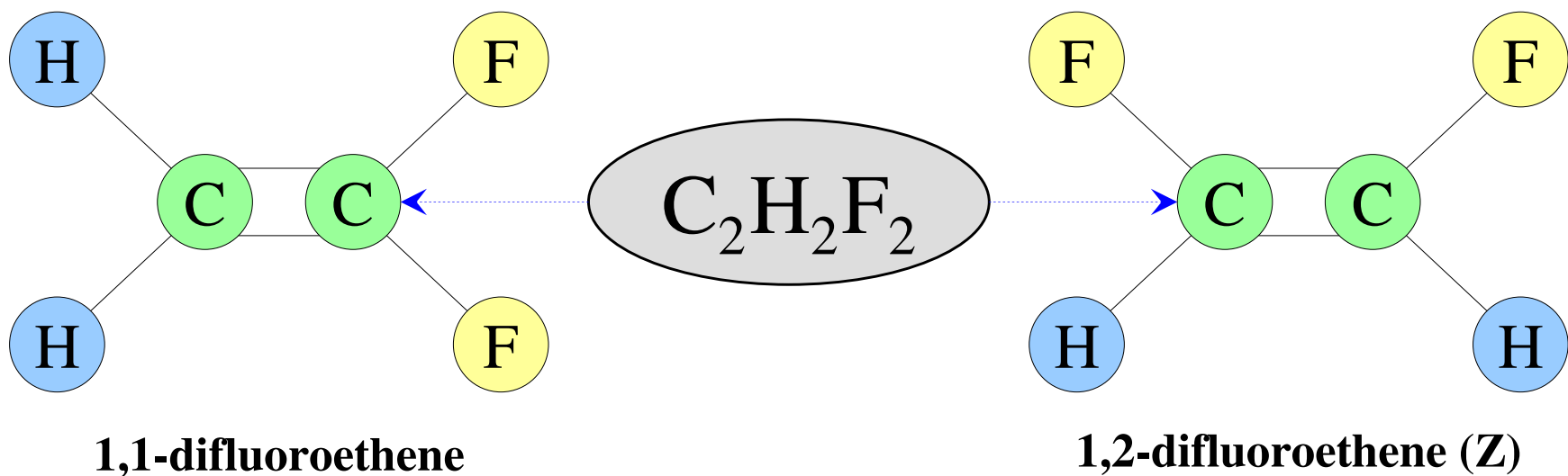
# Why Determine Graph Isomorphism?

- **Classic Uses**

- **Chemical isomer identification:** classic application
- **CAD object matching:** subgraph isomorphism
- **VLSI circuit template selection:** search space pruning
- **Optical Character Recognition (OCR):** segmented image matching

- **Military Uses**

- **Attack Tree Merging:** graph minimization
- **Threat Pattern Detection:** error-tolerant (approximate) isomorphism
- **Network verification:** validate deployed sensor network by a UAV



# A Famous Question

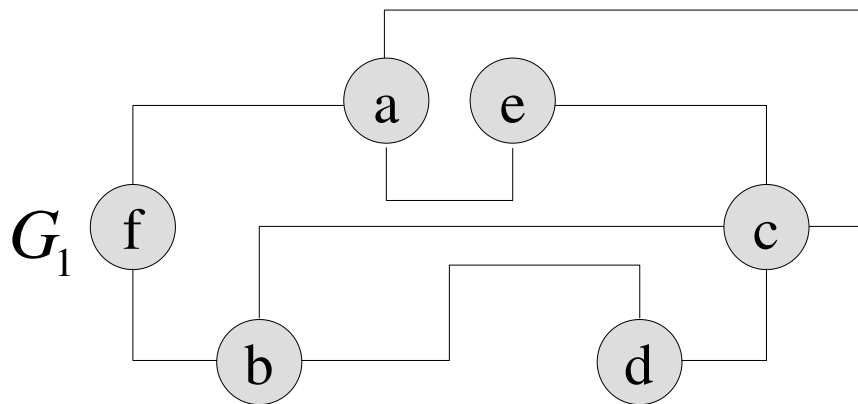
Graph (matrix) isomorphism is decidable in *exponential* time, i.e., in  $O(n!)$  time, by enumerating all possible permutations, but is it decidable in *polynomial* time for arbitrary graphs (matrices), i.e., we know  $\text{ISO} \in \text{NP}$ , (it is verifiable in  $O(n^3)$  time) but is  $\text{ISO} \in \text{P}$ ?

An algorithm for *deciding* isomorphism accepts arbitrary input, whereas an algorithm for *determining* isomorphism does not accept arbitrary input, i.e., it fails on at least one problem instance or is restricted to particular graphs, e.g., a  $\Theta(n)$  algorithm for trees or planar graphs.

# Isomorphism: The “Big” Decision

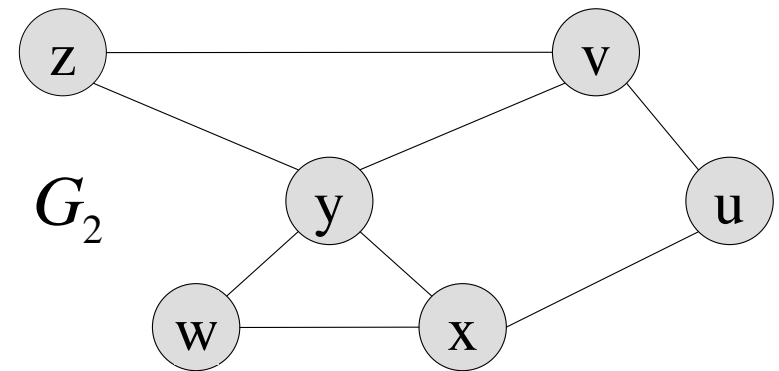
**Q:** Are these two graphs (matrices) isomorphs of each other?

**A:** {*yes, no*}; in this example, *yes*



?

$\cong$



$A_1$

	a	b	c	d	e	f
a	0	0	1	0	1	1
b	0	0	1	1	0	1
c	1	1	0	1	1	0
d	0	1	1	0	0	0
e	1	0	1	0	0	0
f	1	1	0	0	0	0

?

$\cong$

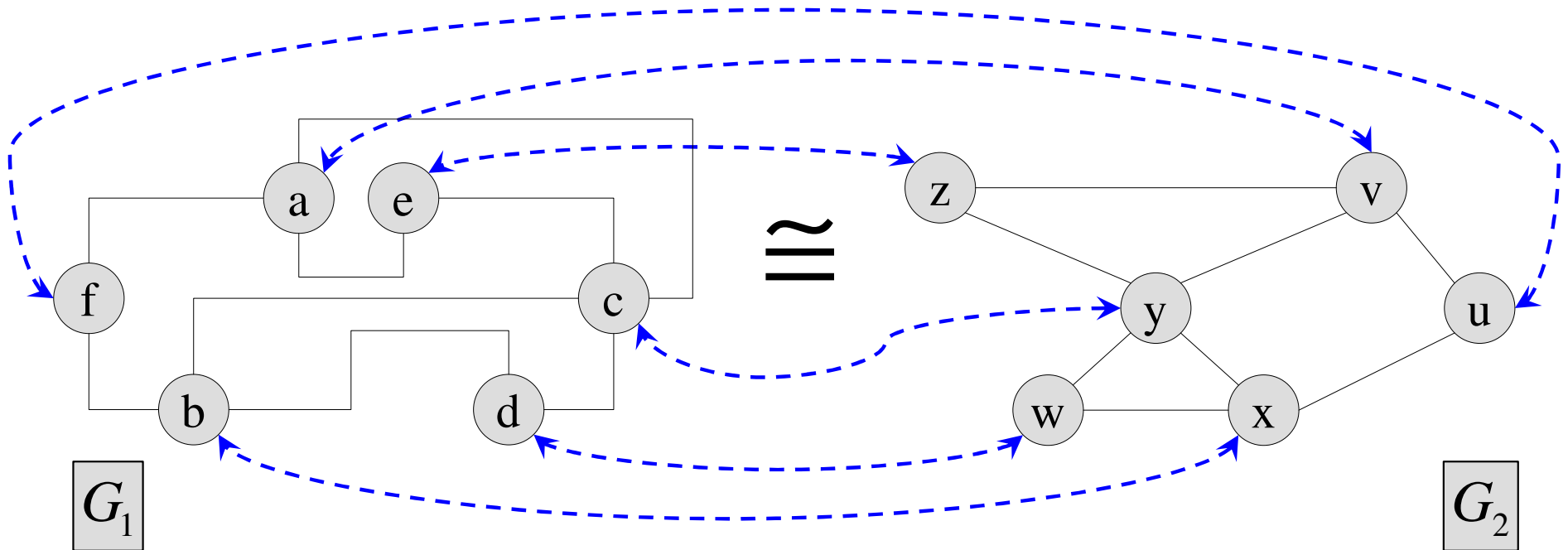
$A_2$

	u	v	w	x	y	z
u	0	1	0	1	0	0
v	1	0	0	0	1	1
w	0	0	0	1	1	0
x	1	0	1	0	1	0
y	0	1	1	1	0	1
z	0	1	0	0	1	0

# Determining Graph Isomorphism

**Q:** What is an isomorphism between  $G_1$  and  $G_2$ ?

**A:**  $V_2 = [u \ v \ w \ x \ y \ z] \cong [f \ a \ d \ b \ c \ e] = \phi([a \ b \ c \ d \ e \ f]) = \phi(V_1)$



$$G_1 \cong G_2 \iff \begin{aligned} &\exists \phi(V_1) = V_2 \text{ s.t.} \\ &\forall e_i = \{v_a, v_b\} \in E_1 \quad \wedge \quad v_a, v_b \in V_1, \\ &\exists e_j = \{\phi(v_a), \phi(v_b)\} \in E_2 \quad \wedge \quad \phi(v_a), \phi(v_b) \in V_2 \end{aligned}$$

# Determining Matrix Isomorphism

- Equivalent:  $G_1 \cong G_2 \leftrightarrow \mathbf{A}_1 \cong \mathbf{A}_2$
- Convenient:  $\mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^{-1}, O(n^3)$
- Efficient:  $\mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^T, O(n^2)$

Assume  $G$  is

1. simple
2. connected

	u	v	w	x	y	z
u	0	1	0	1	0	0
v	1	0	0	0	1	1
w	0	0	0	1	1	0
x	1	0	1	0	1	0
y	0	1	1	1	0	1
z	0	1	0	0	1	0

=

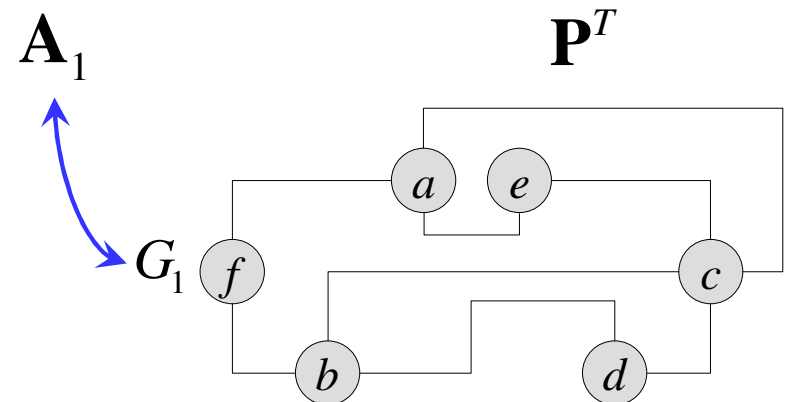
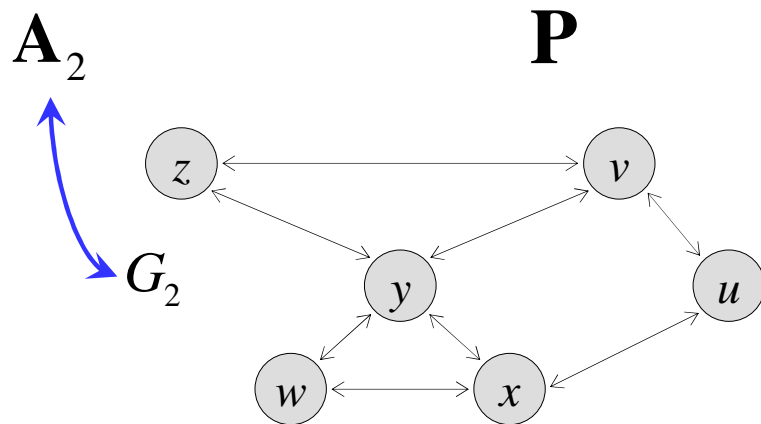
	1	2	3	4	5	6
1	0	0	0	0	0	1
2	1	0	0	0	0	0
3	0	0	0	1	0	0
4	0	1	0	0	0	0
5	0	0	1	0	0	0
6	0	0	0	0	1	0

×

	a	b	c	d	e	f
a	0	0	1	0	1	1
b	0	0	1	1	0	1
c	1	1	0	1	1	0
d	0	1	1	0	0	0
e	1	0	1	0	0	0
f	1	1	0	0	0	0

×

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	1	0	0
3	0	0	0	0	1	0
4	0	0	1	0	0	0
5	0	0	0	0	0	1
6	1	0	0	0	0	0



# A Template for Deciding Isomorphism

1. **Compare** invariants by increasing order of complexity

$$\Psi = \left[ |V|, |E|, \text{sort} \left( \{ \deg(v_1), \deg(v_2), \dots, \deg(v_n) \} \right), \text{eig}(\mathbf{A}), \dots \right]$$

2. **Compute** canonical isomorph,

$$\mathbf{A}_\omega = \mathbf{P}_\omega \cdot \mathbf{A} \cdot \mathbf{P}_\omega^T \leftarrow \textit{The hard part!}$$

3. **Compare** canonical isomorphs

$$(\mathbf{A}_1)_\omega \equiv (\mathbf{A}_2)_\omega \iff \mathbf{A}_1 \cong \mathbf{A}_2$$

```
graph TD; A1((A1)) --> A_omega((A_omega)); A2((A2)) --> A_omega; A1 <-.-> A2; A_omega -.-> HardPart[The hard part!]; HardPart -.-> EqBox["(A1)_omega ≡ (A2)_omega ⇔ A1 ≅ A2"];
```

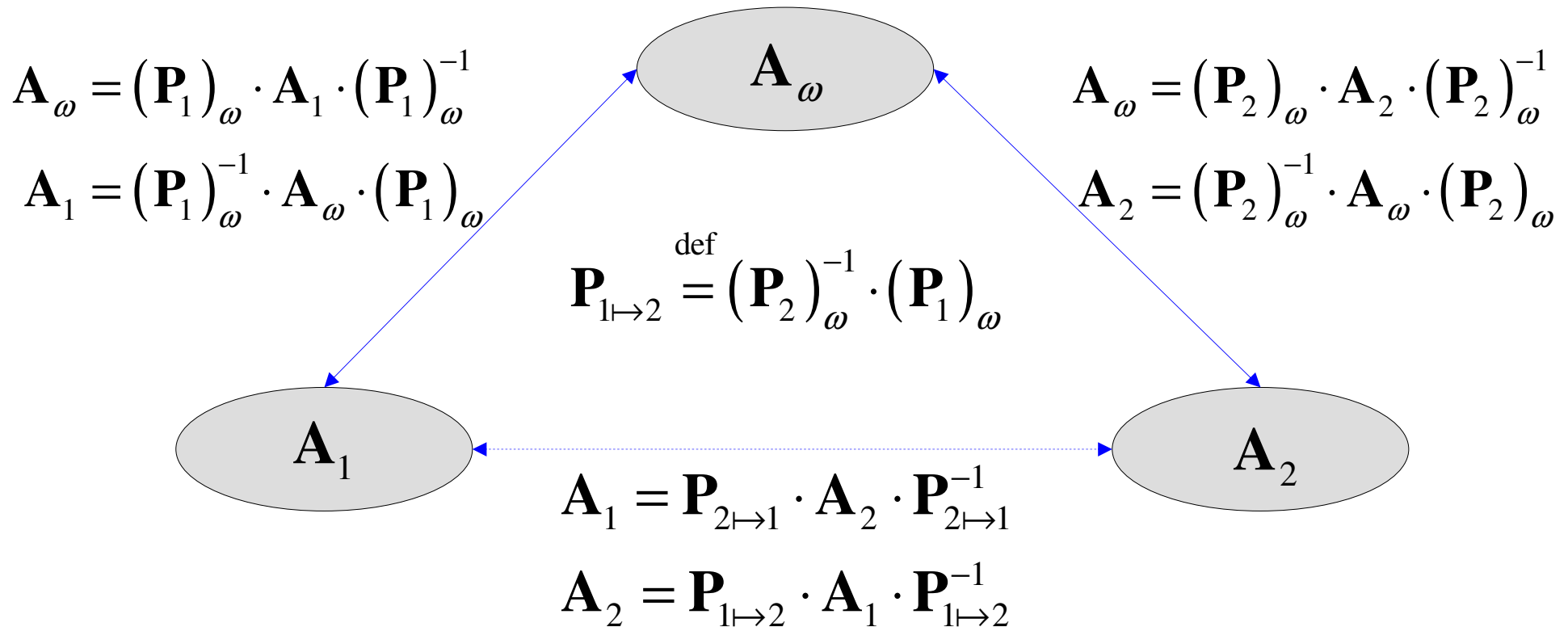
# Permutation Triangle

$\mathbf{A}_1$  –  $G_1$ 's adjacency matrix

$\mathbf{A}_2$  –  $G_2$ 's adjacency matrix

$\mathbf{A}_\omega$  – canonical isomorph

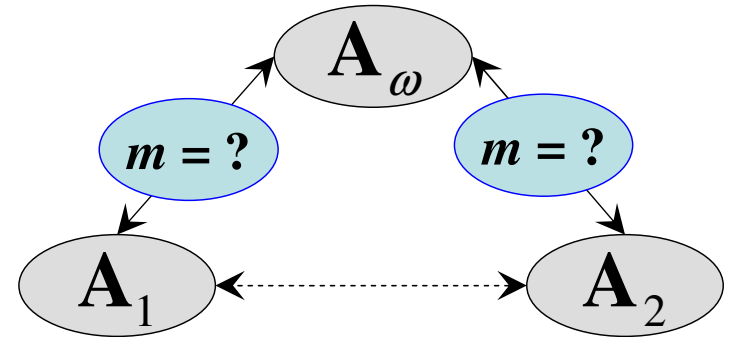
Assume  $G$  is simple & connected, therefore  $\mathbf{A}$  is a symmetric  $\{0,1\}$  matrix w/0s along its diagonal



# Our Approach: A Deterministic Permutation Chain

$$\mathbf{P}_\omega = \mathbf{P}_m \times \dots \times \mathbf{P}_2 \times \mathbf{P}_1$$

$$\mathbf{A}_{i+1} = \mathbf{P}_i \cdot \mathbf{A}_i \cdot \mathbf{P}_i^T = \mathbf{A}_i \xrightarrow{\mathbf{P}_i} \mathbf{A}_{i+1}$$

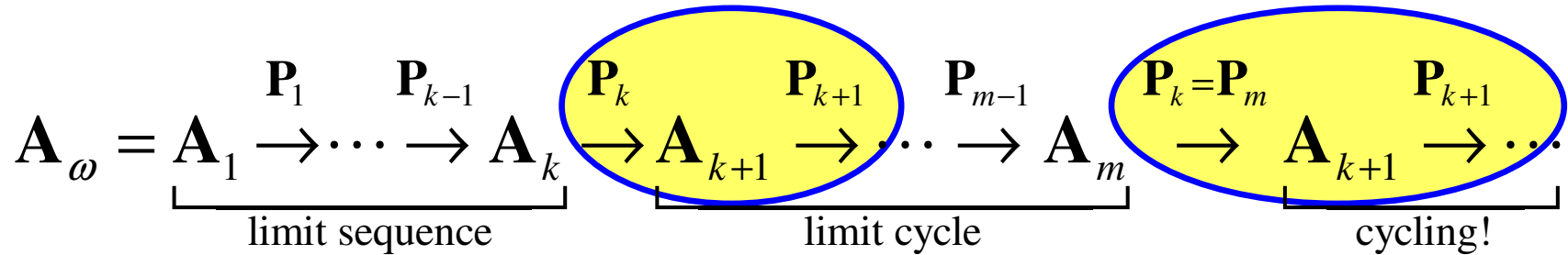


$$\mathbf{A}_\omega = \mathbf{P}_\omega \cdot \mathbf{A} \cdot \mathbf{P}_\omega^T = \mathbf{A}_1 \xrightarrow{\mathbf{P}_1} \mathbf{A}_2 \xrightarrow{\mathbf{P}_2} \dots \xrightarrow{\mathbf{P}_m} \mathbf{A}_m$$

$$\mathbf{A}_\omega = \mathbf{A}_1 \xrightarrow{\mathbf{P}_1} \mathbf{A}_2 \xrightarrow{\mathbf{P}_2} \dots \xrightarrow{\mathbf{P}_k} \mathbf{A}_k \xrightarrow{\mathbf{P}_{k+1}} \dots \xrightarrow{\mathbf{P}_m} \mathbf{A}_m \xrightarrow{\mathbf{P}_k} \mathbf{A}_k \xrightarrow{\mathbf{P}_{k+1}} \dots \xrightarrow{\mathbf{P}_m} \mathbf{A}_m$$

We observe cycling if  $\mathbf{P}_i$   
is *deterministically* computed

# Finding $\mathbf{A}_\omega$ by a Deterministic Permutation Chain



Does it yield

1. *deterministic* permutations, or one iteration, in polynomial time?

i.e.,  $\text{find}(\mathbf{P}_i) \in O(n^a)$ ,  $n = |V|$ ,  $a$  independent of  $n$

2. *short* limit sequences?

i.e.,  $|\text{limit sequence}| \in (k-1) \approx \log_2(n)$

3. *short* limit cycles?

i.e.,  $|\text{limit cycle}| \in (m-k) \approx \log_2(n)$

4. a *small* set of limit cycles (attractors)?

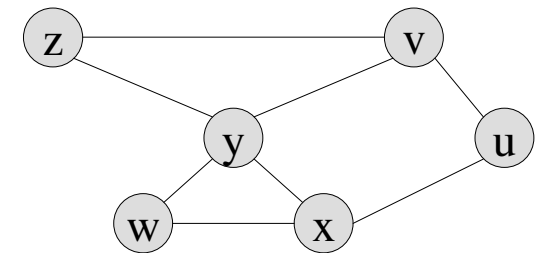
i.e.,  $|\mathbf{A}_\Omega| = 1$ , where  $\mathbf{A}_{\omega_1} = \mathbf{A}_{\omega_2} = \dots = \mathbf{A}_{\omega_r}$ ,  $r = |\text{Iso}(G)|$

# Classic Approaches: MCI & NAUTY

- **Perfect?:** Minimum Canonical Isomorph (MCI)
  - Isomorph yielding the smallest binary number after concatenating the columns of the entries in its upper-right triangle
  - Is guaranteed to exist and defined on an arbitrary subset of  $\text{Iso}(G)$
  - Also yields  $G$ 's maximum independent set &  $\overline{G}$ 's maximum clique
  - BUT, finding  $\text{MCI}(\mathbf{A}) \in \text{NP}$

$$\begin{aligned} \text{MCI}(\mathbf{A}) &= \min(\text{num}(\mathbf{A})) \\ &= 000011101011011_2 \\ &= 1,883_{10} \end{aligned}$$

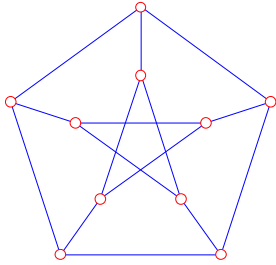
	z	w	u	x	v	y
z	0	0	0	0	1	1
w	0	0	0	1	0	1
u	0	0	0	1	1	0
x	0	1	1	0	0	1
v	1	0	1	0	0	1
y	1	1	0	1	1	0



- **Standard:** No AUTomorphisms, Yes? (*nauty*)
  - Constructs a tree of all possible permutations, then prunes it using a combination of backward search & the discovered automorphisms
  - Partitions vertices until all vertices are in a unit partition
  - Computes  $\text{Aut}(G)$  & finds an MCI (*not necessarily*  $\text{MCI}(\mathbf{A})!$ )
  - Exponential (in worst case), but is *much* faster in practice`

# Related Approaches: Using An “Information Matrix”

Petersen Graph



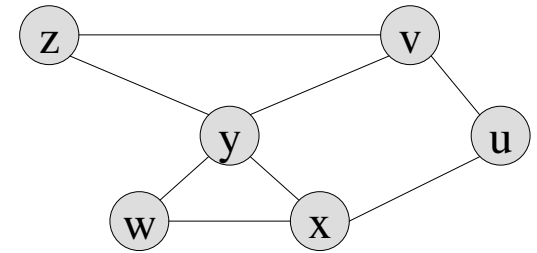
$\lambda_i$	1.00	0.28	0.28	0.28
A	0.32	0.01	-0.63	-0.06
B	0.32	0.40	0.28	-0.39
C	0.32	0.18	-0.35	-0.51
D	0.32	0.08	0.01	0.57

Eigenvector (*subset*)

- **Column-Wise Entry Grouping**
  - Compute *information matrix*
    - Eigenvectors ( $\mathbf{X}$ ) [HZL05]
    - Pseudoinverse ( $\mathbf{A}^\dagger$ ) [BeE96]
  - **Group** equal entries w/in columns to reduce tested permutations
- **Star Partitioning & Lexicographic Sorting [CRS97]**
  - **Find** *canonical* star partition, i.e., map vertices ( $V$ ) to eigenvalues ( $\lambda$ )
  - **Lexicographically sort** partitioned vertices on eigenvectors
  - **Yields** canonical isomorph, but finding *canonical* star partition in NP

# The Motivating Approach: PageRank

- Heart of Google™'s web page rankings
- Ranks pages by leading eigenvector [PBM+99]
  - perturbs matrix s.t. it is positive & row-stochastic
  - by Perron-Frobenius Theorem, it exists, is unique, & is positive
  - computable in  $O(n^3)$  time, e.g., using power iteration
- Yields canonical isomorph on ~75% of random graphs, e.g., WWW
- However, if duplicate entries exist, may not be canonical...



	u	v	w	x	y	z
u	0	1	0	1	0	0
v	1	0	0	0	1	1
w	0	0	0	1	1	0
x	1	0	1	0	1	0
y	0	1	1	1	0	1
z	0	1	0	0	1	0
Σ	2	3	2	3	4	2

	u	v	w	x	y	z
u	0.025	0.308	0.025	0.308	0.025	0.025
v	0.450	0.025	0.025	0.025	0.238	0.450
w	0.025	0.025	0.025	0.308	0.238	0.025
x	0.450	0.025	0.450	0.025	0.238	0.025
y	0.025	0.308	0.450	0.308	0.025	0.450
z	0.025	0.308	0.025	0.025	0.238	0.025
Σ	1.000	1.000	1.000	1.000	1.000	1.000

Λ	1.00		y	v	x	u	w	z
u	0.31	y	0	1	1	0	1	1
v	0.44	v	1	0	0	1	0	1
w	0.31	x	1	0	0	1	1	0
x	0.44	u	0	1	1	0	0	0
y	0.57	w	1	0	1	0	0	0
z	0.31	z	1	1	0	0	0	0

$$\alpha = 0.85$$

$$\delta = (1 - \alpha) / n$$

$$\mathbf{D} = \mathbf{0}^{n,n}$$

$$D_{i,i} = \sum A_{i,:}$$

$$\mathbf{A}' = \alpha \cdot \mathbf{D}^{-1} \cdot \mathbf{A} + \delta$$

$$(\mathbf{A} - \lambda \mathbf{I}) \cdot \mathbf{X} = 0, \mathbf{X} \neq 0$$

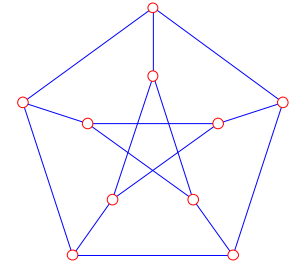
$$\mathbf{A} = \mathbf{X}_\lambda \cdot \Lambda \cdot \mathbf{X}_\lambda^{-1}$$

$$\mathbf{P} = \mathbf{I}^n (\phi(V), :)$$

$$\mathbf{A}_\omega = \mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^T$$

# PageRank & Canonical Isomorphs

- **Extreme cases**, *all* leading eigenvector entries are the same.
- **Thus**, resulting ordering will not generally be canonical.
- **What** if we *lexicographically sort* on every eigenvectors, i.e., if we use the eigenvectors as an *information matrix*?



Petersen Graph

Since all entries of leading eigenvector are equal, ordering yields  $P = I$

$\lambda_i$	1.00	0.28	0.28	0.28	0.28	0.28	-0.57	-0.57	-0.57	-0.57
<b>A</b>	0.32	0.01	-0.63	-0.06	-0.18	0.25	-0.07	0.44	0.05	0.45
<b>B</b>	0.32	0.40	0.28	-0.39	-0.23	-0.24	0.43	-0.06	0.40	0.23
<b>C</b>	0.32	0.18	-0.35	-0.51	0.22	-0.19	0.01	-0.12	-0.45	-0.42
<b>D</b>	0.32	0.08	0.01	0.57	0.24	-0.33	0.37	-0.06	-0.41	0.30
<b>E</b>	0.32	-0.23	0.01	-0.06	0.64	-0.20	-0.38	-0.13	0.45	0.17
<b>F</b>	0.32	-0.35	0.34	-0.16	-0.46	-0.17	-0.44	-0.14	-0.36	0.25
<b>G</b>	0.32	-0.26	-0.29	0.35	-0.40	-0.25	0.07	-0.12	0.36	-0.50
<b>H</b>	0.32	0.57	0.29	0.28	0.00	0.13	-0.43	0.37	0.00	-0.28
<b>I</b>	0.32	-0.48	0.34	-0.12	0.18	0.32	0.38	0.45	-0.05	-0.23
<b>J</b>	0.32	0.09	0.00	0.10	0.00	0.69	0.06	-0.63	0.00	0.03

# Plan B: Other Information Matrices

- Lexicographic sorting on eigenvectors yields a canonical isomorph on 75% of *random* graphs?
- Are there any other information matrices that yield better results using lexicographic sorting?
- If so, it is likely that such a matrix is *unique* up to isomorphism—does such a matrix *exist*?

$$\mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^T \leftrightarrow \mathbf{P} \cdot \mathbf{X} \cdot \mathbf{P}^T$$

- The answer is “yes”; a potential choice is used in network routing—the all-pairs shortest path matrix, computable in  $O(n^3)$  time.

# Some Key Linear Algebra Concepts

<b>A</b>	adjacency matrix
<b>I</b>	identity matrix
<b>0</b> or <b>Z</b> ( <b>z</b> )	all-zero matrix (vector)
<b>1</b> or <b>J</b> ( <b>j</b> )	all-ones matrix (vector)
<b>D</b> ( <b>d</b> )	degree matrix (vector), $\mathbf{D} = \mathbf{0}$ , $\mathbf{D}_{i,i} = \mathbf{d}_i = \text{deg}(v_i)$

Linear System:  $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$

Matrix Inverse:  $\mathbf{A} \cdot \mathbf{X} = \mathbf{B} = \mathbf{I}$ , i.e.,  $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}$

But,  $\mathbf{A}^{-1}$  may not exist—one alternative is the pseudoinverse,

$$\mathbf{A}^\dagger = (\mathbf{A}^T \cdot \mathbf{A})^{-1} \cdot \mathbf{A}^T; \text{ if } \mathbf{A}^{-1} \text{ exists, then } \mathbf{A}^{-1} = \mathbf{A}^\dagger$$

Or, apply *isomorphism-preserving perturbations* to  $\mathbf{A}$  such that

$(\mathbf{A}')^{-1}$  exists, for example...

# The Laplacian & Spectral Graph Theory

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

Laplacian [BeW04]

$$\mathbf{L}^+ = \mathbf{D} + \mathbf{A}$$

signless Laplacian [HaS04]

$$\mathbf{L}^{+\varepsilon} = \mathbf{D} + \mathbf{A} + \varepsilon \cdot \mathbf{I}$$

modified signless Laplacian

If  $\varepsilon > 0$ ,  $\mathbf{L}^{+\varepsilon}$  is *strictly doubly diagonally dominant*, i.e.,  $\mathbf{D}_{i,i} > \sum \mathbf{A}_{i,:}$

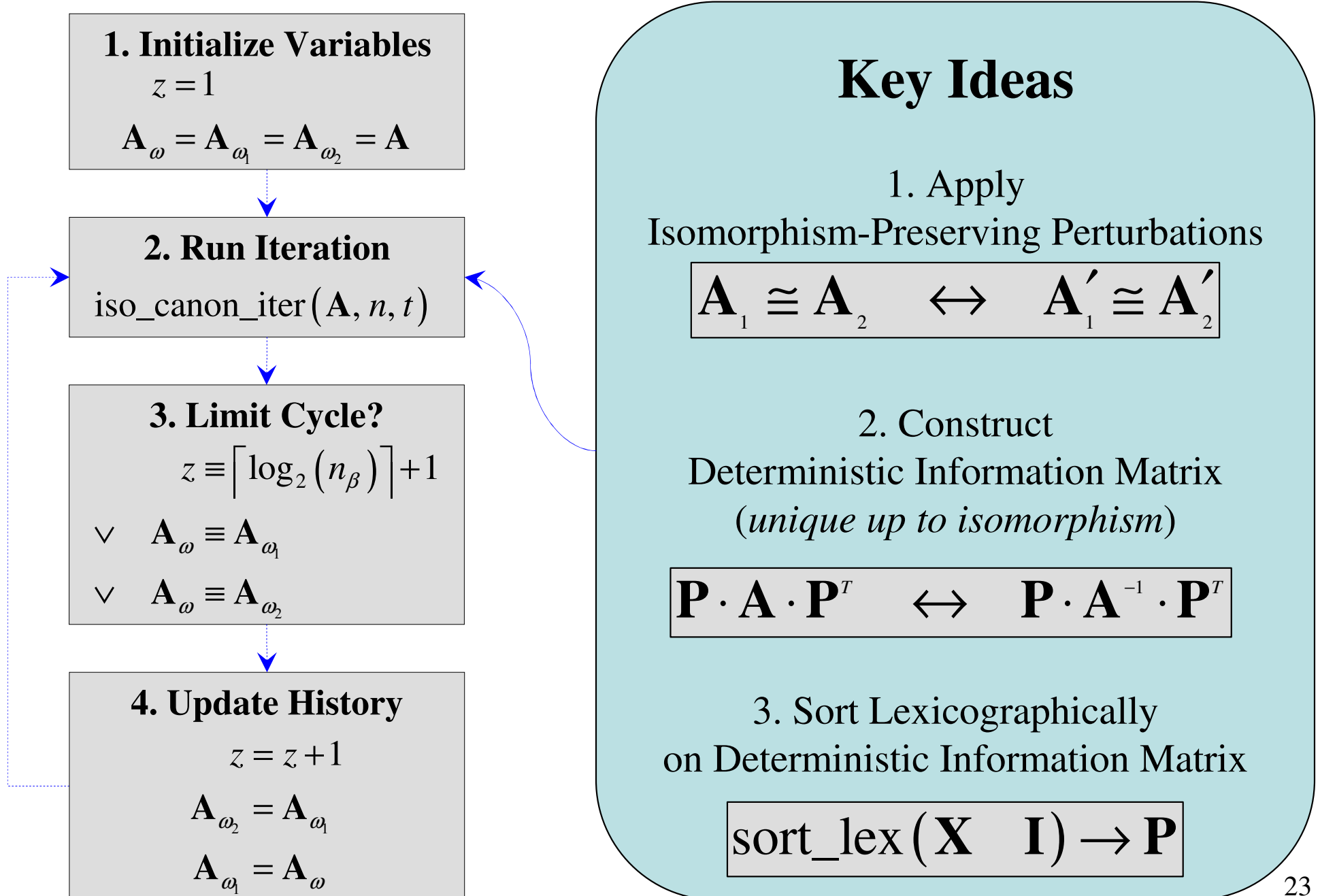
→ positive definite, by Gershgorin Circle Theorem [Var04]

→  $\mathbf{A}^{-1}$  exists and is unique matrix up to isomorphism

→  $\mathbf{A}^{-1}$  is computable using Cholesky decomposition, the most efficient and numerically stable method for directly obtaining the inverse of a symmetric positive definite matrix

→ what is a good value for  $\varepsilon$ , i.e., how do we construct  $\mathbf{L}^{+\varepsilon}$ , or, what is an appropriate *isomorphism-preserving perturbation*?

# IsoCanon: Main Loop



# IsoCanon: One Iteration

*Perturb & Obtain Inverse*

**1. Add  $\beta$ -Vertex**

$$\mathbf{A}_\beta = \begin{bmatrix} \mathbf{0}^{1,1} & \mathbf{1}^{1,n} \\ \mathbf{1}^{n,1} & \mathbf{A} \end{bmatrix}$$

**2. Dominate Diagonal**  
(modified +Laplacian)

$$\mathbf{A}_\beta^{+\varepsilon} = \mathbf{A}_\beta + \mathbf{D}_\beta + \mathbf{D}_\beta^{-1}$$

$$n_\beta = n + 1$$

**3. Compute Inverse**

$$\mathbf{S}_\beta = (\mathbf{A}_\beta^{+\varepsilon})^{-1}$$

$$\mathbf{S}'_\beta = \mathbf{S}_\beta(2:n_\beta, :)$$

*Construct Deterministic Information Matrix*

**6. Construct Information Matrix**

$$\mathbf{X}_\beta = \begin{bmatrix} \mathbf{T}'_\beta & \mathbf{T}_\beta & \mathbf{I}^{n,n} \end{bmatrix}$$

**5. Sort Row Vectors**  
(could be orbits!)

$$\mathbf{T}'_\beta = \text{sort\_vecs}(\mathbf{T}_\beta)$$

**4. Round Entries**  
(due to finite precision)

$$t \in (-\infty, -14]$$

$$\mathbf{T}_\beta = \text{round}(\mathbf{S}'_\beta, t)$$

*Sort Lexicographically on Information Matrix & Apply Permutation*

**7. Sort Lexicographically**

$$n_c = 2 \cdot n_\beta$$

$$\mathbf{X}'_\beta = \text{sort\_lex}(\mathbf{X}_\beta, [1:n_c])$$

**8. Extract P-Matrix**

$$col_s = 2 \cdot n_\beta + 1$$

$$col_e = 2 \cdot n_\beta + n$$

$$\mathbf{P}_\omega = \mathbf{X}_\omega(:, col_s : col_e)$$

**9. Permute Input**

$$\mathbf{A}_\omega = \mathbf{P}_\omega \cdot \mathbf{A} \cdot \mathbf{P}_\omega^T$$

```

function [A_omega] = iso_short(A, n, t)
    % add beta vertex & compute modified signless Laplacian
    A_beta = [0, ones(1, n); ones(n, 1), A];
    d = sum(A_beta, 2);
    D_beta = diag(d + 1./d);
    A_beta = A_beta + D_beta;

    % compute inverse & extract non-beta rows
    S_beta = A_beta \ eye(n + 1);
    S_beta_p = S_beta(2:(n + 1), :);

    % round due to finite precision & sort rows
    T_beta = roundn(S_beta_p, t);
    T_beta_sort = sort(T_beta, 2);

    A_omega = A;
    A_omega_1 = A_omega - 1;
    % iterate on base-2 log relative to |V_beta|
    for i = 1:(ceil(log2(n + 1)) + 1)
        % construct & sort information matrix
        P_omega = sortrowsc([-T_beta_sort, T_beta]);

        % extract & apply induced permutation
        A_omega = A_omega(P_omega, P_omega);

        % check for limit of length {1} vs. update?
        if (any(any(A_omega ~= A_omega_1)) == 0)
            break;
        else
            A_omega_1 = A_omega;
            T_beta_sort = T_beta_sort(P_omega, :);
            T_beta = T_beta(P_omega, [1; (P_omega+1)]);
        end
    end
end
end

```

*IsoCanon*  
Source  
Code  
(MATLAB)

# Steps 1 & 2: Apply Isomorphism-Preserving Perturbations

**$\beta$ -Vertex** [PrD84, CRS97]  
 Aids distinguishing co-spectral graphs

**Diagonal Dominance** [Has04, Var04]  
 $A^{-1}$  exists using *modified* signless Laplacian & Gershgorin Circle theorem

1. Add  $\beta$ -Vertex

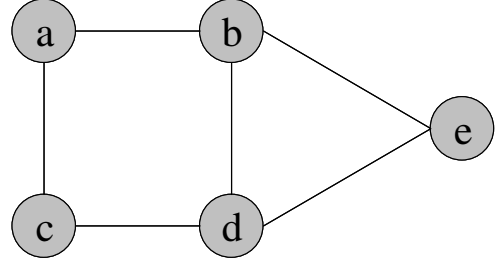
$$A_\beta = \begin{bmatrix} \mathbf{0}^{1,1} & \mathbf{1}^{1,n} \\ \mathbf{1}^{n,1} & A \end{bmatrix}$$

2. Dominate Diagonal

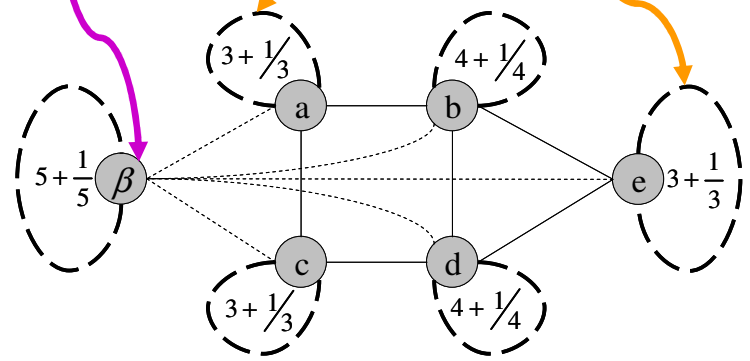
$$A_\beta^{+\varepsilon} = D_\beta + A_\beta + D_\beta^{-1}$$

$$n_\beta = n + 1$$

“House” Graph

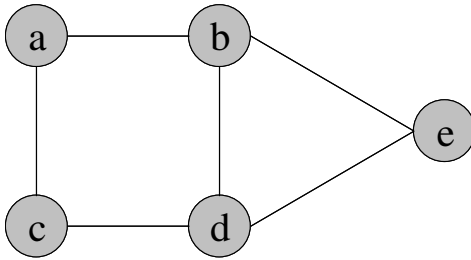


	a	b	c	d	e
a	0	1	1	0	0
b	1	0	0	1	1
c	1	0	0	1	0
d	0	1	1	0	1
e	0	1	0	1	0



	$\beta$	a	b	c	d	e
$\beta$	$5\frac{1}{5}$	1	1	1	1	1
a	1	$3\frac{1}{3}$	1	1	0	0
b	1	1	$4\frac{1}{4}$	0	1	1
c	1	1	0	$3\frac{1}{3}$	1	0
d	1	0	1	1	$4\frac{1}{4}$	1
e	1	0	1	0	1	$3\frac{1}{3}$

# Steps 3 & 4: Compute & Round Inverse



## 3. Compute Inverse & Remove $\beta$ -vertex

$$\mathbf{S}_\beta = (\mathbf{A}_\beta^{+\varepsilon})^{-1}$$

$$\mathbf{S}'_\beta = \mathbf{S}_\beta(2:n_\beta, :)$$

	$\beta$	a	b	c	d	e
$\beta$	$\frac{11490}{49853}$	$-\frac{2370}{49853}$	$-\frac{1220}{49853}$	$-\frac{2370}{49853}$	$-\frac{1220}{49853}$	$-\frac{2715}{49853}$
a	$-\frac{2370}{49853}$	$\frac{1488912}{3938387}$	$-\frac{382068}{3938387}$	$-\frac{455355}{3938387}$	$\frac{216168}{3938387}$	$\frac{1341}{49853}$
b	$-\frac{1220}{49853}$	$-\frac{382068}{3938387}$	$\frac{1153772}{3938387}$	$\frac{216168}{3938387}$	$-\frac{242112}{3938387}$	$-\frac{3096}{49853}$
c	$-\frac{2370}{49853}$	$-\frac{455355}{3938387}$	$\frac{216168}{3938387}$	$\frac{1488912}{3938387}$	$-\frac{382068}{3938387}$	$\frac{1341}{49853}$
d	$-\frac{1220}{49853}$	$\frac{216168}{3938387}$	$-\frac{242112}{3938387}$	$-\frac{382068}{3938387}$	$\frac{1153772}{3938387}$	$-\frac{3096}{49853}$
e	$-\frac{2715}{49853}$	$\frac{1341}{49853}$	$-\frac{3096}{49853}$	$\frac{1341}{49853}$	$-\frac{3096}{49853}$	$\frac{17628}{49853}$

	$\beta$	a	b	c	d	e
a	-0.048	0.378	-0.097	-0.116	0.055	0.027
b	-0.024	-0.097	0.293	0.055	-0.061	-0.062
c	-0.048	-0.116	0.055	0.378	-0.097	0.027
d	-0.024	0.055	-0.061	-0.097	0.293	-0.062
e	-0.054	0.027	-0.062	0.027	-0.062	0.354

## 4. Round Entries

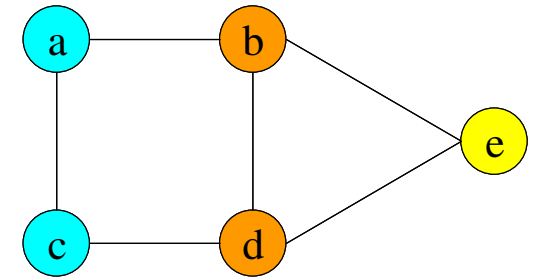
$$t \in (-\infty, -14]$$

$$\mathbf{T}_\beta = \text{round}(\mathbf{S}'_\beta, t)$$

# Step 5: Sort Inverse's Row Vectors

Note same inverse entries for vertices in different orbits!

	$\beta$	a	b	c	d	e
a	-0.048	0.378	-0.097	-0.116	0.055	0.027
b	-0.024	-0.097	0.293	0.055	-0.061	-0.062
c	-0.048	-0.116	0.055	0.378	-0.097	0.027
d	-0.024	0.055	-0.061	-0.097	0.293	-0.062
e	-0.054	0.027	-0.062	0.027	-0.062	0.354



$$\mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^T \leftrightarrow \mathbf{P} \cdot \mathbf{A}^{-1} \cdot \mathbf{P}^T$$

	Equal Rows?					
a	-0.116	-0.097	-0.048	0.027	0.055	0.378
b	-0.097	-0.062	-0.061	-0.024	0.055	0.293
c	-0.116	-0.097	-0.048	0.027	0.055	0.378
d	-0.097	-0.062	-0.061	-0.024	0.055	0.293
e	-0.062	-0.062	-0.054	0.027	0.027	0.354

	Equal Rows?
a	{a, c}
b	{b, d}
c	{a, c}
d	{b, d}
e	{e}

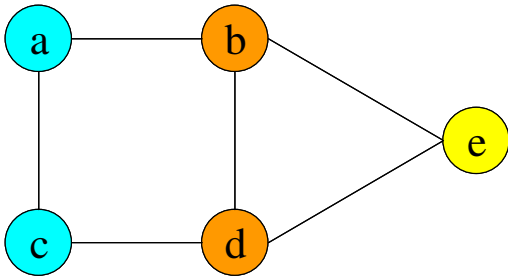
**5. Sort Row Vectors**

$$\mathbf{T}'_{\beta} = \text{sort\_vecs}(\mathbf{T}_{\beta})$$

Sort *within* each row from left to right

We pre-group the rows for *presentation* herein

# Step 6: Construct Information Matrix

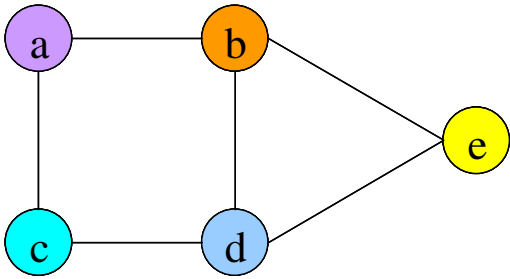


**6. Construct Information Matrix**

$$\mathbf{X}_\beta = [\mathbf{T}'_\beta \quad \mathbf{T}_\beta \quad \mathbf{I}^{n,n}]$$

	$\mathbf{T}'_\beta$	$\mathbf{T}_\beta$						$\mathbf{I}^{n,n}$				
		$\beta$	a	b	c	d	e	a	b	c	d	e
a	{a, c}	-0.048	0.378	-0.097	-0.116	0.055	0.027	1	0	0	0	0
b	{b, d}	-0.024	-0.097	0.293	0.055	-0.061	-0.062	0	1	0	0	0
c	{a, c}	-0.048	-0.116	0.055	0.378	-0.097	0.027	0	0	1	0	0
d	{b, d}	-0.024	0.055	-0.061	-0.097	0.293	-0.062	0	0	0	1	0
e	{e}	-0.054	0.027	-0.062	0.027	-0.062	0.354	0	0	0	0	1

# Step 7: Lexicographically Sort on Constructed Information Matrix



## 7. Sort Lexicographically

$$n_c = 2 \cdot n_\beta$$

$$\mathbf{X}'_\beta = \text{sort\_lex}(\mathbf{X}_\beta, [1:n_c])$$

	$\mathbf{T}'_\beta$	$\mathbf{T}_\beta$						$\mathbf{P}_\omega$				
		$\beta$	a	b	c	d	e	a	b	c	d	e
c	{a, c}	-0.048	-0.116	0.055	0.378	-0.097	0.027	0	0	1	0	0
a	{a, c}	-0.048	0.378	-0.097	-0.116	0.055	0.027	1	0	0	0	0
b	{b, d}	-0.024	-0.097	0.293	0.055	-0.061	-0.062	0	1	0	0	0
d	{b, d}	-0.024	0.055	-0.061	-0.097	0.293	-0.062	0	0	0	1	0
e	{e}	-0.054	0.027	-0.062	0.027	-0.062	0.354	0	0	0	0	1

Sorting terminates,  
unique sorted row determined!

Induced permutation matrix

# Steps 8 & 9: Extract & Apply Permutation to Input Matrix

## 8. Extract Permutation Matrix

$$col_s = 2 \cdot n_\beta + 1$$

$$col_e = 2 \cdot n_\beta + n$$

$$\mathbf{P}_\omega = \mathbf{X}_\omega(:, col_s : col_e)$$

## 9. Permute Input

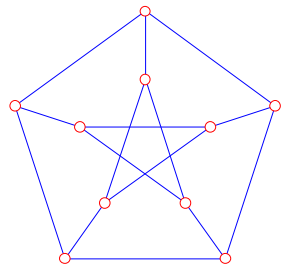
$$\mathbf{A}_\omega = \mathbf{P}_\omega \cdot \mathbf{A} \cdot \mathbf{P}_\omega^T$$

$\mathbf{P}_\omega$					×	$\mathbf{A}$					×	$\mathbf{P}_\omega^T$				
0	0	1	0	0		0	1	1	0	0		0	1	0	0	0
1	0	0	0	0		1	0	0	1	1		0	0	1	0	0
0	1	0	0	0		1	0	0	1	0		1	0	0	0	0
0	0	0	1	0		0	1	1	0	1		0	0	0	1	0
0	0	0	0	1		0	1	0	1	0		0	0	0	0	1

$\mathbf{A}_\omega$					
	c	a	b	d	e
c	0	1	0	1	0
a	1	0	1	0	0
b	0	1	0	1	1
d	1	0	1	0	1
e	0	0	1	1	0

$$\mathbf{A}_\Omega = \{\mathbf{A}_\omega\} = \underbrace{\mathbf{A}_1}_{\text{limit sequence}} \xrightarrow{\mathbf{P}_{k=1}} \underbrace{\mathbf{A}_{k=m}}_{\text{limit cycle}} \xrightarrow{\mathbf{P}_{k=m}} \underbrace{\mathbf{A}_k}_{\text{cycling!}} \rightarrow \dots$$

# Petersen Graph Iterations



Iteration 1

	a	b	c	d	e	f	g	h	i	j
a	0	1	0	0	0	1	1	0	0	0
b	1	0	1	0	1	0	0	0	0	0
c	0	1	0	1	0	0	0	0	1	0
d	0	0	1	0	0	1	0	1	0	0
e	0	1	0	0	0	0	0	1	0	1
f	1	0	0	1	0	0	0	0	0	1
g	1	0	0	0	0	0	0	1	1	0
h	0	0	0	1	1	0	1	0	0	0
i	0	0	1	0	0	0	1	0	0	1
j	0	0	0	0	1	1	0	0	1	0

$A_1$

$A_2$

	q	r	s	t	u	v	w	x	y	z
q	0	0	0	0	1	0	0	0	1	1
r	0	0	0	0	1	1	1	0	0	0
s	0	0	0	0	0	1	0	1	0	1
t	0	0	0	0	0	0	1	1	1	0
u	1	1	0	0	0	0	0	1	0	0
v	0	1	1	0	0	0	0	0	1	0
w	0	1	0	1	0	0	0	0	0	1
x	0	0	1	1	1	0	0	0	0	0
y	1	0	0	1	0	1	0	0	0	0
z	1	0	1	0	0	0	1	0	0	0

$A_{\omega?}$

Iteration 2

	a	c	e	j	d	i	h	b	f	g
a	0	0	0	0	0	0	0	1	1	1
c	0	0	0	0	1	1	0	1	0	0
e	0	0	0	1	0	0	1	1	0	0
j	0	0	1	0	0	1	0	0	1	0
d	0	1	0	0	0	0	1	0	1	0
i	0	1	0	1	0	0	0	0	0	1
h	0	0	1	0	1	0	0	0	0	1
b	1	1	1	0	0	0	0	0	0	0
f	1	0	0	1	1	0	0	0	0	0
g	1	0	0	0	0	1	1	0	0	0

$A_{\omega?}$

Iteration 3

	a	j	c	h	e	d	i	g	f	b
a	0	0	0	0	0	0	0	1	1	1
j	0	0	0	0	1	0	1	0	1	0
c	0	0	0	0	0	1	1	0	0	1
h	0	0	0	0	1	1	0	1	0	0
e	0	1	0	1	0	0	0	0	0	1
d	0	0	1	1	0	0	0	0	1	0
i	0	1	1	0	0	0	0	1	0	0
g	1	0	0	1	0	0	1	0	0	0
f	1	1	0	0	0	1	0	0	0	0
b	1	0	1	0	1	0	0	0	0	0

$A_{\omega!}$

	a	j	c	h	e	d	i	g	f	b
a	0	0	0	0	0	0	0	1	1	1
j	0	0	0	0	1	0	1	0	1	0
c	0	0	0	0	0	1	1	0	0	1
h	0	0	0	0	1	1	0	1	0	0
e	0	1	0	1	0	0	0	0	0	1
d	0	0	1	1	0	0	0	0	1	0
i	0	1	1	0	0	0	0	1	0	0
g	1	0	0	1	0	0	1	0	0	0
f	1	1	0	0	0	1	0	0	0	0
b	1	0	1	0	1	0	0	0	0	0

# Permutation Chain Performance I

$$\mathbf{A}_\omega = \underbrace{\mathbf{A}_1 \xrightarrow{\mathbf{P}_1} \cdots \xrightarrow{\mathbf{P}_{k-1}} \mathbf{A}_k}_{\text{limit sequence}} \xrightarrow{\mathbf{P}_k} \underbrace{\mathbf{A}_{k+1} \xrightarrow{\mathbf{P}_{k+1}} \cdots \xrightarrow{\mathbf{P}_{m-1}} \mathbf{A}_m}_{\text{limit cycle}} \xrightarrow{\mathbf{P}_k = \mathbf{P}_m} \underbrace{\mathbf{A}_{k+1} \xrightarrow{\mathbf{P}_{k+1}} \cdots}_{\text{cycling!}}$$

Does IsoCanon yield

1. *deterministic* permutations in polynomial time?

Yes,  $\text{find}(\mathbf{P}_i) \in O(n^3 \cdot \log(n))$ ,  $n = |V|$

- i.  $\text{compute}(\mathbf{A}^{-1}) \in O(n^3)$ , using Cholesky decomposition
- ii.  $\text{sort\_lex}(\mathbf{S}) \in O(n^3 \cdot \log(n))$ , using *quicksort*

2. *short* limit sequences?

3. *short* limit cycles?

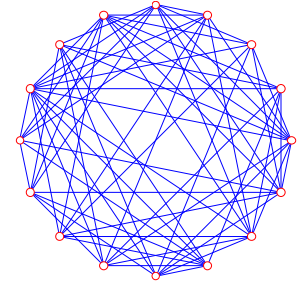
4. *small* sets of limit cycles?

**TBD (experiments)**

**Graphs:**  $\mathbf{A}_{i,j} \in \{0,1\} \wedge \kappa(G) = 1$   
**Implementation:** MATLAB (BLAS)  
**Rounding:** 13 digits, hardware double

# Experiment 1: *Random Graphs*

- Select graph edge probability,  $\Pr(E)$
- Insert edge  $e_i$  if  $\Pr(e_i) \geq \Pr(E)$  for  $C(n, 2)$  edges
- 1,024 graphs for  $\Pr(e_i) = 0.5$  and  $\Pr(e_i) \in [0, 1]$
- (2) random isomorphs of *each* of above graphs



Random Graph  
 $n = 16, \Pr(e_i) = 0.5$

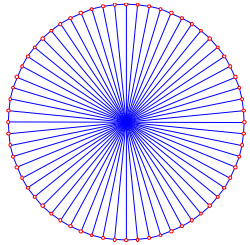
Graph Family	$n =  V $	limit sequence	limit cycle	$ \mathbf{A}_\Omega $
$\Pr(e_i) = 0.5$	64	0	1	1
$\Pr(e_i) \in [0, 1]$	64	0	1	1

In sum, finds canonical isomorph on 1<sup>st</sup> iteration!

$$\mathbf{A}_\Omega = \{\mathbf{A}_\omega\} = \underbrace{\mathbf{A}_1}_{\text{limit sequence}} \xrightarrow{\mathbf{P}_{k=1}} \underbrace{\mathbf{A}_{k=m}}_{\text{limit cycle}} \xrightarrow{\mathbf{P}_{k=m}} \underbrace{\mathbf{A}_k}_{\text{cycling!}} \rightarrow \dots$$

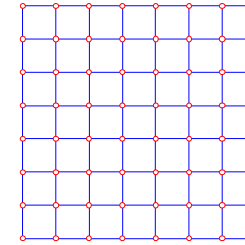
# Exp. 2: Regular & Strongly Regular Graphs

- Often cited as difficult graphs to determine isomorphism for
- 1,024 isomorphs of each of (3) regular / strongly regular graphs

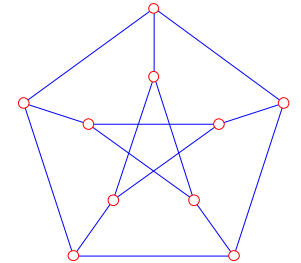


Mobius Ladder,  $n = 32$

$m : y :: \text{length} : \text{tally}$   
(tally of 1024)



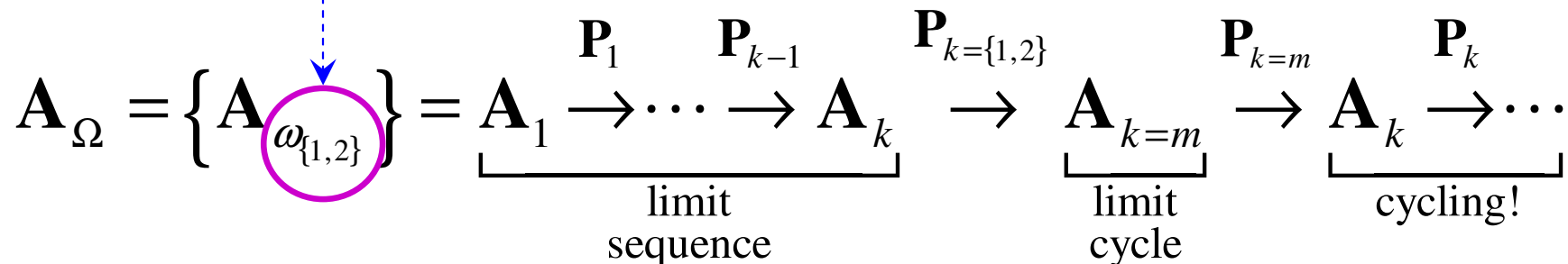
Grid (8 x 8)



Petersen

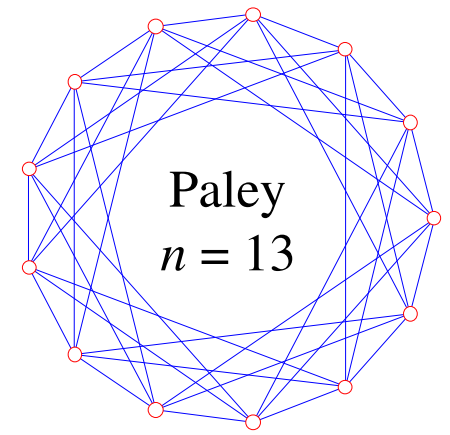
Graph Family	$n =  V $	limit sequence	limit cycle	$ \mathbf{A}_\Omega $
Mobius Ladder	64	0	1	1
Grid (8 x 8)	64	0:245, 1:779	1	1
Petersen	10	0:527, 1:497	1	2

Use random permutations to fix w.h.p., may also be due to  $\kappa(\mathbf{A})$



# Exp. 3: Really Strongly Regular Graphs

- What about a more “difficult” graph?
- 1,024 isomorphs of (2) Paley graphs



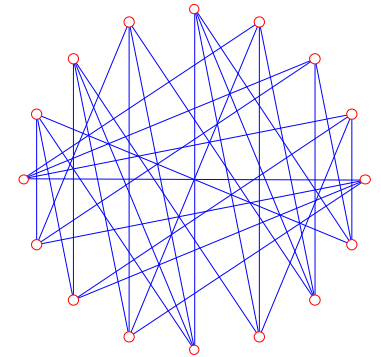
Graph Family	$n =  V $	limit sequence	limit cycle	$ \mathbf{A}_\Omega $
Paley	13	0:16, 1:500, 2:508	1	2
Paley	82	2:14, 3:296, 4:584, 5:130	1	60

First sign of *non-polynomial* behavior; random permutation resolves w.h.p.

$$\mathbf{A}_\Omega = \left\{ \mathbf{A}_{\omega_{[1,60]}} \right\} = \underbrace{\mathbf{A}_1 \xrightarrow{\mathbf{P}_1} \cdots \xrightarrow{\mathbf{P}_{k-1}} \mathbf{A}_k}_{\text{limit sequence}} \xrightarrow{\mathbf{P}_{k=[1,5]}} \underbrace{\mathbf{A}_{k=m}}_{\text{limit cycle}} \xrightarrow{\mathbf{P}_{k=m}} \underbrace{\mathbf{A}_k \rightarrow \cdots}_{\text{cycling!}}$$

# Exp. 4: Pathologically Difficult Graphs

- Often considered among the “most difficult” graphs
- Construction used to show Hadamard equivalence [McK81]
- For Hadamard matrix of order  $h$ , we have  $n = 4 \cdot h = |V|$
- 1,024 isomorphs of (2) Hadamard-based graphs



Hadamard  
 $h = 4, n = 16$

Graph Family	$n =  V $	limit sequence	limit cycle	$ \mathbf{A}_\Omega $
Hadamard, $h = 4$	16	$\lceil \log_2(n+1) \rceil + 1$	$\{1, 2\}$	$\gg n$
Hadamard, $h = 5$	20			

Too large to fix by random permutation!

$$\mathbf{A}_\Omega = \left\{ \mathbf{A}_{\omega_{[1, ???]}} \right\} = \underbrace{\mathbf{A}_1 \xrightarrow{\mathbf{P}_1} \cdots \xrightarrow{\mathbf{P}_{k-1}} \mathbf{A}_k}_{\text{limit sequence}} \xrightarrow{\mathbf{P}_{k \leq \sim \log_2(n)}} \underbrace{\mathbf{A}_{k=m}}_{\text{limit cycle}} \xrightarrow{\mathbf{P}_{k=m}} \underbrace{\mathbf{A}_k \rightarrow \cdots}_{\text{cycling!}}$$

# Permutation Chain Performance II

$$\mathbf{A}_\omega = \underbrace{\mathbf{A}_1 \xrightarrow{\mathbf{P}_1} \cdots \xrightarrow{\mathbf{P}_{k-1}} \mathbf{A}_k}_{\text{limit sequence}} \xrightarrow{\mathbf{P}_k} \underbrace{\mathbf{A}_{k+1} \xrightarrow{\mathbf{P}_{k+1}} \cdots \xrightarrow{\mathbf{P}_{m-1}} \mathbf{A}_m}_{\text{limit cycle}} \xrightarrow{\mathbf{P}_k = \mathbf{P}_m} \underbrace{\mathbf{A}_{k+1} \xrightarrow{\mathbf{P}_{k+1}} \cdots}_{\text{cycling!}}$$

Does IsoCanon yield

1. *deterministic* permutations in polynomial time?

yes, determined, in *all* cases,  $\text{find}(\mathbf{P}_i) \in O(n^3 \cdot \log(n))$ ,  $n = |V|$

2. *short* limit sequences?

yes, observed, in *all* cases,  $|\text{limit sequence}| \in k \leq \lceil \log_2(n+1) \rceil + 1$

3. *short* limit cycles?

yes, observed, in *all* cases,  $|\text{limit cycle}| \in (m-k) = \{1, 2\}$

4. *small* sets of limit cycles?

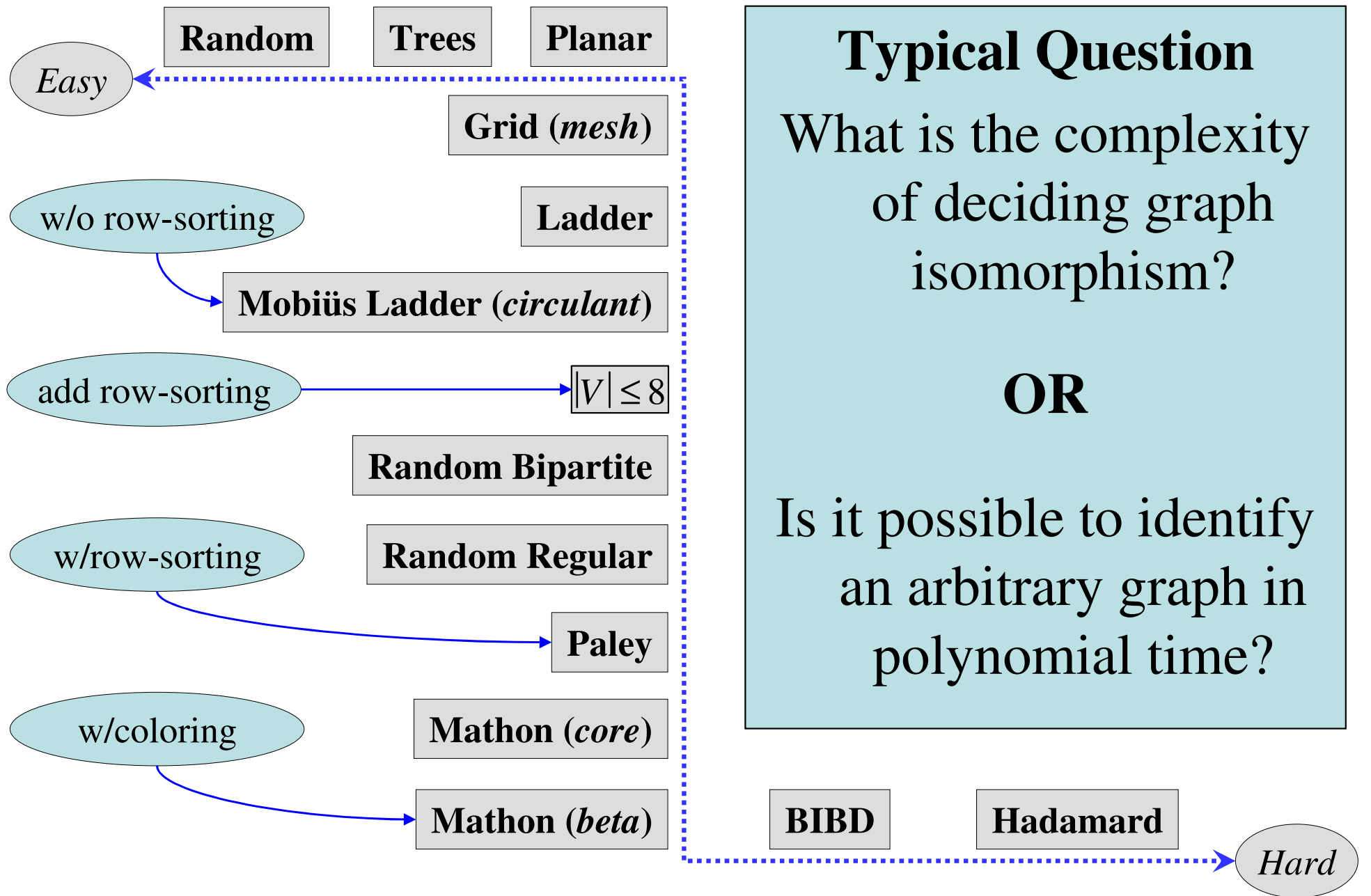
yes, in *average* case,  $|\mathbf{A}_\Omega| = 1$ ,  $\mathbf{A}_{\omega_1} = \mathbf{A}_{\omega_2} = \cdots = \mathbf{A}_{\omega_r}$ ,  $r = |\text{Iso}(G)|$

# Timing with Parallel BLAS

	# CPUs			
<i>n</i>	1	2	4	8
1	0.001	0.001	0.001	0.001
2	0.001	0.001	0.001	0.001
4	0.001	0.001	0.001	0.001
8	0.001	0.001	0.001	0.001
16	0.001	0.002	0.002	0.003
32	0.003	0.003	0.003	0.004
64	0.006	0.006	0.007	0.008
128	0.021	0.020	0.020	0.023
256	0.112	0.100	0.099	0.102
512	0.564	0.461	0.456	0.430
1024	3.328	2.572	2.497	2.208
2048	23.146	16.013	14.501	12.082
	<b>Duration (secs)</b>			

<i>ISOCANON</i>
<b>MATLAB</b>
<b>LAPACK</b>
<b>BLAS</b> (MKL, ACML)
<b>OS</b> (Windows)
<b>HW</b> (CPU)

# Graph Difficulty



# A New Paradigm

What is the complexity of causing an arbitrary graph to be *easy* to identify?

**OR**

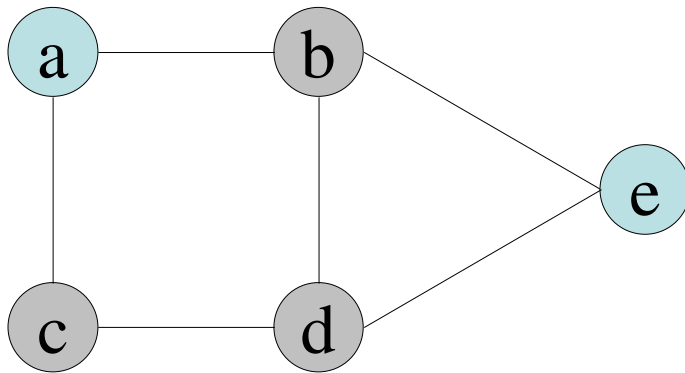
How much *noise* must be added such that an arbitrary graph is *easy* to identify?

**OR**

How many vertices must be colored such that a graph can be identified in polynomial time?

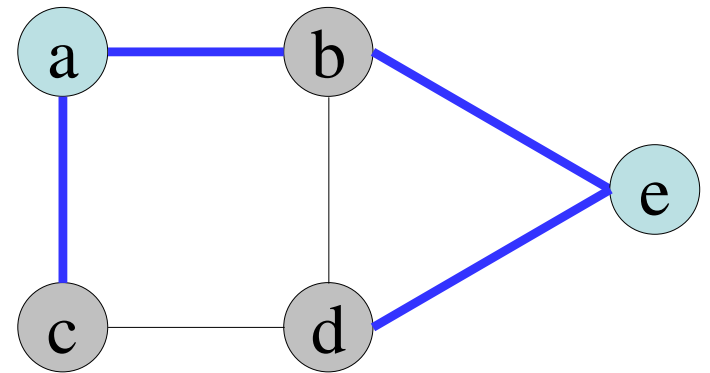
# Random Coloring Methods (*with constant*)

color  $k$  vertices



	a	b	c	d	e
a	2	1	1	0	0
b	1	0	0	1	1
c	1	0	0	1	0
d	0	1	1	0	1
e	0	1	0	1	2

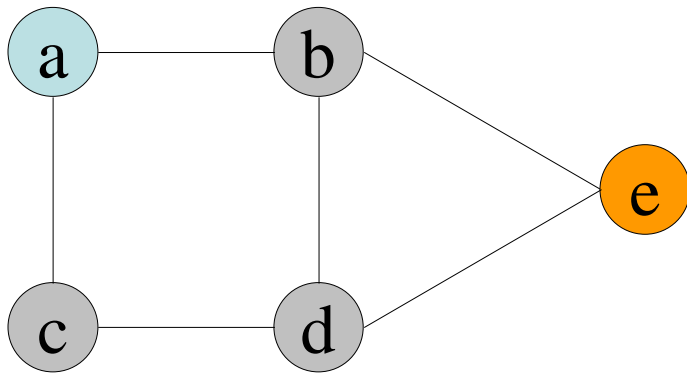
color incident edges  
of  $k$  vertices



	a	b	c	d	e
a	0	2	2	0	0
b	2	0	0	1	2
c	2	0	0	1	0
d	0	1	1	0	2
e	0	2	0	2	0

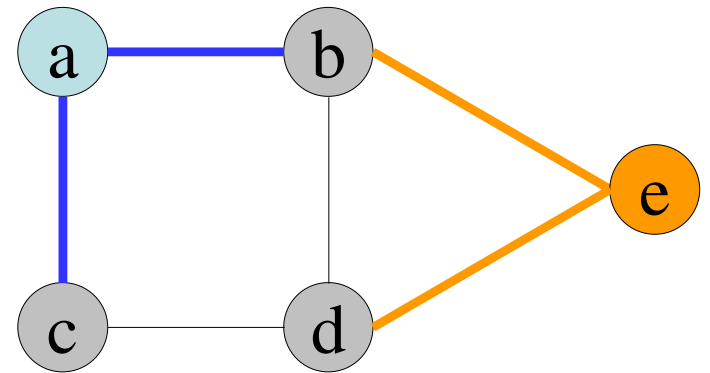
# Random Coloring Methods (*with primes*)

color  $k$  vertices



	a	b	c	d	e
a	2	1	1	0	0
b	1	0	0	1	1
c	1	0	0	1	0
d	0	1	1	0	1
e	0	1	0	1	3

color incident edges  
of  $k$  vertices



	a	b	c	d	e
a	0	2	2	0	0
b	2	0	0	1	3
c	2	0	0	1	0
d	0	1	1	0	3
e	0	3	0	3	0

# Random Coloring Results

1. Randomly permute, i.e., “shuffle”, graph  $\log_2(n)$  times
2. Color  $k$  random vertices by some method  $X$

Result	$k$	Method
<b>Unique Rows</b>	$\lceil \log_2(n) \rceil \cdot \lceil \sqrt{n} \rceil$	$V$ (primes)
<b>IsoCanon</b>	$2 \cdot \lceil \log_2(n) \rceil$	$V$ (constant)
<b>IsoCanon</b>	$2 \cdot \lceil \ln(n) \rceil$	$V$ (primes)
<b>IsoCanon</b>	$\lceil \ln(n) \rceil$	$E$ (primes)
<b>IsoCanon</b>	$\lceil \ln(n) \rceil$	$E$ (constant)

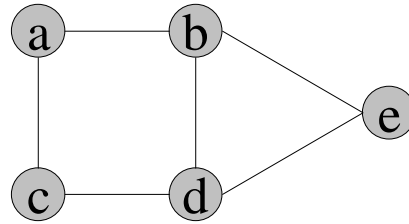
# DISCLAIMER

What is about to be shown is only *one* method of applying the effect we observed by randomly coloring a logarithmic set of vertices.

However, it is the most effective method we have discovered thus far with respect to efficiency, effectiveness, and numerical stability.

# Deterministic Coloring I (naïve version)

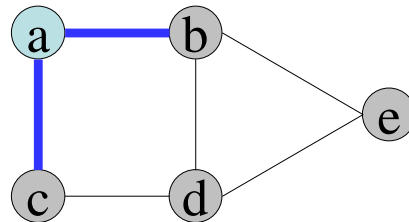
	$\beta$	a	b	c	d	e
$\beta$	1	2	2	2	2	2
a	2	1	2	2	1	1
b	2	2	1	1	2	2
c	2	2	1	1	2	1
d	2	1	2	2	1	2
e	2	1	2	1	2	1



**Matrix Shift:  $A = A + 1$**   
 “spreads” coloring info faster  
**Choose Vertex:** from the smallest equivalent row set

	Equal Rows?						
a	1	-0.1080	0.0012	0.0035	0.0118	0.0150	0.0170
b	1	-0.0993	0.0012	0.0095	0.0121	0.0138	0.0150
c	1	-0.1080	0.0012	0.0035	0.0118	0.0150	0.0170
d	1	-0.0993	0.0012	0.0095	0.0121	0.0138	0.0150
e	1	-0.1074	0.0035	0.0035	0.0121	0.0138	0.0138

	$\beta$	a	B	c	d	e
$\beta$	2.4	2.8	2	2	2	2
a	2.8	1.4	2.8	2.8	1.4	1.4
b	2	2.8	2.1	1	2	2
c	2	2.8	1	1	2	1
d	2	1.4	2	2	2.1	2
e	2	1.4	2	1	2	1.7

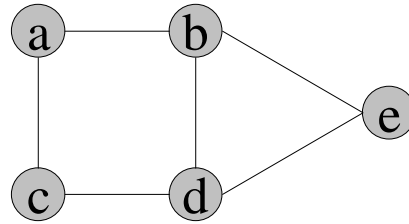


**Compute Inverse:  $A'$**   
 assess effect of coloring  
**Done Coloring?:** if all rows unique  $\parallel \ln(n)$  vertices colored

	Equal Rows?						
a	1	-0.0783	0.0019	0.0039	0.0090	0.0115	0.0169
b	3	-0.0705	0.0007	0.0045	0.0066	0.0081	0.0115
c	1	-0.0994	0.0007	0.0031	0.0072	0.0103	0.0169
d	3	-0.0719	0.0019	0.0051	0.0066	0.0085	0.0103
e	2	-0.0838	0.0031	0.0039	0.0069	0.0081	0.0085

# Deterministic Coloring II (not so naïve)

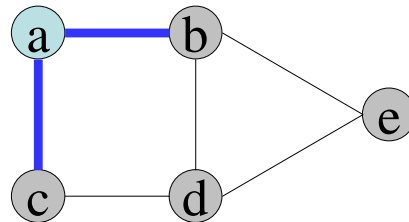
	$\beta$	a	b	c	d	e
$\beta$	1	2	2	2	2	2
a	2	1	2	2	1	1
b	2	2	1	1	2	2
c	2	2	1	1	2	1
d	2	1	2	2	1	2
e	2	1	2	1	2	1



In this example, {a, c} are in the same orbit & there are only 2 available vertices. If more than 2 vertices, we...

		Equal Rows?					
a	1	-0.1080	0.0012	0.0035	0.0118	0.0150	0.0170
b	1	-0.0993	0.0012	0.0095	0.0121	0.0138	0.0150
c	1	-0.1080	0.0012	0.0035	0.0118	0.0150	0.0170
d	1	-0.0993	0.0012	0.0095	0.0121	0.0138	0.0150
e	1	-0.1074	0.0035	0.0035	0.0121	0.0138	0.0138

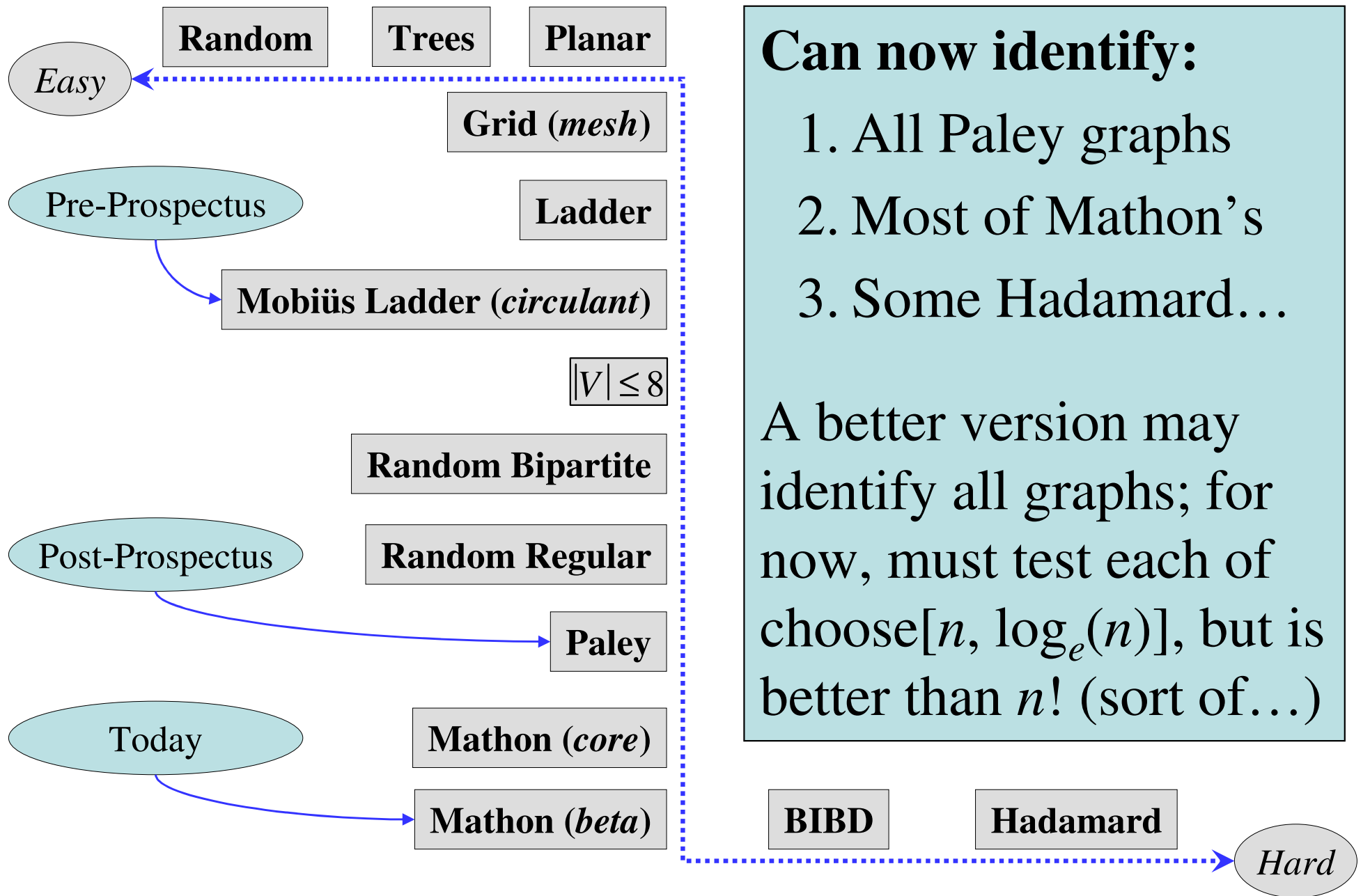
	$\beta$	a	B	c	d	e
$\beta$	2.4	2.8	2	2	2	2
a	2.8	1.4	2.8	2.8	1.4	1.4
b	2	2.8	2.1	1	2	2
c	2	2.8	1	1	2	1
d	2	1.4	2	2	2.1	2
e	2	1.4	2	1	2	1.7



color each vertex separately in the equivalent set and select one yielding the smallest new inverse—this can be extended.

		Equal Rows?					
a	1	-0.0783	0.0019	0.0039	0.0090	0.0115	0.0169
b	3	-0.0705	0.0007	0.0045	0.0066	0.0081	0.0115
c	1	-0.0994	0.0007	0.0031	0.0072	0.0103	0.0169
d	3	-0.0719	0.0019	0.0051	0.0066	0.0085	0.0103
e	2	-0.0838	0.0031	0.0039	0.0069	0.0081	0.0085

# Graph Difficulty



# Do We Solve Our Original Problem?

- What does IsoCanon do?

*Simultaneously* determines isomorphism *and* ranks vertices

- So why not just use *nauty* and then PageRank?

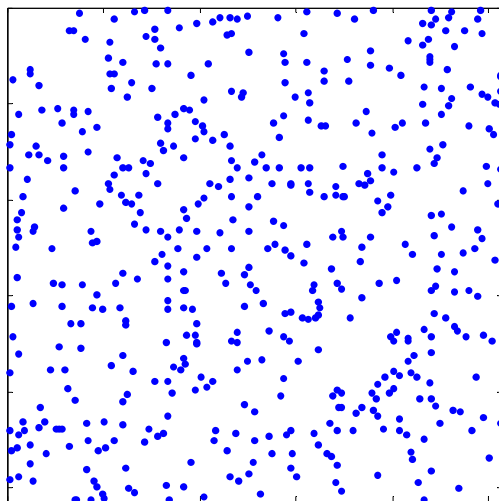
Doesn't *simultaneously* solve our problem.

- So what “ranking” do we obtain?

(4) options: degree, closeness, betweenness, eigenvector

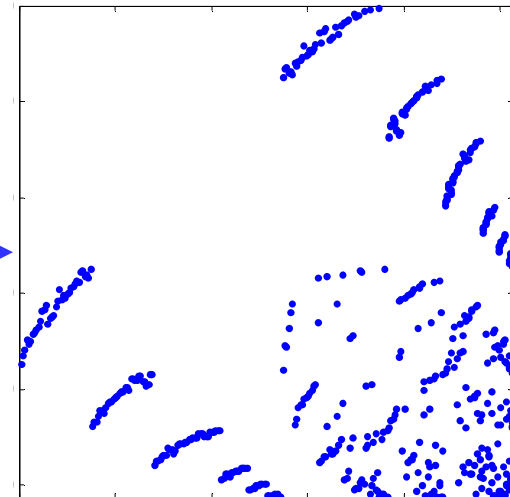
- What about “criticality”?

(2) options: max components vs. min bandwidth



UAV Swarm

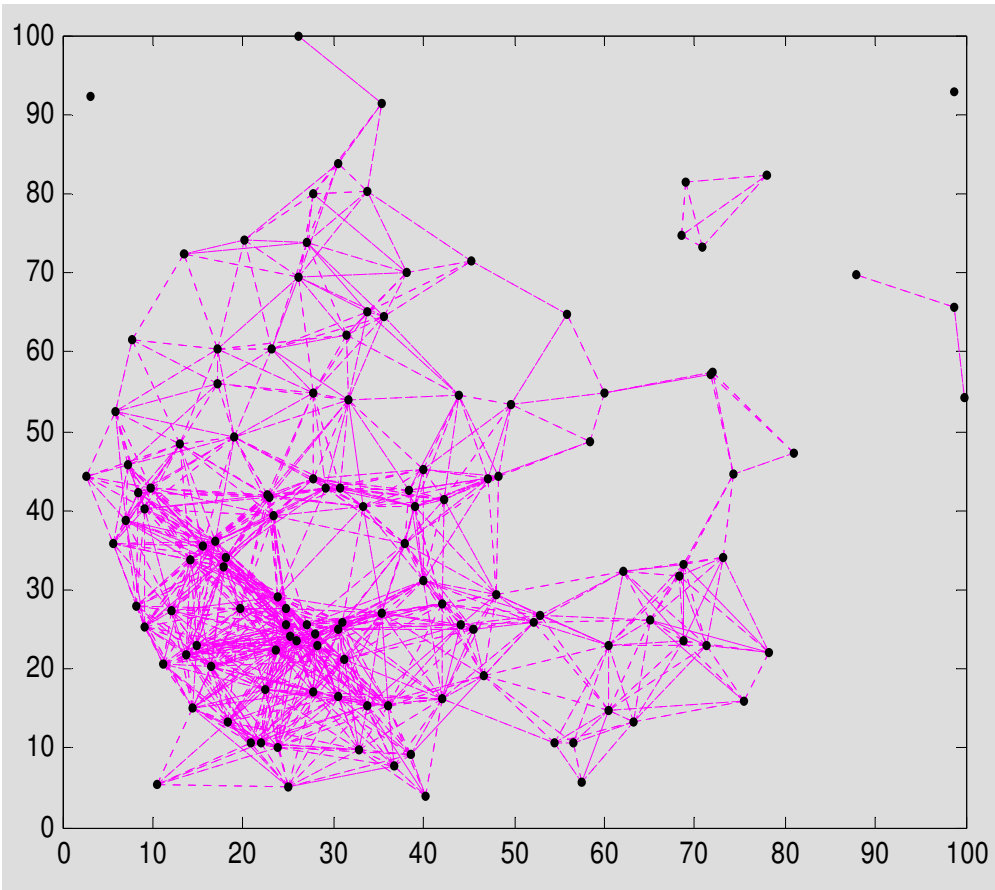
*Vertex  
Ordering*



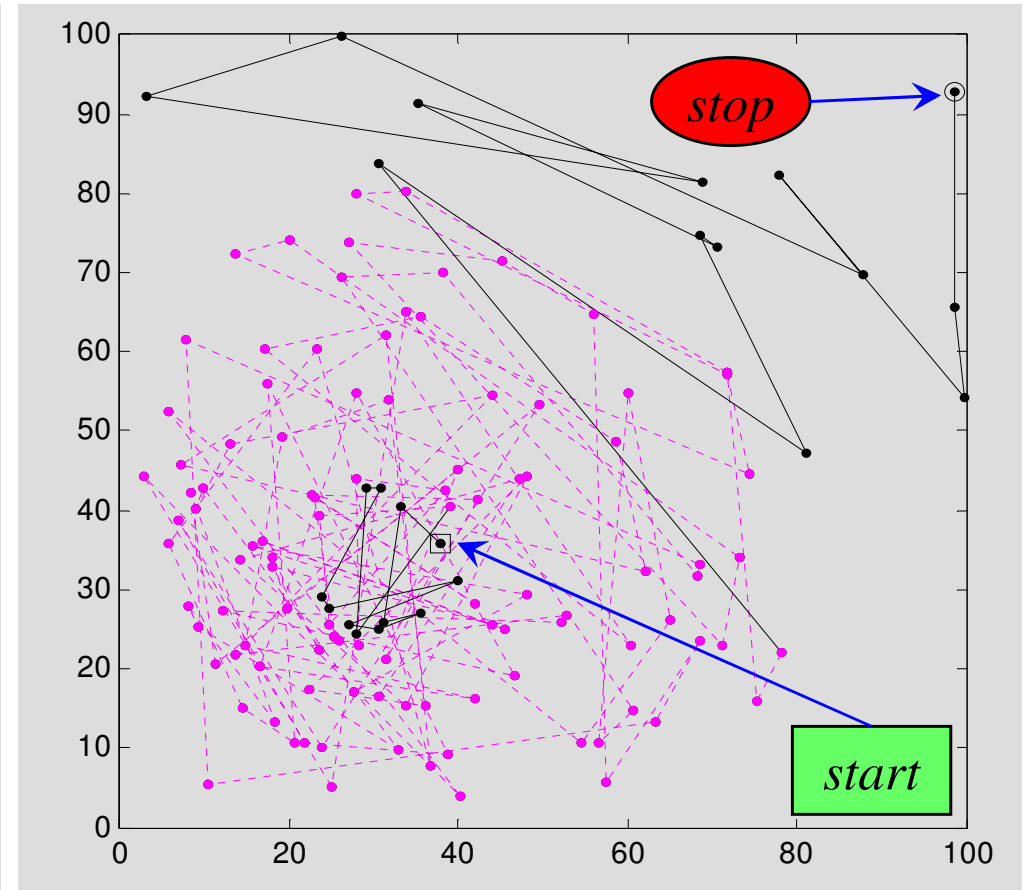
A Possible Linearization

# Connectivity (Critical Nodes)

$$|V| = 128$$



**Network Connectivity**



**Network Linearization**

# MIGHTY XLIV Plug

C. Augeri\*, B. Mullins, L. Baird, D. Bulutoglu, & R. Baldwin

Logarithmic Coloring: How Hard is it to Determine Isomorphism?

*to be presented at the*

11–12 May, MIGHTY XLIV Conference @ WSU

<http://homepages.udayton.edu/%7Esrithara/mighty44>

*organized by*

K. Arasu (WSU), D. Slilaty (WSU), & R. Sritharan (UD)

# What's Next?

- Extend logarithmic coloring method
- Study behavior w.r.t. numerical conditioning
- Assess usefulness of linearization
  - Data mining
  - Critical nodes
  - User queries
- Prepare optimized version

# References

- [BeE96] J. M. Bennett and J. J. Edwards. A graph isomorphism algorithm using pseudoinverses. *BIT Numerical Mathematics*. Springer-Verlag, 36(1):41–53, 1996.
- [BeW04] Lowell W. Beineke and Robin J. Wilson, eds., with Peter J. Cameron. *Topics in Algebraic Graph Theory*. Cambridge University Press, 2004.
- [HaS04] W. H. Haemers and E. Spence. Enumeration of cospectral graphs. *European Journal of Combinatorics*, 25(2):199–211, 2004.
- [HZL05] P. R. He, W. J. Zhang, and Q. Li. Some further development on the eigensystem approach for graph isomorphism detection. *J. of the Franklin Inst.*, Elsevier, 342(6), 2005.
- [McK81] B. D. McKay. Practical Graph isomorphism. In *Proc. of the 10th Manitoba Conf. on Num. Mathematics and Comp.*, 45–87. *Congressus Numerantium*, vol. 30, 1981.
- [PBM+99] L. Page, S. Brin, R. Motwani and T. Winograd. *The PageRank Citation Ranking: Bringing Order to the Web*. TR 1999-66, Stanford University, 1999.
- [PrD85] G. M. Prabhu and N. Deo. On the power of a perturbation for testing non-isomorphism of graphs. *BIT*, Springer, 24(3):302–307, 1984.
- [Var04] R. S. Varga. *Geršgorin and His Circles*. Springer, 2004.