CptS 591: Elements of Network Science

Graph Similarity

OVERVIEW

Similarity in Graphs

- Similarity between nodes
  - Structural equivalence
  - Regular equivalence
  - Homophily
- Similarity between graphs
  - With known node correspondence
  - With unknown node correspondence

Graph Similarity Applications

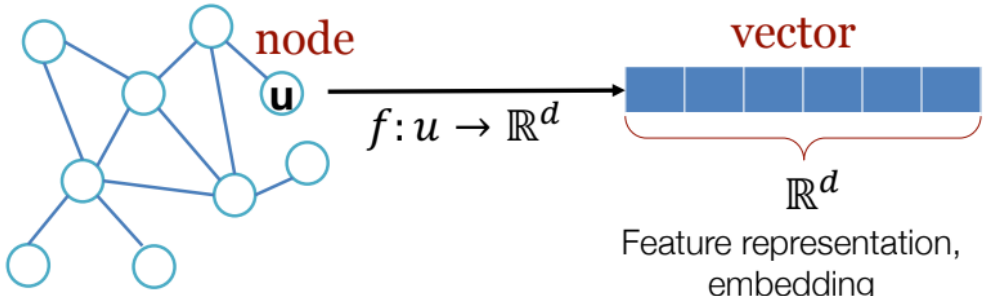
- Re-identification across networks
- Anomaly Detection
- Classification
- Clustering
- Cross-network analytics
- Recommendation systems
- ...

Similarity in Graphs

- Similarity between nodes
  - Structural equivalence
  - Regular equivalence
  - Homophily
  - **Graph embedding – Representation Learning**
- Similarity between graphs
  - With known node correspondence
  - With unknown node correspondence

Embedding based techniques

- Feature Engineering is one of the most important parts of Machine life cycle
- Possible features in graph: node degree, page rank score, degree of neighbors, clustering coefficient, …
- How to extract some features automatically from raw data?
- Graph Embedding:
  - Given a graph G = (V, E), a graph embedding is a mapping f : $v_i \rightarrow y_i \in R^d \forall i \in [n]$ such that d<<|V| and the function **f** preserves some proximity measure defined on graph G.



Jure Leskovec - Graph Representation Learning

Embedding based techniques

- Direct encoding approaches
  - **Factorization-based approaches**
  - Random walk approaches
- Generalized encoder-decoder architectures
  - Neighborhood auto-encoder methods
  - Neighborhood aggregation and convolutional encoders
- Others:

  - GraphSage, GraphWave, …

DETAILS

Related papers and a tutorial

1. **Graph Factorization:** Ahmed, Amr, et al. "Distributed large-scale natural graph factorization." Proceedings of the 22nd international conference on World Wide Web. ACM, 2013.

2. **GraRep:** Cao, Shaosheng, Wei Lu, and Qiongkai Xu. "Grarep: Learning graph representations with global structural information." Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM, 2015.

3. **Hope:** M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In KDD, 2016

4. **Graph Convolutional Network**: Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).

5. **Node2vec**: Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016.

6. **Deepwalk:** Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.

7. **Deltacon**: Koutra, Danai, Joshua T. Vogelstein, and Christos Faloutsos. "Deltacon: A principled massive-graph similarity function." *Proceedings of the 2013 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 2013.

8. **FaBP**: Koutra, Danai, et al. "Unifying guilt-by-association approaches: Theorems and fast algorithms." *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, Berlin, Heidelberg, 2011.

9. **Netsimile**: Berlingerio, Michele, et al. "Netsimile: A scalable approach to size-independent network similarity." *arXiv preprint arXiv:1209.2684* (2012).

10. **Big-align**: Koutra, Danai, Hanghang Tong, and David Lubensky. "Big-align: Fast bipartite graph alignment." *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 2013.

11. **Graph2vec**: Narayanan, Annamalai, et al. "graph2vec: Learning Distributed Representations of Graphs." *arXiv preprint arXiv:1707.05005* (2017).

12. **A good Tutorial**: http://web.eecs.umich.edu/~dkoutra/tut/icdm14.html

Factorization Based Methods

- Graph Factorization[1], GraRep[2], Hope[3] are popular examples
- Directly inspired by classic techniques for dimensionality reduction (PCA)
- Laplacian eigenmaps, Inner-product methods
- Loss function: $$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{D}} \|\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\|_2^2,$$
- Decoder $\text{DEC}(\mathbf{z}_i, \mathbf{z}_j)$ could be as simple as an inner product
- graph proximity measure $s_{\mathcal{G}}(v_i, v_j)$ used could be based on adj. matrix, its various powers, neighborhood overlaps, …

Embedding based techniques

- Direct encoding approaches
  - Factorization-based approaches
  - **Random walk approaches**
- Generalized encoder-decoder architectures
  - Neighborhood auto-encoder methods
  - Neighborhood aggregation and convolutional encoders
- Others:

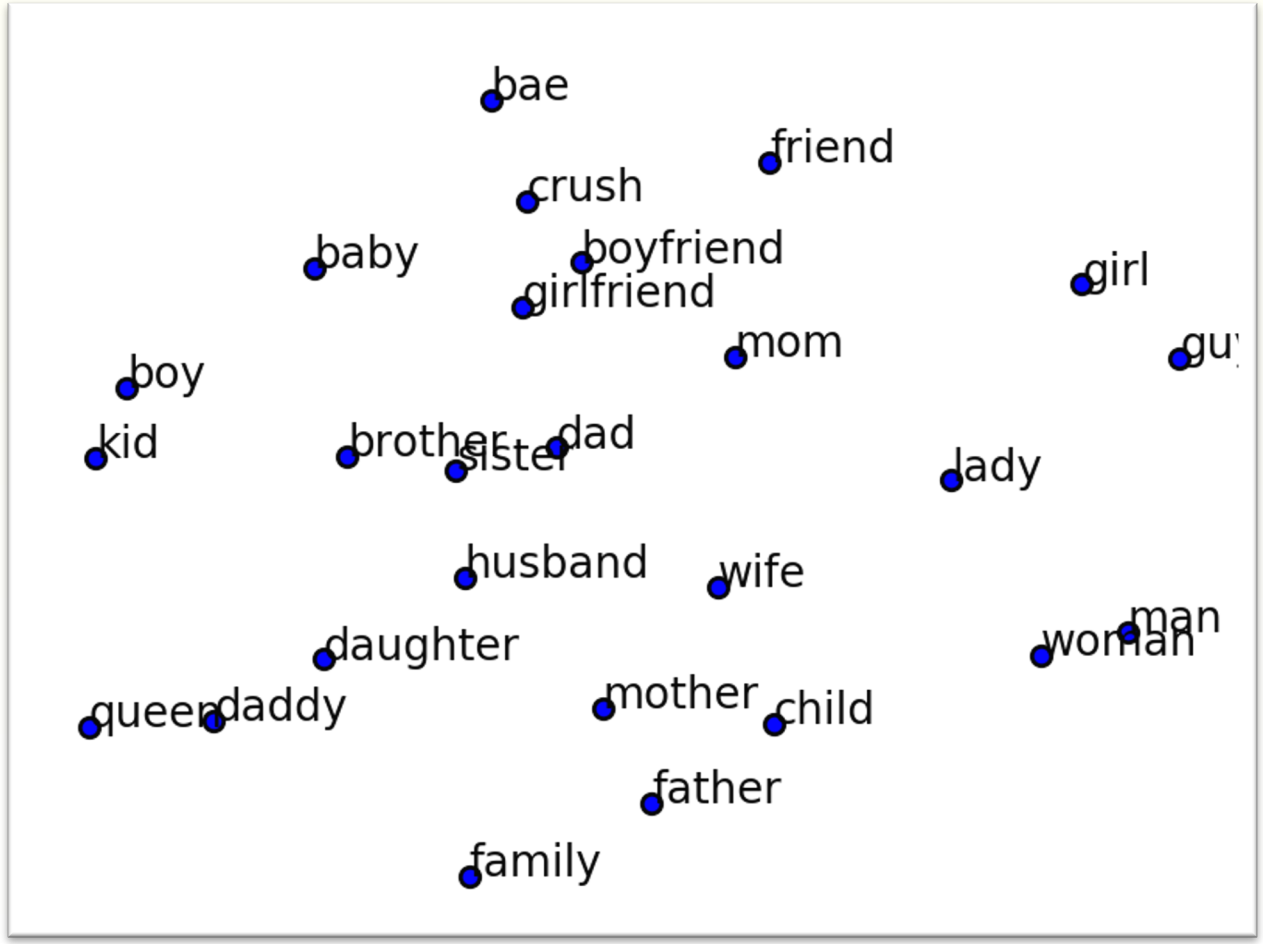
  - GraphSage, GraphWave, …

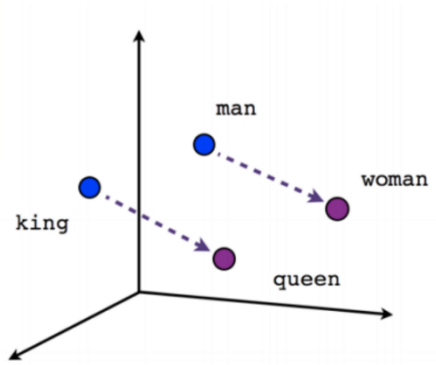
Random Walk methods

- Node2vec[5] and Deepwalk[6] are popular examples
- Node2vec: using a famous framework called Word2vec
  - Word2vec is proposed by Google Research
  - A word embedding approach to map each word to a vector of size d
  - If two words have the same context, the embedded vector would be similar
  - Input is a set of sentences, output is the embedding

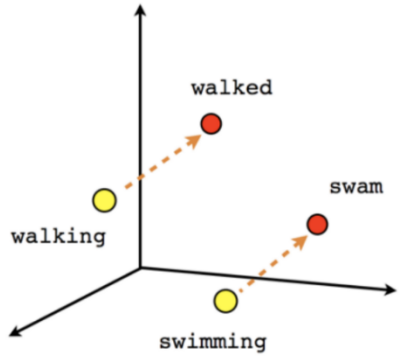
    (architecture behind it: skip-gram algorithm for sampling, neural network, negative sampling,…)

Random Walk methods





Male-Female



Verb tense

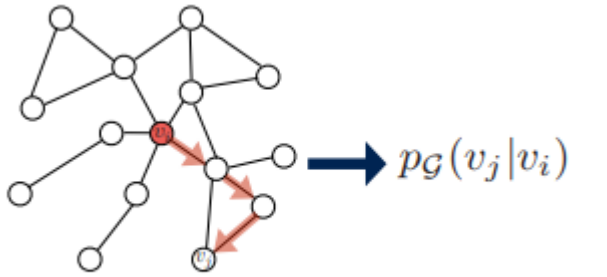
Tensorflow.org/tutorials/word2vec

Random Walk methods

- Idea: doing a (customized) random walk starting from each node can make a sentence closed by nodes
- Use word2vec to generate such embedding of nodes

Goal: Find embedding $f(u)$ that predicts nearby nodes $N_S(u)$:

$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u))$$



1. Run random walks to obtain co-occurrence statistics.

Jure Leskovec - Graph Representation Learning

Embedding based techniques

- Direct encoding approaches
  - Factorization-based approaches
  - Random walk approaches
- Generalized encoder-decoder architectures
  - **Neighborhood auto-encoder methods**
  - Neighborhood aggregation and convolutional encoders
- Others:

  - GraphSage, GraphWave, …

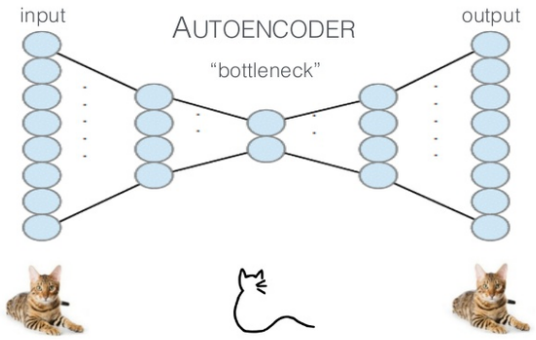
Auto-encoder Based

- Examples: Deep Neural Graph Representations (DNGR) and Structural Deep Network Embedding (SDNE)
- Deep learning auto-encoder approach to encode data and then decode to reconstruct the graph
- Objective:

$$\mathrm{DEC}(\mathrm{ENC}(\mathbf{s}_i)) = \mathrm{DEC}(\mathbf{z}_i) \approx \mathbf{s}_i$$

- Loss function:

$$\mathcal{L} = \sum_{v_i \in \mathcal{V}} \|\mathrm{DEC}(\mathbf{z}_i) - \mathbf{s}_i\|_2^2.$$



- S could be any representation of information on graph
  - Random walk mutual information
  - Adjacency vector
  - Laplacian eigenmaps
- the structure and size of the auto-encoder is fixed, strictly transductive and cannot cope with evolving graphs, nor can they generalize across graphs.

Embedding based techniques

- Direct encoding approaches
  - Factorization-based approaches
  - Random walk approaches
- Generalized encoder-decoder architectures
  - Neighborhood auto-encoder methods
  - **Neighborhood aggregation and convolutional encoders**
- Others:

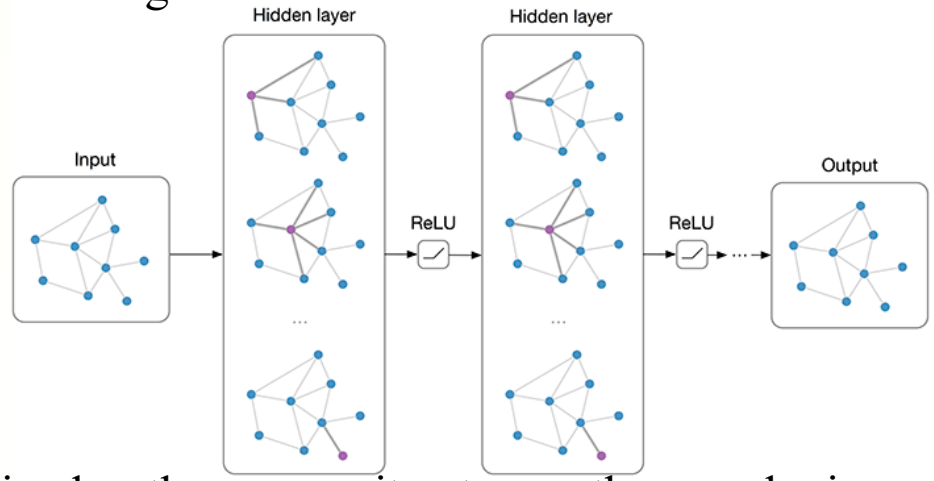
  - GraphSage, GraphWave, …

Graph Convolutional Based Approaches

- Example: Graph Convolutional Networks (GCN)
- Intuition behind these approaches is that they generate embeddings for a node by aggregating information from its local neighborhood
- Called **convolutional** because they represent a **node as a function of its surrounding neighborhood**, in a manner similar to the receptive field of a center-surround convolutional kernel in computer vision
- Can support meta-data features on nodes (profiles in social networks)
- In encoding phase, the neighborhood aggregation methods build up the representation for a node in an iterative, or recursive, fashion

Graph Convolutional Based Approaches

1. Node embeddings are initialized to be equal to the input node attributes.
2. At each iteration of the encoder algorithm, nodes aggregate the embeddings of their neighbors, using an aggregation function that operates over sets of vectors.
3. Every node is assigned a new embedding, equal to its aggregated neighborhood vector combined with its previous embedding from the last iteration.
4. Finally, this combined embedding is fed through a dense neural network layer and the process repeats.



- As each iteration, the node embeddings contain information aggregated from further and further reaches of the graph.
- However, the dimensionality of the embeddings remains constrained as the process iterates, so the encoder is forced to compress all the neighborhood information into a low dimensional vector.
- After K iterations the process terminates and the final embedding vectors are output as the node representations.

Similarity in Graphs

- Similarity between nodes
  - Structural equivalence & regular equivalence
  - Homophily
  - Graph embedding – Representation Learning
- **Graph Similarity**
  - With known node correspondence
  - With unknown node correspondence
    - With finding an alignment of graphs
    - Without finding an alignment of graphs

Simple Similarity functions

- Need to have a function to calculate similarity of graphs

- Direct structural comparison
- Ranking-Based
- Distribution-Based
- Aggregation-Based

Simple Similarity functions

- Ranking-Based Similarity
  - Calculate the centrality measure of each node

  (in/ out degree, in/out weight, betweenness, pagerank, …)
  - Compute top-k central nodes intersection as a score

Simple Similarity functions

- Distribution-Based Similarity Measures
  - Calculates the quantized distribution of whole graph on:

(in/ out degree, in/out weight, betweenness, pagerank, …)

  - Compute the distance of two graphs' distribution using:

(Euclidean, Jensen-Shannon, Hellinger distance)

Simple Similarity functions

- Aggregation-Based Similarity Measures
  - an attribute can be associated with a node or an edge. Hence, for the same attribute type x, we can construct a vector V(x,G) and compute the similarity of those attributes

Simple Similarity functions

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

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

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

VR = rank correlation of node pagerank

3. "… their edge weights are similar."

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

Simple Similarity functions

4. "… they have similar subgraphs." Maximum Common Subgraph *NP-complete*

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

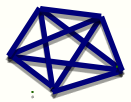
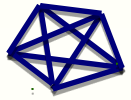
$$GED(G_A, G_B) = |V_A| + |V_B| - 2|V_A \cap V_B| + |E_A| + |E_B| - 2|E_A \cap E_B|$$

Similarity function properties

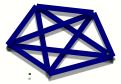
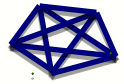
What **properties** should
a **good** similarity
function have?

Similarity function properties

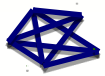
- **Identity Property**:

Similarity( , ) = 1

- **Symmetry Property**:

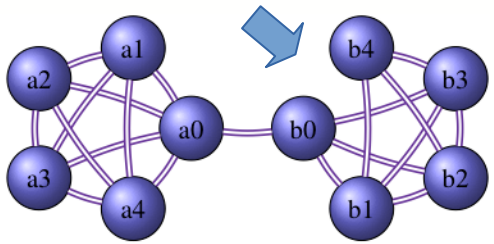
Similarity( , ) =Similarity( , )

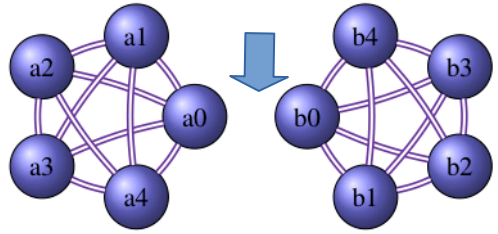
- **Zero Property**:

Similarity( , ) = 0

- *Edge Importance*

(Weight, sub-modularity, ...)





Similarity in Graphs

- Similarity between nodes
  - Structural & regular equivalence
  - Homophily concept
  - Graph embedding – Representation Learning
- **Graph Similarity**
  - **With known node correspondence**
  - With uknown node correspondence

Graph Similarity with Known Node Correspondence

- Example: DeltaCon [7]
- Intuition:
  - Find node affinity between every two nodes in graph (pairwise node influence)
  - Influence score is calculated through n hops
  - Pairwise comparison of the influence scores can give us a node level similarity
  - Aggregation of distances using Euclidean distance
- Supports identity, symmetric and zero properties
- Considers edge importance, weights, bridging edges
  - Changes that create disconnected components should be penalized more than changes that maintain the connectivity properties of the graphs.
  - A specific change is more important in a graph with few edges than in a much denser, but equally sized graph.

An Example: DeltaCon

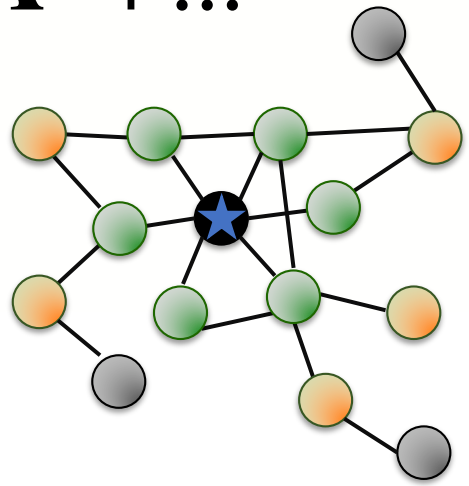
- How to find node influence?
  - Using Fast Belief Propagation Algorithm [8]
- An epsilon value is added to each node's score
- Epsilon is decreasing as we consider further nodes (lower effect)

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

1-hop 2-hops …

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

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



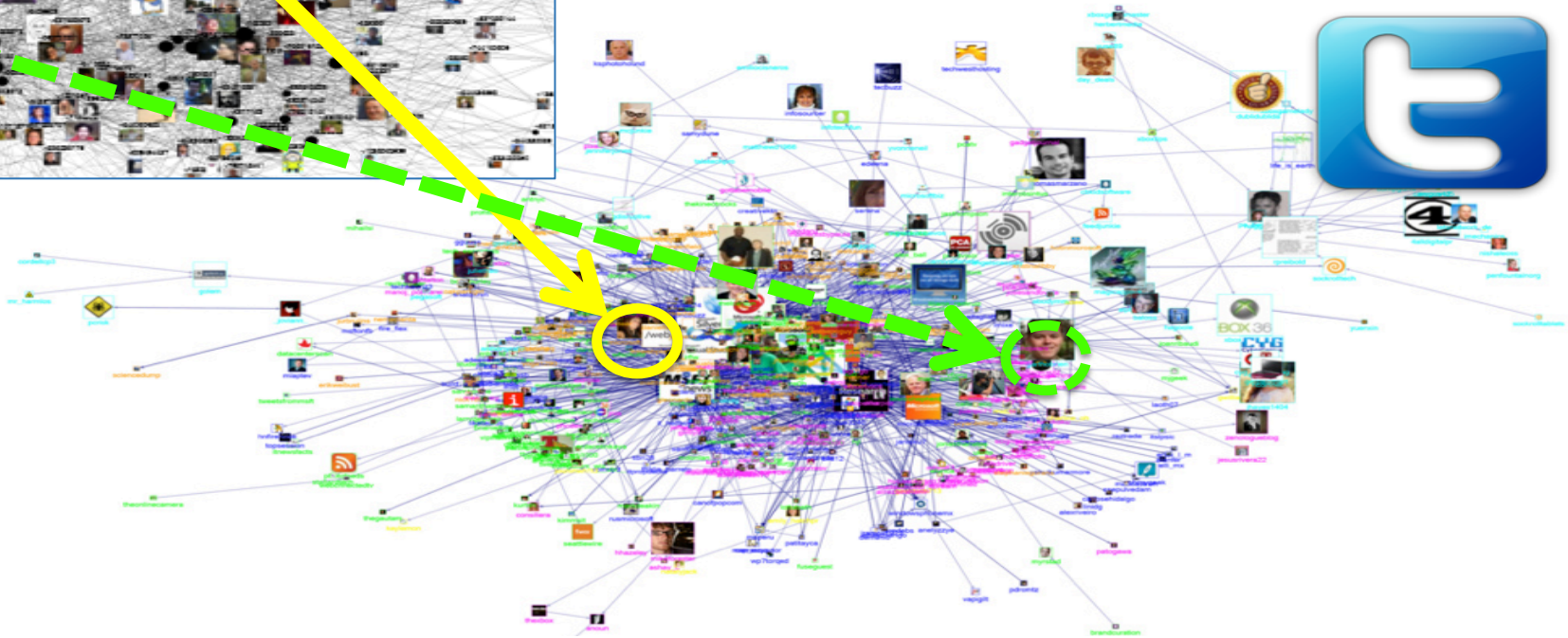
$$sim(G_A, G_B) = \frac{1}{1 + \sqrt{Euclidean\ Dist.}} = \frac{1}{1 + \sqrt{\sum_{i,j} \left(\sqrt{s_{A,ij}} - \sqrt{s_{B,ij}} \right)^2}}$$

Similarity in Graphs

- Similarity between nodes
  - Structural & regular equivalence
  - Homophily concept
  - Graph embedding – Representation Learning
- **Graph Similarity**
  - With known node correspondence
  - **With unknown node correspondence**

Can we identify users across social networks?

Same or “similar” users?



Applications

Smarter Commerce

information network

social network

transfer learning

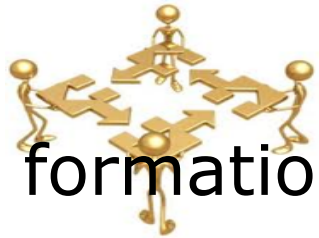
internal

external networks

better anom detection

better `INPUT` for graph similarity + graph kernel

healthcare

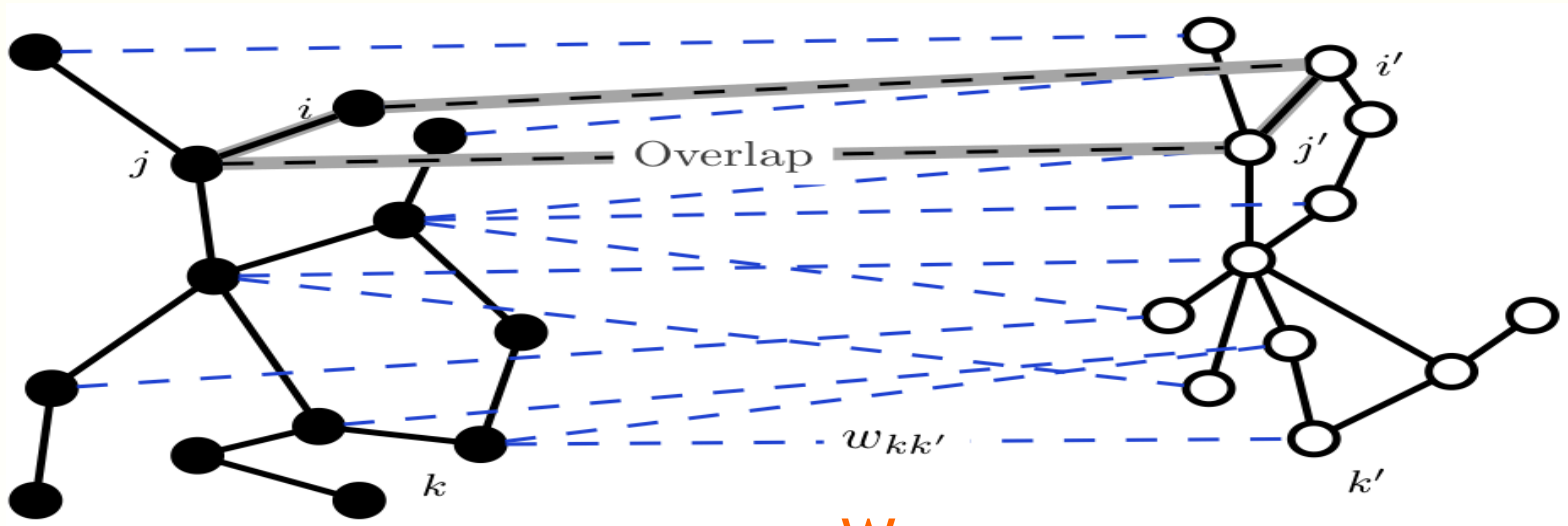


team formatio

Similarity in Graphs

- Similarity between nodes
  - Structural & regular equivalence
  - Homophily concept
  - Graph embedding – Representation Learning
- **Graph Similarity**
  - With known node correspondence
  - **With unknown node correspondence**
    - **Based on Alignment**
      - **Examples: NetAlign, Big-Align**
    - **Avoiding Alignment**
      - **Example: NetSmilie**

NetAlign



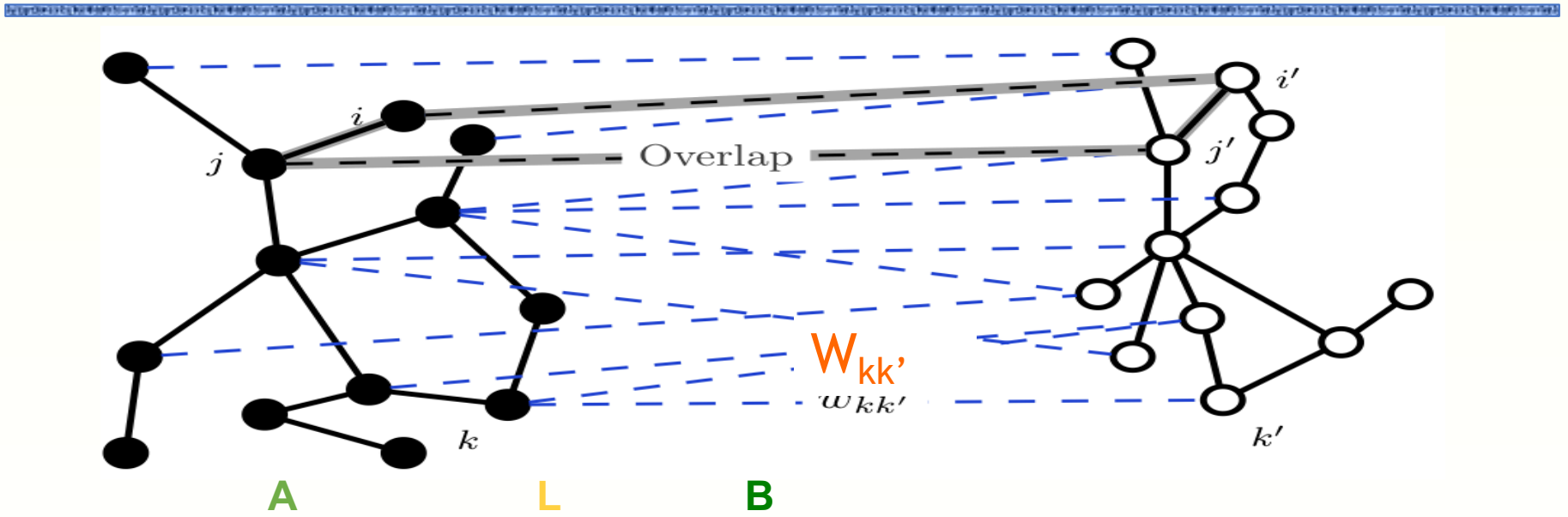
$W_{kk'}$

A L B

Possible matchings!

[Bayati+ ’11]

NetAlign



Goal: find matching **M** ⊂ **L**

s.t. it maximizes

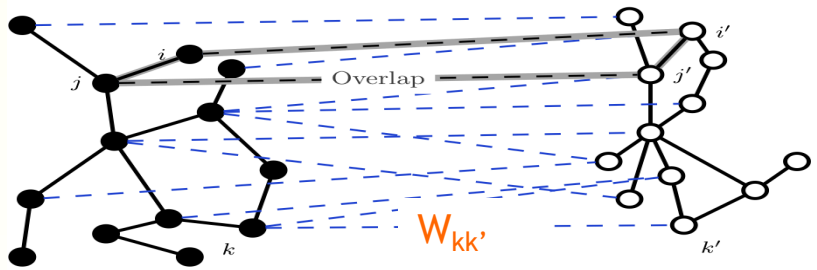
a linear combination of the weights w

& the number of overlapped edges.

NP-hard

[Bayati+ '11]

Two Solutions



Approach	Sparse L	Dense L	Speed
Belief Propagation [Bayati+'11]		✓	30% faster
Lagrangian [Klau+'09]	✓		

Bipartite Graph Alignment: Big-Align

- Given two bipartite graphs, G_A and G_B, with adjacency matrices A and B, find the correspondence permutation matrices P, Q that minimize the cost function f:

$$\min_{\mathbf{P},\mathbf{Q}} f(\mathbf{P},\mathbf{Q}) = \min_{\mathbf{P},\mathbf{Q}} ||\mathbf{PAQ} - \mathbf{B}||_F^2$$

- Each matrix element is the probability of an alignment
- Matrices should be sparse!
- A and B can have different sizes

- Using Alternating Projected Gradient Descent to solve the optimization problem

Main Idea

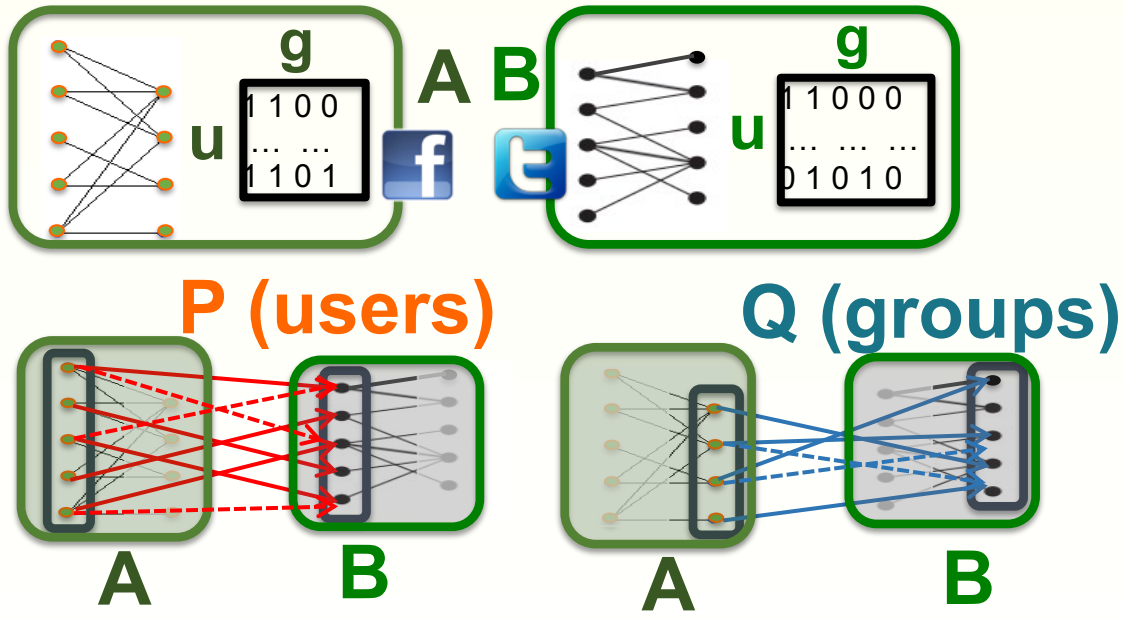
INPUT:

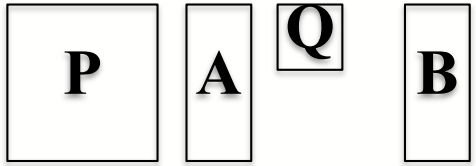
A, B

OUTPUT:

P, Q

correspondence matrices

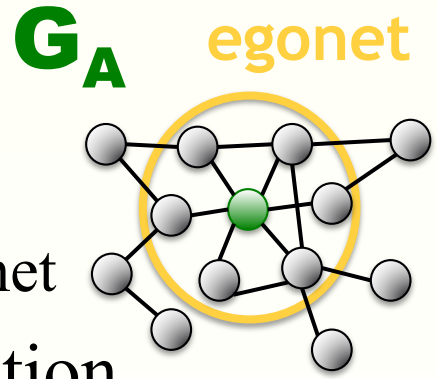




Avoiding Alignment, An example: NetSimile

- It gives similarity scores that are size-invariant
- It is scalable, being linear in the number of edges for “signature” vector extraction
- It does not need to solve the node-correspondence problem (NP)
- A small number of numerical features, which capture the topology of the graph as moments of distributions over its structural features
- Similarity of graphs => similarity of their “signature” feature vectors

An Example: NetSimile



- **Feature extraction**: a set of structural features for each node based on its local and egonet-based features
  - Degree, clustering coefficient, number of 2hop away neighbors, ego-net
- **Feature aggregation:** aggregate median, mean, standard deviation, skewness and kurtosis of each feature for the whole graph

'signature' vector P =

- **Comparison, similarity score:** Canberra Distance (sensitive to small changes near zero)

$$d(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{n} \frac{|p_i - q_i|}{|p_i| + |q_i|}$$

Related papers and a tutorial

1. **Graph Factorization:** Ahmed, Amr, et al. "Distributed large-scale natural graph factorization." Proceedings of the 22nd international conference on World Wide Web. ACM, 2013.
2. **GraRep:** Cao, Shaosheng, Wei Lu, and Qiongkai Xu. "Grarep: Learning graph representations with global structural information." Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM, 2015.
3. **Hope:** M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In KDD, 2016
4. **Graph Convolutional Network**: Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).
5. **Node2vec**: Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016.
6. **Deepwalk:** Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.
7. **Deltacon**: Koutra, Danai, Joshua T. Vogelstein, and Christos Faloutsos. "Deltacon: A principled massive-graph similarity function." *Proceedings of the 2013 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 2013.
8. **FaBP**: Koutra, Danai, et al. "Unifying guilt-by-association approaches: Theorems and fast algorithms." *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, Berlin, Heidelberg, 2011.
9. **Netsimile**: Berlingerio, Michele, et al. "Netsimile: A scalable approach to size-independent network similarity." *arXiv preprint arXiv:1209.2684* (2012).
10. **Big-align**: Koutra, Danai, Hanghang Tong, and David Lubensky. "Big-align: Fast bipartite graph alignment." *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 2013.
11. **Graph2vec**: Narayanan, Annamalai, et al. "graph2vec: Learning Distributed Representations of Graphs." *arXiv preprint arXiv:1707.05005* (2017).
12. **A good Tutorial**: http://web.eecs.umich.edu/~dkoutra/tut/icdm14.html