电子科技大学
University of Electronic Science and Technology of China

A Brief Introduction To Graph Similarity

Reporter: Xinzuo Wang

Data Mining Lab, Big Data Research Center, UESTC
Email：junmshao@uestc.edu.cn
http://staff.uestc.edu.cn/shaojunming

Problem Definition:

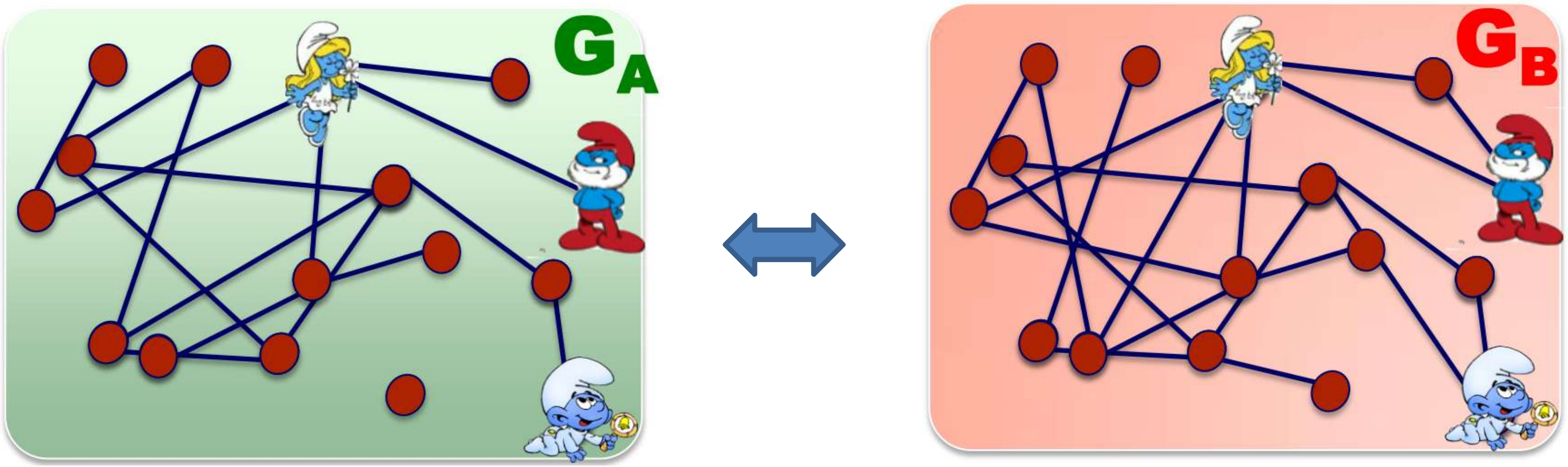
- Informally, given two different networks (graphs) how do we access their similarity?

- Actually, the problem can be divided into two categories:
  - Graph similarity with **known node correspondence**
  - Graph similarity with **unknown node correspondence**

Graph similarity with known node correspondence

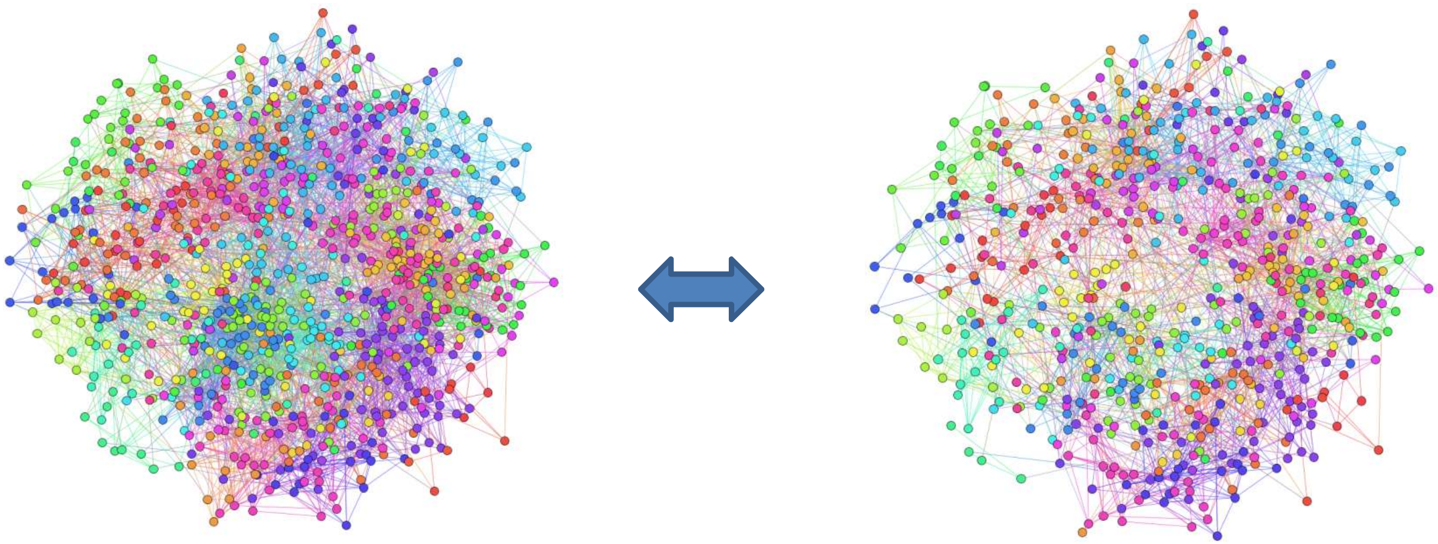
数据挖掘实验室
Data Mining Lab

- **Given**:
  I. 2 graphs with the **same nodes** and different edge sets
  II. node correspondence
- **Find**: similarity scores [0,1]



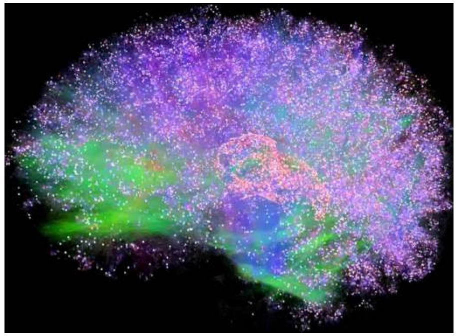
Graph similarity with unknown node correspondence:

- **Given**: 2 anonymized networks (**without node correspondence**)
- **Find**: structural similarity score or node mapping



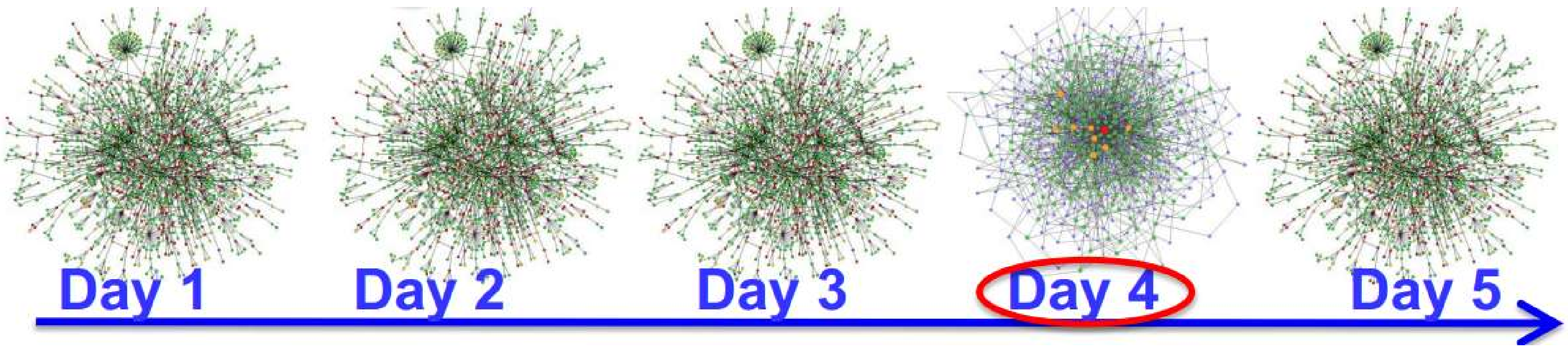
Application:

- Brain network analysis



Different network wiring between different people ?

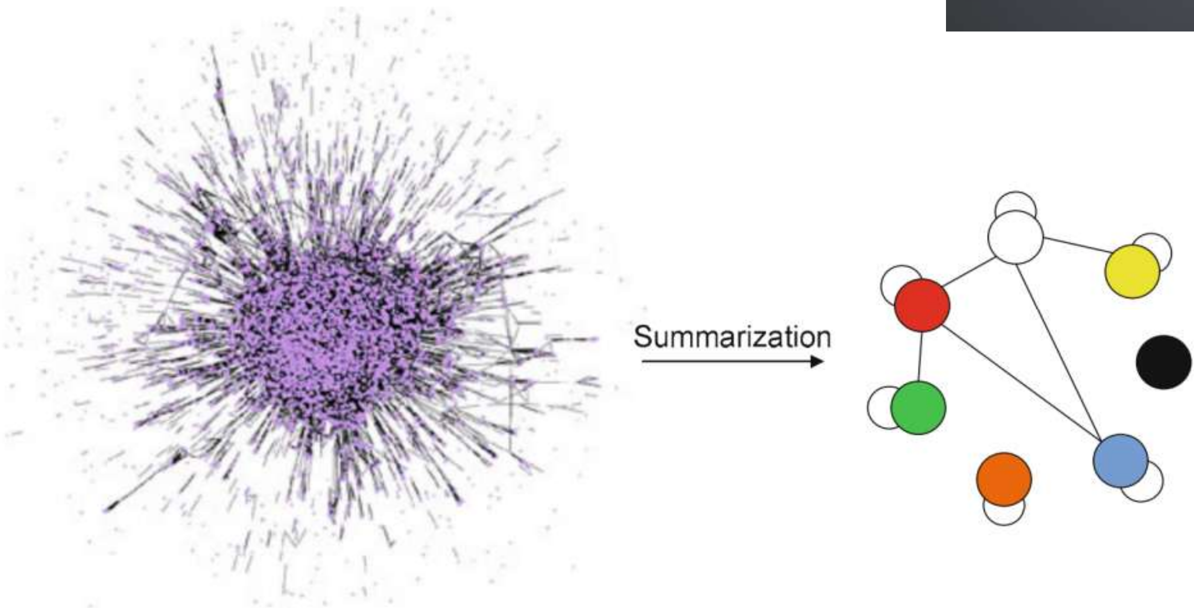
- Discontinuity detection



Application:

- Behavioral patterns analysis

- Large network compression



- **Known node correspondence**
  - **Simple features**
  - Complex features
- Unknown node correspondence
  - Avoiding node correspondence problem
  - Finding node correspondence

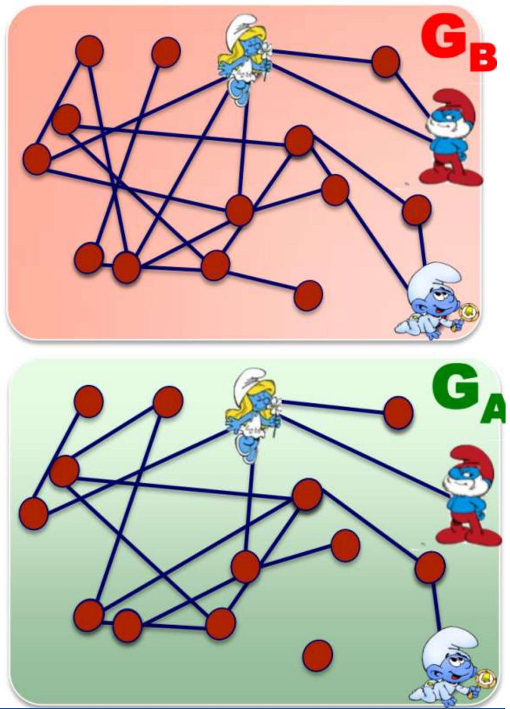
Simple Features:

- Simple, and sometimes naïve...

- Some examples:
  1. **Edge Overlap(EO)**: number of common edges.
  2. **EVO**: Number of common edges and vertices.

$$VEO = \frac{|E_A \cap E_B| + |V_A \cap V_B|}{|E_A| + |E_B| + |V_A| + |V_B|}$$

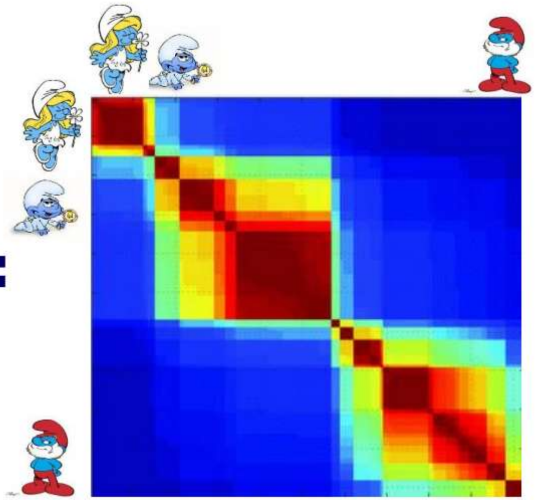
  3. **Graph Edit Distance**: number of node/edge additions/deletions to transform G_A to G_B.
  4. ……

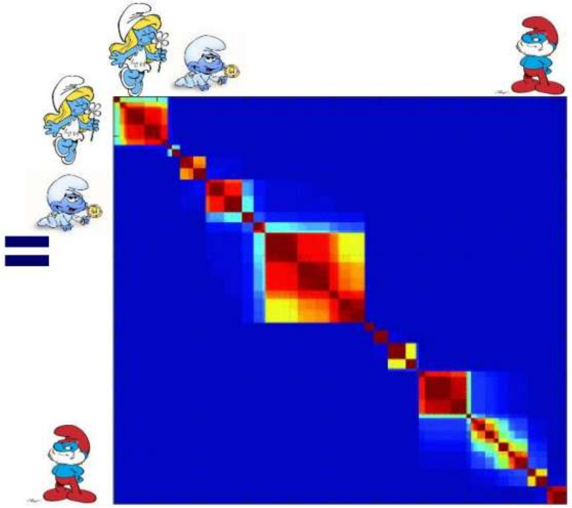


More complex features--- an example

1. Find the **pairwise node influence**, S_A& S_B
2. Find the **similarity** between S_A& S_B

$$sim(\mathbf{S_A},\mathbf{S_B}) = \frac{1}{1+\sqrt{Euclidean\ Dist.}} = \frac{1}{1+\sqrt{\sum_{i,j}\left(\sqrt{s_{A,ij}}-\sqrt{s_{B,ij}}\right)^2}}$$

$\mathbf{S_A} =$ 

$\mathbf{S_B} =$ 

[Koutra, Faloutsos, Vogelstein SDM'13]

- Known node correspondence
  - Simple features
  - Complex features
- **Unknown node correspondence**
  - **Avoiding node correspondence problem**
  - Finding node correspondence

Spectral methods: λ-distance

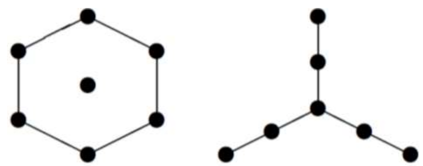
- $d(G_{A,} G_B) = \sqrt{\sum(\lambda_{Ai} - \lambda_{Bi})^2}$

- $\lambda_{Ai} = eigenvalues\ of:$
  - $Adjacency \qquad A$
  - $Laplacian \qquad L = D - A$
  - $Normalized\ Laplacian$

$$L_{norm} = D^{-1/2} A\, D^{-1/2}$$

λ-distance : disadvantages

- Co-spectral graphs with **different** structure



- Subtle changes in the graphs => big differences in spectra

- $O(n^3)$ runtime --- SVD

Other Methods :

数据挖掘实验室
Data Mining Lab

- Extracting features from graph
  --NETSIMILE
  [Berlingerio, Koutra, Eliassi-Rad, Faloutsos '13]

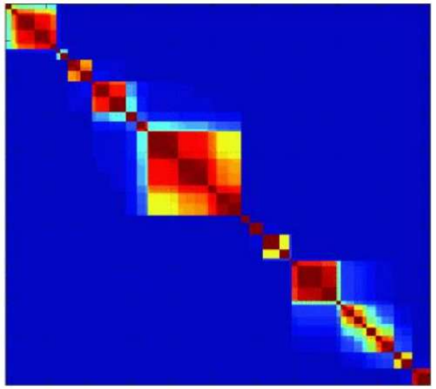
  For every node extract:

  a) degree,
  b) clustering coefficient,
  c) average degree of neighbors,
  d) average clustering coefficient of neighbors,
  e) number of edges in ego-network,
  f) number of outgoing edges of egonetwork,
  g) number of neighbors of ego-network.

Outline:

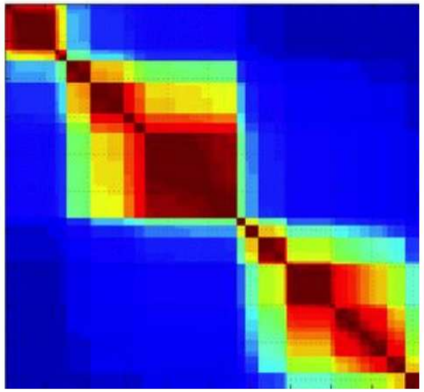
- Known node correspondence
  - Simple features
  - Complex features
- **Unknown node correspondence**
  - Avoiding node correspondence problem
  - **Finding node correspondence**

Eigen-Decomposition Approach:



A

G_A G_B



B

- Goal: $\min_P ||PAP^T - B||$

(P is an **permutation matrix,** replacing the index of each node)

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$$

[Umeyama ' 88]

- **Conclusion**: when G_A and G_B they are **isomorphic**, the **optimum** P can be obtained by **maximizing** $\boldsymbol{tr(P^T|U_A||U_B|)}$ (using Hungarian Method),

  where
  $$A = U_A \Lambda_A {U_A}^T$$
  $$B = U_B \Lambda_B {U_B}^T$$

- **Optimal** for **isomorphic** graphs, **nearly optimal** for noiseless (nearly isomorphic) graphs.
- $O(n^3)$ runtime.
- Only for graphs of the **same size**.

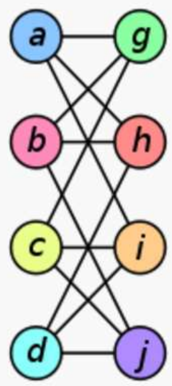
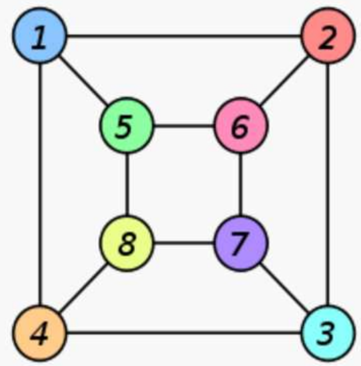
Some tips about graph isomorphism (图同构)

- **Isomorphism**: Intuitively, two objects are isomorphic *if they cannot be distinguished by using only the properties used to define morphism* (i.e. same) (except possibly in their representations).
- **Isomorphism of 2 linear vector spaces**: if their exists an invertible linear map between them.
- **Isomorphism of 2 graphs**: an isomorphism of graphs G and H is a bijection between the vertex sets of G and H.

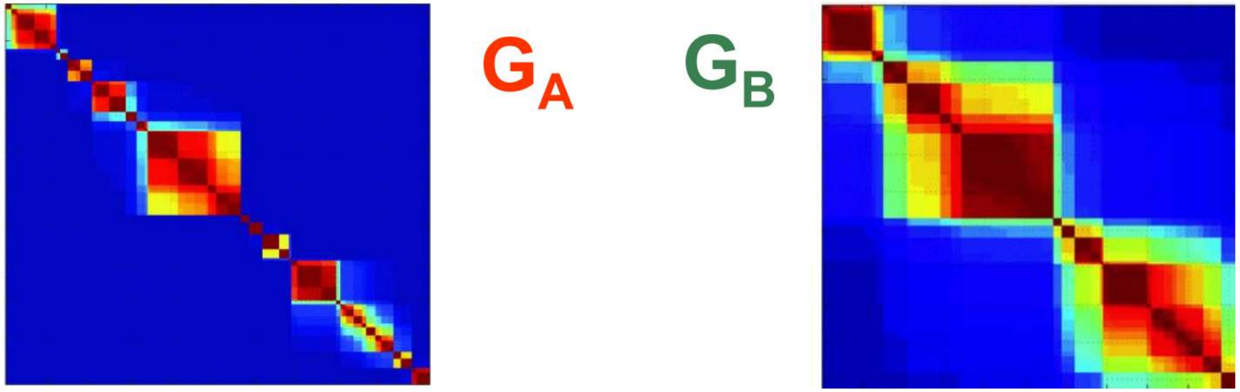
The **graph isomorphism problem** is the computational problem of determining whether two finite graphs are isomorphic.

László shows that Graph Isomorphism is in Quasipolynomial Time: that is time of the form $2^{O(\log(n))^c}$.

Polynomial time is the case when $c = 1$, but any c is a **huge improvement** over the previous best result.

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

NMF-based Approach:



A B

- Goal: $\min_P ||PAP^T - B||$

$$PP^T = I$$

$$P \geq 0$$

(P is a **permutation matrix**)

[Ding+ ‘ 08]

NMF-based Approach:

数据挖掘实验室
Data Mining Lab

- Step 1: $P_0 = |U_A|\left|{U_B}^T\right|$
- Step 2: Non-Negative Matrix Factorization to find P^∞

$$P_{ij} \leftarrow P_{ij}\sqrt{\frac{(APB)_{ij}}{(P\alpha)_{ij}}}$$

$$\alpha \equiv \frac{P^TAPB + (P^TAPB)^T}{2}$$

1. $O(n^3)$ runtime
2. Guaranteed convergence
3. Much better results than the **Eigen-Decomposition Approach**

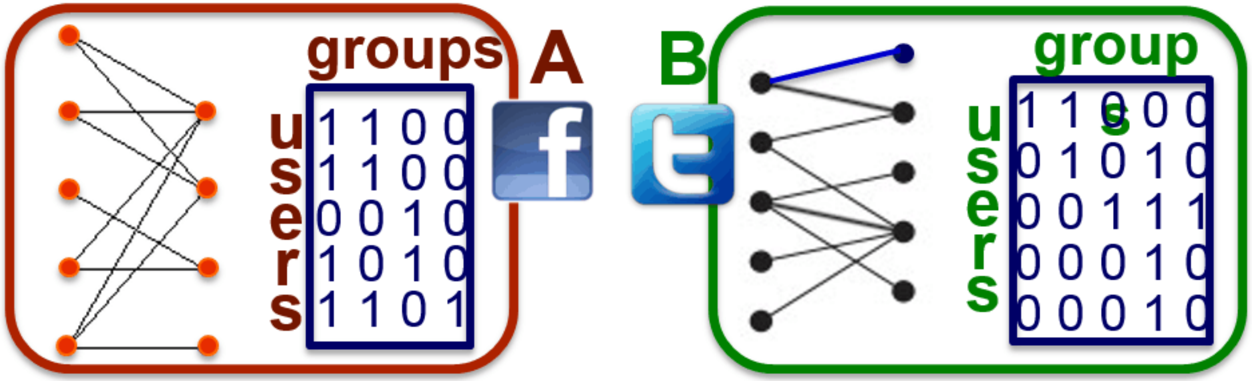
N	ϵ	P_0	NMF
10	0.1	0.72	0.97
10	0.2	0.19	0.69
10	0.3	0.04	0.36
10	0.4	0.00	0.19
20	0.2	0	0.74

Table 1. Success rate for different sizes (N) and noise levels (ϵ). P_0: using P_0 and the Hungarian algorithm. NMF: using NMF and the Hungarian algorithm.

Bipartite Graph Alignment :

- Input:

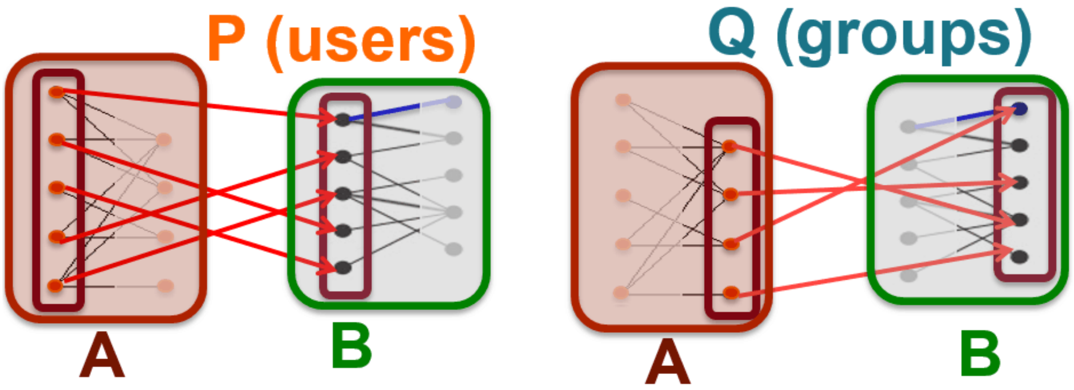
  Bipartite Graph
  A, B



- Output:

  Permutation matrix **P, Q**

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$$



$$\min \|PAQ - B\|$$

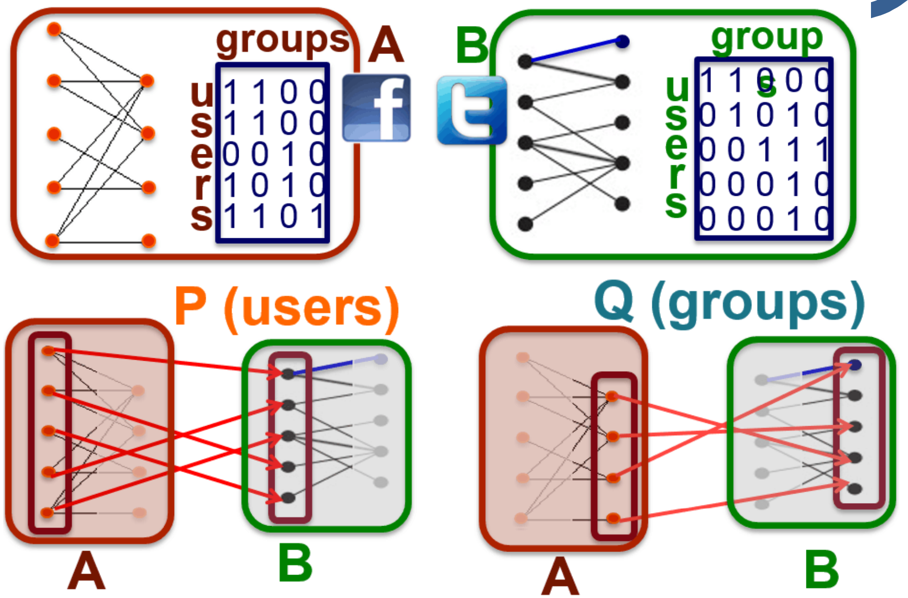
Bipartite Graph Alignment :

- Input:

  Bipartite Graph
  A, B

- Output:

  Permutation
  matrix **P, Q**



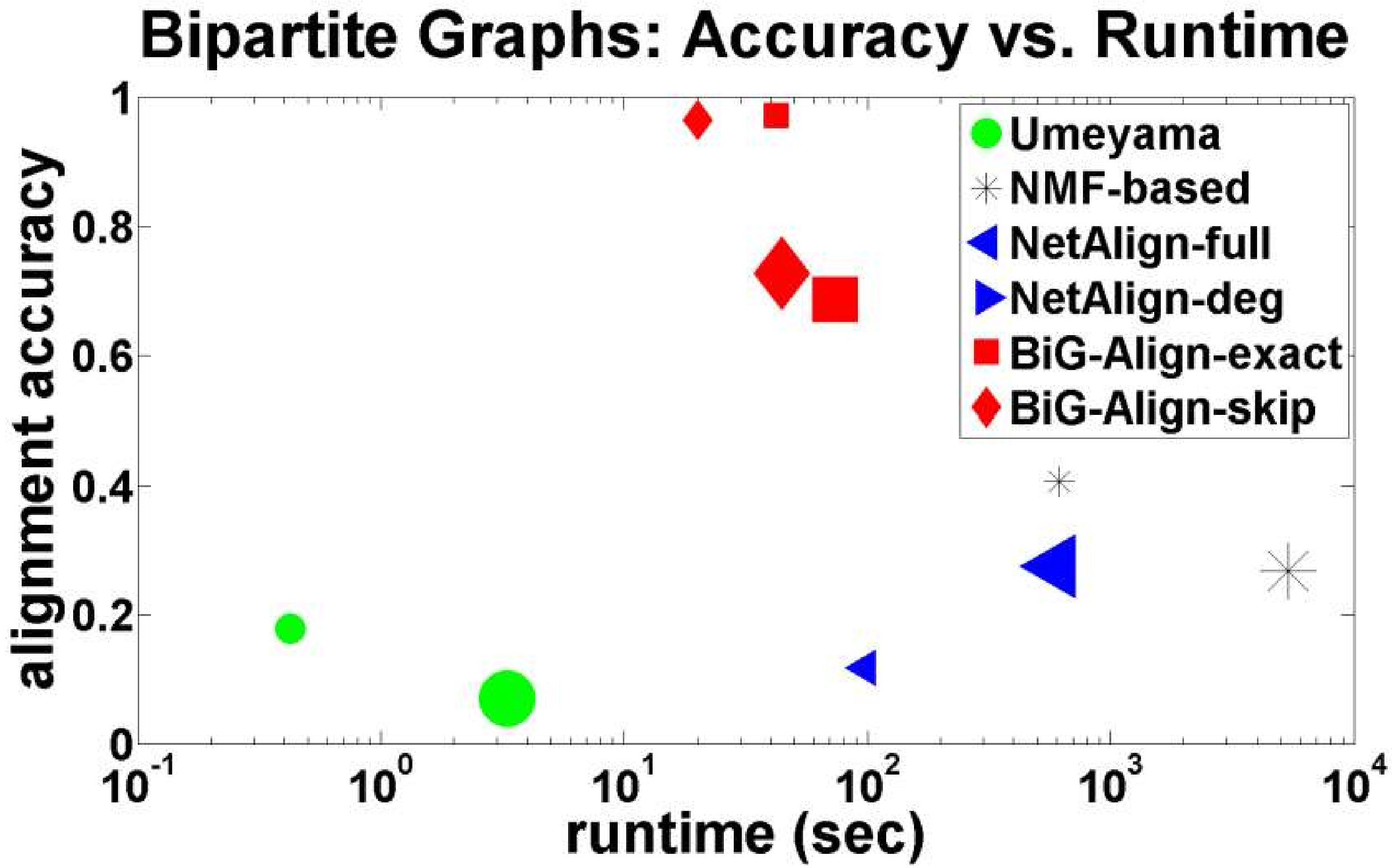
2 constrains

1. $P_{ij}, Q_{ij} \epsilon \; [0,1]$ (i.e. probabilities (similarities between nodes)
2. P, Q should be sparse (i.e. more efficient for large graphs)

$$\boxed{\min_{P,Q} ||PAQ - B|| + \lambda ||P||_1 + \mu ||Q||_1}$$

[Koutra, Tong, Lubensky ’13]

Bipartite Graph Alignment :



Further Reference:

数据挖掘实验室
Data Mining Lab

- http://db.cs.cmu.edu/projects/graph-similarity-with-attribution-and-alignment
- http://icdm2014.sfu.ca/program_tutorials.html

Thanks

F&Q ?

- Finished?
- Maybe not (if you like) 

- ***<u>WHY</u>*** Eigenvalues and Eigenvectors Are So Important ?

- Given a vector space V:

  $\forall v \in V, \exists\ a\ basis\ of\ V, i.e.\ v_1, v_2, \dots, v_m,$

  $$s.t. \qquad v = a_1 v_1 + a_2 v_2 + \cdots + a_m v_m,$$

  given a linear transformation $T \in \mathcal{L}(V, V)$

  $$\begin{aligned} Tv &= a_1 T v_1 + a_2 T v_2 + \cdots + a_m T v_m \\ &= a_1 \alpha_1 v_1 + a_2 \alpha_2 v_2 + \cdots + a_m \alpha_m v_m \\ &\in span(v_1, v_2, \dots, v_m). \end{aligned}$$

- WHY Eigenvalues and Eigenvectors Are So Important ?

- They give the **simplest description** of an linear transformation in a specific space.

- How do we do with those imperfect linear transformation?

- The Jordan Normal Form!

- If V is a complex vector space, if $T \in \mathcal{L}(V, V)$, then V have a Jordan basis of T.

$$\begin{bmatrix} A_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_m \end{bmatrix}, A_m = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_k \end{bmatrix}$$