CentraleSupélec

ESSEC BUSINESS SCHOOL

Network Science Analytics

Option Applied Math and M.Sc. in DSBA

Lecture 5B

Graph similarity

Fragkiskos Malliaros

Friday, March 2, 2018

Acknowledgements

- The lecture is partially based on material by
  - Jure Leskovec, Stanford University
  - Aaron Clauset, CU Boulder
  - Manos Papaggelis, York University
  - Gonzalo Mateos, University of Rochester
  - Albert-László Barabási, Northeastern University
  - Christos Faloutsos, CMU
  - Panagiotis Tsaparas, University of Ioannina
  - Danai Koutra, University of Michigan
  - Michalis Vazirgiannis, Ecole Polytechnique
  - Karsten M. Borgwardt, University of Munich
  - R. Zafarani, M. A. Abbasi, and H. Liu, Social Media Mining: An Introduction, Cambridge University Press, 2014. Free book and slides at http://socialmediamining.info/

Thank you!

In This Lecture

- Graph similarity
  - Problem definition and applications
  - Known node correspondence
  - Unknown node correspondence

Graph similarity

Graph Similarity

Given two graphs G_1 and G_2 from the space of graphs $\mathcal{G}$ the problem of graph similarity is to find a mapping

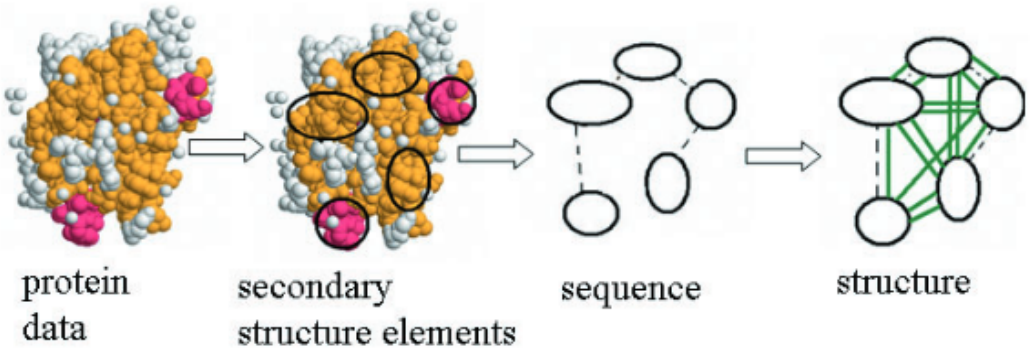
$$s : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$$

such that $s(G_1, G_2)$ quantifies the similarity of G_1 and G_2

- Very important problem with numerous applications
  - Recall that data is often represented as graphs
  - Most machine learning algorithms make decisions based on the similarity or distance between pairs of data points (i.e., instances)

Applications – Protein Function Prediction

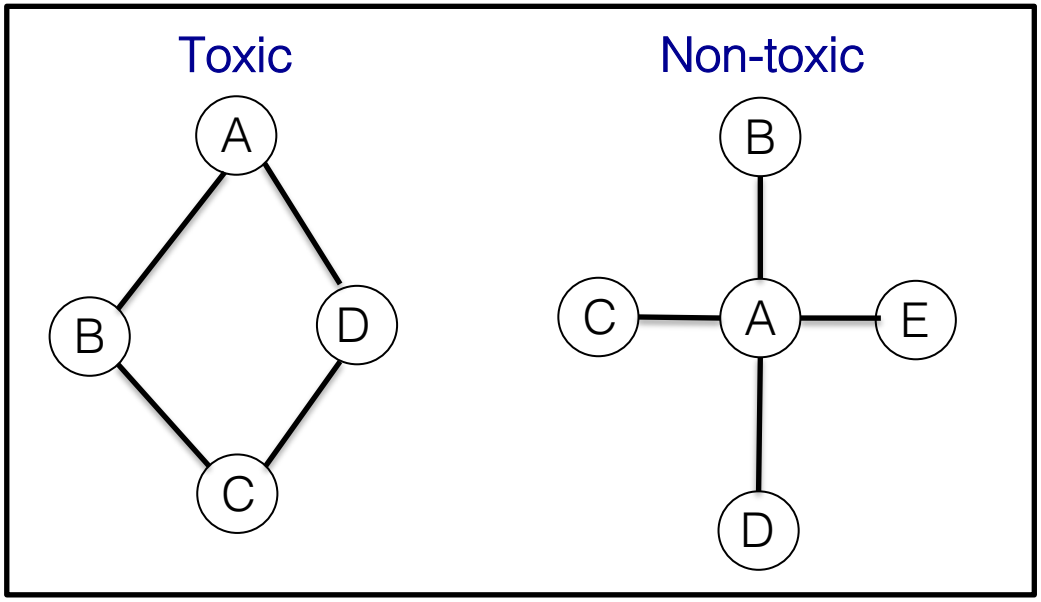
- For each protein, create a graph that contains information about its
  - Structure
  - Sequence
  - Chemical properties



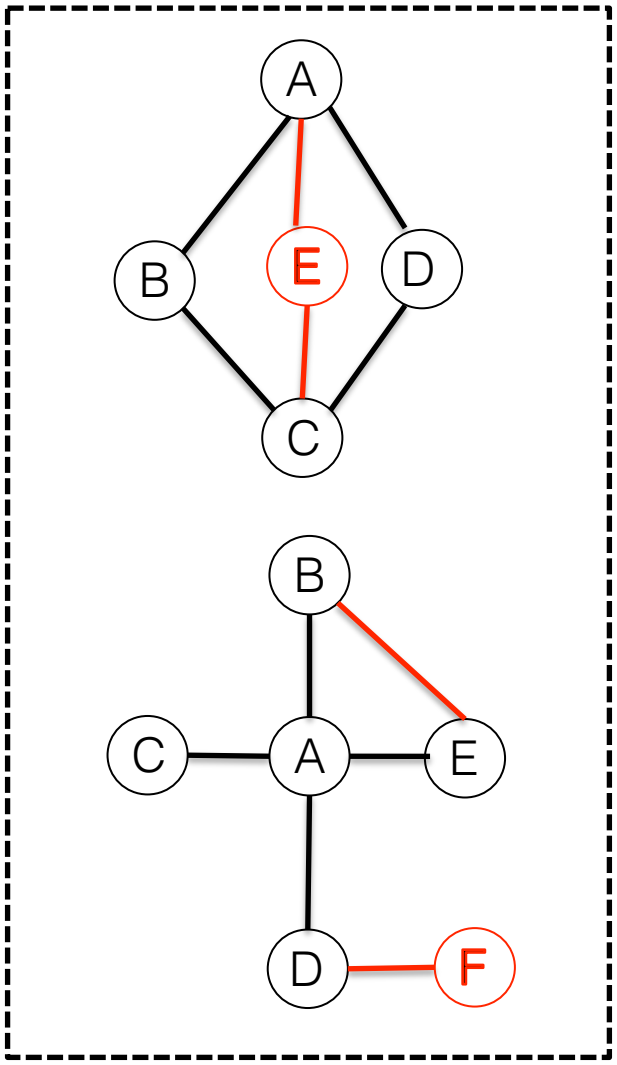
- Use graph similarity methods
  - To measure the structural similarity between proteins
  - To predict the function of proteins

Applications – Classification of Molecules

Dataset of known molecules



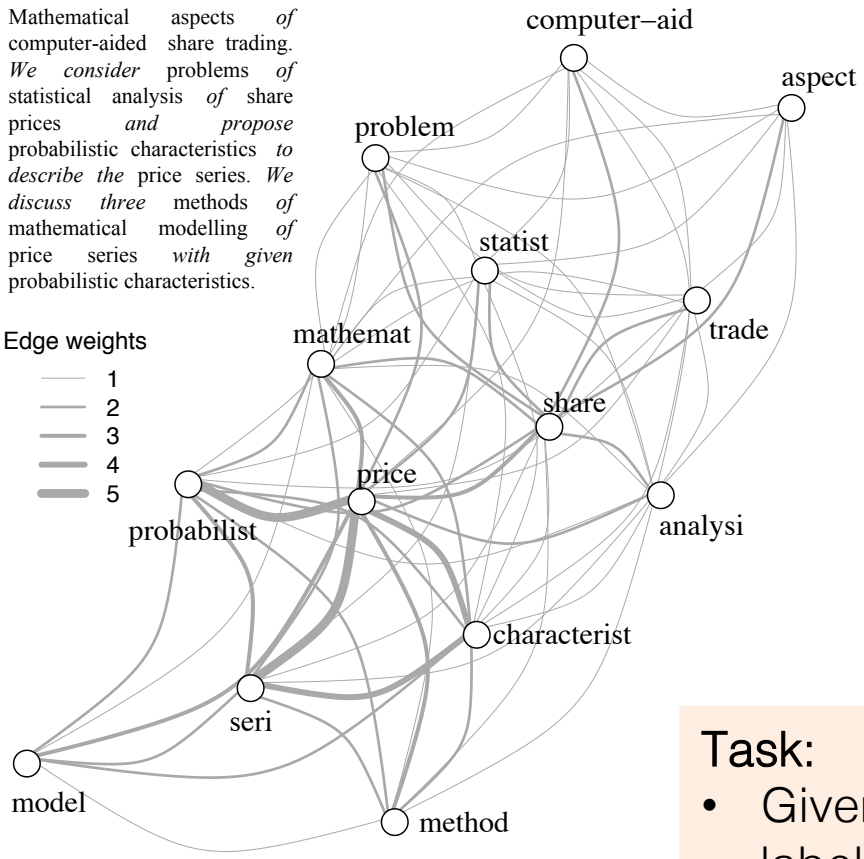
Unknown molecules



Task:

- Given a set of molecules that are either toxic or non-toxic
- Predict the class of unknown molecules

Applications – Text Categorization



Represent a document as a graph

- Nodes: terms of the document
- Edges: co-occurrence within a fixed-size sliding window

Task:

- Given a set of documents and their class labels (e.g., positive/negative reviews)
- Correctly classify documents

Applications – Malware Detection

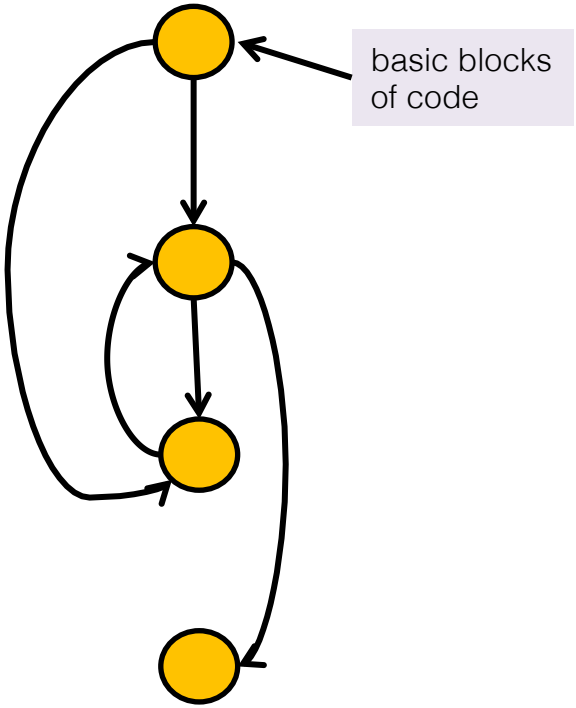
Computer program

```
# a scheduler class, to schedule and run events after a delay
class Scheduler:
    def __init__(self):
        # begin with no events
        self.events = []

    # after the delay, run the function
    def schedule(self, delay, function):
        if delay <= 0:
            # if no delay, run function straight away
            function()
        else:
            # queue the function
            self.events.append([delay, function])
            # sort the queue, smallest delay first
            self.events.sort(key = lambda event: event[0])

    def run(self):
        # loop while there are events to process
        while len(self.events) > 0:
            # get the first event
            delay, function = self.events.pop(0)
            # subtract time from remaining events
            for event in self.events:
                event[0] -= delay
            # wait for required delay
            pyb.delay(delay)
            # run function
            function()
```

Control flow graph



- Compare the control flow graph of the program against a set of control flow graphs of known malware
- If it contains a similar subgraph to these graphs, then there is probably malicious code running in the program

More Applications

- Bioinformatics
- Chemo-informatics
- Comparison of electric circuits
- Anomaly detection
- Relational entity disambiguation
- XML document similarity
- Entity classification
- Object tracking
- Process mining in the business domain
- …

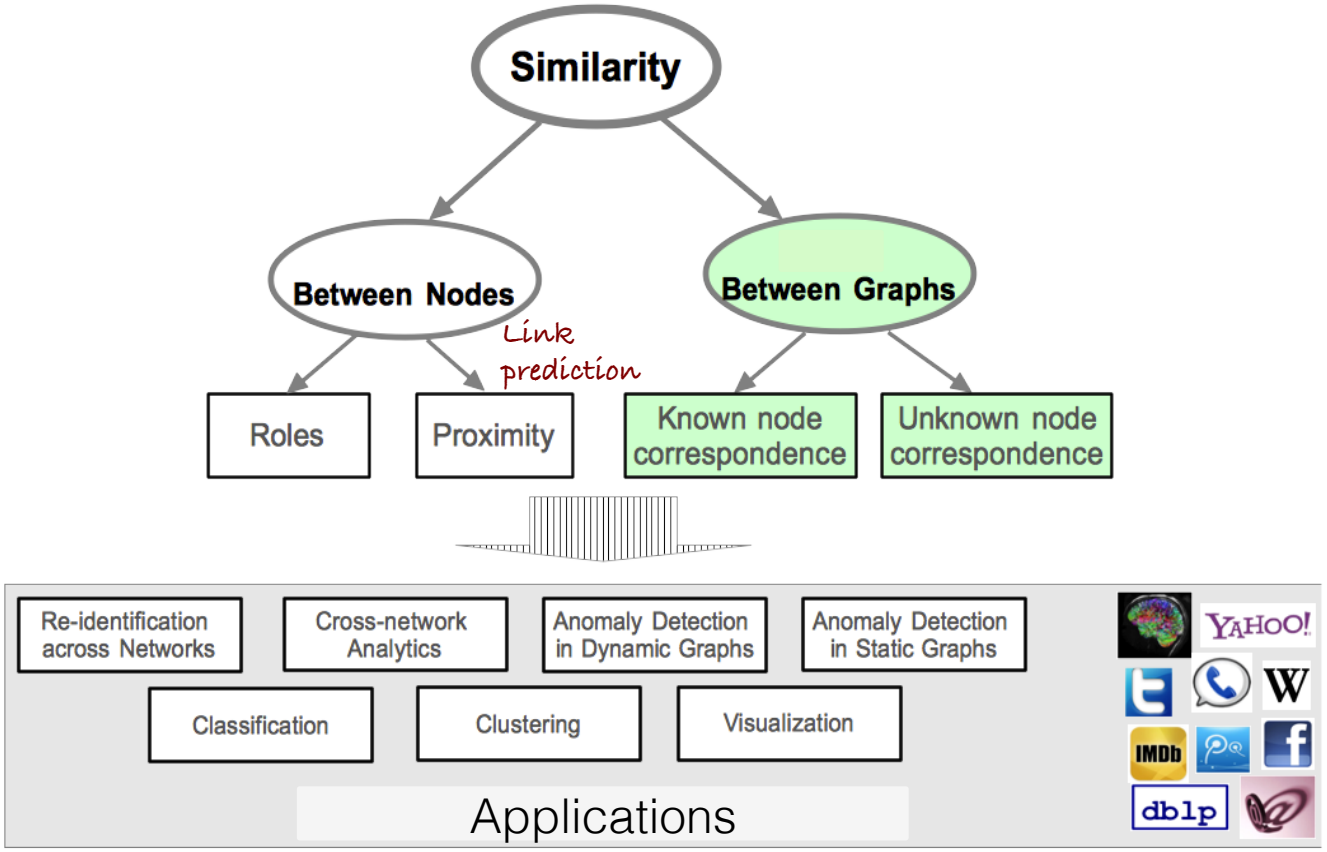
It's Not a Trivial Task

- Although graph similarity (GS) seems to be a tractable problem, in fact it is a complex task

- Many problems related to GS are computationally difficult
  - Subgraph isomorphism [will be defined later]
  - Finding the largest common subgraph

- We are mainly interested in GS algorithms that run in polynomial time
  - However, in large scale networks even poly-time algorithms are not applicable (e.g., algorithms with running time $O(n^4)$)

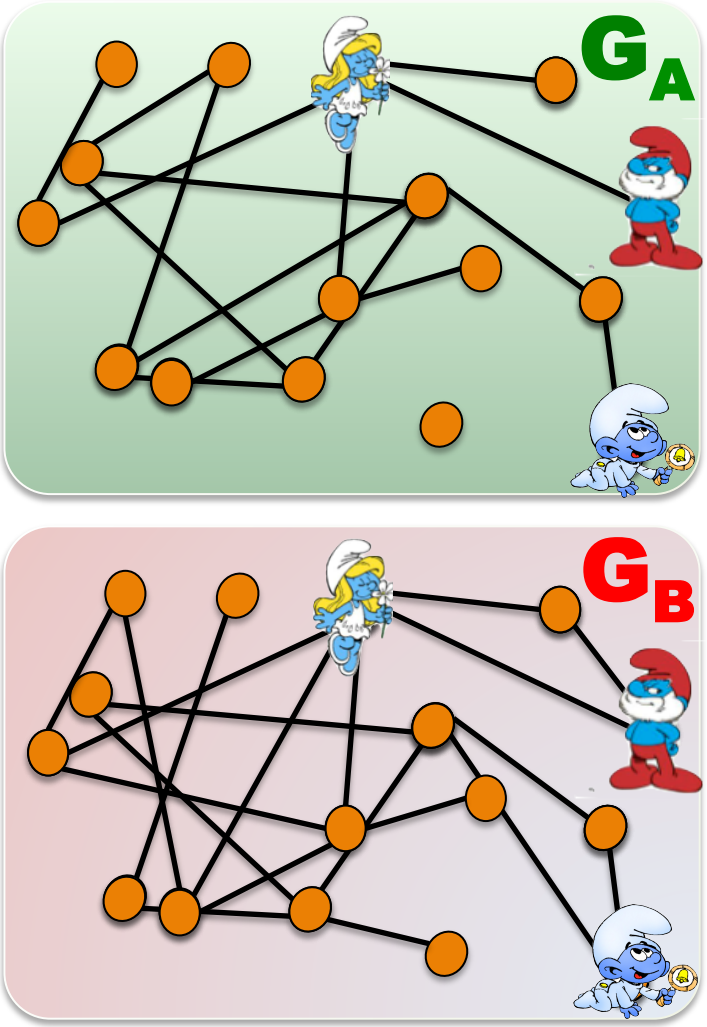
Similarity in Graphs



[Koutra et al. '14]

Graph similarity:
known node correspondence

Problem Definition: Graph Similarity



Given:

(i) two graphs with the

same nodes and

different edge sets

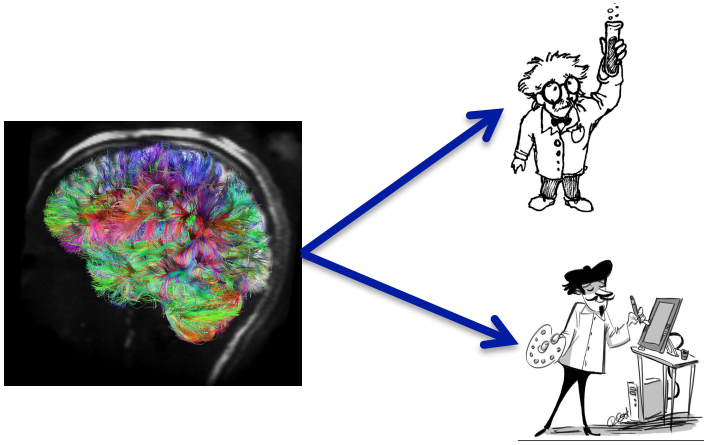
(ii) node correspondence

Find: similarity score

$$s \in [0, 1]$$

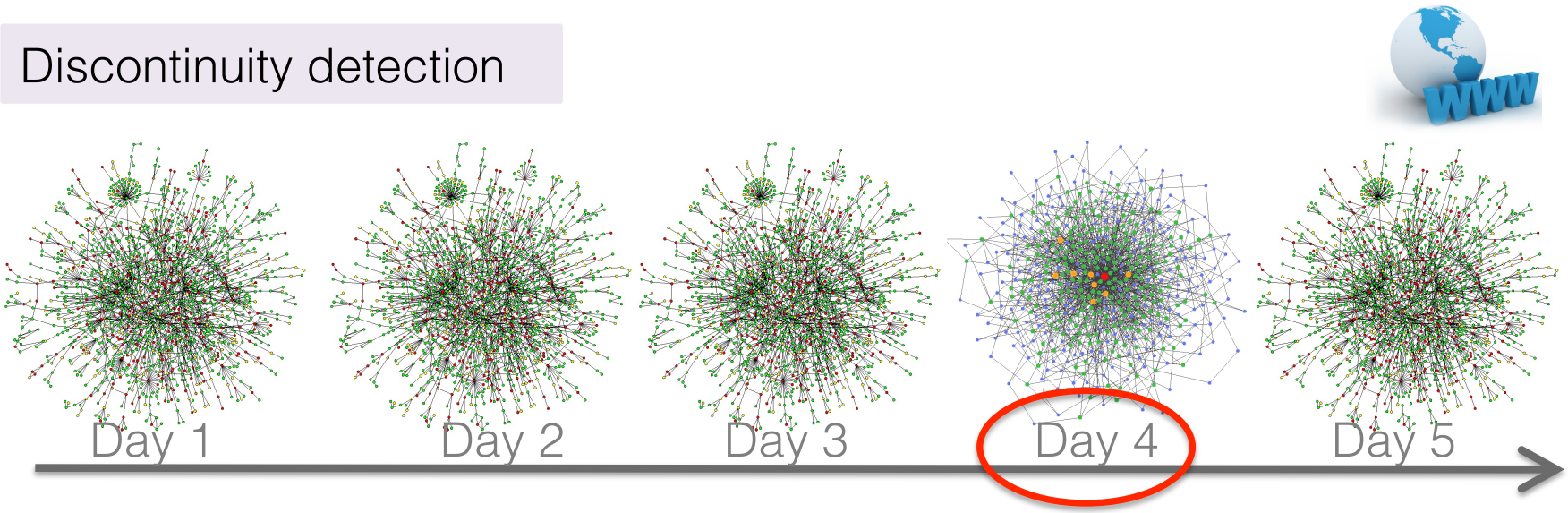
Applications (1/2)

Classification



different brain wiring?

Discontinuity detection

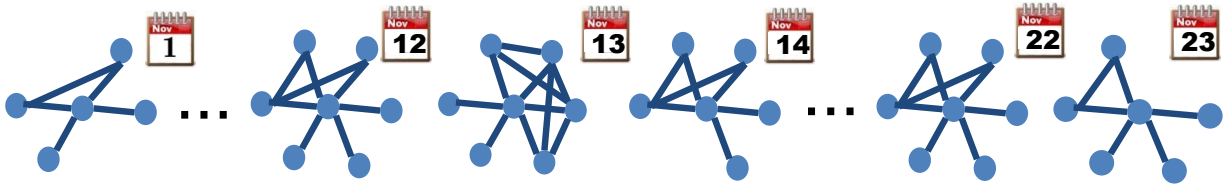


Applications (2/2)

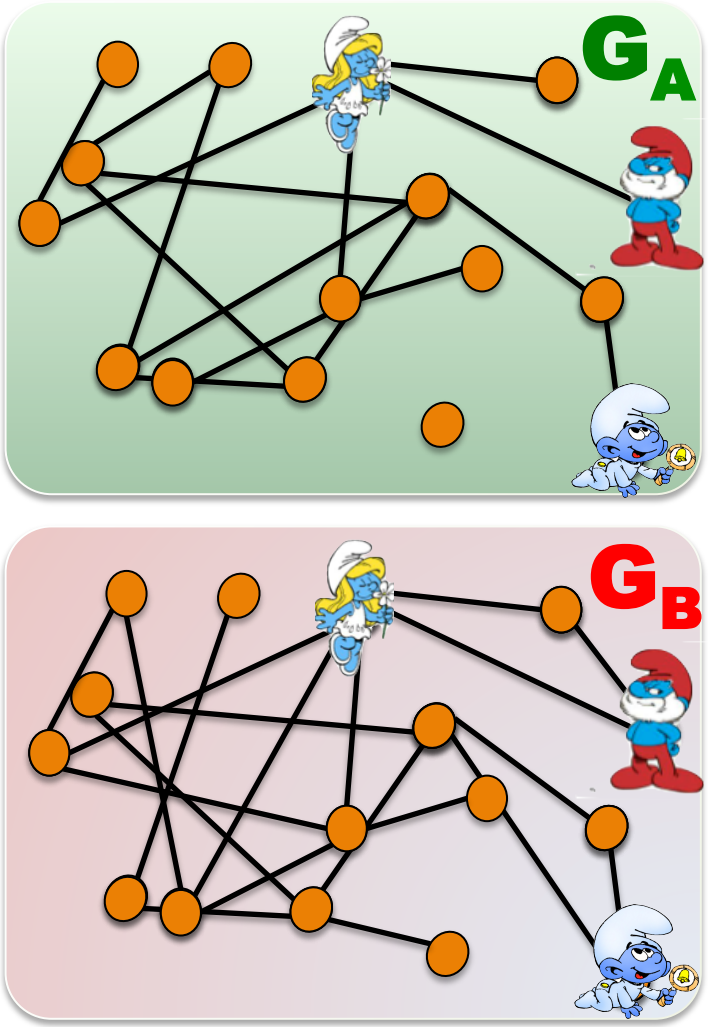
Behavioral Patterns

FB message graph vs. wall-to-wall network

Intrusion detection



A First Solution



Edge Overlap (EO)

of common edges

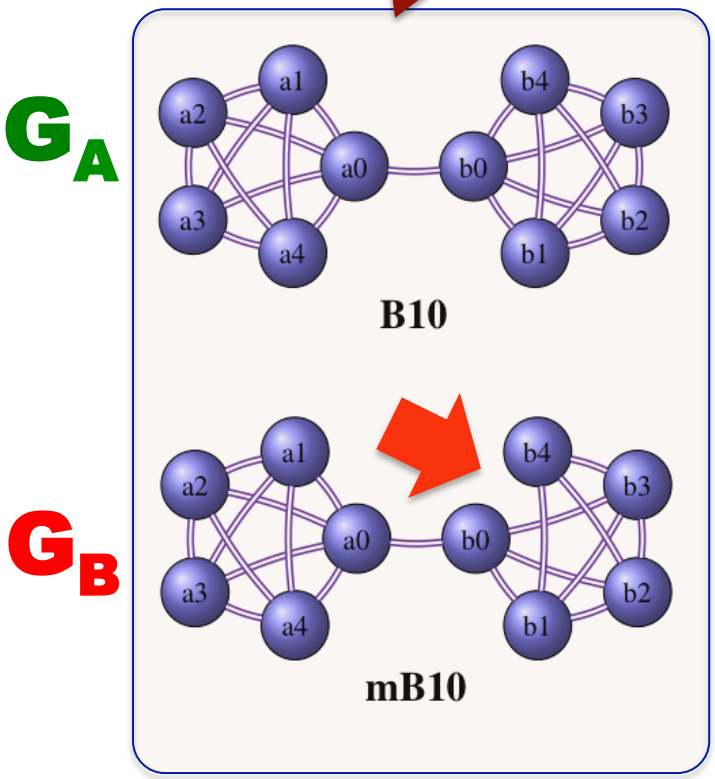
(normalized or not)

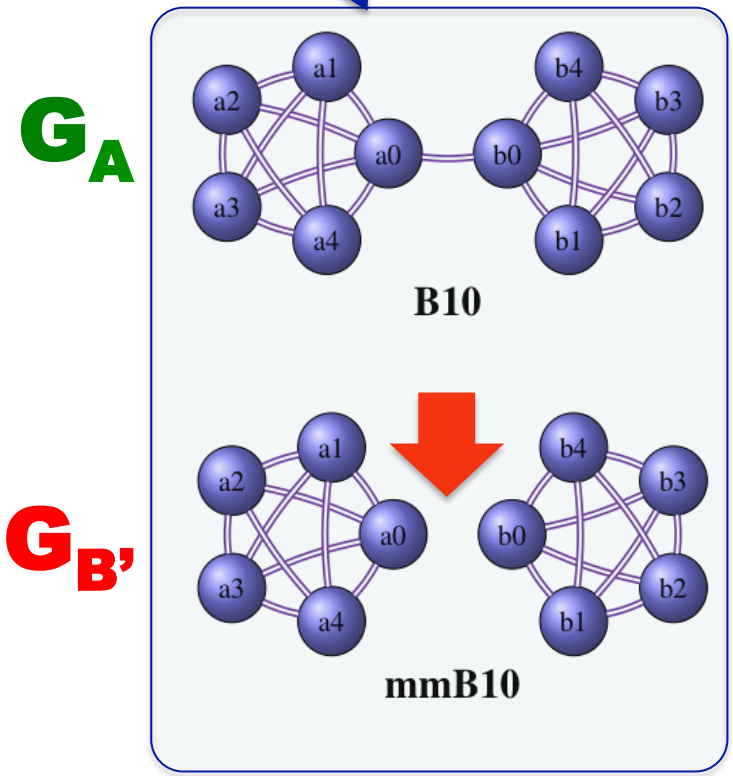
Is it a good measure?

Edge Overlap – Example

EO(B10, mB10) == EO(B10, mmB10)

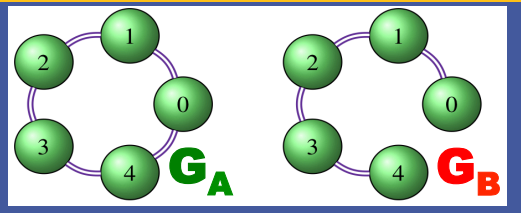
Barbell graph





Different impact in information flow

Similar If …



1. "… they share many vertices and/or edges"

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

Vertex/Edge Overlap:
$O(|V_A|+|V_B|+|E_A|+|E_B|)$

2. "… the rankings of their vertices are similar"

VR = rank correlation of node PageRank

Vertex Ranking:
$O(|V_A|+|V_B|)$

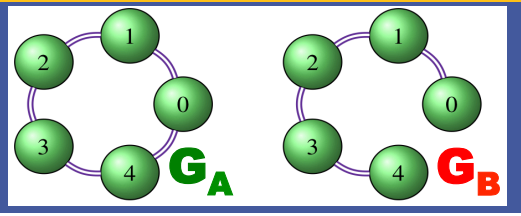
3. "… their edge weights are similar"

$$d(G_A, G_B) = \frac{1}{|E_A \cup E_B|} \sum_e \frac{|w_{GA}(e) - w_{GB}(e)|}{\max\{w_{GA}(e), w_{GB}(e)\}}$$

Weighted distance:
$O(|E_A|+|E_B|)$

[Papadimitriou, Dasdan, Garcia-Molina '10; Bunke '06, Shoubridge et al. '02, Dickinson et al. '04]

Similar If …



4. "… they have similar subgraphs"

Maximum Common Subgraph (MCS)
(NP-complete problem)

Vertex MCS Distance

$$d(G_A, G_B) = 1 - \frac{|mcs(V_A, V_B)|}{\max\{|V_A|, |V_B|\}}$$

Edge MCS Distance

$$d(G_A, G_B) = 1 - \frac{|mcs(E_A, E_B)|}{\max\{|E_A|, |E_B|\}}$$

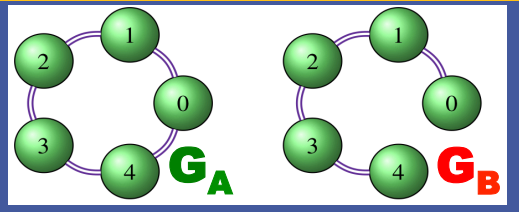
5. "… if we need few node/edge additions/deletions to transform G_A to G_B"

Graph Edit Distance

$$GED(G_A, G_B) = |V_A| + |V_B| - 2|V_A \cap V_B| + |E_A| + |E_B| - 2|E_A \cap E_B|$$

[Bunke '06, Shoubridge et al. '02, Dickinson et al. '04; [Bunke+ '98, '06, Riesen '09, Gao '10, Fankhauser '11; Kapsabelis et a. '07]

Similar If …



6. "… they have similar fingerprints"

Signature similarity

b-bit fingerprint of G_A :

1	0	1	0	1

b-bit fingerprint of G_B :

0	0	1	0	1

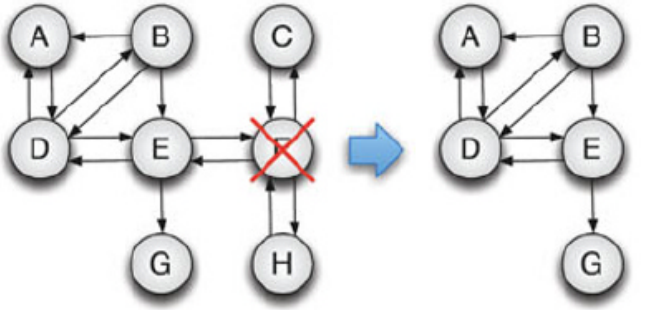
Hamming distance: 1

The fingerprint is created by applying a hash function on the nodes and edges (features):

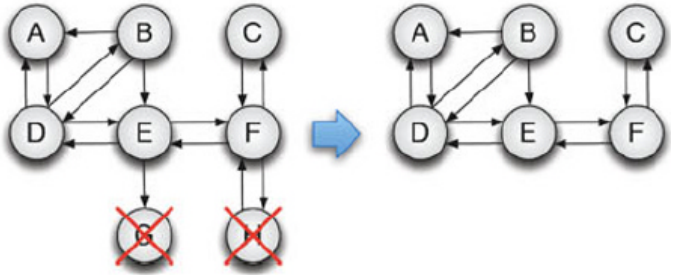
- Nodes: use the PageRank score
- Edges (u, v): The PageRank score of u multiplied by the out-degree of u

[Papadimitriou, Dasdan, Garcia-Molina '10]

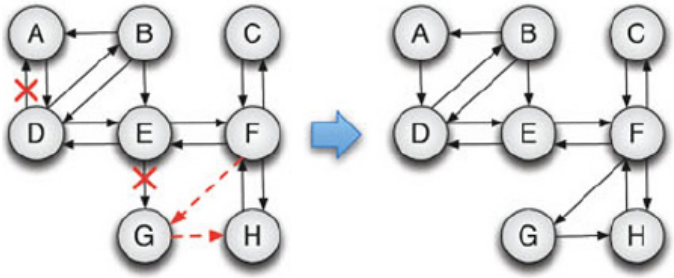
Application: Web Graph Anomaly Detection



Anomaly I : Missing connected subgraph



Anomaly II: Missing random nodes



Anomaly III: Connectivity change
(same # of nodes/edges but different connectivity for some nodes)

	Missing Subgraph	Missing Vertices	Topological Changes
Vertex/Edge Overlap	✓✓	✓	×
Vector Similarity	✓✓	✓	✓
Vertex Ranking	✓✓	×	×
Sequence Similarity	✓	×	✓✓
Signature Similarity	✓✓	✓✓	✓✓

Anomaly detection in the Web graph:

- Given a sequence of web graphs $G_1, \ldots, G_n$
- Compute similarity between G_i and G_{i+1}
- Construct time series based on similarity scores
- Detect anomalies

[Papadimitriou, Dasdan, Garcia-Molina '10]

Overview of Graph Similarity Functions

- Many similarity functions can be defined
- What properties should a good similarity function have?

Axioms of Graph Similarity Functions

A1. **Identity** property

sim( , ) = 1

A2. **Symmetric** property

sim( , ) = sim( , )

A3. **Zero** property

sim( , ) = 0

[Koutra, Faloutsos, Vogelstein '13]

Desired Properties

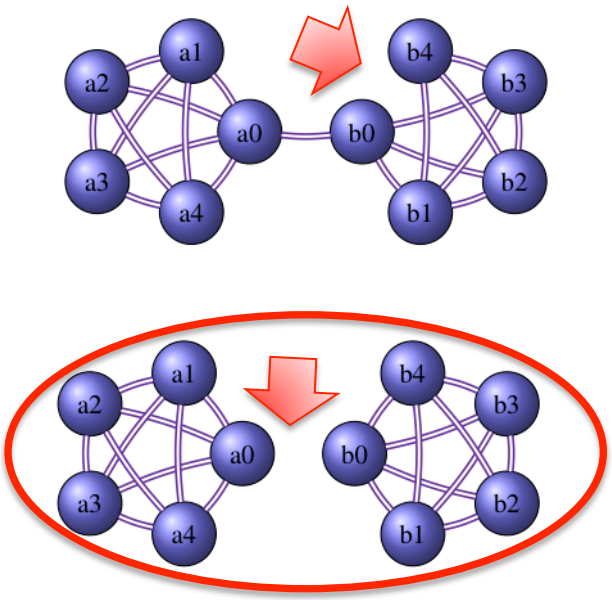
- Intuitiveness
  - P1. Edge Importance
  - P2. Weight Awareness
  - P3. Edge-"Submodularity"
  - P4. Focus Awareness

- Scalability

[Koutra, Faloutsos, Vogelstein '13]

Desired Properties

- Intuitiveness
  - P1. Edge Importance
  - P2. Weight Awareness
  - P3. Edge-"Submodularity"
  - P4. Focus Awareness
- Scalability

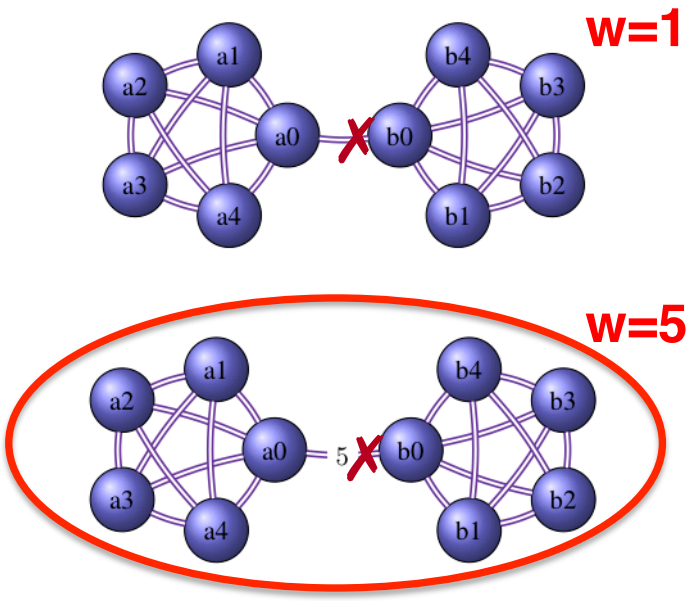


Creation of disconnected components matters more than small connectivity changes

[Koutra, Faloutsos, Vogelstein '13]

Desired Properties

- Intuitiveness
  - P1. Edge Importance
  - P2. Weight Awareness
  - P3. Edge-"Submodularity"
  - P4. Focus Awareness
- Scalability

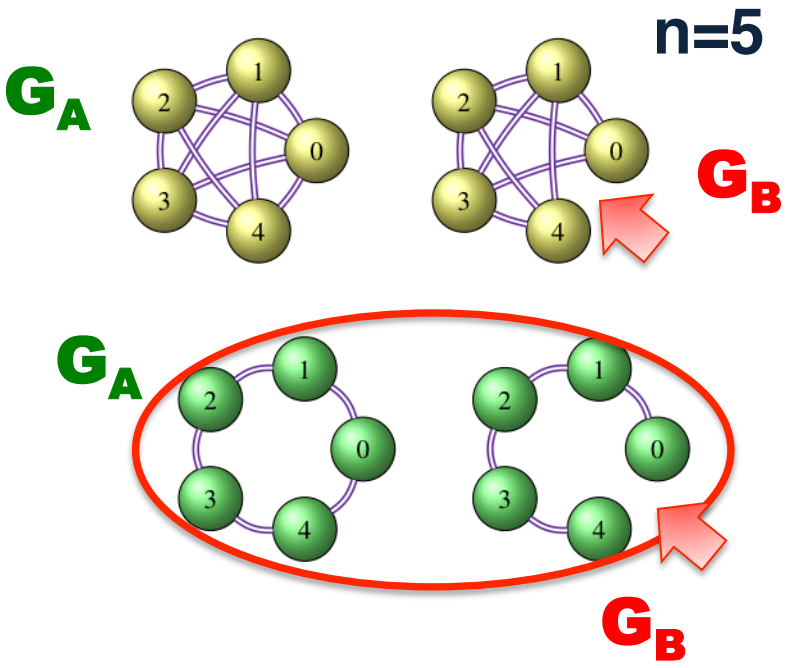


The bigger the edge weight, the more the edge change matters

[Koutra, Faloutsos, Vogelstein '13]

Desired Properties

- Intuitiveness
  - P1. Edge Importance
  - P2. Weight Awareness
  - P3. Edge-"Submodularity"
  - P4. Focus Awareness
- Scalability

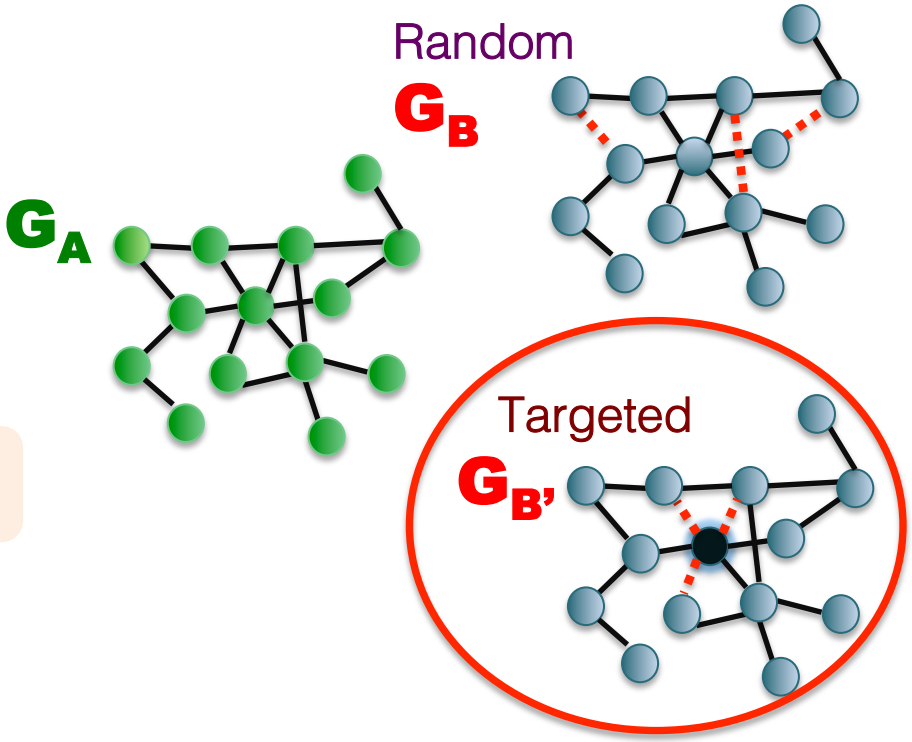


The sparser the graphs, the more important is a "fixed" change (e.g., removal of an edge)

[Koutra, Faloutsos, Vogelstein '13]

Desired Properties

- Intuitiveness
  - P1. Edge Importance
  - P2. Weight Awareness
  - P3. Edge-"Submodularity"
  - P4. Focus Awareness
- Scalability



Targeted changes are more important than random changes of the same extend

[Koutra, Faloutsos, Vogelstein '13]

Properties of State-of-the-Art Methods

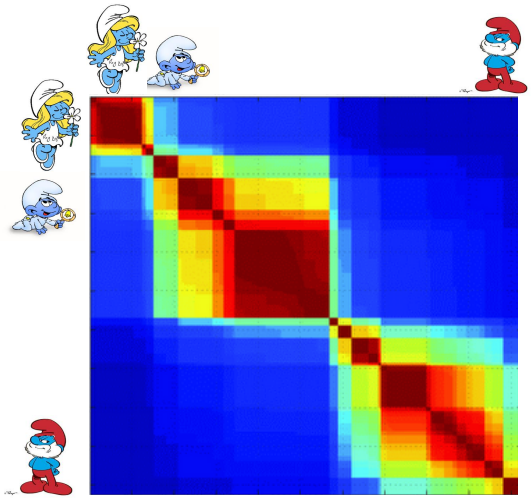
	importance	weight	sparsity	focus
Metric	P1	P2	P3	P4
Vertex/Edge Overlap	✗	✗	✗	?
Graph Edit Distance (XOR)	✗	✗	✗	?
Signature Similarity	✗	✔	✗	?

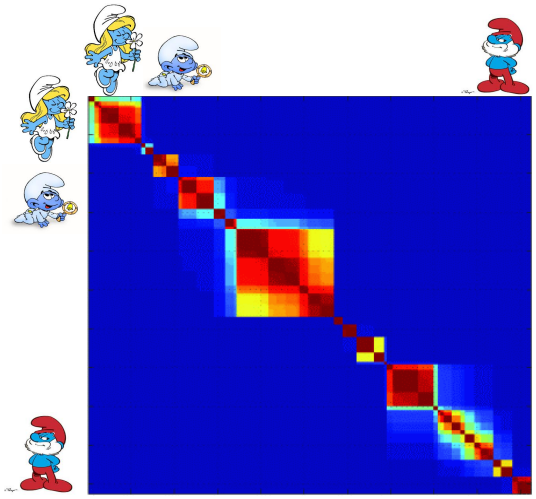
Looking for similarity functions that satisfy these properties

[Koutra, Faloutsos, Vogelstein '13]

DELTACON

1. Find the pairwise node influence, S_A and S_B
   - Node influence = node affinity
2. Find the similarity between S_A and S_B

$S_A =$ 

$S_B =$ 

[Koutra, Faloutsos, Vogelstein '13]

How to Measure Node Affinity

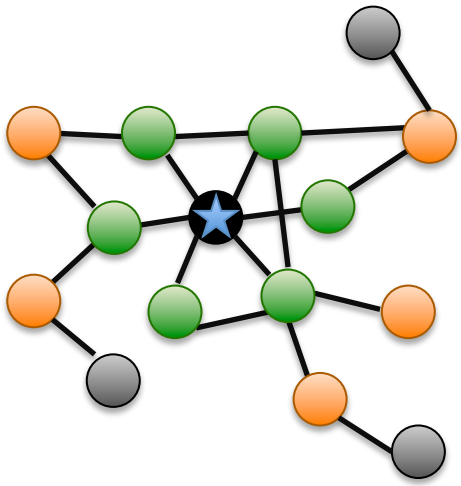
- Any random walk-based method can be used
  - E.g., PageRank, Random walk with restarts (RWR), personalized PR
  - Use Fast Belief Propagation (FaBP) Similar to Personalized RWR
- Attenuating Neighboring Influence for small ε:

$$S = [\mathbf{I} + \varepsilon^2 \mathbf{D} - \varepsilon \mathbf{A}]^{-1} \approx$$

S : similarity matrix

1-hop 2-hops ...

$$\approx [\mathbf{I} - \varepsilon \mathbf{A}]^{-1} = \mathbf{I} + \varepsilon \mathbf{A} + \varepsilon^2 \mathbf{A}^2 + \dots$$



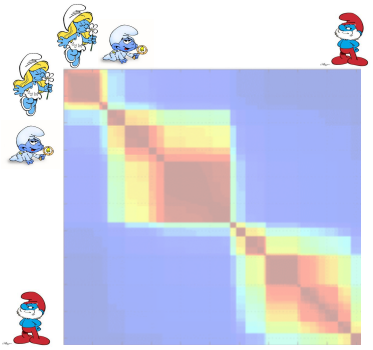
Note: $\varepsilon > \varepsilon^2 > \dots, 0<\varepsilon<1$

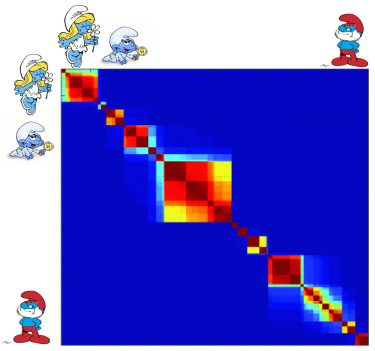
Why FaBP?
- Sound theoretical background
- Fast
- Intuitive: consider k-hop away information
  - Differences in long paths have smaller effect in the similarity

Graph Similarity with DeltaCon

1. Find the pairwise node influence, S_A and S_B
   - Node influence = node affinity
2. Find the similarity between S_A and S_B

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

$\mathbf{S_A}$ = 

$\mathbf{S_B}$ = 

$s(\mathbf{S_A}, \mathbf{S_B}) = 0.3$

[Koutra, Faloutsos, Vogelstein '13]

DELTACON - The Algorithm

Input: graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$

Output: similarity $s(G_1, G_2)$

1. $\mathbf{S}_1 = [\mathbf{I} + \epsilon^2 \mathbf{D}_1 - \epsilon \mathbf{A}_1]^{-1}$
2. $\mathbf{S}_2 = [\mathbf{I} + \epsilon^2 \mathbf{D}_2 - \epsilon \mathbf{A}_2]^{-1}$
3. $d(G_1, G_2) = \text{Root Euclidean Distance}(\mathbf{S}_1, \mathbf{S}_2)$
4. $s(G_1, G_2) = \frac{1}{1 + d}$

$S_{1,ij}$: affinity/influence of node i to node j in G_1

- Quadratic complexity (n^2 affinity scores need to be computed)
- DELTACON_0 : faster, linear approximation algorithm
  - Split the node set into g groups and compute n x g affinity scores

Code: http://www.cs.cmu.edu/~dkoutra/CODE/deltacon.zip

Properties of Similarity Methods

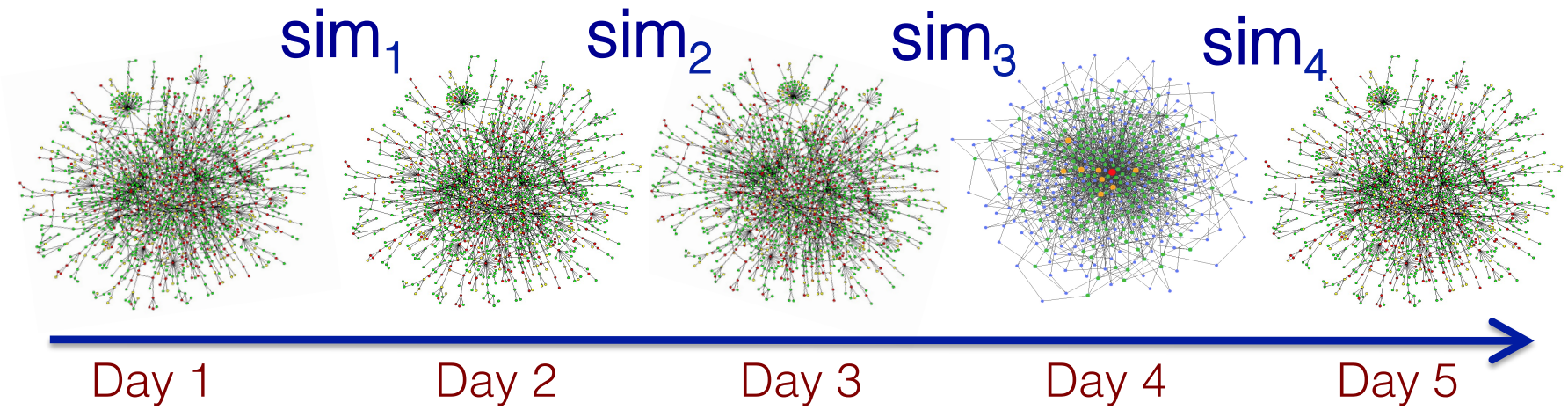
			importance	weight	sparsity	focus
Metric	P1	P2	P3	P4
Vertex/Edge Overlap	✗	✗	✗	?
Graph Edit Distance (XOR)	✗	✗	✗	?
Signature Similarity	✗	✔	✗	?
DELTACON	✔	✔	✔	✔
DELTACON$_0$	✔	✔	✔	✔

[Koutra, Faloutsos, Vogelstein '13]

Temporal Anomaly Detection (1/2)

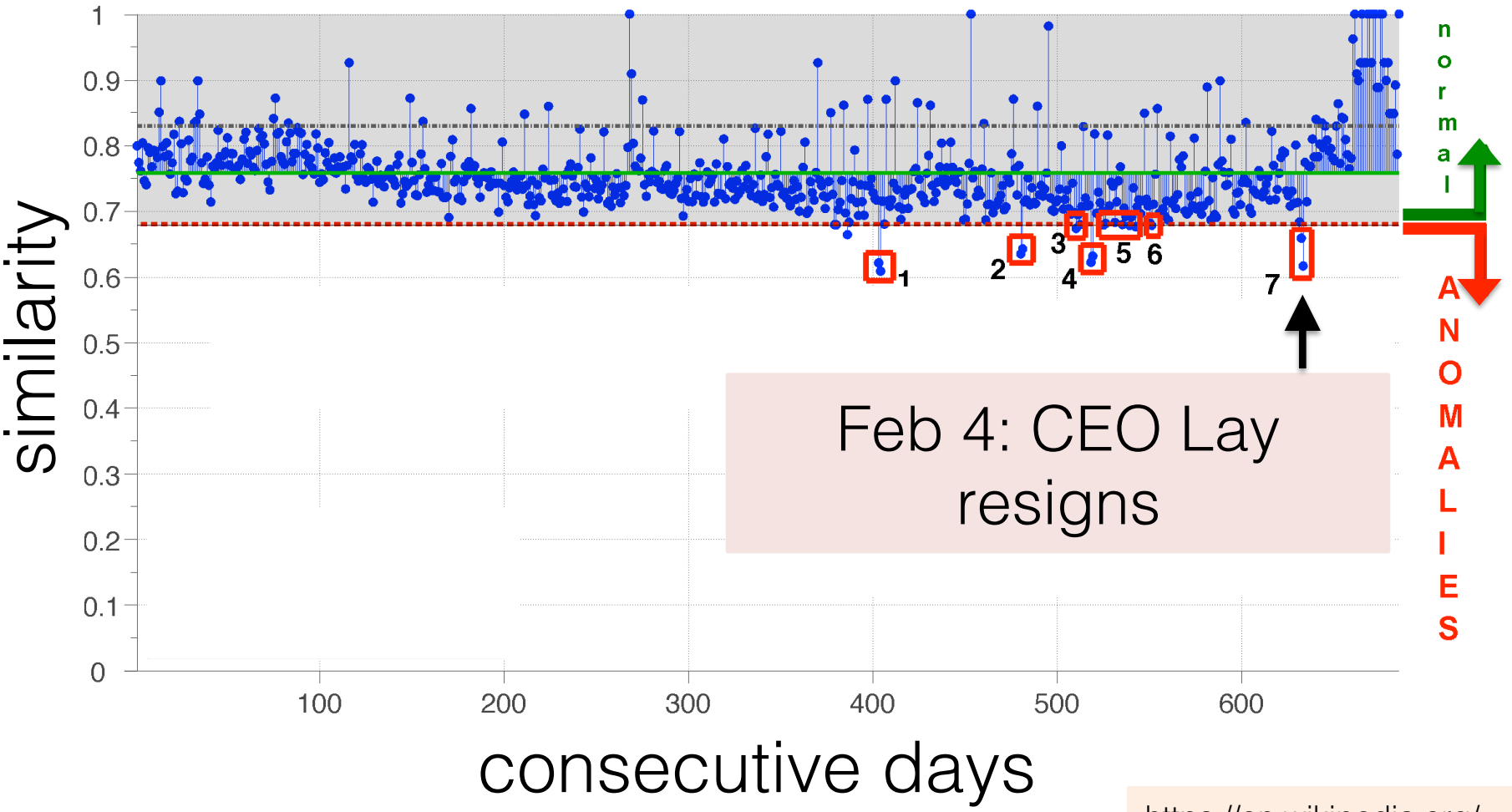
Nodes: employees

Edges: email exchange



[Koutra, Faloutsos, Vogelstein '13]

Temporal Anomaly Detection (2/2)

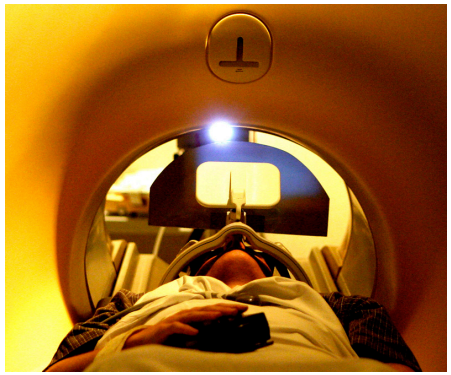


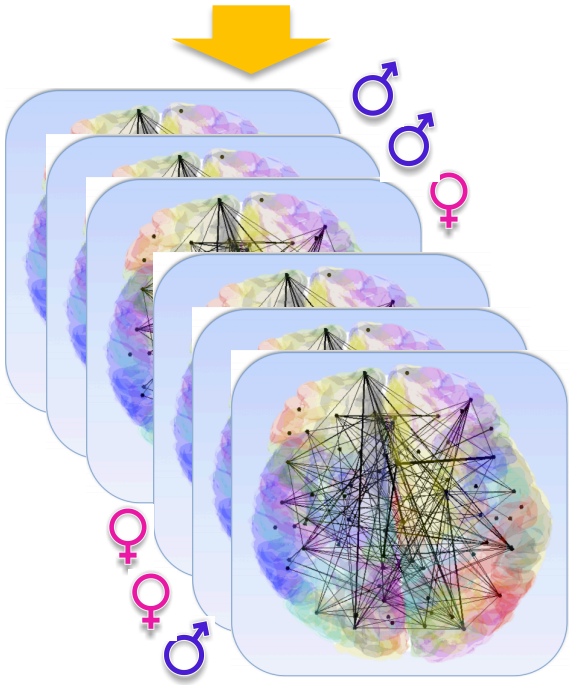
https://en.wikipedia.org/wiki/Enron_scandal

Use the method of Quality Control with Individual Moving Range to define the threshold below which days are anomalous (red line): median ± 3σ

Brain Connectivity Graph Clustering (1/2)

- 114 brain graphs (Multimodal MRI)
  - One graph per person
  - Nodes: 70 cortical regions
  - Edges: connections
- Attributes: gender, IQ, age…



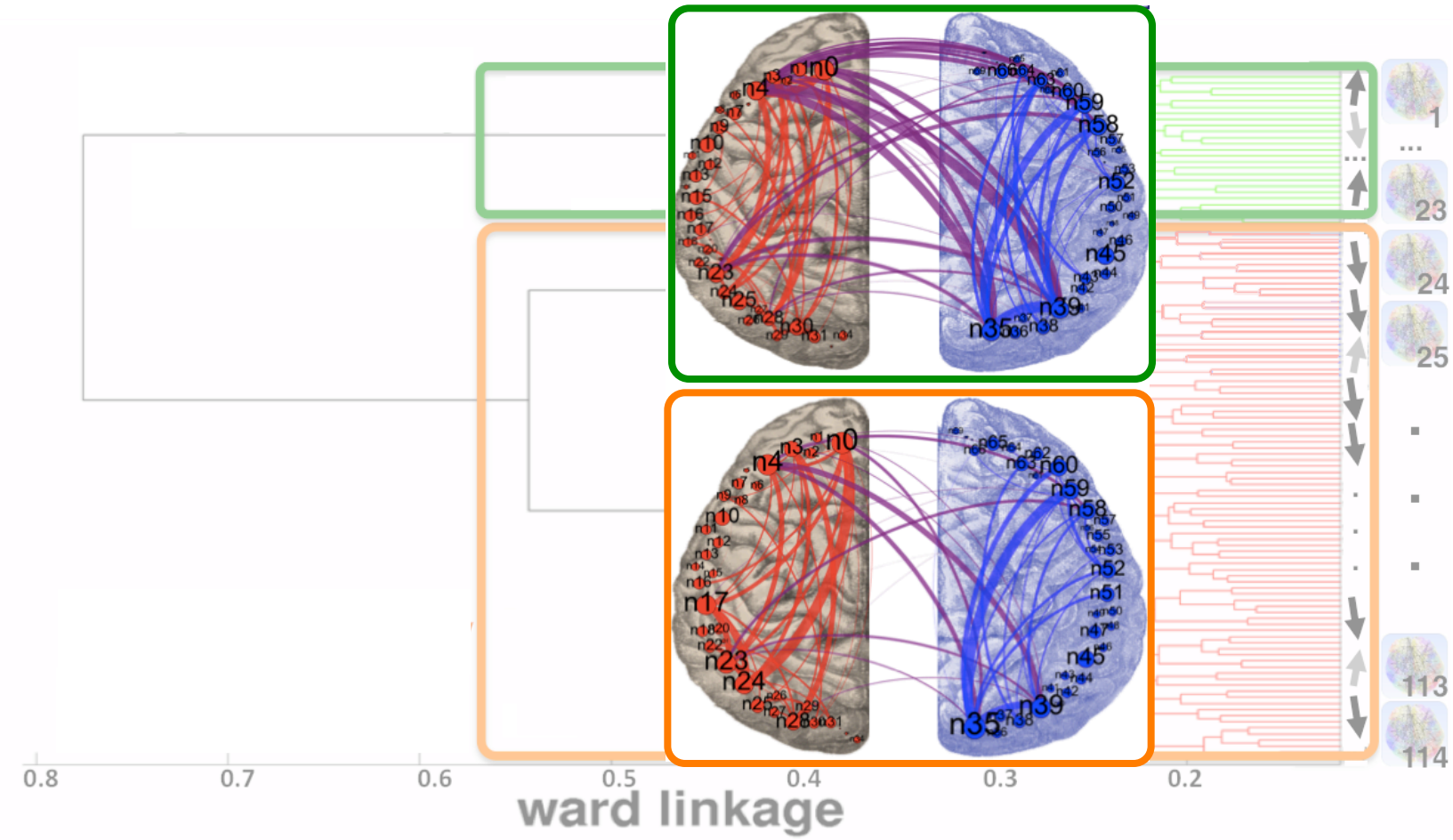


Experimental setup

- Compute similarities with DELTACON
- Perform hierarchical clustering using Ward's method (https://en.wikipedia.org/wiki/Ward%27s_method)
- Apply t-test on the attributes of the extracted groups to detect differences

[Koutra, Faloutsos, Vogelstein '13]

Brain Connectivity Graph Clustering (2/2)



Significant difference in t-test:
p-value = 0.0057

[Koutra, Faloutsos, Vogelstein '13]

CCI: Composite Creative Index
- Related to a person's performance on a series of creative tasks

Graph similarity:
unknown node correspondence

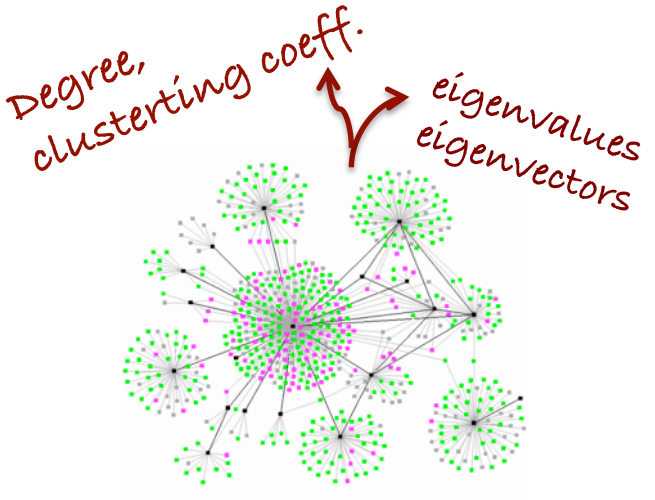
Main Idea

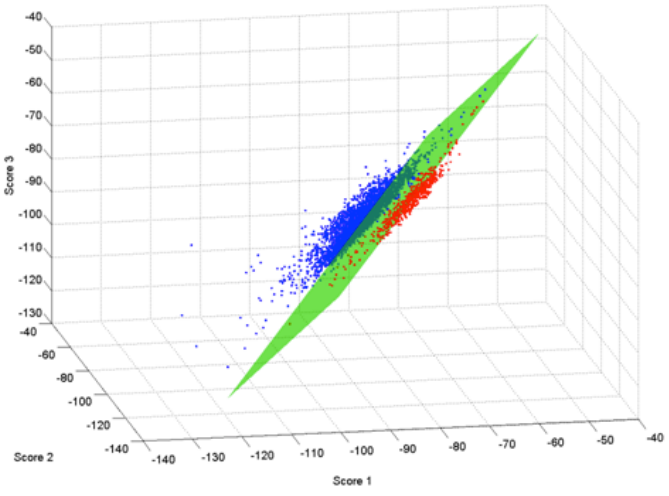
1. Extract graph features

2. s(G_A, G_B) = "similarity" between the features

Or

apply some classifier





Which features?
- Consider spectral information

λ-distance

- Eigenvalues of G_A: $\lambda_{A1} \leq \ldots \leq \lambda_{At}$
- Eigenvalues of G_B: $\lambda_{B1} \leq \ldots \leq \lambda_{Bs}$

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

[Bunke '06, Wilson '08, ElGhawalby '08]

λ-distance: Variations

- Eigenvalues of different matrices
  - Adjacency

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

  - Laplacian

$$0 = \lambda_{A1} \leq \ldots \leq \lambda_{At}$$

$$L = D - A = \begin{bmatrix} d_1 & & \\ & d_2 & \\ & & d_3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

  - Normalized Laplacian

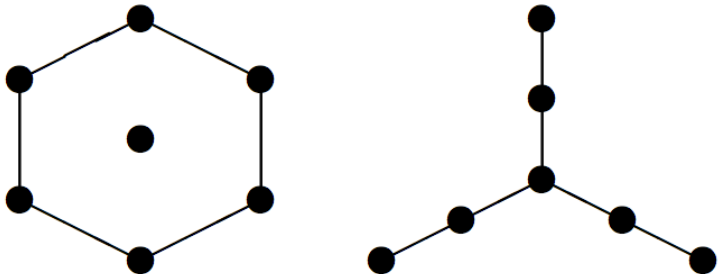
$$0 = \lambda_{A1} \leq \ldots \leq \lambda_{At} \leq 2$$

$$L_{norm} = D^{-1/2} A D^{-1/2} = \begin{bmatrix} d_1^{-.5} & & \\ & d_2^{-.5} & \\ & & d_3^{-.5} \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} d_1^{-.5} & & \\ & d_2^{-.5} & \\ & & d_3^{-.5} \end{bmatrix}$$

[Bunke '06, Wilson '08, ElGhawalby '08]

λ-distance: Disadvantages

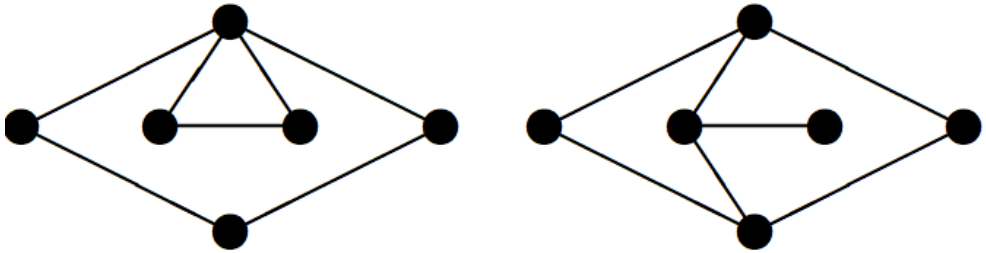
1) co-spectral graphs with **different** structure



Same spectrum

Non-isomorphic graphs, co-spectral w.r.t. A

Non-isomorphic graphs, co-spectral w.r.t. L



[Haemers+ '04, Brouwer '09]

λ-distance: Disadvantages

1) co-spectral graphs with **different** structure

2) subtle changes in the graphs =>
big differences in the spectrum

3) $O(n^3)$ runtime
   - E.g., using SVD

4) Labeled graphs: nodes can be associated with labels
   - How can you take into account this information?
   - Use graph kernels

[Haemers+ '04, Brouwer '09]

Graph Kernels

Graph Isomorphism

- Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijective mapping (one to one correspondence) $f: V_1 \rightarrow V_2$ such that $(v_i, v_j) \in E_1$ iff $(f(v_1), f(v_2)) \in E_2$

- The Graph Isomorphism (GI) is an "extreme" case of the graph similarity problem
  - It asks if two graphs are structurally equivalent to each other
  - No polynomial algorithm for GI is known
    - Quasipolynomial time $2^{(\log n)^{O(1)}}$ by L. Babai [Babai '15]

Example

The following graphs are isomorphic

Function f:

- f(1) = a
- f(2) = b
- f(3) = d
- f(4) = c

One-to-one correspondence between the nodes of the two graphs

Subgraph Isomorphism

- The Subgraph Isomorphism problem asks if there is a subset of edges and nodes of G_1 that is isomorphic to a smaller graph G_2
  - NP-complete problem

- Practical problems
  - The runtime may grow exponentially
  - Can we have a polynomial time similarity measure?

Graph Kernels – The Idea

- Compare substructures of graphs that are computable in polynomial time
  - Walks
  - Shortest paths
  - Cyclic patterns
  - Trees
  - Graphlets

Our goal: Learning on graph data

- Represent a graph with a fixed-length feature vector
- Typical representation required by most of ML algorithms
- There is no straightforward way to transform graphs to such a representation

[Vishwanathan et al '10]

Kernel Methods in ML

- The function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a kernel if it is:
  - Symmetric $k(x, y) = k(y, x)$
  - Positive semi-definite $\forall x_1, x_2, \ldots x_n \in \mathcal{X}$, the matrix $\mathbf{K}_{ij} = k(x_i, x_j)$ is positive semi-definite

- If a function satisfies the above two conditions on a set $\mathcal{X}$, it is known that there exists a mapping $\phi : \mathcal{X} \rightarrow \mathbb{H}$ into a Hilbert space $\mathbb{H}$ such that:

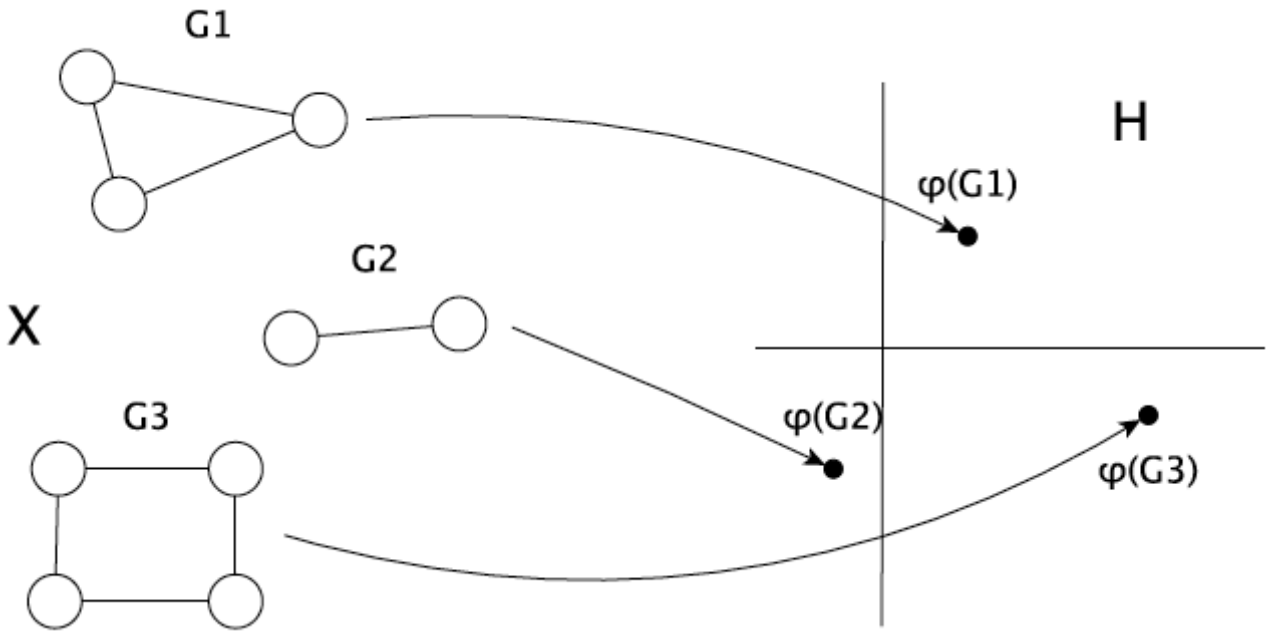
$$k(x, y) = \langle \phi(x), \phi(y) \rangle$$

  for all $(x, y) \in \mathcal{X}^2$ where $\langle \cdot, \cdot \rangle$ is the inner product in $\mathbb{H}$

- Informally, $k(x, y)$ is a measure of similarity between x and y

Graph Kernels

- A graph kernel $k : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$ is a kernel function over a set of graphs $\mathcal{G}$



Makes the family of kernel methods applicable to graphs

[Vishwanathan et al '10]

Kernel Trick in ML

- Many machine learning algorithms can be expressed only in terms of inner products between vectors

- Computing the explicit mappings $\phi(x_1)$, $\phi(x_2)$ and their inner product $\langle\phi(x_1), \phi(x_2)\rangle$ for the pair of graphs can be computationally expensive

- Kernel trick: avoid the explicit mapping by directly computing the inner product $\langle\phi(x), \phi(y)\rangle$ via the kernel function

Random Walk Kernel – The Idea

- Idea: count common walks in $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$
  - Walks are sequences of nodes that allow repetitions of nodes
  - Recall that, walks of length ℓ can be computed by taking the adjacency matrix A to the power of k
  - $A^\ell(i, j) = c$ means that c walks of length ℓ exist between nodes i and j

- How to find common walks in two graphs?
  - Use the Product graph G_x of G_1 and G_2

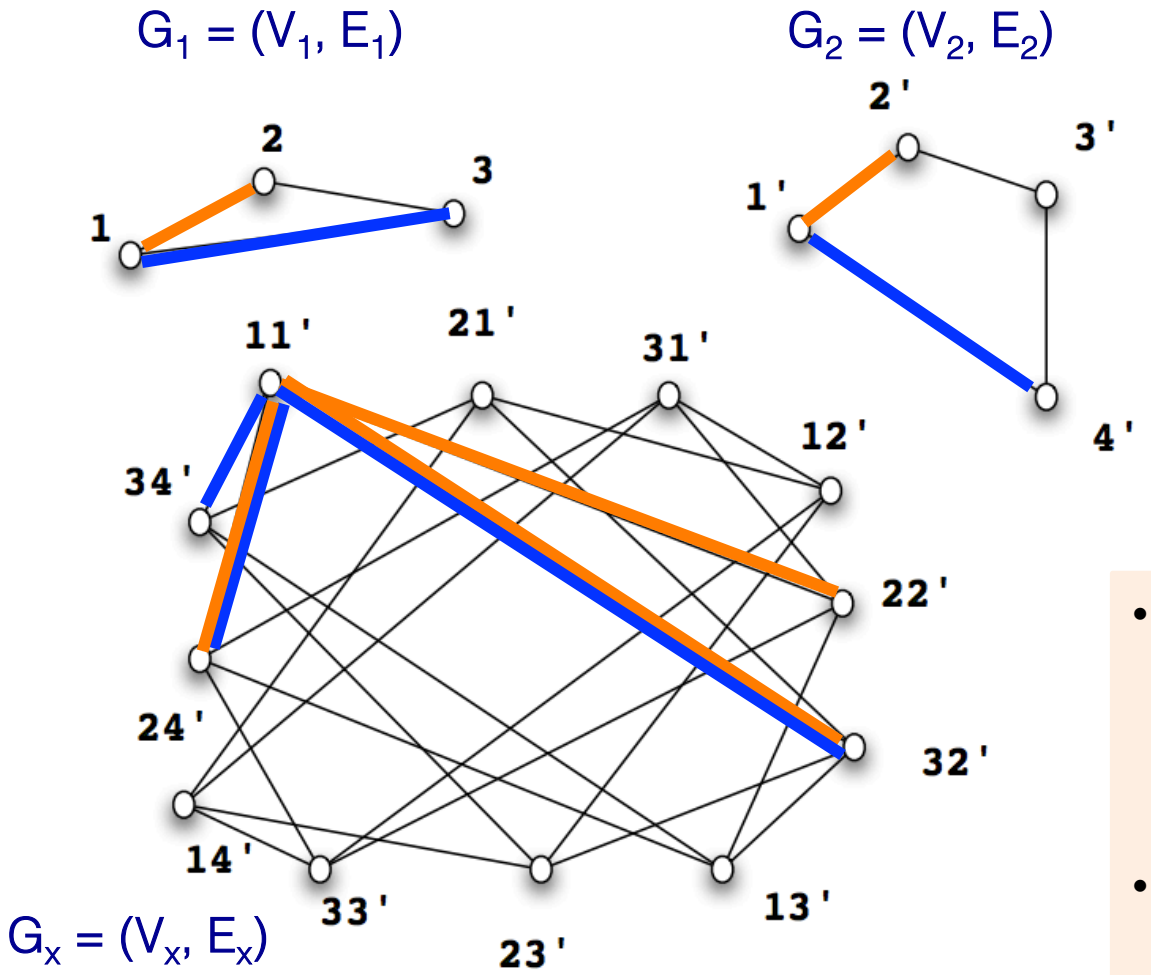
$$V_\times = \{(v_1, v_2) : v_1 \in V_1, v_2 \in V_2\}$$

$$E_\times = \{((v_1, v_2), (u_1, u_2)) : (v_1, u_1) \in E_1 \quad , (v_2, u_2) \in E_2\}$$

Nodes: pairs of vertices from G_1 and G_2
Draw edge if corresponding nodes of G_1 and G_2 are adjacent in G_1 and G_2

[Vishwanathan et al '10]

Product Graph



- Performing a random walk on G_x is equivalent to performing a simultaneous random walk on G_1 and G_2
- Common walks of length ℓ can be computed using A_x^{ℓ}

$$V_\times = \{(v_1, v_2) : v_1 \in V_1, v_2 \in V_2\}$$
$$E_\times = \{((v_1, v_2), (u_1, u_2)) : (v_1, u_1) \in E_1 \in V_1, (v_2, u_2) \in E_2\}$$

Random Walk Kernel – Computation

- Random walk kernel
  - Construct the product graph G_x
  - Count walks in this product graph
  - The random walk kernel can be computed as follows:

$$k_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{\ell=0}^{\infty} \lambda^\ell \mathbf{A}_\times^\ell \right]_{ij}$$

$$= \mathbf{1}^T (\mathbf{I} - \lambda \mathbf{A}_\times)^{-1} \mathbf{1}$$

λ is a convergence factor

- If both graphs have size n (number of nodes)
  - $O(n^4)$ for computing the product graph (comparison for all pairs of edges)
  - Matrix multiplication or inversion of an $n^2 \times n^2$ matrix
  - Total complexity: $O(n^6)$
  - Can be reduced to $O(n^3)$ (using the Sylvester equation)

Shortest Path Kernels (1/3)

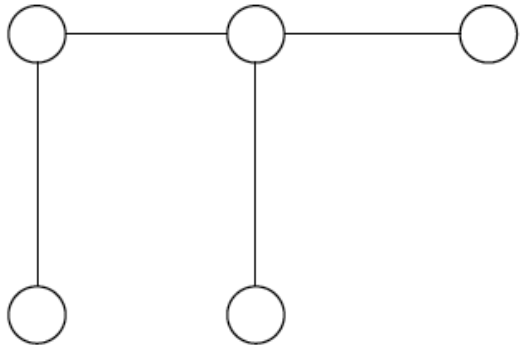
- The similarity of G_1 and G_2 is determined by the similarity of the length of shortest paths in the two graphs

- Floyd transformation
  - Compute the shortest-paths between all pairs of nodes of the input graph G using the Floyd-Warshall algorithm
  - Create a shortest-path graph S which contains the same set of nodes as the input graph G
  - All nodes which are connected by a walk in G are linked with an edge in S
  - Each edge in S is labeled by the shortest distance between its endpoints in G

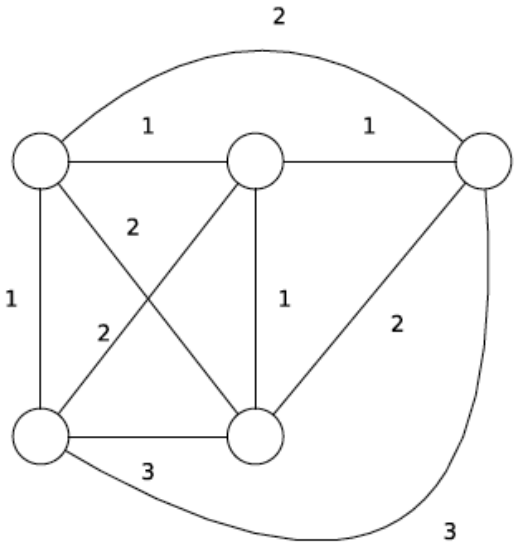
[Borgwardt and Kriegel '05]

Shortest Path Kernels (2/3)

Floyd transformation



G



S

Shortest Path Kernels (3/3)

- Given the Floyd-transformed graphs $S_1 = (V_1, E_1)$ and $S_2 = (V_2, E_2)$ of G_1 and G_2, the shortest path kernel is defined as

$$k(G_1, G_2) = \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} k^{(1)}_{walk}(e_1, e_2)$$

where $k^{(1)}_{walk}$ is a kernel on edge walks of length 1

Complexity: $O(n^4)$

edge labels (shortest path values)

$$k^{(1)}_{walk} = \begin{cases} 1, & \text{if } \ell(e_1) = \ell(e_2) \\ 0, & \text{otherwise} \end{cases}$$

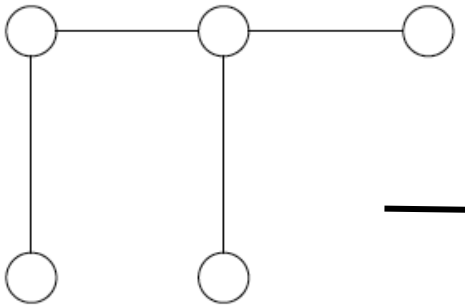
Unlabeled graphs

Labeled graphs

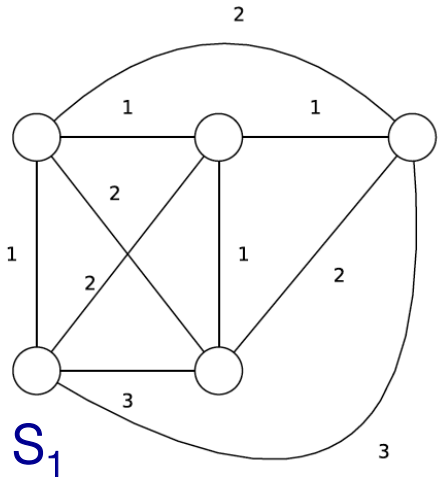
$$k^{(1)}_{walk} = \begin{cases} 1, & \text{if } \ell(e_1) = \ell(e_2) \text{ and } \ell(u_1) = \ell(u_2) \text{ and } \ell(v_1) = \ell(v_2) \\ 0, & \text{otherwise} \end{cases}$$

edge endpoint labels: $e_1 = (u_1, v_1)$

Shortest Path Kernels – Example

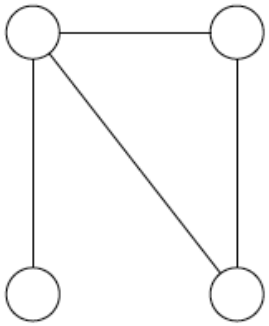


G_1

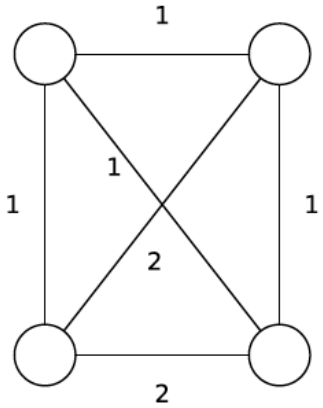


S_1

- In S_1 we have:
  - 4 edges with label 1
  - 4 edges with label 2
  - 2 edges with label 3
- In S_2 we have:
  - 4 edges with label 1
  - 2 edges with label 2



G_2



S_2

$$k(G_1, G_2) = \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} k^{(1)}_{walk}(e_1, e_2)$$
$$= 4 \times 4 + 4 \times 2 = 24$$

Graph Similarity – What to Remember

- Numerous applications in graph mining and learning
  - Network monitoring, anomaly detection, network intrusion, behavioral studies
- Although seems easy problem, it's not
  - Some measures are counter-intuitive
  - Graph isomorphism is a computationally difficult problem
  - DeltaCon [Koutra et al. '13] (based on node proximity) satisfies several intuitive properties
- There are multiple measures, but which one to use?
  - Depends on the application!
  - Good news according to the guide of [Soundarajan et al. '14]
- Graph kernels and application in graph classification
  - Kernel trick: they can directly be applied in well known learning algorithms (e.g., SVMs)
  - Scalability issues

Thank You!

DiscoverGreece.com