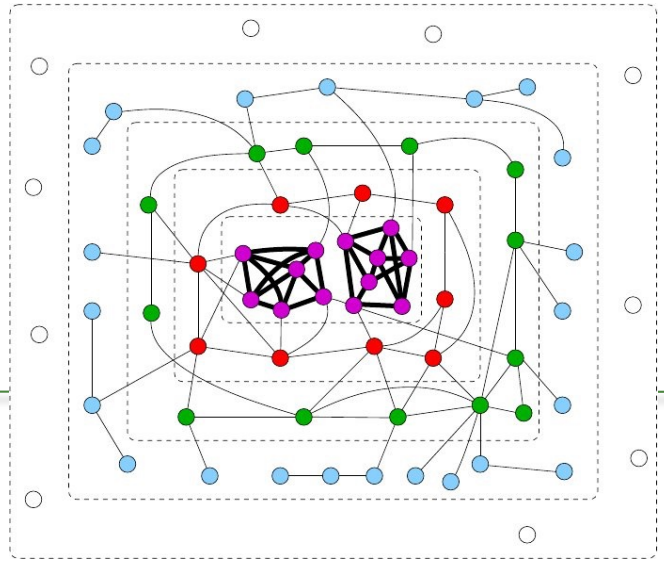
Graph Mining and ML an introduction



Michalis Vazirgiannis

LIX @ Ecole Polytechnique

Data Science and Mining Team (DASCIM),

LIX ÉcolePolytechnique http://www.lix.polytechnique.fr/dascim

Google Scholar: https://bit.ly/2rwmvQU

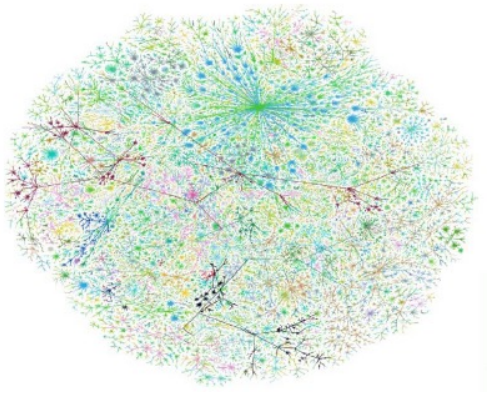
Twitter: @mvazirg

January 2025

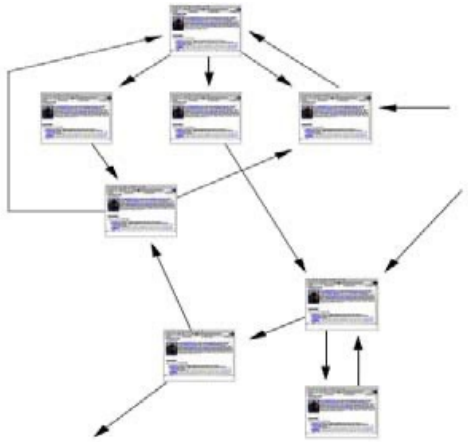
Learning for Graphs

1. **Introduction & Motivation**
2. Graph Generators
3. Unsupervised learning
    1. Community detection
4. Graph Kernels and node embeddings

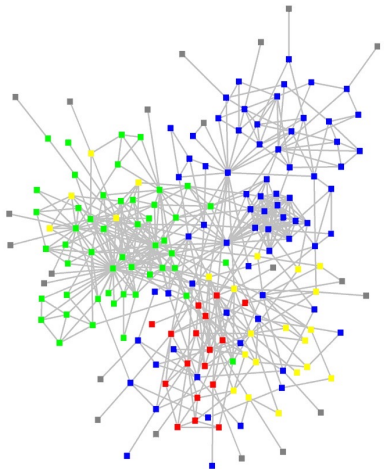
Networks are Everywhere



Internet

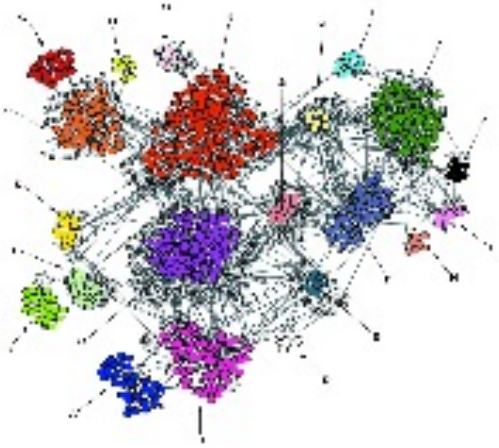


World Wide Web



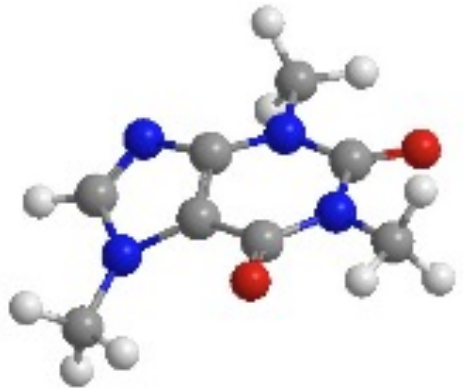
Email network

Social network



Magwene *et al. Genome Biology* 2004 **5**:R100

Co-expression network



Chemical network

Social Networking Data

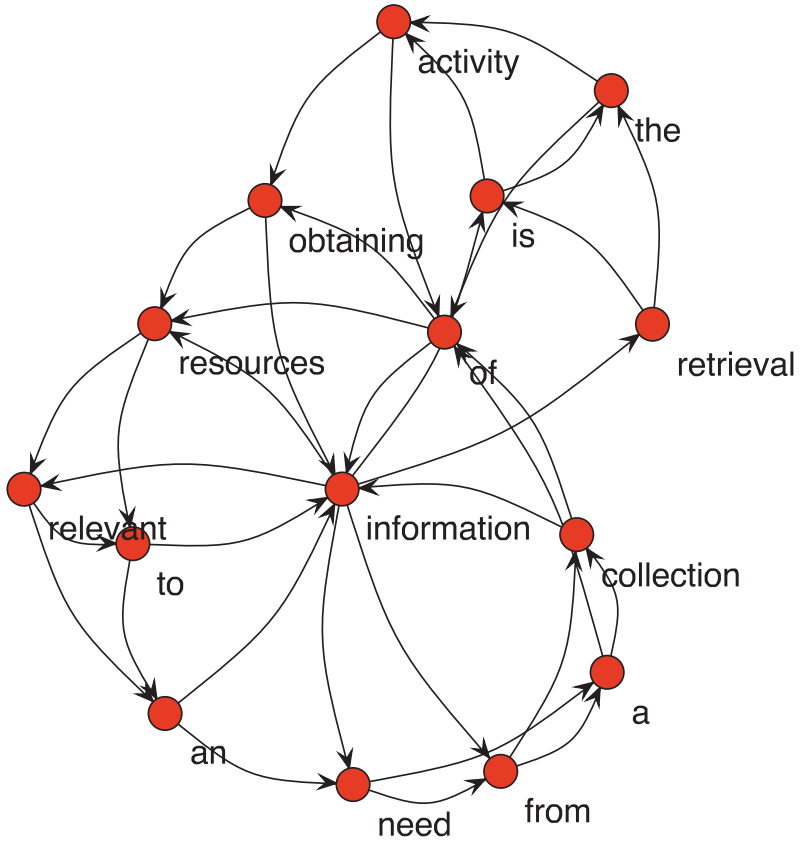
- Online social networks and social media
- Easily accessible network data at **large scale**
- Opportunity to scale up observations
- Large amounts of data raise new questions

Even representing text - Graph-of-word

information retrieval is the activity of obtaining

information resources relevant to an information need

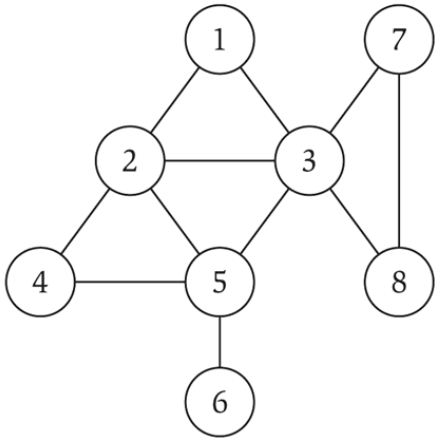
from a collection of information resources



"Graph of word approach for ad-hoc information retrieval", F. Rousseau, M. Vazirgiannis, Best paper mention award ACM CIKM 2013

Graph Representation: Adjacency Matrix

- A graph can be represented by the adjacency matrix **W**
  - Matrix of size **n x n**, where **n** is the number of nodes
  - $W_{ij} > 0$, if **i** and **j** are connected
  - $W_{ij} = 0$, if **i** and **j** are not connected
  - In case of unweighted graphs, $W_{ij} = 1$, if **(i, j)** is an edge of the graph
  - Space proportional to n^2



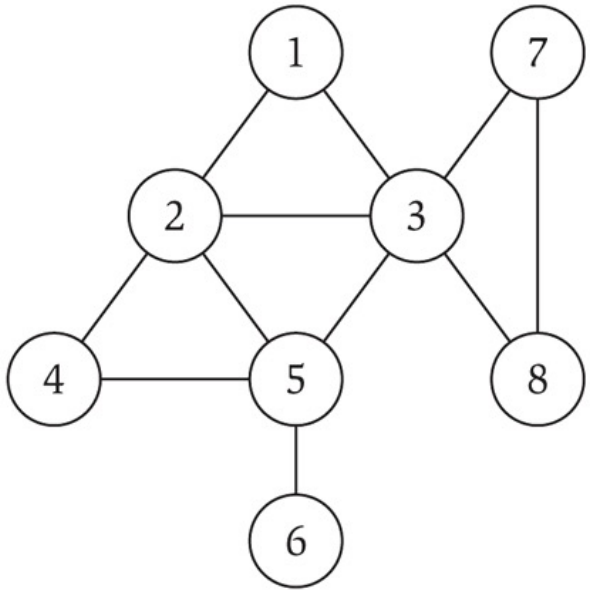
Undirected graph

0	1	1	0	0	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	1	1
0	1	0	0	1	0	0	0
0	1	1	1	0	1	0	0
0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1
0	0	1	0	0	0	1	0

Adjacency matrix

Cycles in Graphs

- **Definition:** A cycle is a path $v_1, v_2, \ldots, v_k$ in which $v_1 = v_k$, $k > 2$ and the first **k-1** nodes are all distinct



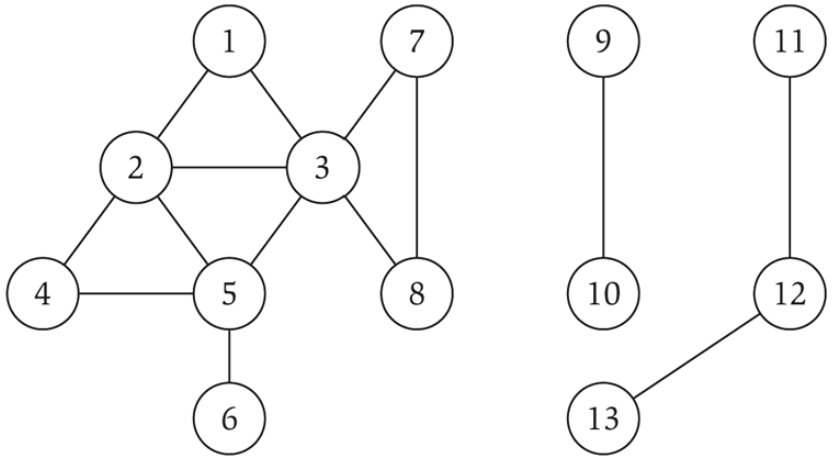
Cycle C = 1 – 2 – 4 – 5 – 3 – 1

Cycle issues:

1.Infinite Loops
2.Ambiguity in Path Definitions: in a cyclic graph, there might be multiple paths between two vertices, and determining the "shortest path" or "simplest path" becomes non-trivial.
3.Unintended Behavior in Graph Algorithms: graph algorithms applied to cyclic graphs, these algorithms might exhibit unexpected behavior or fail to terminate (i.e. topological sorting - **Complexity**
4.Deadlock in Resource Allocation: a set of processes are blocked because each is holding a resource and waiting for another resource acquired by some other process in the cycle.

Connected Components

- A **connected component** is a maximal connected subgraph of a graph **G** (there is a path between any pair of nodes)



Graph with 3 connected components

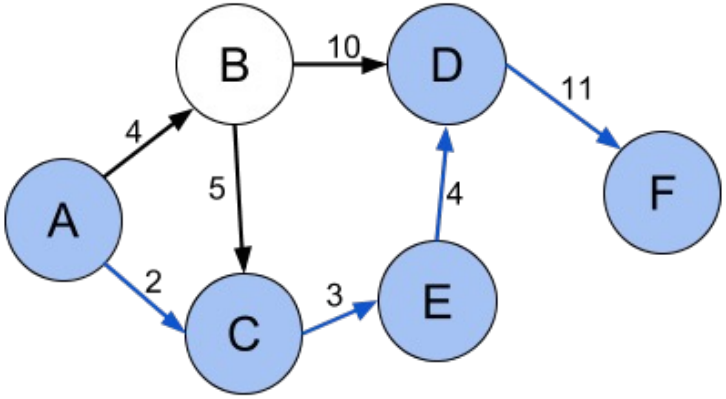
Connected component containing node 1:
{1, 2, 3, 4, 5, 6, 7, 8}

Question: How can we compute the connected components of a graph?

A: Apply BFS - complexity O(|V| + |E|)

Shortest Paths

- **Definition:** find a path between two nodes in a graph, in such a way that the sum of the weights of its constituent edges is minimized
  - Many applications (e.g., road networks)
  - **Single-source** shortest path problem
  - **Single-destination** shortest path problem
  - **All-pairs** shortest path problem

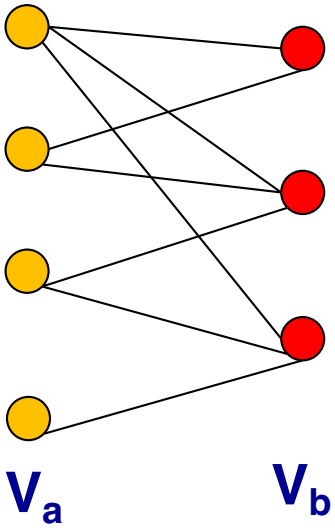


Shortest path (A, C, E, D, F) between vertices A and F in the weighted directed graph

Many algorithms:
- Dijkstra
- Bellman-Ford

Bipartite Graphs

- **Definition:** A graph **G=(V,E)** is called **bipartite** if the node set **V** can be partitioned into two disjoint sets V_a, V_b and every edge **(u,v)** connects a node of V_a to a node of V_b



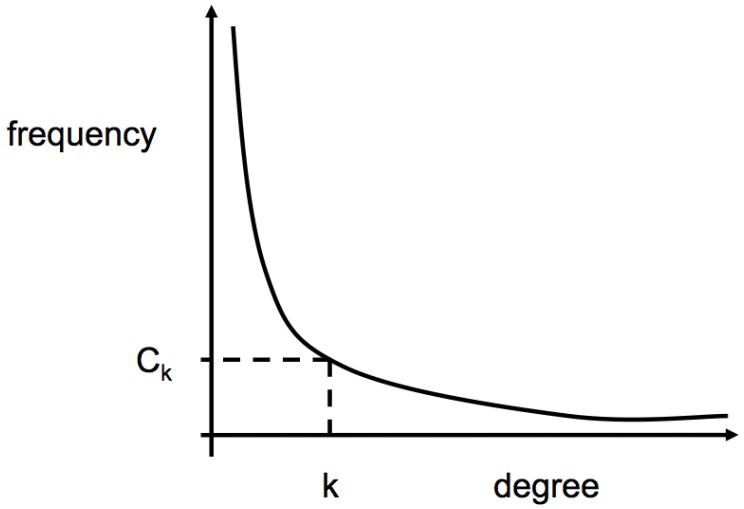
- Strong modeling capabilities and many real-world applications
- E.g., **Collaborative filtering** in recommender systems
  - Model the customer-product space using a bipartite graph (who-purchased-what)
  - If a user A has purchased the same product with a user B, then it is more likely to purchase another product as B did, than of a person selected randomly

Properties of Real-World Graph

- Networks arising from **real-world** applications obey fascinating properties
- **Static networks**
  - Heavy-tailed degree distribution
  - Small diameter
  - Giant connected component (GCC)
  - Triangle Power Law
  - Community structure
  - …
- **Dynamic networks**
  - Densification
  - Small and shrinking diameter

Degree Powel Law Distributions

- The **probability distribution** of the degrees over the network

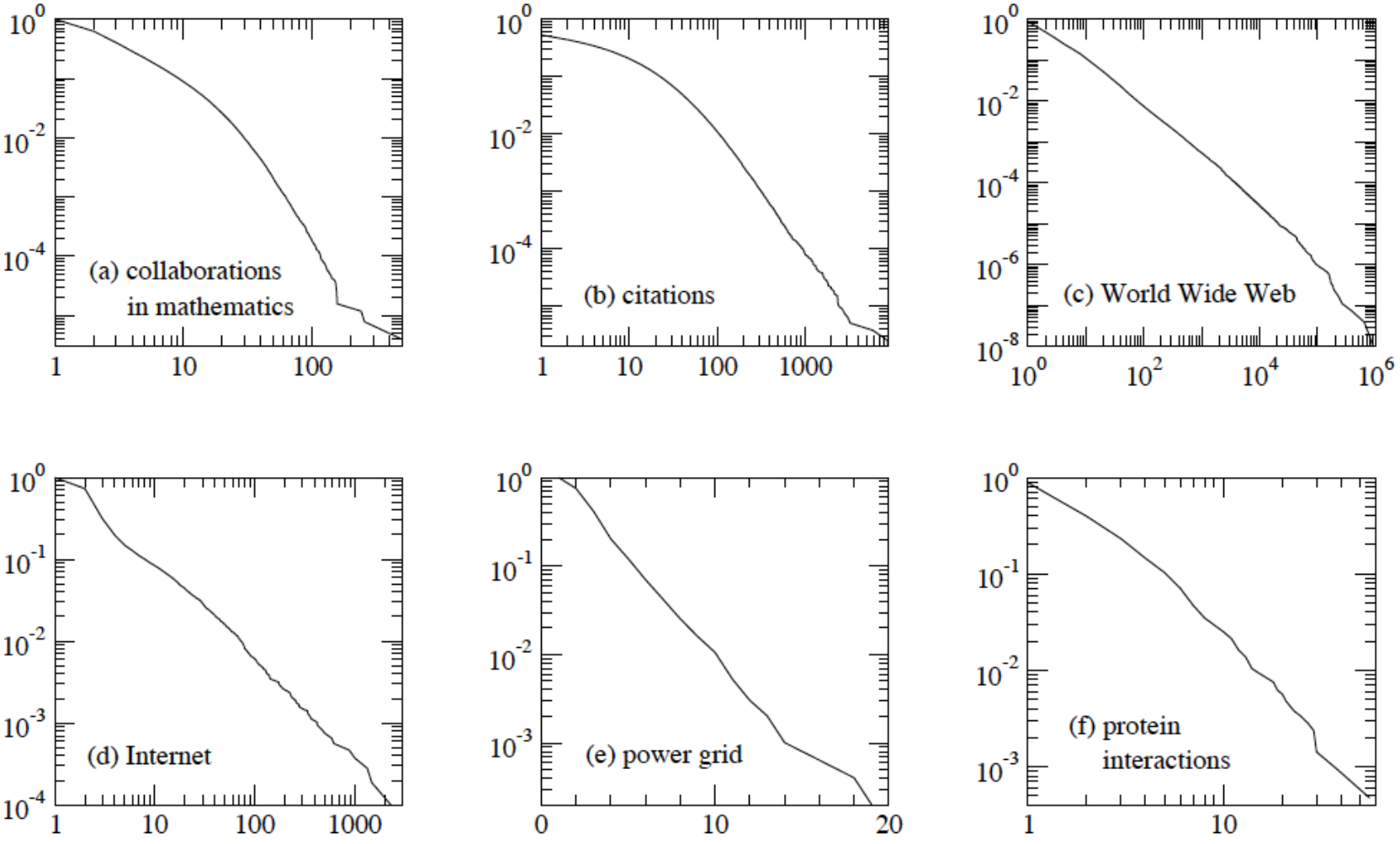


- **Problem:** find the probability distribution the **fits** best the **observed data**
- C_k = #of nodes with degree **k**
- $C_k = c\ k^{-\gamma}$ - $\gamma > 1$ and **c** a constant
- How to recognize a power-law distribution?

$$\ln C_k = \ln c - \gamma \ln k$$

  - Plotting $\ln C_k$ versus $\ln k$ gives a straight line with slope $-\gamma \ln k$

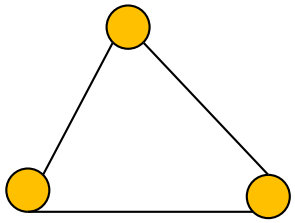
Power-law Degree Distribution in Real-Networks (2/2)



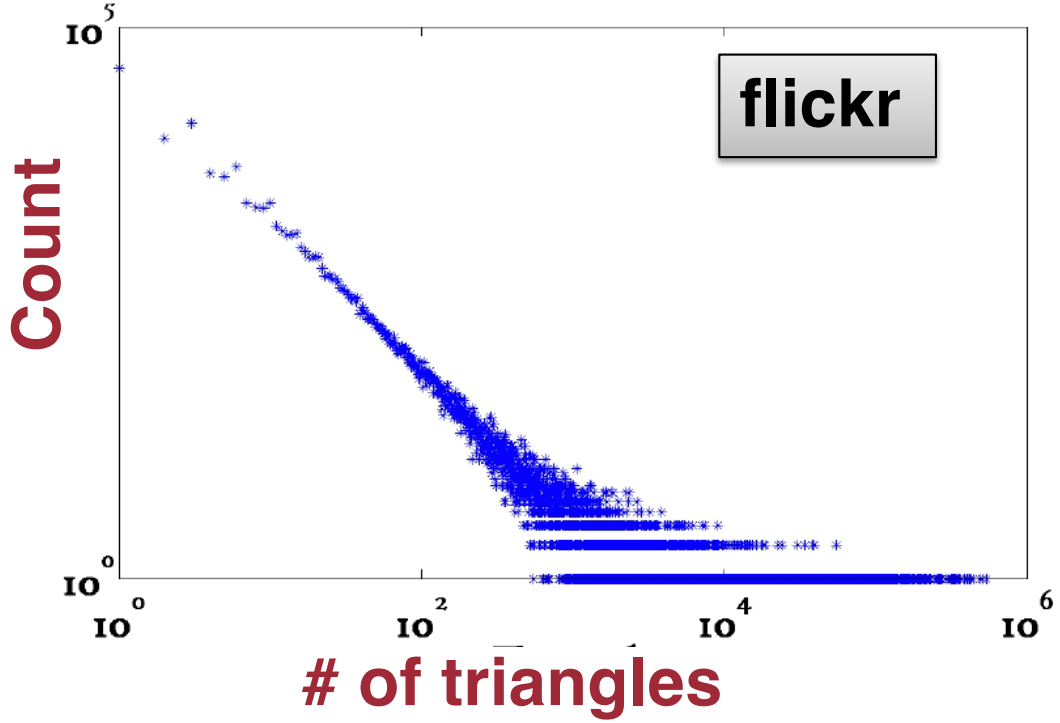
Cumulative degree distribution for six different networks [Newman 2003]

18

Triangle Participation Distribution



Complete graph with 3 nodes: triangle



- Number of nodes that participate in **k** triangles vs. **k** in log-log scale
- **Heavy-tailed** distribution

[Tsourakakis, 2008]

Clustering Coefficient

- Captures the tendency of the nodes of a graph to cluster together

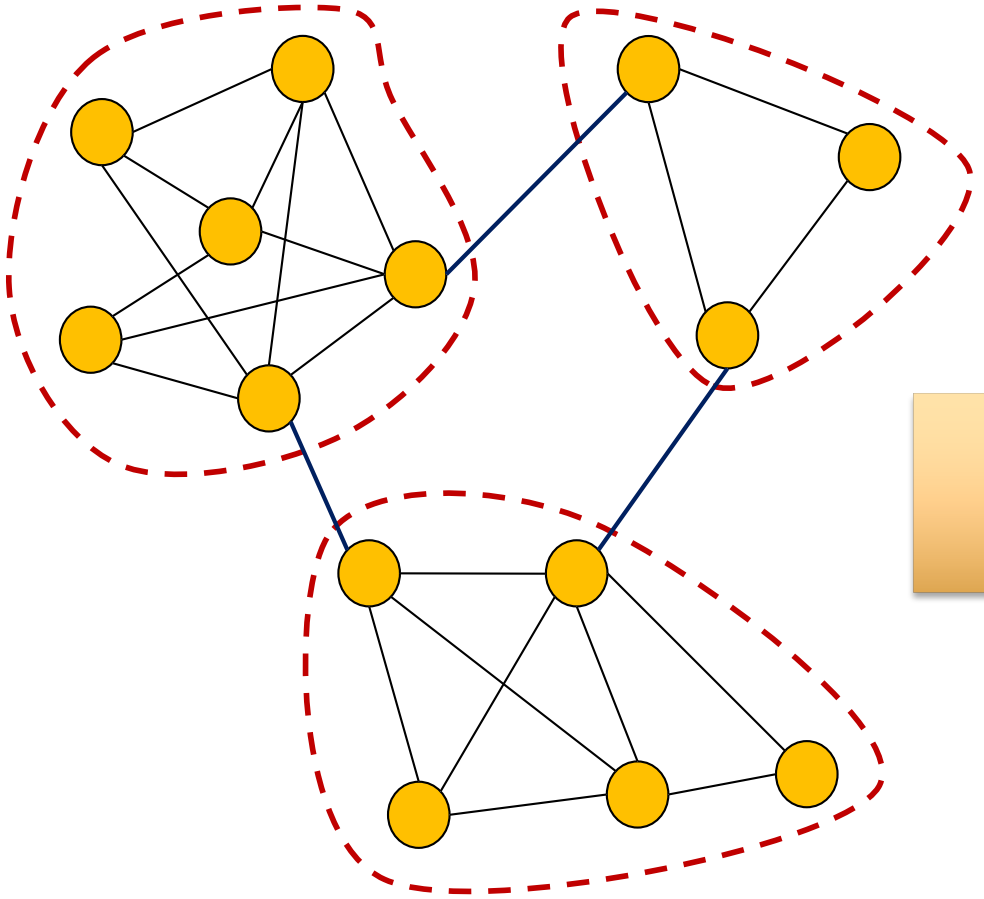
  T(G) = 3 x # of triangles in G / # of connected triplets

- Captures the transitivity of clustering
  - If **u** is connected to **v** and **v** is connected to **w** …
  - ... it is likely that **u** is also connected to **w**

- Real-world networks tend to have high clustering coefficient
  - Connections to the existence of clustering and community structure property

[Frieze, Gionis, Tsourakakis, 2013]

Community Structure

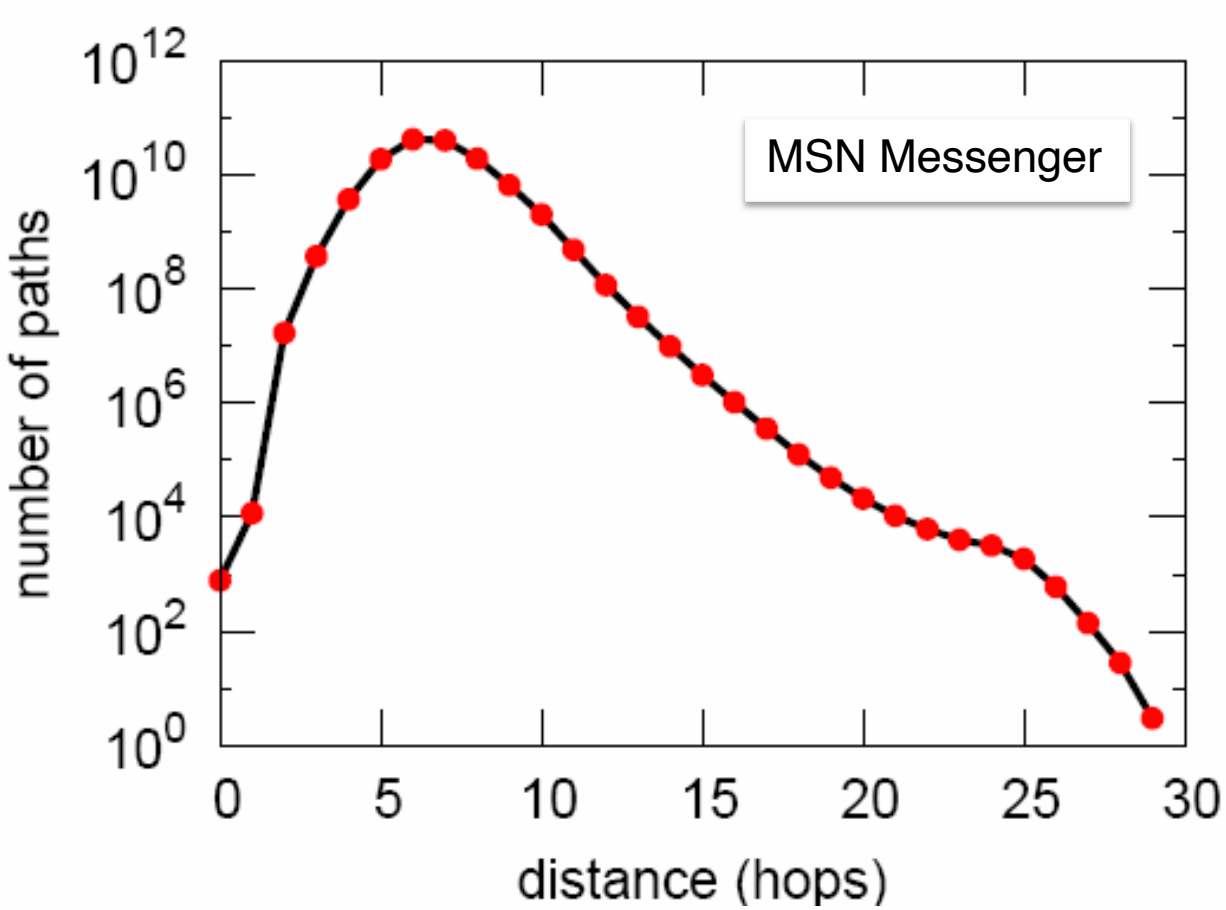


Example graph with three communities

- Will be covered later on in detail

Small-world Phenomenon (1/2)

- The small-world phenomenon appears in various network settings

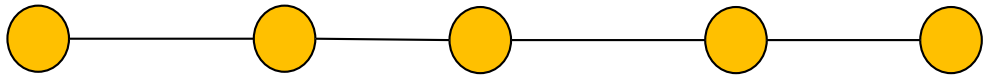


- **Average path length** is **6.6**
- 90% of the nodes are reachable in less than 8 steps
- **Facebook** network:
  - Average distance is **4.7**
  - **[Ugander et al., 2011]**

[Leskovec, 2009]

Small Diameter

- **Diameter** is the largest shortest path in the graph
  - Diameter is often sensitive to **chains** of nodes



- In practice, we use the **effective diameter**
  - Upper bound of the shortest path over 90% of the pairs of nodes

- As an effect of the small-world phenomenon, real networks have **small diameter**

[Leskovec, 2009]

Learning for Graphs

- Introduction & Motivation
- **Graph Generators**
- Unsupervised learning
  - Community detection
- Graph Kernels and node embeddings

Graph generating models

Goal: Characterize, model and understand the structure of real networks

- How do real-world networks look like?
  1. **Empirical: statistical properties of networks** (e.g., degree distribution, diameter)
  2. **Generative models of network structure**
     - Mechanisms that reproduce the underlying generative processes

Graph generating models

- Creating models for real-world graphs is important for several reasons
  - Help us to **understand** and **reason** about the observed properties
  - Create **artificial data** for simulation purposes
  - **Predict** the evolution of networks
  - **Privacy preservation:** release the parameters of the generative model, instead of the network itself

Erdős–Rényi Random Graph Model

- Suppose that we want to generate a network with **n** nodes
- The $G_{n,\,p}$ model:
  - Graph with **n** nodes and edge probability **p**
  - For each pair of nodes **(u, v)**, add the edge **(u, v)** **independently** with probability **p**
  - Family of graphs, in which a graph with **m** edges appears with probability

$$p^m(1-p)^{\binom{n}{2}-m}$$

Degree Distribution of the ER Model (1/2)

- **Q:** Do Erdős–Rényi graphs look **realistic**?
- The degree distribution is **Binomial**
  - Let C_k denote the number of nodes with degree **k**

$$C_k = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

- What if **n → infinity** and we fix the expected degree = **c**?

$$\text{If } n \to \infty \text{ and } np \to c \text{ (with } c > 0\text{) then}$$

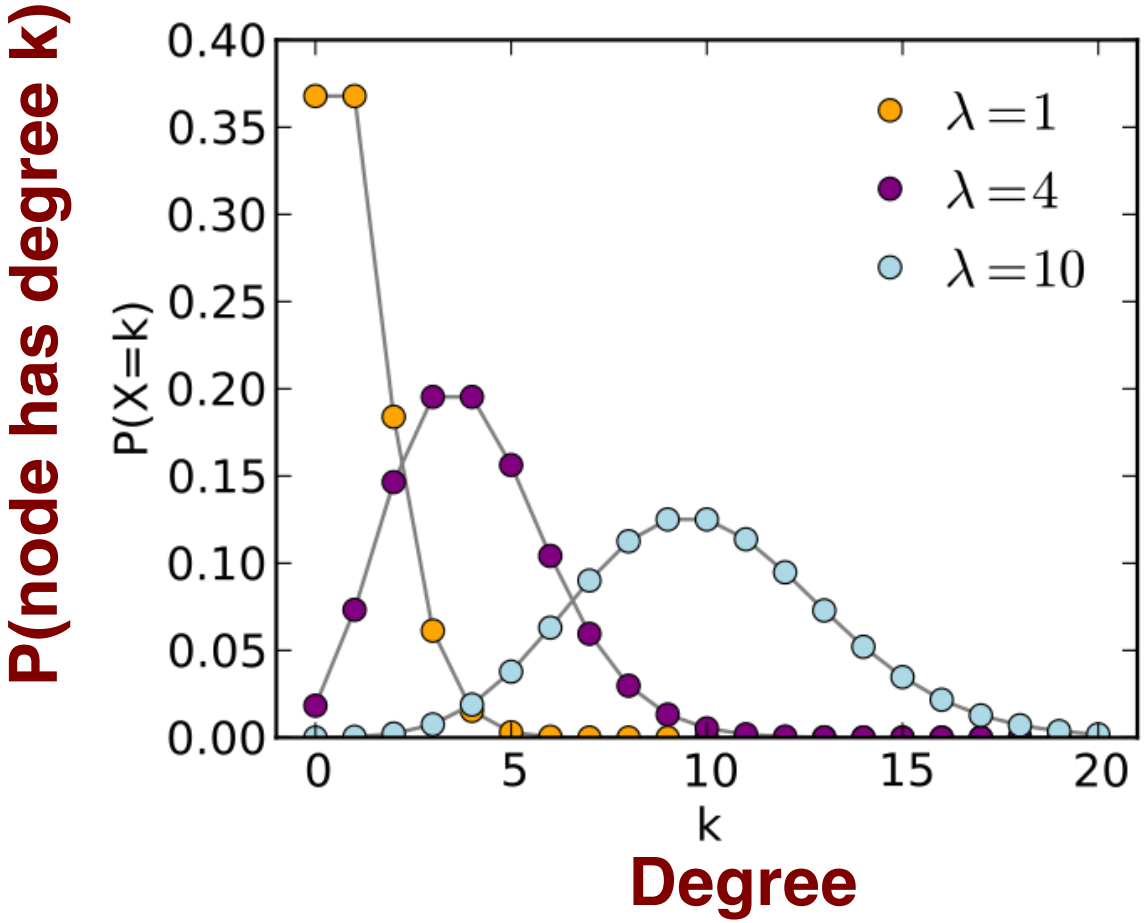
$$\frac{n!}{(n-k)!k!} p^k (1-p)^{n-k} \to e^{-c} \frac{c^k}{k!}$$

Poisson distribution

Degree Distribution of the ER Model (2/2)

Poisson distribution

$$\frac{\lambda^k e^{-\lambda}}{k!}$$



The degree distribution of ER random graph model is **not realistic** for real-world graphs

Source: Wikipedia

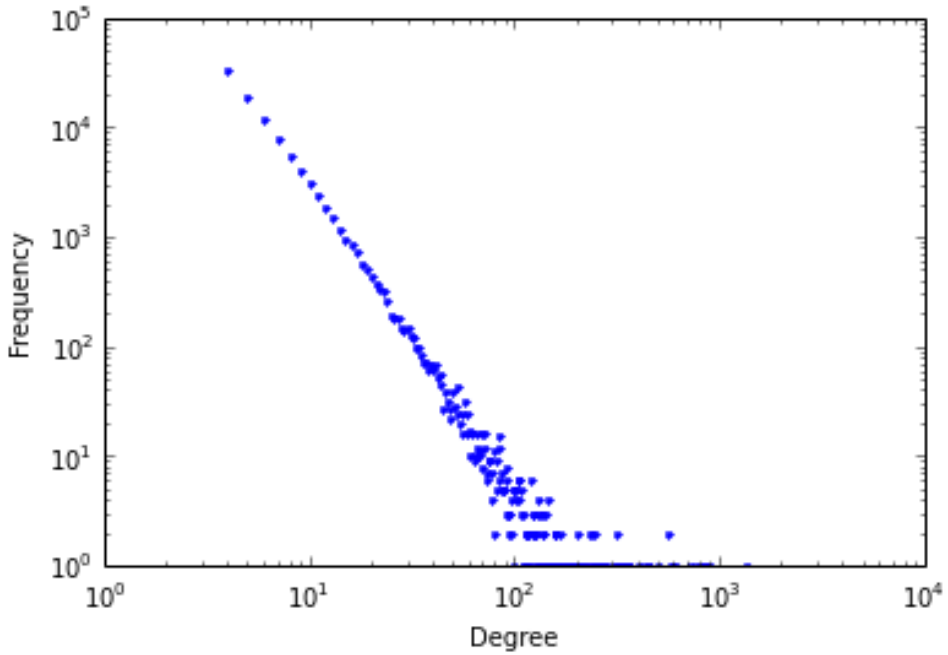
Preferential Attachment - Barabasi-Albert (1/2)

- The **Barabasi-Albert** model:
  - **Input:** subgraph **G_0** , **m**: degre / new node
  - The process:
    - The nodes are created - one at the time
    - Each new node connects to **m** existing nodes selected with probability proportional to their degree
    - Let **[d1, d2, …, dt]** : degree sequence at time **t**.
      - node at **t+1** will be connected to node **i** with probability

$$p_i = \frac{d_i}{\sum_i d_i}$$

Preferential Attachment - Barabasi-Albert (2/2)

- known as the **rich get richer** effect
  - E.g., a web page that already has many incoming hyperlinks is likely to get more in the future
- The BA model produces graphs with **power-law** degree distribution $C_k = k^{-\gamma}$, where $\gamma = 3$



- Barabasi-Albert graph
- **n = 100,000** nodes
- **m = 4**

The BA model holds for several real-world networks (flickr, Delicious, LinkedIn) [Leskovec et al., 2008]

Network Models and Temporal Evolution

- Most of the existing models (e.g., BA) consider that
  - The **number of edges** grows **linearly** with respect to the number of nodes
  - The **diameter increases** based on a factor of **log n** or **log log n**
- In real networks we have observed
  - **Densification power law**
  - **Shrinking diameter**

Kronecker Model of Graphs (1/4)

- Reminder: **Kronecker product** of matrices
  - **A = [a_{ij}]** an **n x m** matrix
  - **B = [b_{ij}]** an **p x q** matrix
  - Then **C = A ⊠ B** is defined as the **np x mq** matrix

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} \begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \cdots & a_{1,m}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \cdots & a_{2,m}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & a_{n,2}\mathbf{B} & \cdots & a_{n,m}\mathbf{B} \end{pmatrix}$$

- **Intuition:** repeat the Kronecker product between the adjacency matrix of an initial graph to get the final graph

[Leskovec et al., 2010]

Kronecker Model of Graphs (2/4)

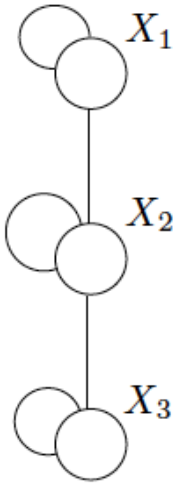
- **Kronecker** model:
  - Start by an initiator adjacency matrix $\mathbf{A_1}$ of size **p x p**
  - The Kronecker product of two graphs is defined as the Kronecker product of their adjacency matrices
  - The Kronecker graph after **k** iterations is defined as the graph with the following adjacency matrix

$$\mathbf{A}_k = \underbrace{\mathbf{A}_1 \otimes \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_1}_{k \text{ iterations}} = \mathbf{A}_{k-1} \otimes \mathbf{A}_1$$

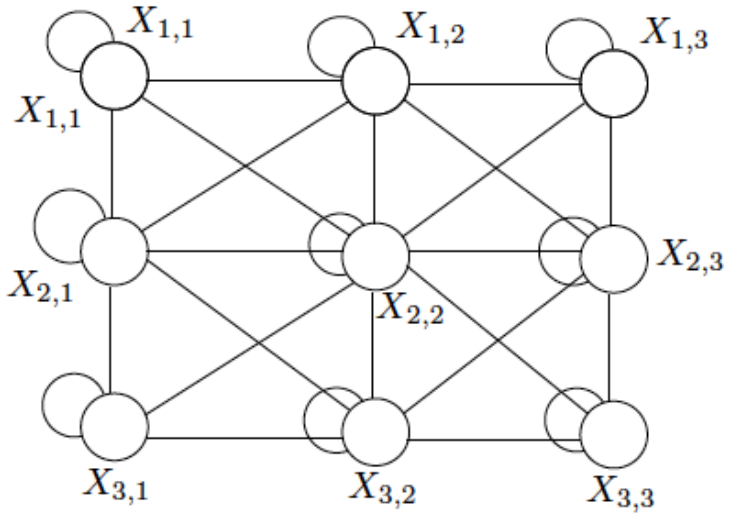
  - Each Kronecker multiplication exponentially increases the size of the graph

[Leskovec et al., 2010]

Kronecker Model of Graphs (3/4)



Graph G_1



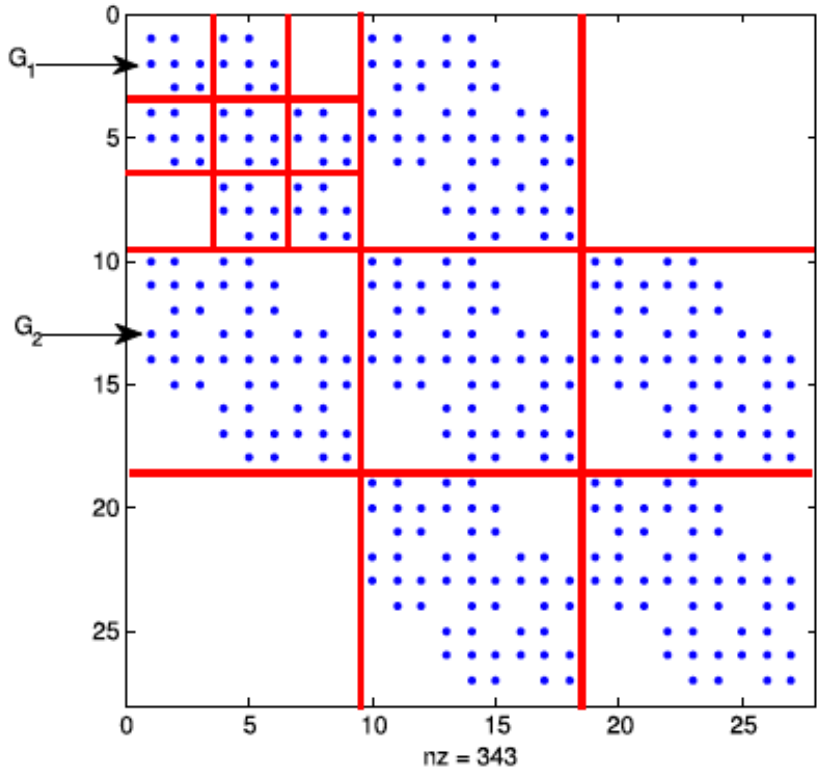
Graph $G_2 = G_1 \boxtimes G_1$

1	1	0
1	1	1
0	1	1

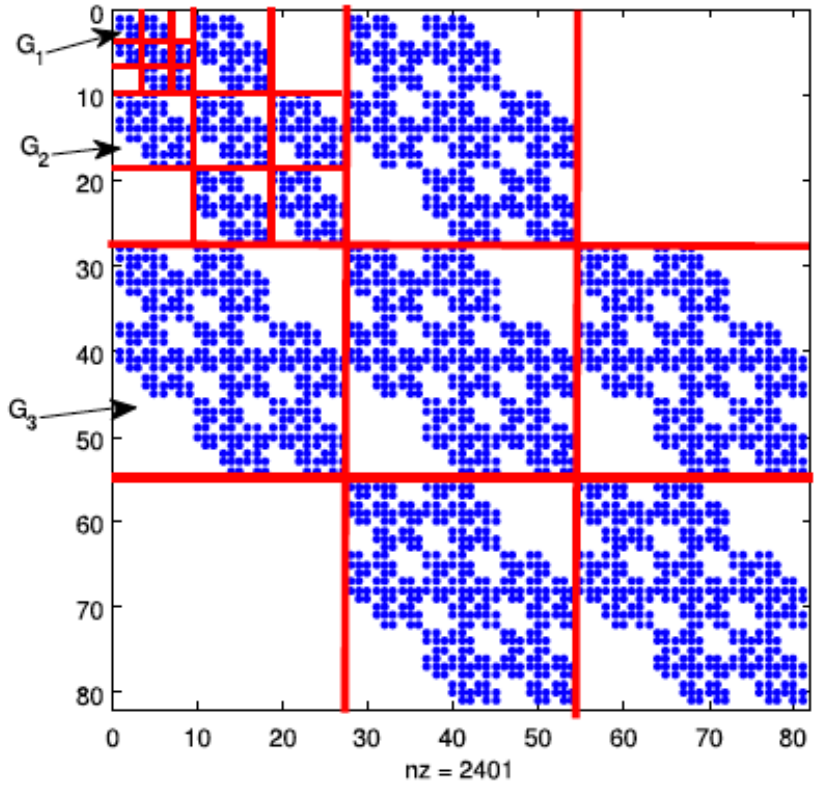
G_1	G_1	0
G_1	G_1	G_1
0	G_1	G_1

[Leskovec et al., 2010]

Kronecker Model of Graphs (4/4)



$(\alpha)\ A(G_3) = A(G_2) \otimes A(G_1)$



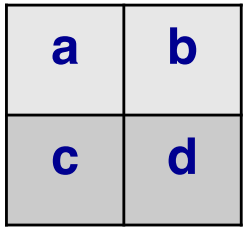
$(\beta)\ A(G_4) = A(G_3) \otimes A(G_1)$

Intuition: Recursion and self-similarity

[Leskovec et al., 2010]

Stochastic Kronecker Model

- In practice, the **stochastic Kronecker graph** is used
  - Start by an initiator matrix **θ**



  - We obtain a graph with $\mathbf{n = 2^k}$ nodes by repeating **k** times the Kronecker product: $\mathbf{A_{k,\theta} = \theta \boxtimes \ldots \boxtimes \theta}$
  - Consider the value **(i, j)** of the matrix $\mathbf{A_{k,\theta}}$ as the probability of existence of the edge **(i, j)** (applying randomized rounding)
  - Typically, **2 x 2** initiator matrices produce good results

[Leskovec et al., 2010]

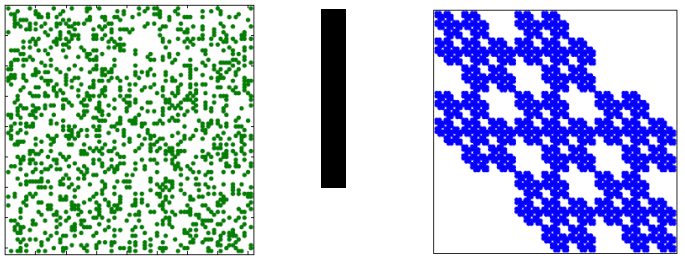
Generate Realistic Kronecker Graphs

- Given a network **G**, how can we find a "good" initiator matrix **θ**, such that $\mathbf{A_G \sim= \theta \boxtimes \ldots \boxtimes \theta}$?
  - Fit the parameters **θ** of the model
  - Idea: use **maximum-likelihood estimation**

$$\arg\max_{\Theta} P(G|\Theta)$$

After Kronecker products

$$\arg\max_{G_1}\left(\quad \boxtimes \quad \right)$$



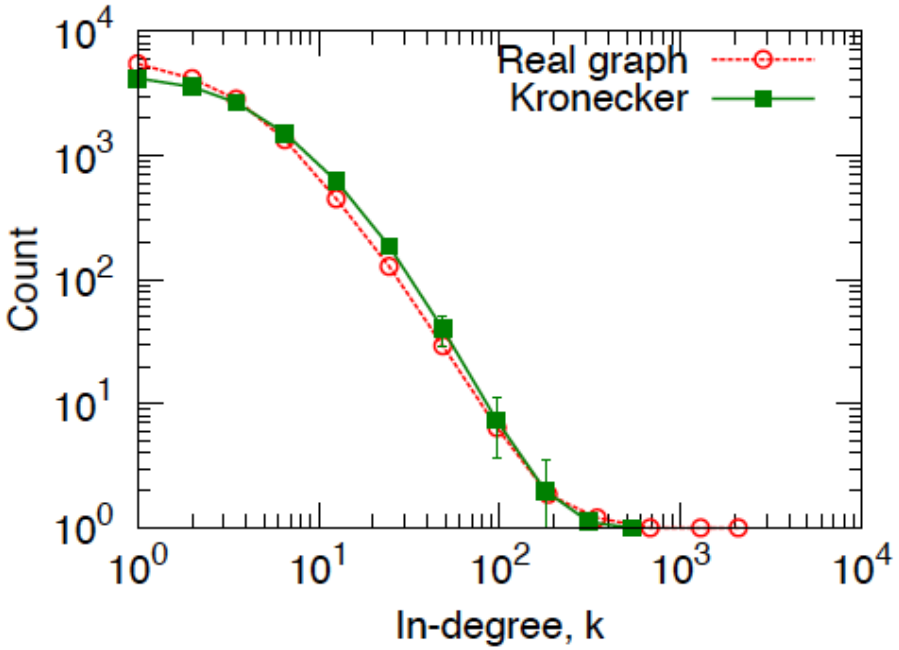
[Leskovec et al., 2010]

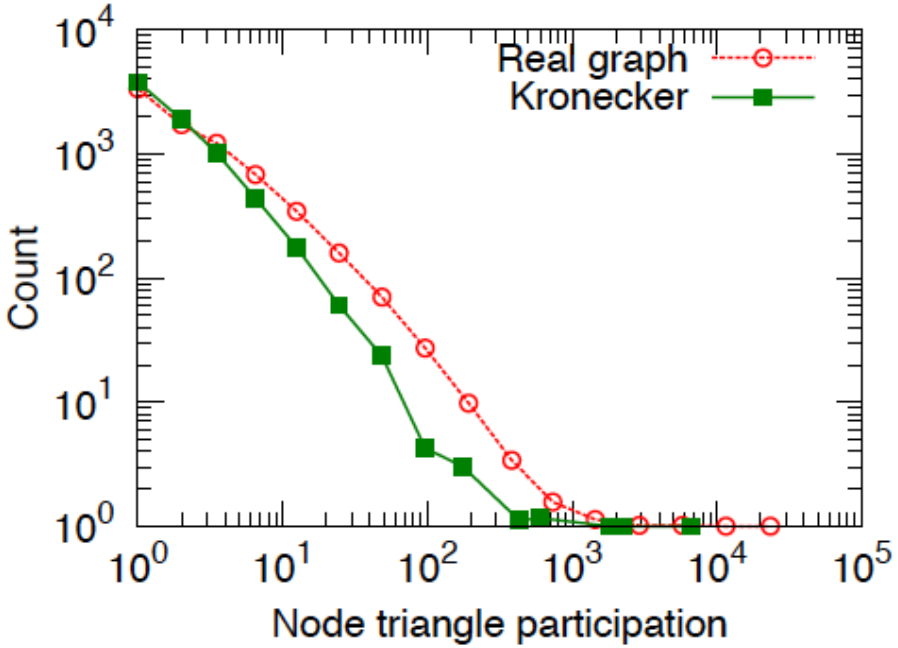
Properties of Kronecker Model

- The Kronecker (stochastic) graph model is able to reproduce a plethora of properties
  - Power-law degree distribution
  - Small diameter
  - Shrinking diameter
  - Densification power-law
  - Triangle participation
  - …

[Leskovec et al., 2010]

Example: Fitting Kronecker Model to a Graph



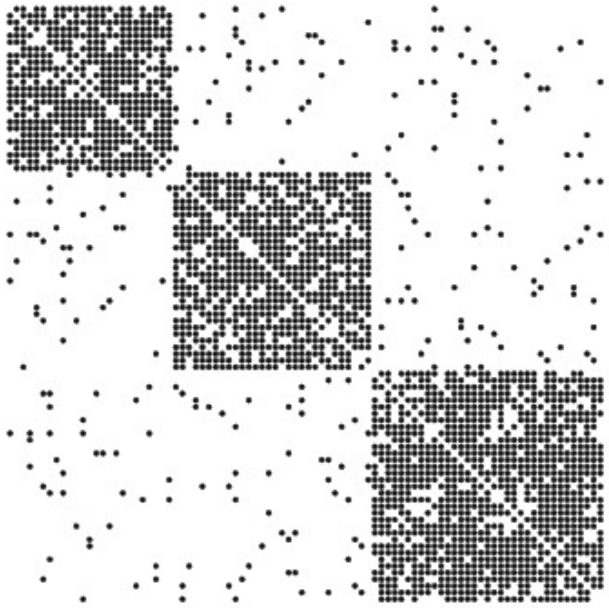


Blog-to-Blog network

[Leskovec et al., 2010]

Stochastic Block Models (1/3)

- Graphs with nodes belonging to unobserved groups that describe the connectivity
- Adjacency Matrix - Block Stochastic Matrix



- Group membership of the nodes
- Conditional probability of edge occurrence between two nodes (conditioned on group memberships).

[Lee and Wilkinson, 2019]

Stochastic Block Models (2/3)

Mathematically given,

- N nodes, K groups
- $Z \in M^{N \times K}$ group membership
- $Z_i = (0,0, \dots, 1, \dots 0)$
- $C \in M^{K \times K}$ with $C_{ij} \in (0,1)$ the probability of an edge exist between a node in group i and a node in group j

Then

- $A \in M^{N \times N}$ is the adjacency matrix with

$$A_{ij} \sim Bernoulli(\mathrm{Z}_i C Z_j)$$

[Lee and Wilkinson, 2019]

Stochastic Block Models (3/3)

- Controllable graph structure generation. C
  - Directed/Undirected, Homophilic/Heterophilic, …
- For simulation: generate synthetic graphs of desired structure
- For modelling: describe real-world networks with precision
  - Clustering/Community detection: infer the group membership of nodes by maximizing the likelihood
  - Topic Modelling for NLP

[Lee and Wilkinson, 2019]

Deep Graph Generators (1/4)

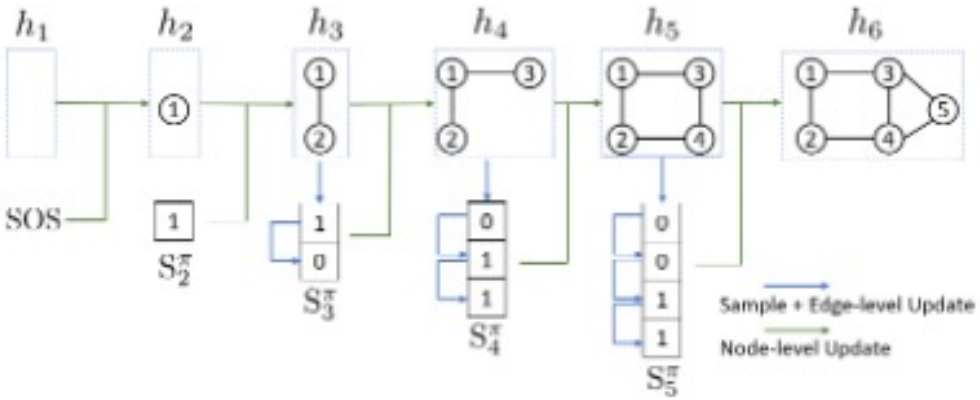
- To generate real-like synthetic graphs
- Enjoy the power of deep learning: learn from data instead of hand-designed procedures
- An active research front:
  - **Autoregressive generators**
    - Recurrent models: node by node or edge by edge RNN
    - Nonrecurrent models: attention-based
  - **Auto-encoder based generators**
  - Reinforcement Learning based generators
  - **Adversarial generators**
  - Flow-based generators

[Faez et al., 2021]

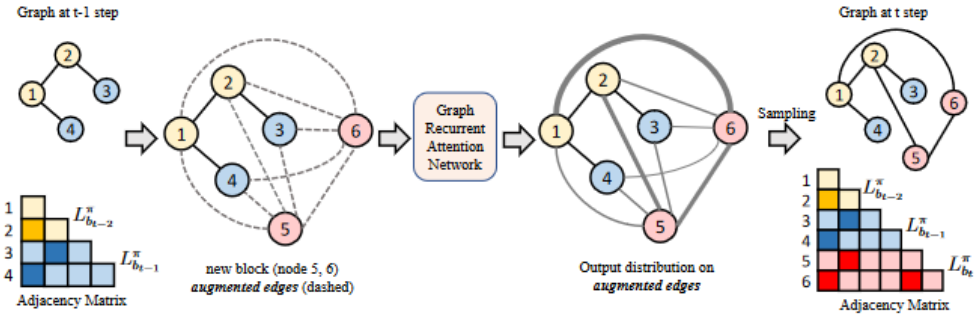
Deep Graph Generators (2/4)

- Autoregressive generators: generate using sequential models
- Recurrent model: GraphRNN
  - Hierarchical RNN model (graph level + edge level) that learns to sample $G \sim p(G)$
  - A graph G is represented as a sequence of adjacency vectors $(S_1^\pi, \dots S_n^\pi)$. π is an ordering of nodes. S_i^π is a (i-1) dimensional vector represents edges between $\pi(i)$ and previous nodes
  - $p(G)$ is related to $p(S^\pi)$ with $p(S^\pi) = \prod_{i=1}^{n+1} p(S_i^\pi | S_1^\pi, \dots, S_{i-1}^\pi)$. This product can be modeled by a RNN (graph-level) with state transition possibly modeled by another RNN (edge-level), i.e., $h_{i,j}^{edge} = \mathrm{RNN}(h_{i,j-1}^{edge}, S_{i,j-1}^\pi)$
  - π is limited to BFS ordering in order to reduce computational cost

Deep Graph Generators (3/4)



[You et al., 2018]



[Liao et al., 2019]

Deep Graph Generators (4/4)

- AE-based generator: generate using latent space variables
  - Variational graph auto-encoder [Kipf and Welling, 2016]
  - Latent variable $z_i \sim N(\mu, \sigma^2)$ with $\mu = \mathrm{GCN}_\mu(X, A),\ \log(\sigma) = \mathrm{GCN}_\sigma(X, A)$. Then $\tilde{A} = \mathrm{sigmoid}(ZZ^T)$
  - VAE Loss: $L = \mathbf{E}_{q_\varphi(Z|G)}\left[-\log\left(p_\theta(G|Z)\right)\right] + \mathrm{KL}(q_\varphi(Z|G)||p(Z))$
  - For the reconstruction error, a graph (approximate) matching procedure is needed due to the permutation invariant nature of graphs
- Adversarial generator: generate using generative adversarial networks
  - MolGAN: First implicit generative model for small molecule graphs [De Cao and Kipf, 2018]
  - The generator takes as input the latent vector z sampled from $N(0, \mathbf{1})$
  - In a way similar to VGAE, the generator output a probabilistic graph from z
  - A permutation-invariant discriminator (constructed by GCN and MLP) tries to discriminate between generated graphs and real ones
  - Does not need expensive graph matching procedure: the use of GAN omit the need to calculate likelihood

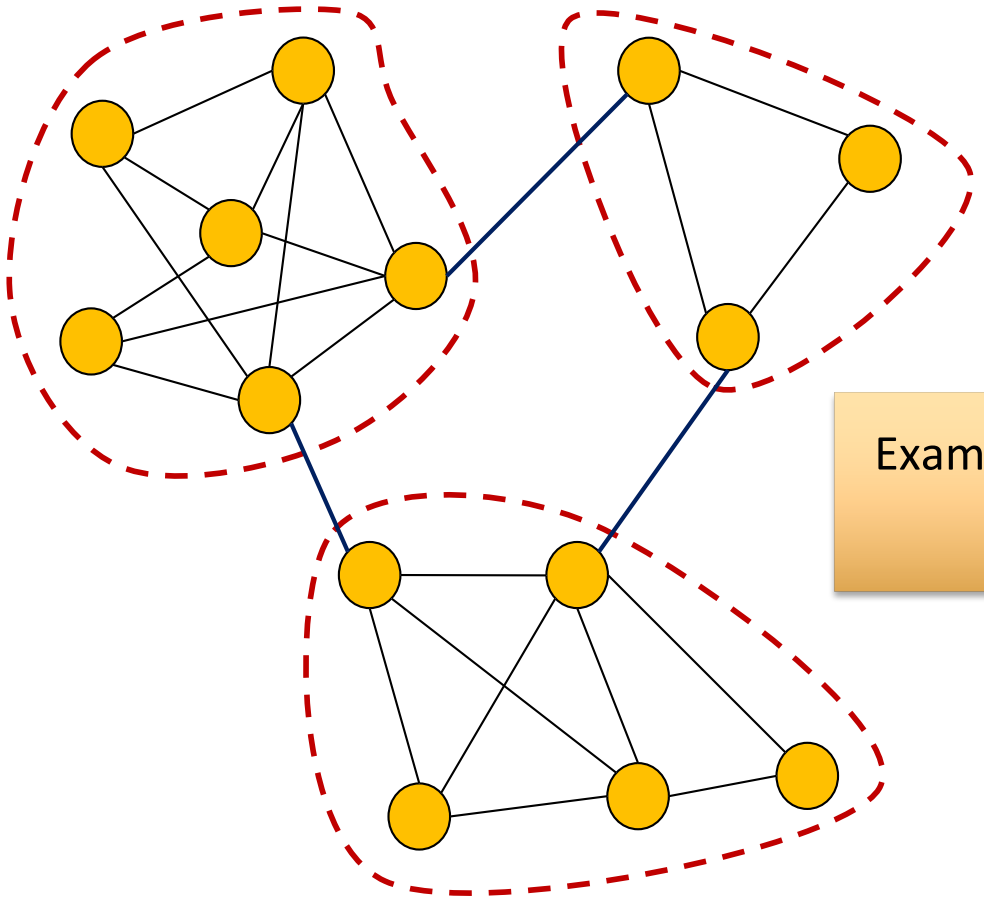
References

- J. Leskovec. Modeling Large Social and Information Networks. Tutorial at ICML, 2009.
- J. McAuley. Data Mining and Predictive Analytics, UCSD, 2015.
- D. Easley and J. Kleinberg. Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press, 2010.
- J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, Z. Ghahramani. Kronecker Graphs: An approach to modeling networks. JMLR, 2010.
- P. Holland, K. B. Laskey , and S. Leinhardt . Stochastic blockmodels : Some first steps. Social Networks 5, 1983
- Clement Lee and Darren J. Wilkinson. A Review of Stochastic Block Models and Extensions for Graph Clustering. Applied Network Science, 2019
- Faez, F., Ommi, Y., Baghshah, M. S., and Rabiee, H. R. Deep graph generators: A survey. IEEE Access, 2021.
- Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. International Conference on Machine Learning, 2018
- Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. Advances in Neural Information Processing System, 2019
- Thomas N Kipf and Max Welling. Variational graph auto-encoders. NeurIPS Workshop on Bayesian Deep Learning, 2016
- Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models, 2018

Outline

1. Introduction & Motivation
2. Graph Generators
3. **Unsupervised learning**
    1. Community detection
4. Graph kernels and Node embeddings

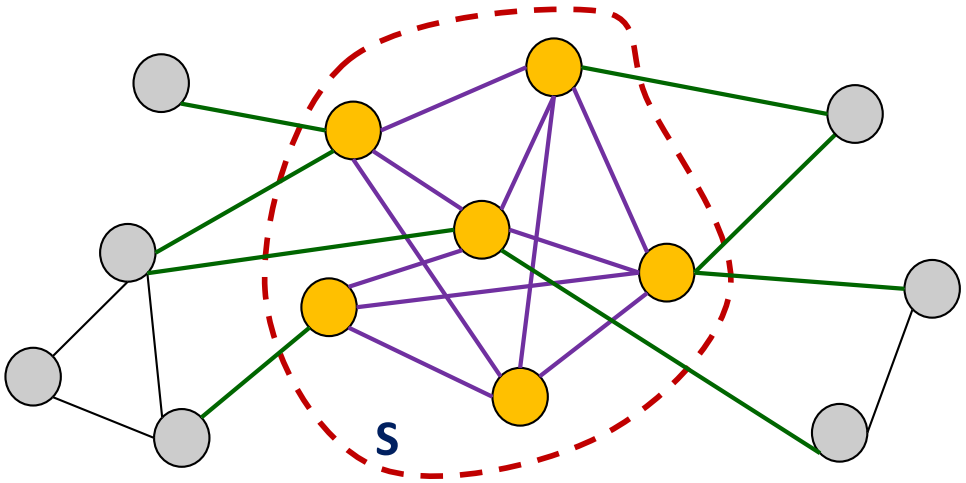
Schematic representation of communities

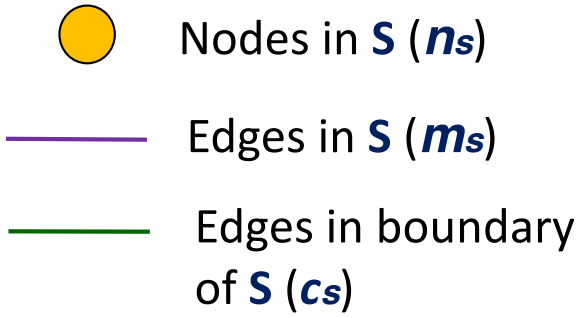


Example graph with three communities

Notation

- **G = (V, E)** is an undirected graph, **|V| = n**, **|E| = m**
- **S** is the set of nodes in the cluster
- n_s **= |S|** is the number of nodes in **S**
- m_s is the number of edges in **S**, $m_s = \left|\{(u,v): u \in S, v \in S\}\right|$
- c_s is the number of edges on the boundary of **S**, $c_s = \left|\{(u,v): u \in S, v \notin S\}\right|$
- d_u is the degree of node u
- f **(S)** represent the clustering quality of set S



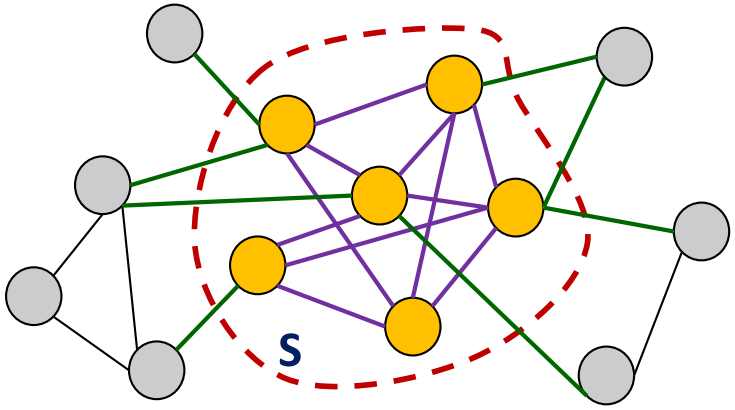


Evaluation based on external connectivity

- **Expansion** [Radicchi et al. '04]

$$f(S) = \frac{c_s}{n_s}$$

Measures the number of edges per node that point outside **S**



- **Cut ratio** [Fortunato '10]

$$f(S) = \frac{c_s}{n_s(n - n_s)}$$

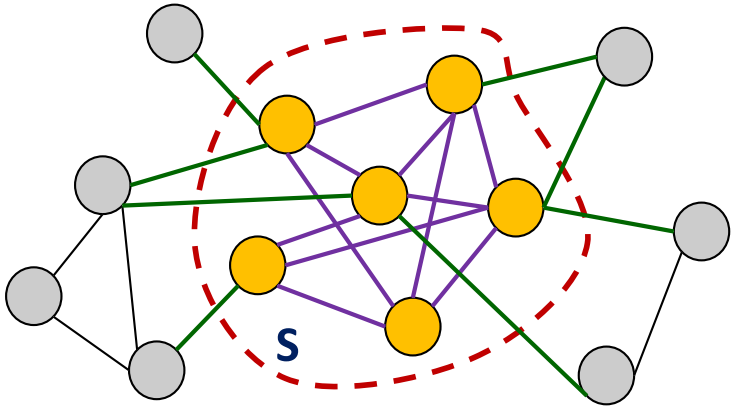
Fraction of existing edges – out of all possible edges – that leaving **S**

Evaluation based on internal connectivity (1)

- **Internal density** [Radicchi et al. '04]

$$f(S) = \frac{m_s}{n_s(n_s - 1)/2}$$

Captures the internal edge density of community **S**



- **Edges inside** [Radicchi et al. '04]

$$f(S) = m_s$$

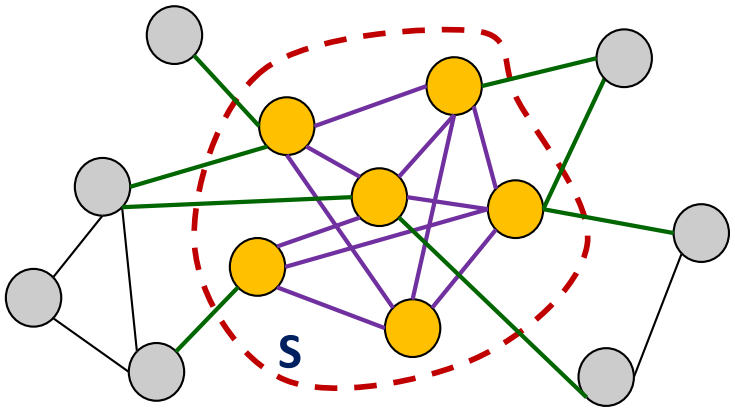
Number of edges between the nodes of **S**

Evaluation based on internal and external connectivity (2)

Conductance [Chung '97]

$$f(S) = \frac{c_s}{2m_s + c_s}$$

Measures the fraction of total edge volume that points outside **S**



Normalized cut [Shi and Malic '00]

$$f(S) = \frac{c_s}{2m_s + c_s} + \frac{c_s}{2(m - m_s) + c_s}$$

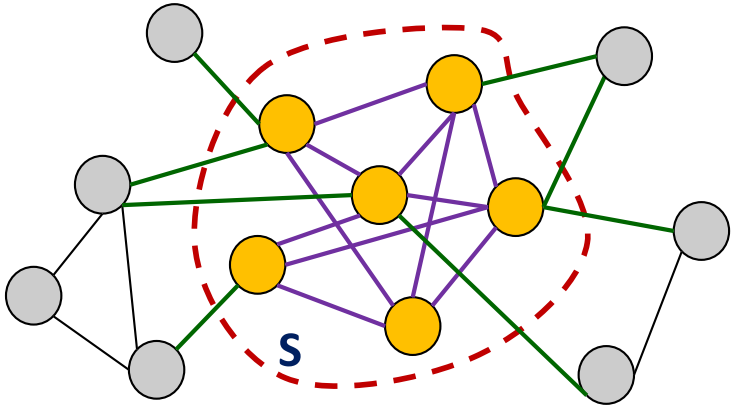
Measures the fraction of total edge volume that points outside **S** normalized by the size of **S**

Evaluation based on internal connectivity (3)

- **Triangle participation ratio (TPR)** [Yang and Leskovec '12]

$$f(S) = \frac{\left|\left\{u : u \in S, \left\{(v,w) : v,w \in S, (u,v) \in E, (u,w) \in E, (v,w) \in E\right\} \neq \varnothing\right\}\right|}{n_s}$$

Fraction of nodes in **S** that belong to a triangle



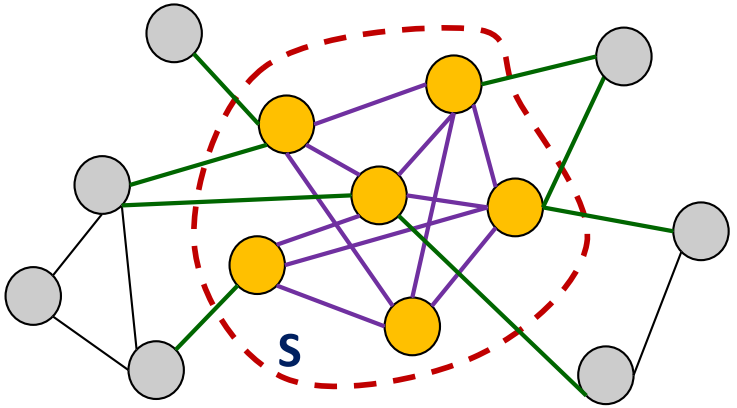
Evaluation based on network model

Modularity [Newman and Girvan '04], [Newman '06]

$$f(S) = \frac{1}{4}\left(m_s - E(m_s)\right)$$

Measures the difference between the number of edges in **S** and the expected number of edges **E(m_s)** in case of a configuration model

- Typically, a random graph model with the same degree sequence

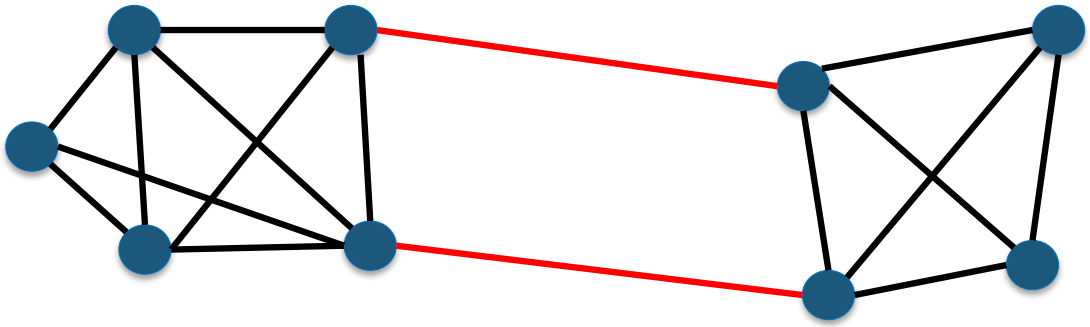


Notations

- Given Graph G=(V,E) undirected:
  - Vertex Set V={v_1,…..v_n}, Edge e_{ij} between v_i and v_j
    - we assume weight $w_{ij}>0$ for e_{ij}
  - |V| : number of vertices
  - d_i degree of v_i : $d_i = \sum_{v_j \in V} w_{ij}$
  - $v(V) = \sum_{v_i \in V} d_i$
  - for $A \subset V$ $\overline{A} = V - A$
  - Given $A, B \subset V$ & $A \cap B = \emptyset$ $w(A,B) = \sum_{v_i \in A, vj \in B} w_{ij}$
  - D: Diagonal matrix where D(i,i)=d_i
  - W: Adjacency matrix W(i,j)=w_{ij}

Graph-Cut

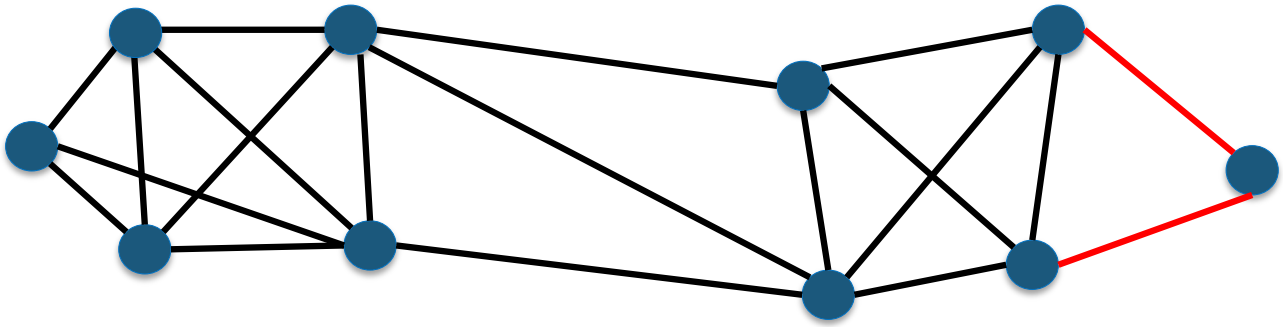
- For k clusters:
  - $cut(A_1, .., Ak) = 1/2 \sum_{i=1}^{k} w(Ai, \overline{A_i})$
    - undirected graph:1/2 we count twice each edge



- Min-cut:Minimize the edges' weight a cluster shares with the rest of the graph

Min-Cut

- Easy for k=2 : Mincut(A_1,A_2)
  - Stoer and Wagner: “A Simple Min-Cut Algorithm”
- In practice one vertex is separated from the rest
  - The algorithm is drawn to outliers



Normalized Graph Cuts

- We can normalize by the size of the cluster (size of sub-graph) :
  - number of Vertices (Hagen and Kahng, 1992):

    $$Ratiocut(A_1, \dots Ak) = \sum_{i=1}^{k} \frac{cut(Ai, \overline{A_i})}{|Ai|}$$

  - sum of weights (Shi and Malik, 2000) :

    $$Ncut(A_1, \dots Ak) = \sum_{i=1}^{k} \frac{cut(Ai, \overline{A_i})}{v(A_i)}$$

- Optimizing these functions is NP-hard
- Spectral Clustering provides solution to a relaxed version of the above

From Graph Cuts to Spectral Clustering

- For simplicity assume k=2:
  - Define $f: V \to \mathbb{R}$ for Graph G :

$$f_i = \begin{cases} 1 & v_i \in A \\ -1 & v_i \in \overline{A} \end{cases}$$

- Optimizing the original cut is equivalent to an optimization of:

$$\sum_{i,j=1}^{n} w_{ij}(f_i - fj)^2$$

$$= \sum_{v_i \in A, vj \in \overline{A}} w_{ij}(1+1)^2 + \sum_{v_i \in \overline{A}, vj \in A} w_{ij}(-1-1)^2$$

$$= \mathbf{8} * \boldsymbol{cut}\left(\boldsymbol{A}, \overline{\boldsymbol{A}}\right)$$

Graph Laplacian

- How is the previous useful in Spectral clustering?

$$\sum_{i,j=1}^{n} w_{ij}(f_i - fj)^2$$

$$= \sum_{i,j=1}^{n} w_{ij} {f_i}^2 \; -2 \sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{i,j=1}^{n} w_{ij} {f_j}^2$$

$$= \sum_{i,j=1}^{n} d_i {f_i}^2 - 2 \sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{i,j=1}^{n} d_j {f_j}^2$$

$$= 2\left(\sum_{i,j=1}^{n} d_{ii} {f_i}^2 - \sum_{i,j=1}^{n} w_{ij} f_i f_j \right)$$

$$= 2(\boldsymbol{f^T D f - f T W f}) = 2\boldsymbol{f^T (D - W) f} = 2\boldsymbol{f^T L f}$$

- **f:**a single vector with the cluster assignments of the vertices
- **L=D-W :** the Laplacian of a graph

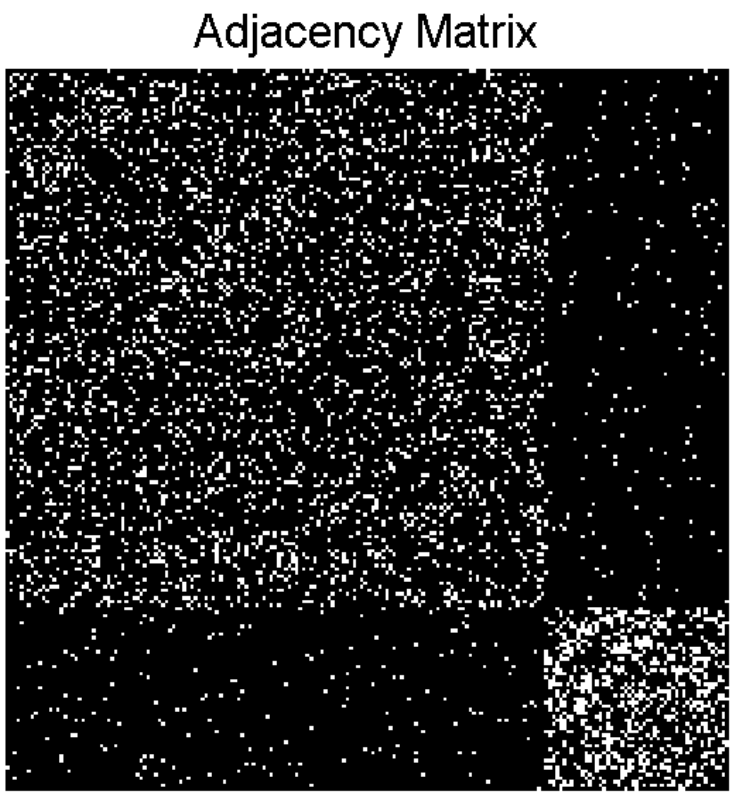
Properties of L

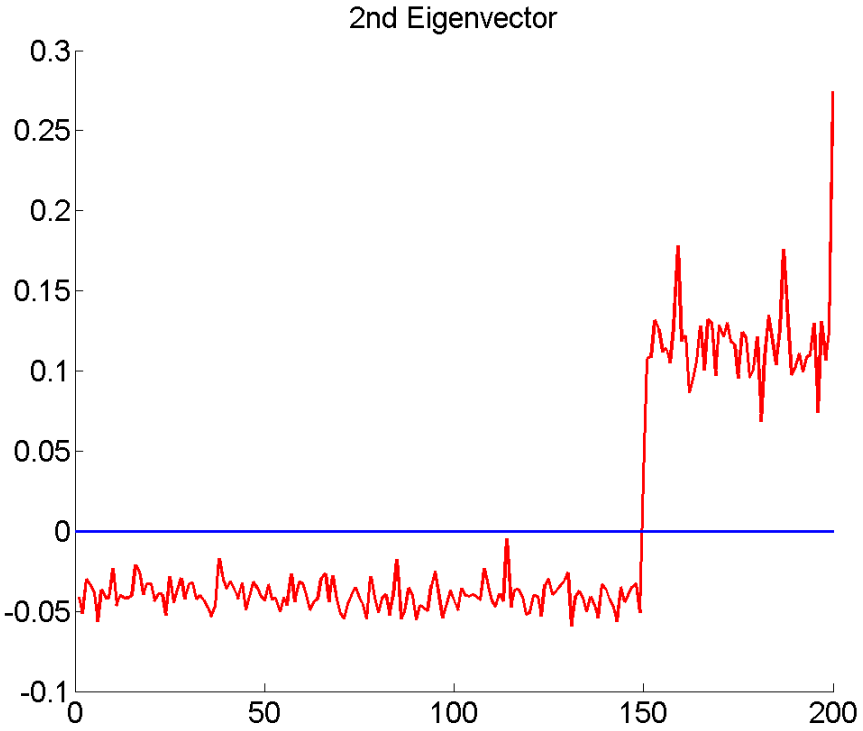
- L is
  - Symmetric
  - Positive
  - Semi-definite
- The smallest eigenvalue of L is 0
  - The corresponding eigenvector is $\mathbb{1}$
- L has n non-negative, real valued eigenvalues
  - $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$

Two Way Cut from the Laplacian

- We could solve $min_f f^T L f$ where $f \in \{-1,1\}^n$
- NP-Hard for discrete cluster assignments
  - Relax the constraint $to\ f \in R^n$:
    $min_f f^T L f$ subject to $f^T f = n$
- The solution to this problem is given by:
  - (**Rayleigh-Ritz Theorem**) the eigenvector corresponding to smallest eigenvalue: 0 TRIVIA as it offers no information
- We use the second eigenvector as an approximation
  - $f_i > 0$ the vertex belongs to one cluster , $f_i < 0$ to the other

Example


Adjacency Matrix

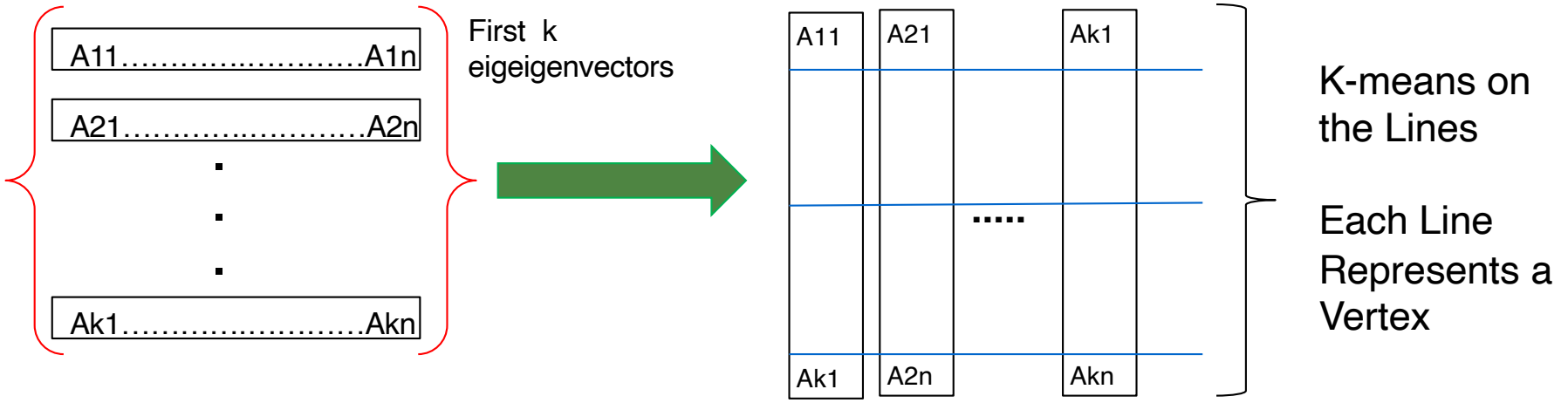

2nd Eigenvector
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
50
100
150
200

Multi-Way Graph Partition

- Define $f_{ij} = \begin{cases} \frac{1}{\sqrt{|Aj|}} & vi \in Aj \\ 0 & othewise \end{cases}$
  - we have a vector indicating the cluster a vertex belongs to
- Similarly to the other equations we can deduce:
  - $f_i^T L f_i = cut(Ai, \overline{A_i})/|Ai|$
  - $\sum_{i=1}^{k} f_i^T L f_i = \sum_{i=1}^{k} (F^T L F)_{ii} = Tr(F^T L F)$
    - Where Tr is the Trace of a Matrix
- So now the RatioCut becomes:
  $min(F^T L F)\ subject\ to\ F^T F = I$

Multi-Way Graph Partition

- The solution can now be given by the first k eigenvectors of L as columns
- The real values need to be converted to cluster assignments
  - We use k-means to cluster the rows
  - We can substitute *L* with L_{sym}



References

- UIrike von Luxburg, A Tutorial on Spectral Clustering, Statistics and Computing, 2007
- Davis, C., W. M. Kahan (March 1970). The rotation of eigenvectors by a perturbation. III. SIAM J. Numerical Analysis 7
- Shi, Jianbo, and Jitendra Malik. "Normalized cuts and image segmentation."*Pattern Analysis and Machine Intelligence, IEEE Transactions on* (2000).
- Mechthild Stoer and Frank Wagner. 1997. A simple min-cut algorithm. *J. ACM*
- Ng, Jordan & Weiss, K-means algorithm on the embeded eigen-space, NIPS 2001
- Hagen, L. Kahng, , "New spectral methods for ratio cut partitioning and clustering," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , 1992

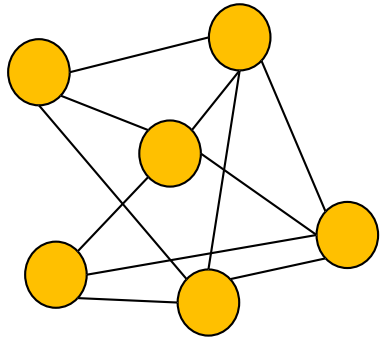
Graph Clustering Algorithms

- Modularity Based Methods
- Louvain Algorithm

Modularity based clustering

- **Modularity** function [Newman and Girvan '04], [Newman '06]
- Initially introduced as a measure for assessing the strength of communities
  - **Q = (fraction of edges within communities) –**

    (expected number of edges within communities)
- What is the **expected** number of edges?
- Consider a configuration model
  - **Random graph** model with the same degree distribution
  - Let P_{ij} = probability of an edge between nodes **i** and **j** with degrees k_i and k_j respectively
  - Then $P_{ij} = k_i k_j / 2m$, where $m = |E| = \frac{1}{2} \sum_i k_i$



Formal definition of modularity

Modularity Q

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j)$$

where

- **A** is the adjacency matrix
- k_i, k_j the degrees of nodes **i** and **j** respectively
- **m** is the number of edges
- C_i is the community of node **i**
- **δ(.)** is the Kronecker function: 1 if both nodes **i** and **j** belong on the same community ($C_i = C_j$), 0 otherwise

[Newman and Girvan '04], [Newman '06]

Properties of modularity

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j)$$

- **Larger** modularity **Q** indicates **better** communities (more than random intra-cluster density)
  - The community structure would be better if the number of internal edges exceed the expected number
- Modularity value is always **smaller than 1**
- It can also take **negative values**
  - E.g., if each node is a community itself
  - No partitions with positive modularity → No community structure
  - Partitions with large negative modularity → Existence of subgraphs with small internal number of edges and large number of inter-community edges

[Newman and Girvan '04], [Newman '06], [Fortunato '10]

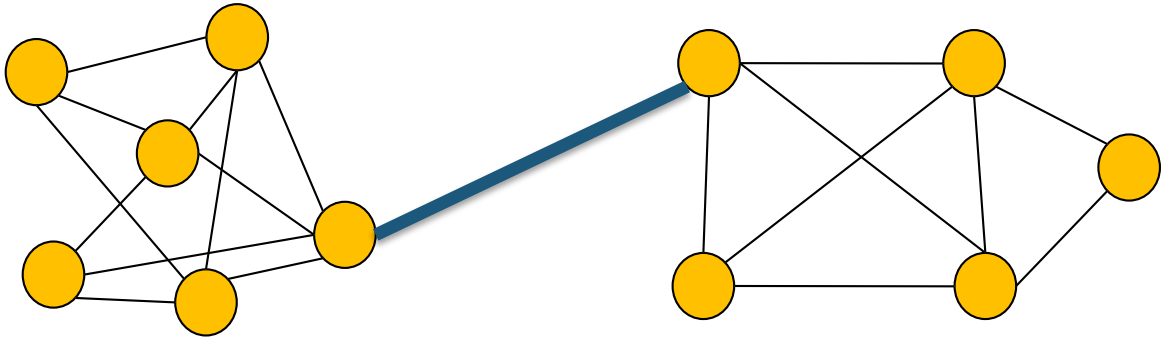
Applications of modularity

- Modularity can be applied:
  - As **quality function** in clustering algorithms
  - As **evaluation measure** for comparison of different partitions or algorithms
  - As a community detection tool itself
    - **Modularity optimization**
  - As criterion for reducing the size of a graph
    - Size reduction preserving modularity [Arenas et al. '07]

[Newman and Girvan '04], [Newman '06], [Fortunato '10]

Modularity-based community detection

- Modularity was first applied as a **stopping criterion** in the Newman-Girvan algorithm
- Newman-Girvan algorithm [Newman and Girvan '04]
  - A **divisive** algorithm (detect and remove edges that connect vertices of different communities)
  - **Idea:** try to identify the edges of the graph that are most between other vertices → responsible for connecting many node pairs
  - Select and remove edges based to the value of **betweenness centrality**
  - **Betweenness centrality:** number of **shortest paths** between every pair of nodes, that pass through an edge



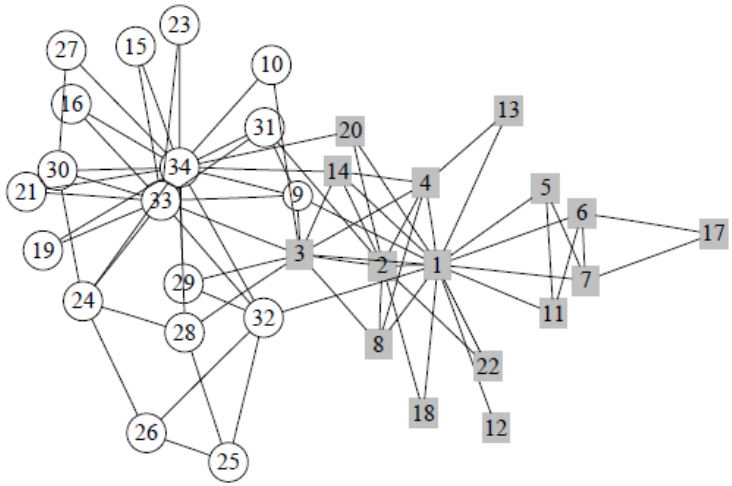
Edge betweenness is higher for edges that connect different communities

Newman-Girvan algorithm (1)

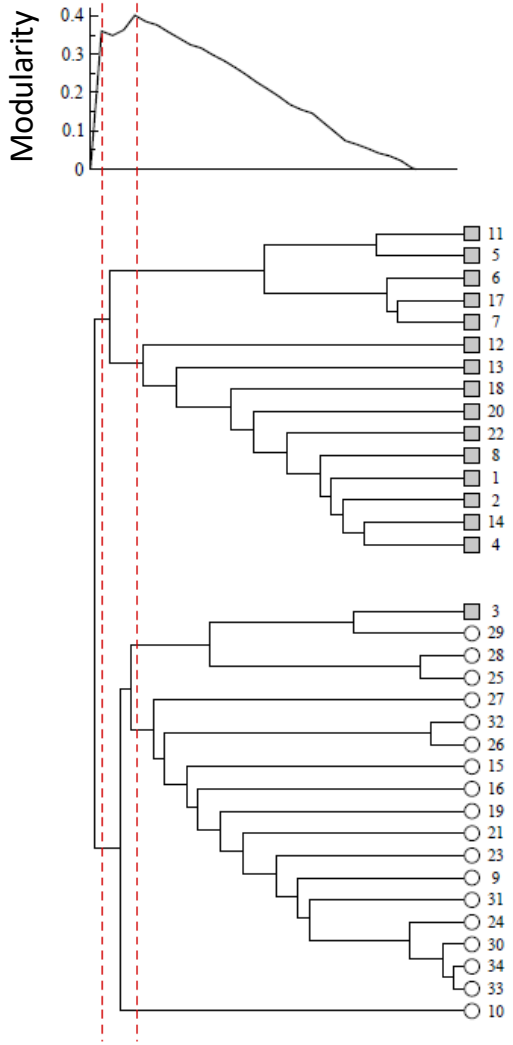
- **Basic steps:**
  1. Compute betweenness centrality for all edges in the graph
  2. Find and remove the edge with the highest score
  3. Recalculate betweenness centrality score for the remaining edges
  4. Go to step 2
- How do we know if the produced communities are **good ones** and stop the algorithm?
  - The output of the algorithm is in the form of a **dendrogram**
  - Use **modularity** as a criterion to cut the dendrogram and terminate the algorithm (Q ~= 0.3-0.7 indicates good partitions)
- Complexity: **$O(m^2n)$** (or **$O(n^3)$** on a sparse graph)

[Newman and Girvan '04], [Girvan and Newman '02]

Newman-Girvan algorithm (2)



Zachary's karate club



Community structure

[Newman and Girvan '04]

Modularity optimization

- High values of modularity indicate good quality of partitions
- **Goal:** find the partition that corresponds to the maximum value of modularity
- **Modularity maximization** problem
  - Computational difficult problem [Brandes et al. '06]
  - Appoximation techniques and heuristics
- Four main categories of techniques
  1. Greedy techniques
  2. **Spectral optimization**
  3. Simulated annealing
  4. Extremal optimization

[Fortunato '10]

Spectral optimization (1)

- **Idea:** Spectral techniques for modularity optimization
- **Goal:** Assign the nodes into two communities, **X** and **Y**
- Let $s_i, \forall i \in V$ an indicator variable where $s_i = +1$ if **i** is assigned to **X** and $s_i = -1$ if **i** is assigned to **Y**

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j)$$

$$= \frac{1}{4m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)(s_i s_j + 1)$$

$$= \frac{1}{4m}\sum_{ij} B_{ij} s_i s_j = \frac{1}{4m} s^T \mathrm{B} s$$

- **B** is the **modularity matrix**

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

[Newman '06], [Newman '06b]

Spectral optimization (2)

- Modularity matrix **B**

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

- Vector **s** can be written as a linear combination of the eigenvectors $\mathbf{u_i}$ of the modularity matrix **B**

where $a_i = u_i^T s$

$$s = \sum_i a_i \, u_i$$

- Modularity can now expressed as

$$Q = \frac{1}{4m} \sum_i a_i u_i^T \; B \; \sum_j a_j u_j^T = \frac{1}{4m} \sum_{i=1}^{n} \left(u_i^T s \right)^2 \beta_i$$

Where $\boldsymbol{\beta_i}$ is the eigenvalue of **B** corresponding to eigenvector $\mathbf{u_i}$

[Newman '06], [Newman '06b]

Spectral optimization (3)

- Spectral modularity optimization algorithm
  1. Consider the eigenvector **u_1** of **B** corresponding to the largest eigenvalue
  2. Assign the nodes of the graph in one of the two communities **X** (**$s_i = +1$**) and **Y** (**$s_i = -1$**) based on the **signs** of the corresponding components of the eigenvector

$$s_i = \begin{cases} 1 & if\ u_1(i) \geq 0 \\ -1 & if\ u_1(i) < 0 \end{cases}$$

  - More than two partitions?
  1. **Iteratively**, divide the produced partitions into two parts
  2. If at any step the split does not contribute to the modularity, leave the corresponding subgraph as is
  3. End when the entire graph has been splinted into no further divisible subgraphs
  - Complexity: **$O(n^2 \log n)$** for sparse graphs

[Newman '06], [Newman '06b]

Louvain Method (1)

- **Idea:** Heuristic for modularity optimization
- Similar to hierarchical clustering:
  - *recursively merge communities into single nodes*
  - *apply modularity clustering on the condensed graph*
- Fast algorithm: linear on sparse data
- Costly on memory though…

[Blondel et al. 2008]

Louvain Method (2)

- Step 1: Assign different communities to each node of the graph
- Step 2: For each node, evaluate modularity gain by moving this node from its current community to its neighbor's community. Its community is changed if the gain is positive and maximized. Otherwise remains.

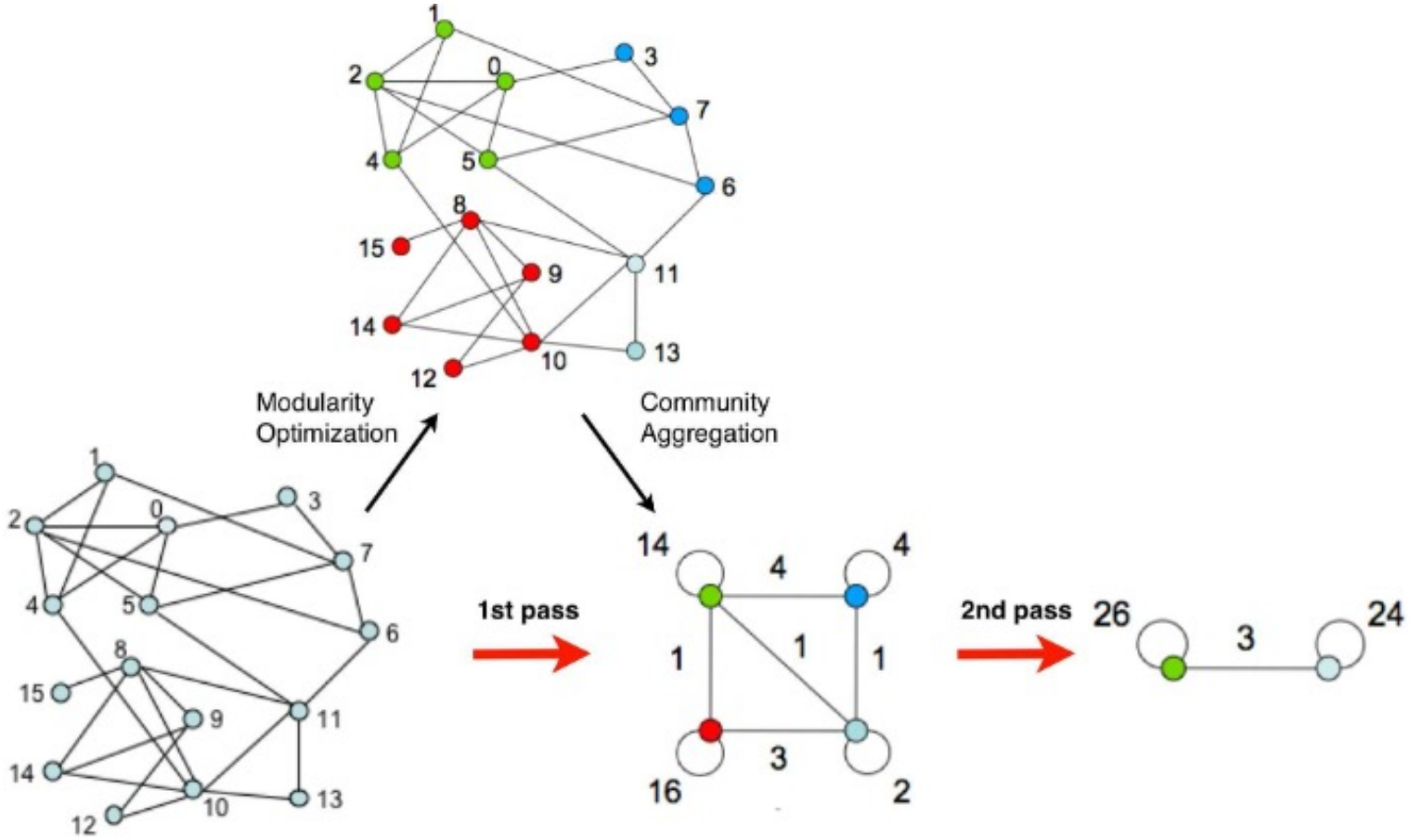
$$\Delta Q = \left[\frac{\Sigma_{in}+k_{i,in}}{2m} - \left(\frac{\Sigma_{tot}+k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

- Repeat Step 2 until a local maxima of modularity is obtained (no further improvement)
- Step 3: Condense the graph by merging nodes of the same community into one single node
- Step 4: Apply Step 2 on the new graph

Repeat until there are no changes in the graph and maximum modularity is obtained

[Blondel et al. 2008]

Louvain Method (3)



[Blondel et al. 2008]

References – Graph clustering

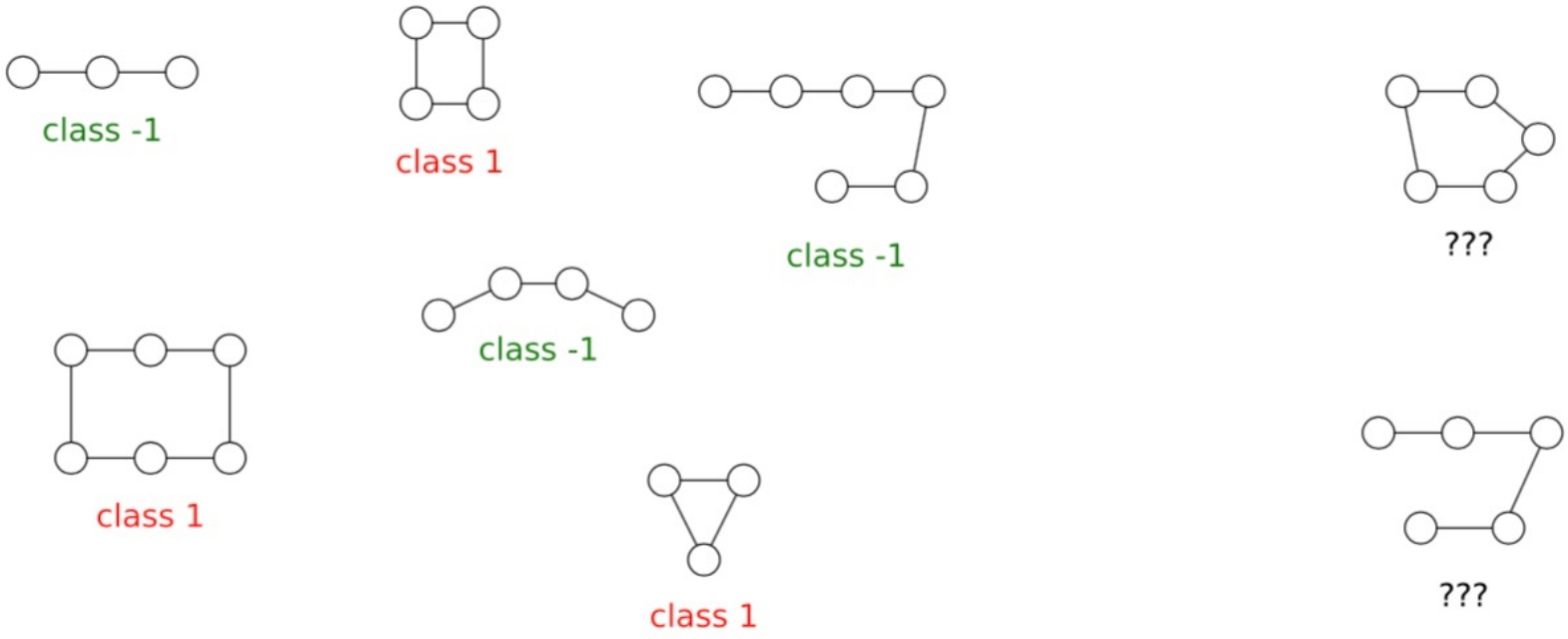
- UIrike von Luxburg, A Tutorial on Spectral Clustering, Statistics and Computing, 2007
- Davis, C., W. M. Kahan (March 1970). The rotation of eigenvectors by a perturbation. III. SIAM J. Numerical Analysis 7
- Shi, Jianbo, and Jitendra Malik. "Normalized cuts and image segmentation, "*Pattern Analysis and Machine Intelligence, IEEE Transactions on* (2000).
- Mechthild Stoer and Frank Wagner. 1997. A simple min-cut algorithm. *J. ACM*
- Ng, Jordan & Weiss, K-means algorithm on the embedded eigen-space, NIPS 2001
- Hagen, L. Kahng, , "New spectral methods for ratio cut partitioning and clustering," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , 1992
- Blondel, Vincent D., et al. "Fast unfolding of communities in large networks." Journal of statistical mechanics: theory and experiment, 2008
- Traag, V.A., Waltman, L. & van Eck, N.J. “From Louvain to Leiden: guaranteeing well-connected communities.” Sci Rep, 2019
- Raghavan, U. N., Albert, R. & Kumara, S. “Near linear time algorithm to detect community structures in large-scale networks.” Physical Review E, 2007
- Pons, P. & Latapy, M. “Computing communities in large networks using random walks.” Computer and Information Sciences-ISCIS, 2005
- Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. “Structural deep clustering network.” Proceedings of The Web Conference, 2020
- Di Jin, Ziyang Liu, Weihao Li, Dongxiao He, and Weixiong Zhang. “Graph convolutional networks meet markov random fields: Semi-supervised community detection in attribute networks.” Proceedings of the 33rd AAAI Conference on Artificial Intelligence, 2019
- Binbin Zhang, Zhizhi Yu, and Weixiong Zhang. “Community-centric graph convolutional network for unsupervised community detection.” IJCAI, 2020

Learning for Graphs

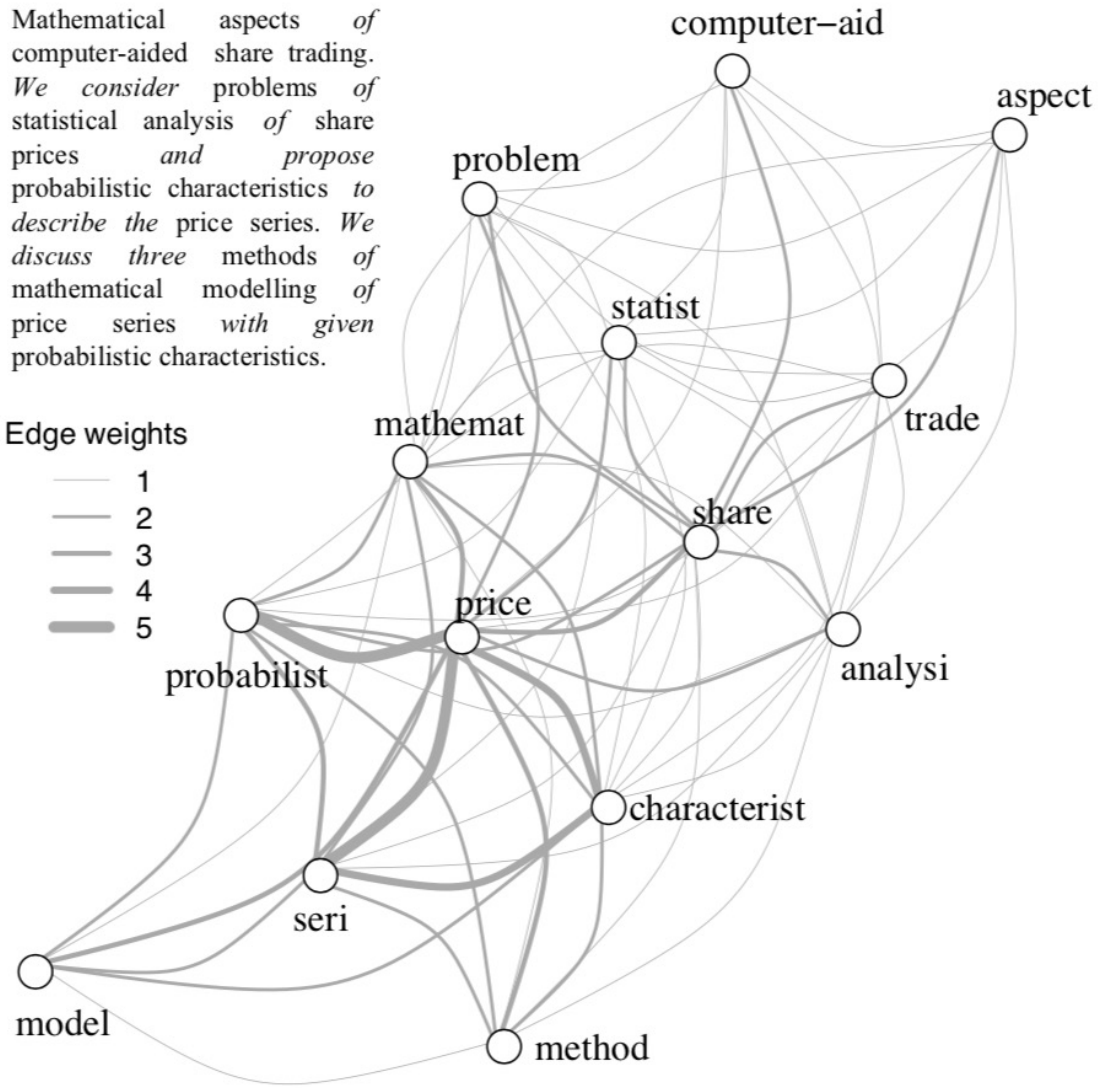
- Introduction & Motivation
- Graph Generators
- Unsupervised learning
  - Community detection
- **Graph Kernels and node embeddings**

Machine Learning for Graphs

- Node classification
- Graph clustering
- Link Prediction:
- Graph classification



Motivation – Text categorization



Given a text, create a graph where

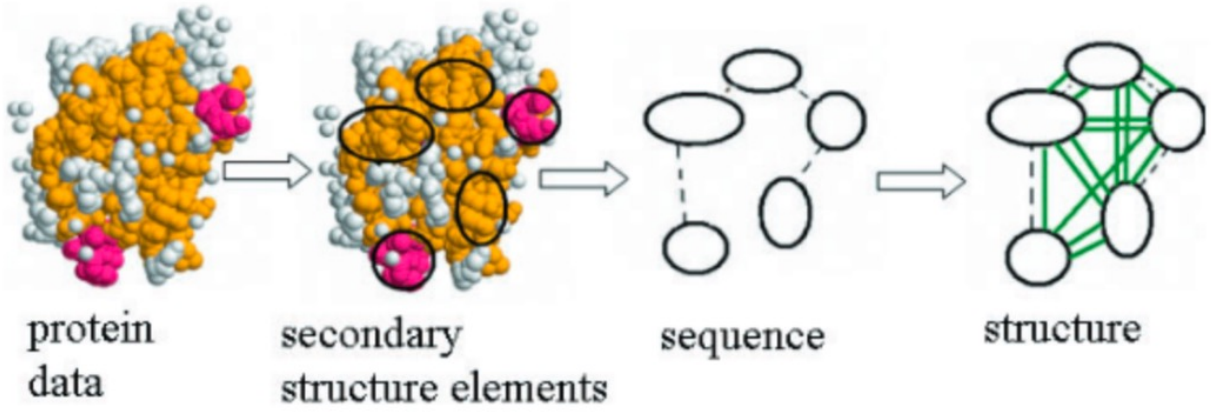
- vertices correpond to terms
- two terms are linked to each other if they co-occur within a fixed-size sliding window

Rousseau et al. "Text categorization as a graph classification problem.". In ACL'15

Motivation – Protein Function Prediction

For each protein, create a graph that contains information about its

- structure
- sequence
- chemical properties



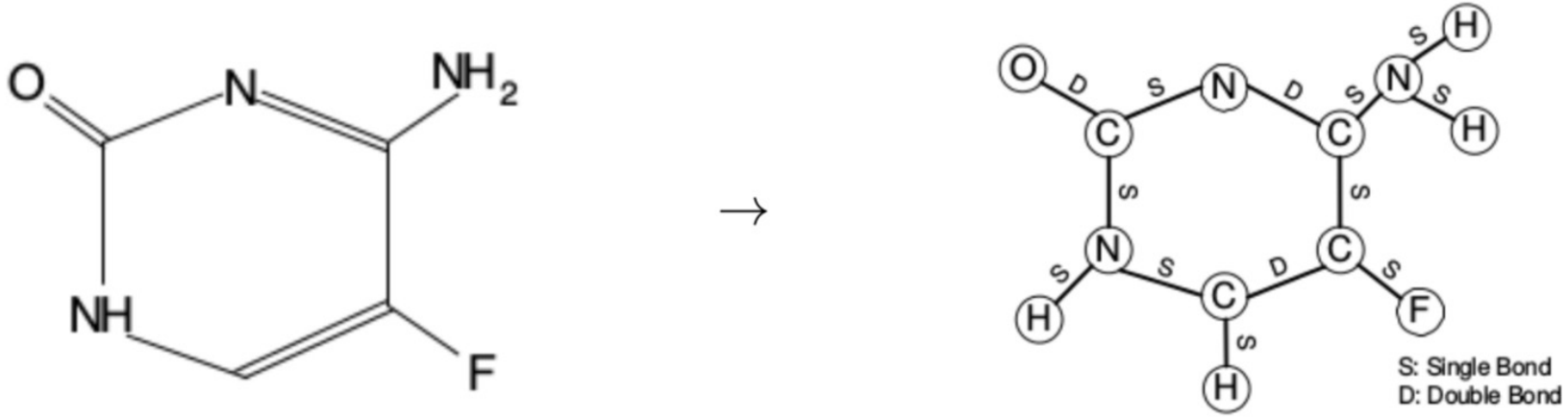
Use graph kernels to

- measure structural similarity between proteins
- predict the function of proteins

Borgwardt et al. "Protein function prediction via graph kernels". Bioinformatics 21

Motivation – Chemical compound classification

Represent each chemical compound as a graph



Use a frequent subgraph discovery algorithm to discover the substructures that occur above a certain support constraint

Perform feature selection

Use the remaining substructures as features for classification

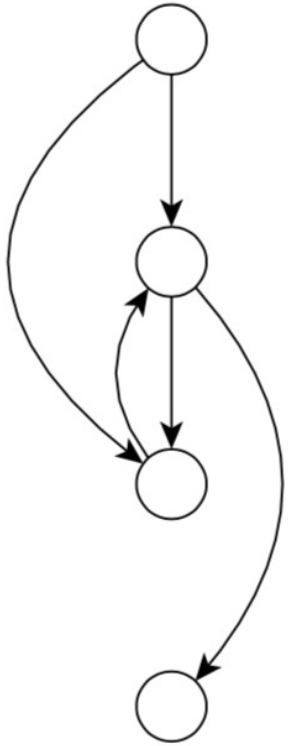
Deshpande et al. "Frequent substructure-based approaches for classifying chemical compounds". TKDE 17(8)

Motivation – Malware detection

Given a computer program, create its control flow graph

```
            processed_pages.append(processed_page)
            visited += 1
            links = extract_links(html_code)
            for link in links:
                if link not in visited_links:
                    links_to_visit.append(link)

        return create_vocabulary(processed_pages)


def parse_page(html_code):
    punct = re.compile(r'([^A-Za-z0-9])')
    soup = BeautifulSoup(html_code, 'html.parser')
    text = soup.get_text()
    processed_text = punct.sub(" ", text)
    tokens = processed_text.split()
    tokens = [token.lower() for token in tokens]
    return tokens


def create_vocabulary(processed_pages):
    vocabulary = {}
    for processed_page in processed_pages:
        for token in processed_page:
            if token in vocabulary:
                vocabulary[token] += 1
            else:
                vocabulary[token] = 1

    return vocabulary
```

→



Compare the control flow graph of the problem against the set of control flow graphs of known malware

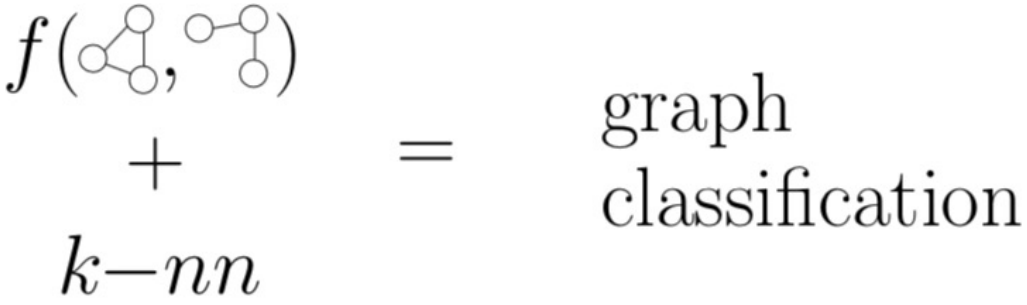
If it contains a subgraph isomporphic to these graphs → malicious code inside the program

Gascon et al. "Structural detection of android malware using embedded call graphs". In AISec'13

Graph similarity

Graph classification very related to graph comparison

Example



Although graph comparison seems a tractable problem, it is very **complex**

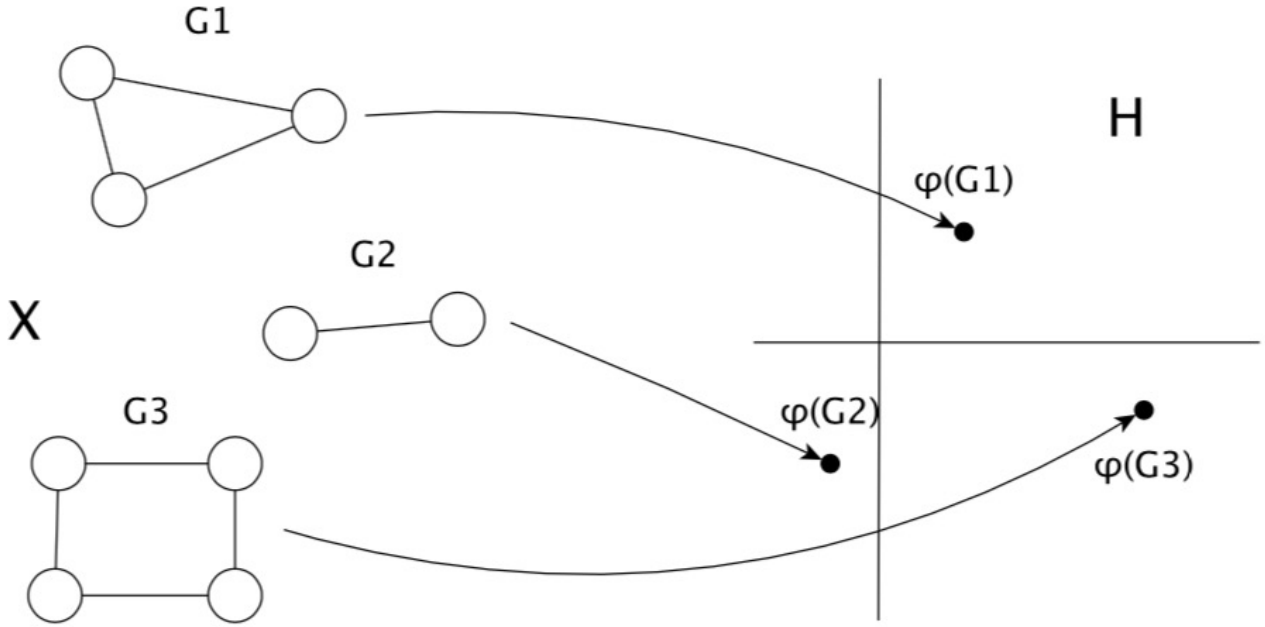
We are interested in algorithms capable of measuring the similarity between two graphs in **polynomial** time

Graph Kernels

Definition (Graph Kernel)

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

- It is equivalent to an inner product of the embeddings $\phi : \mathcal{X} \rightarrow \mathbb{H}$ of a pair of graphs into a Hilbert space: $k(G_1, G_2) = \langle \phi(G_1), \phi(G_2) \rangle$
- Makes the whole family of kernel methods (e.g. SVMs) applicable to graphs



Graph invariants

We saw that proving that two graphs are isomorphic is not a simple task

It is much simpler to show that two graphs are not isomorphic by finding a property that only one of the two graphs has. Such a property is called a *graph invariant*

Definition (Graph Invariant)

A graph invariant is a numerical property of graphs for which any two isomorphic graphs must have the same value

Some examples of graph invariants include:

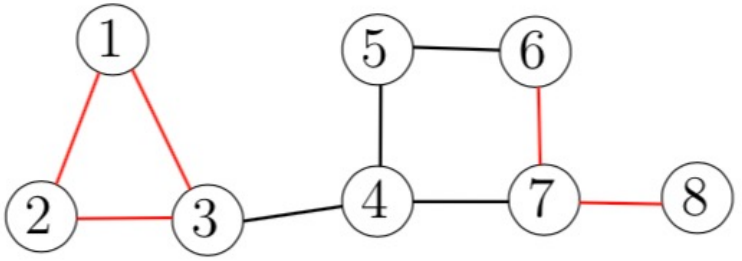
1. number of vertices
2. number of edges
3. number of spanning trees
4. degree sequence
5. spectrum

Substructures for similarity

A large number of graph kernels compare substructures of graphs that are computable in polynomial time:

- walks
- shortest path lengths
- cyclic patterns
- rooted subtrees
- graphlets

⋮



Shervashidze et al. "Efficient graphlet kernels for large graph comparison.". In AISTATS'09

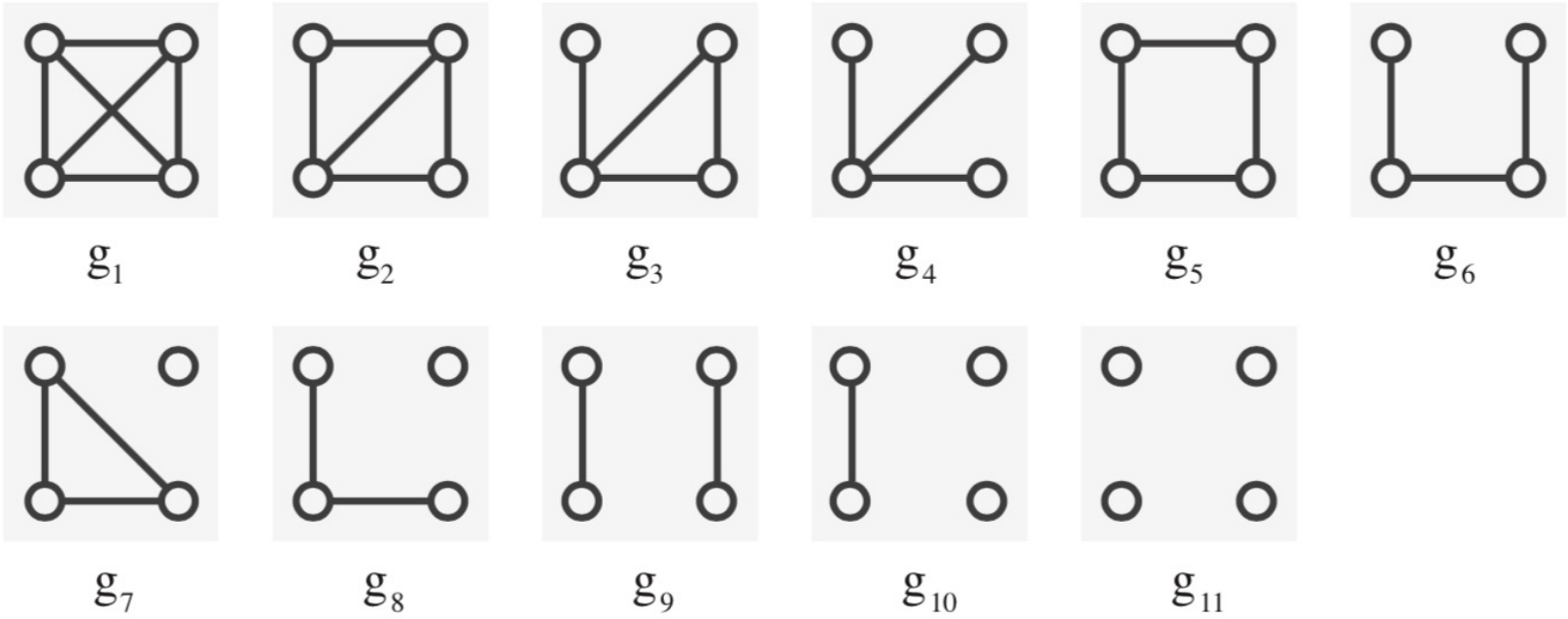
Graphlet Kernel

The graphlet kernel compares graphs by counting *graphlets*

A graphlet corresponds to a small subgraph

- typically of 3,4 or 5 vertices

Below is the set of graphlets of size 4



Shervashidze et al. "Efficient graphlet kernels for large graph comparison". In AISTATS'09

Graphlet Kernel

Let $\mathcal{G} = \{graphlet_1, graphlet_2, \ldots, graphlet_r\}$ be the set of size-k graphlets

Let also $f_G \in \mathcal{N}^r$ be a vector such that its i-th entry is $f_{G,i} = \#(graphlet_i \sqsubseteq G)$

The graphlet kernel is defined as:

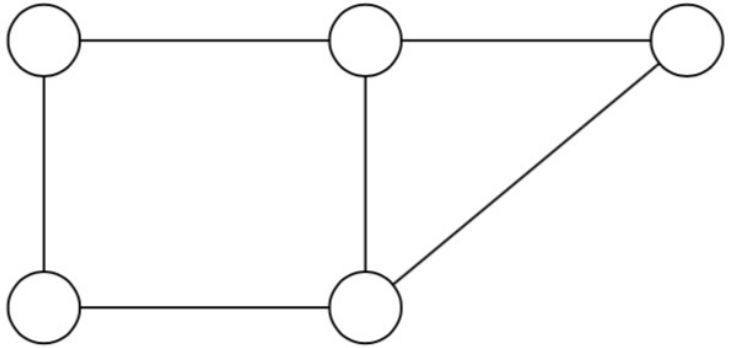
$$k(G_1, G_2) = f_{G_1}^\top f_{G_2}$$

Problems:

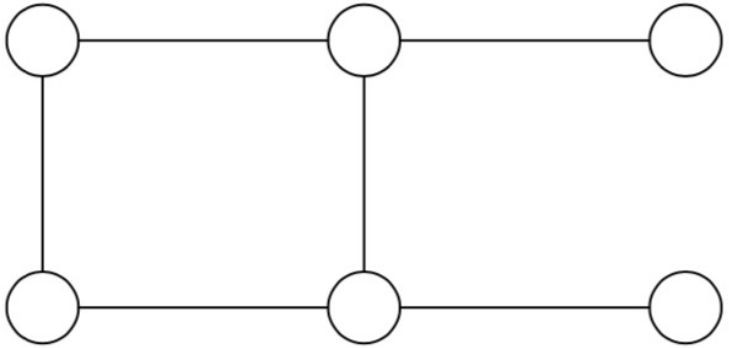
- There are $\binom{n}{k}$ size-k subgraphs in a graph
- Exaustive enumeration of graphlets is very expensive

  Requires $O(n^k)$ time
- For labeled graphs, the number of graphlets increases further

Graphlet Kernel



G_1



G_2

The vector representations of the graphs above according to the set of graphlets of size 4 is:

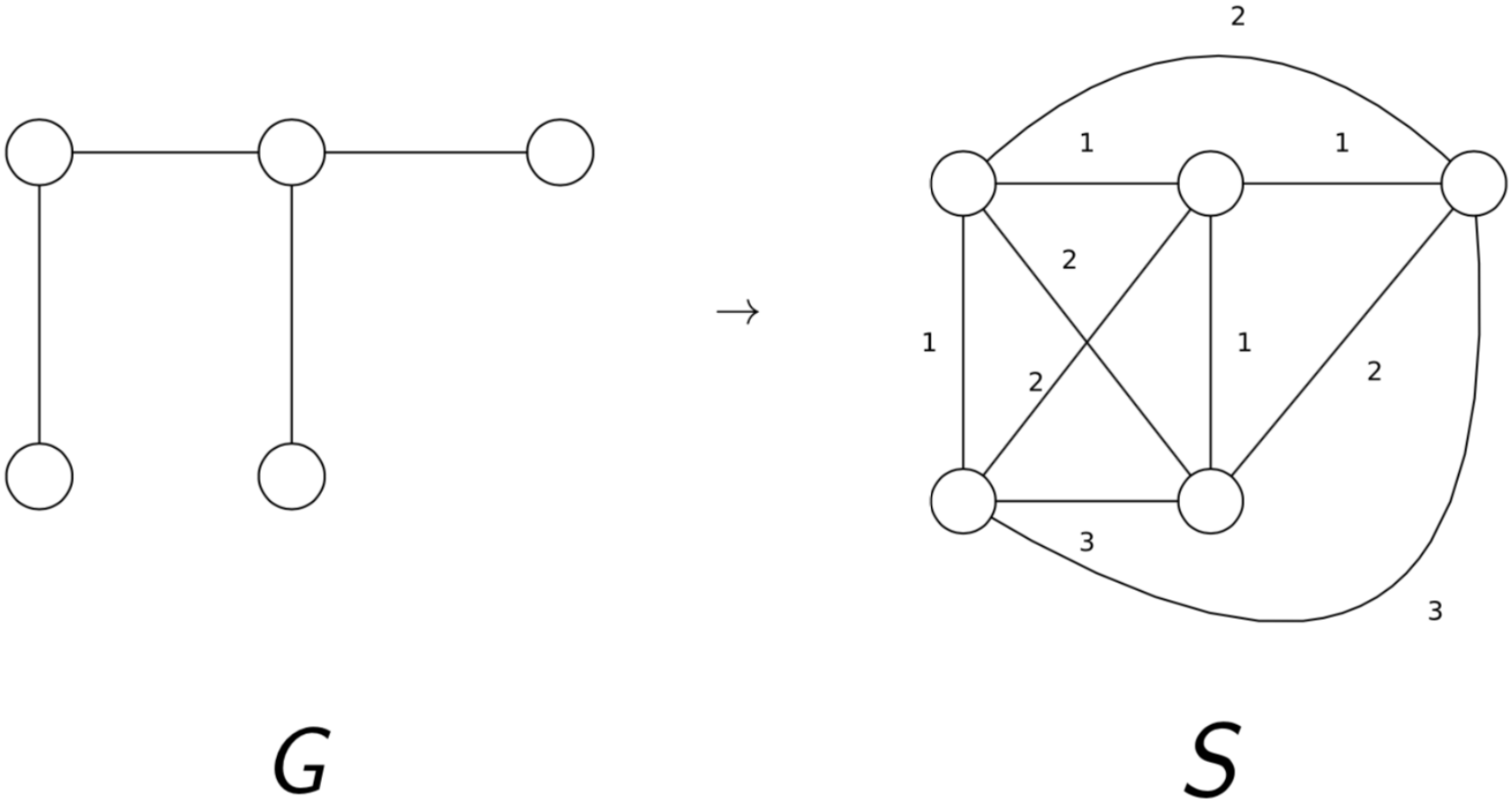
$$f_{G_1} = (0, 0, 2, 0, 1, 2, 0, 0, 0, 0, 0)^T$$
$$f_{G_2} = (0, 0, 0, 2, 1, 5, 0, 4, 0, 3, 0)^T$$

Hence, the value of the kernel is:

$$k(G_1, G_2) = f_{G_1}^\top f_{G_2} = 11$$

Shortest Path Kernel

Floyd-transformation



Shortest Path Kernel

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

- For unlabeled graphs, it can be:

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

  where $\ell(e)$ gives the label of edge e

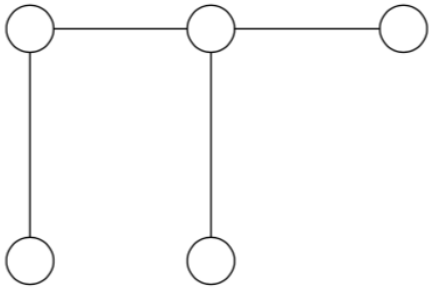
- For labeled graphs, it can be:

$$k^{(1)}_{walk}(e_1, e_2) = \begin{cases} 1 & \text{if } \ell(e_1) = \ell(e_2) \wedge \ell(e_1^1) = \ell(e_2^1) \wedge \ell(e_1^2) = \ell(e_2^2), \\ 0 & \text{otherwise} \end{cases}$$

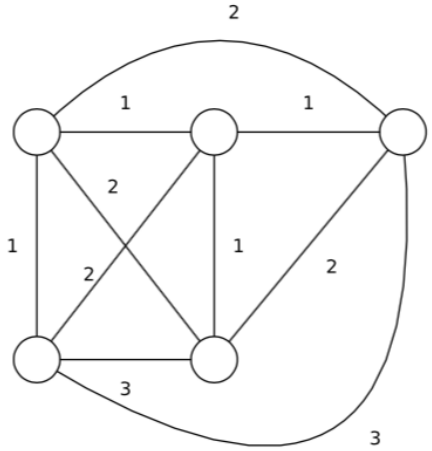
  where e^1, e^2 are the two endpoints of e

Shortest Path Kernel

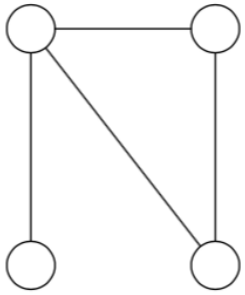
Floyd-transformations



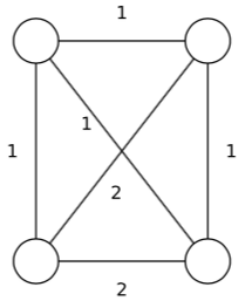
G_1



S_1



G_2



S_2

Shortest Path Kernel

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

Hence, the value of the kernel is:

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

Shortest Path Kernel

Computing the shortest path kernel includes:

- Computing shortest paths for all pairs of vertices in the two graphs: $\mathcal{O}(n^3)$
- Comparing all pairs of shortest paths from the two graphs: $\mathcal{O}(n^4)$

Hence, runtime is $\mathcal{O}(n^4)$

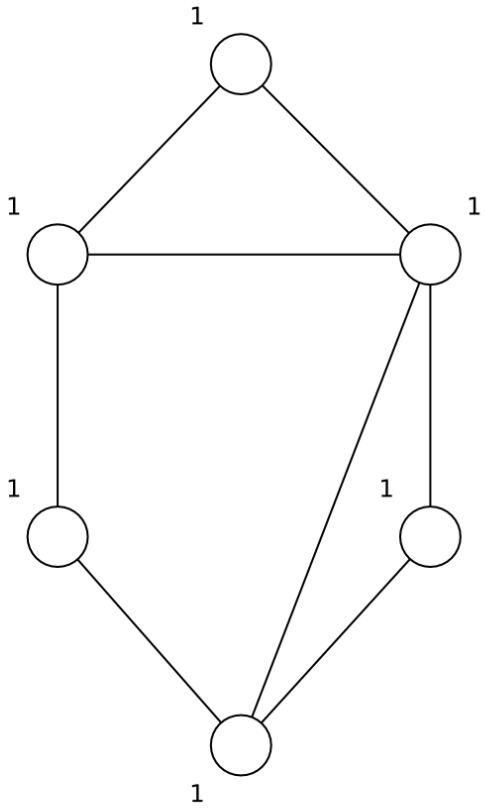
Problems:

- Very high complexity for large graphs
- Shortest-path graphs may lead to memory problems on large graphs

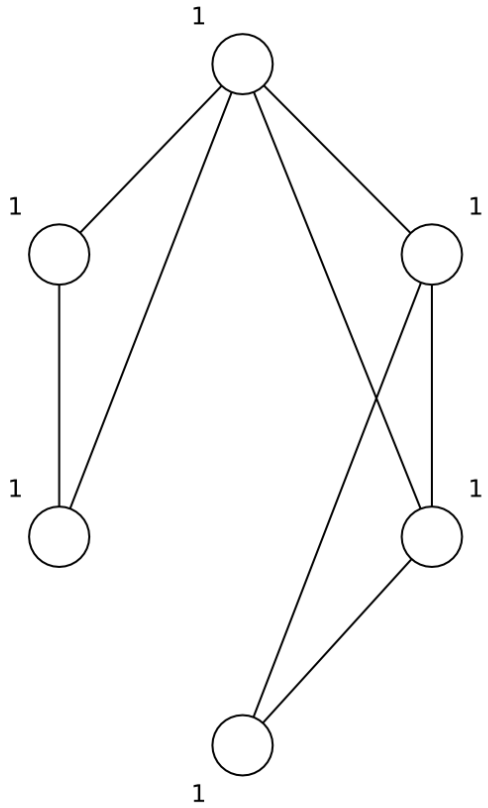
Weisfeiler-Lehman Test of Isomorphism

May answer if two graphs are not isomorphic

Run the Weisfeiler-Lehman algorithm for the following pair of graphs



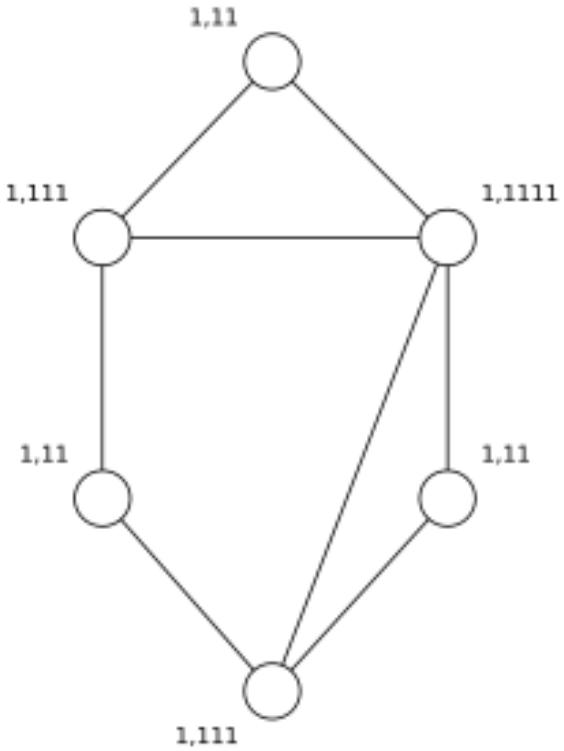
G_1



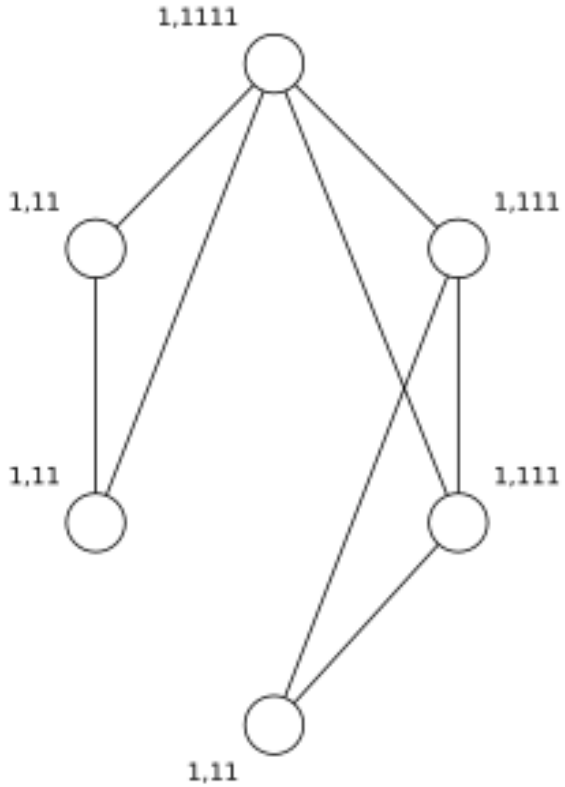
G_2

Weisfeiler-Lehman Test of Isomorphism

First step: Augment the labels of the vertices by the sorted set of labels of neighbouring vertices



G_1

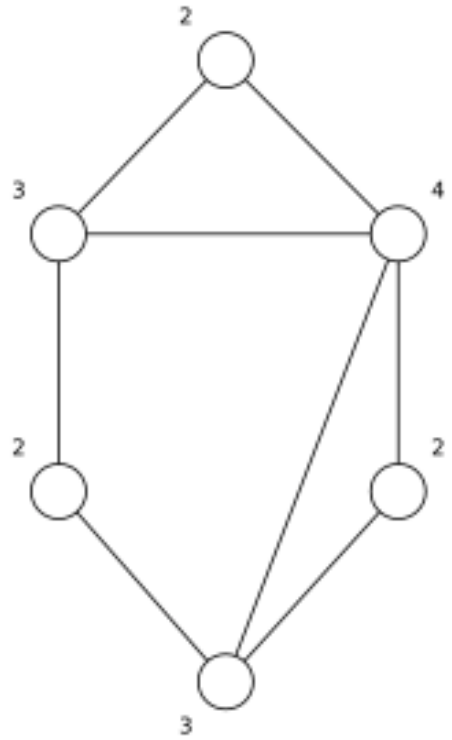


G_2

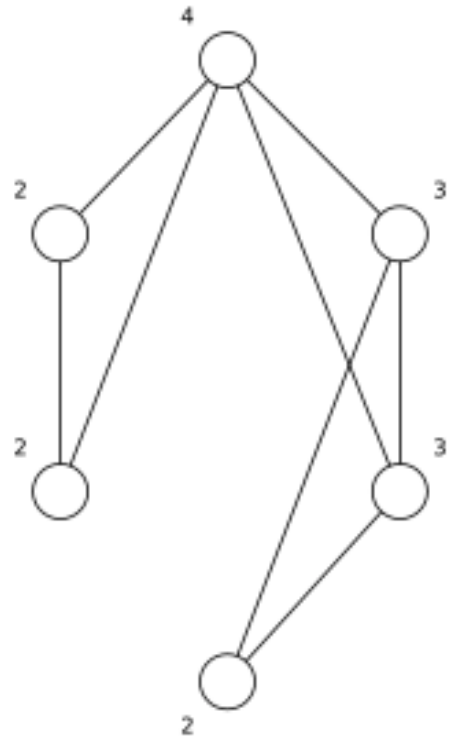
Weisfeiler-Lehman Test of Isomorphism

Second step: Compress the augmented labels into new, short labels:

- $1, 11 \rightarrow 2$
- $1, 111 \rightarrow 3$
- $1, 1111 \rightarrow 4$



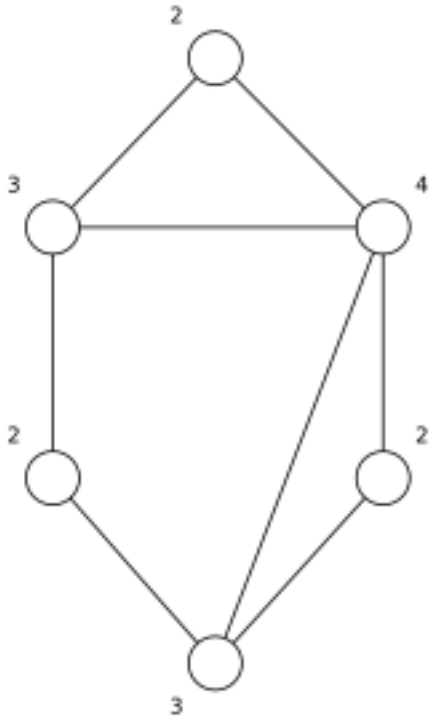
G_1



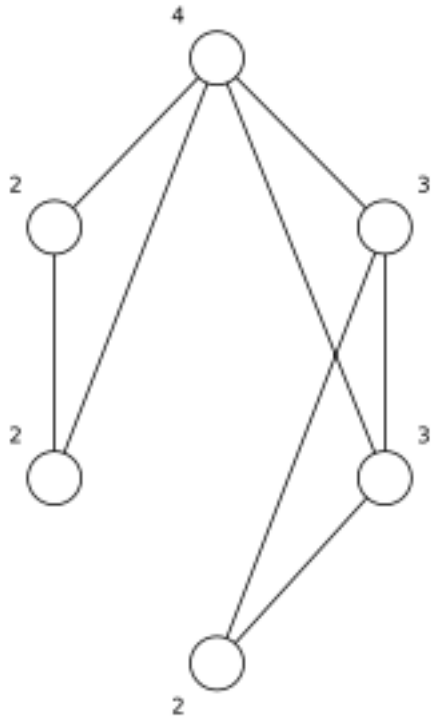
G_2

Weisfeiler-Lehman Test of Isomorphism

Are the label sets of G_1 and G_2 identical?



G_1



G_2

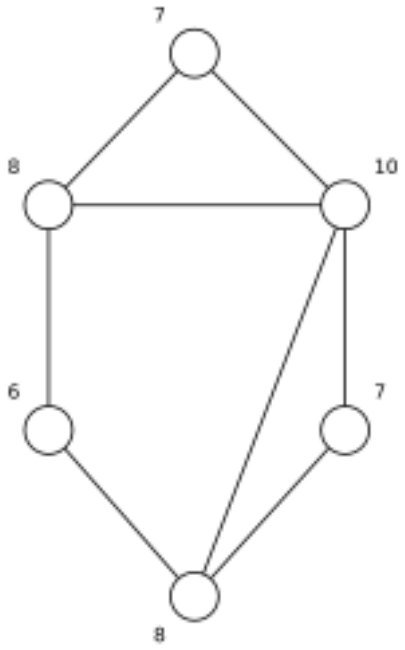
Yes!!!

Continue to the next iteration

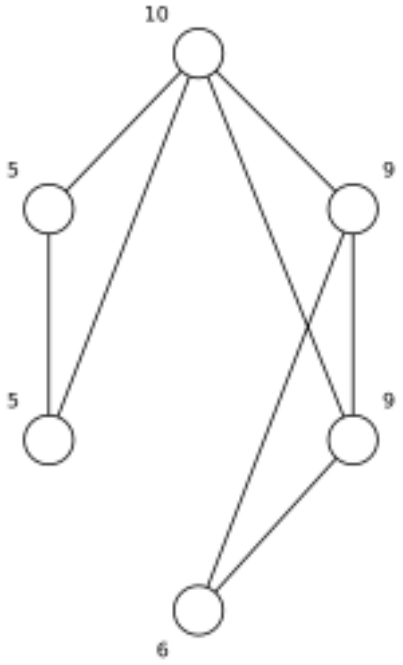
Weisfeiler-Lehman Test of Isomorphism

Second step: Compress the augmented labels into new, short labels:

- $2, 24 \rightarrow 5$
- $2, 33 \rightarrow 6$
- $2, 34 \rightarrow 7$
- $3, 234 \rightarrow 9$
- $4, 2233 \rightarrow 10$



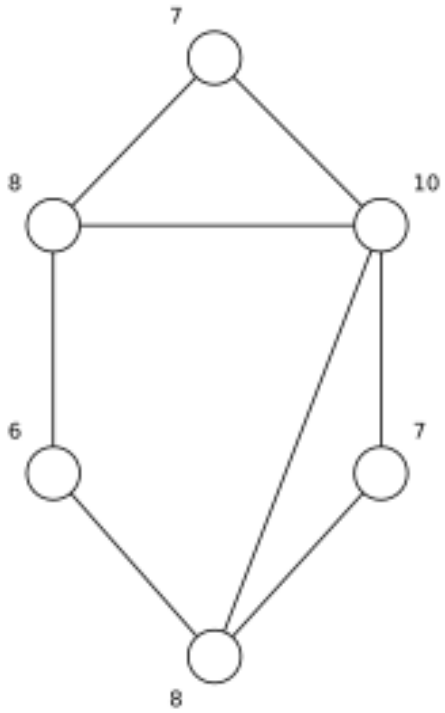
G_1



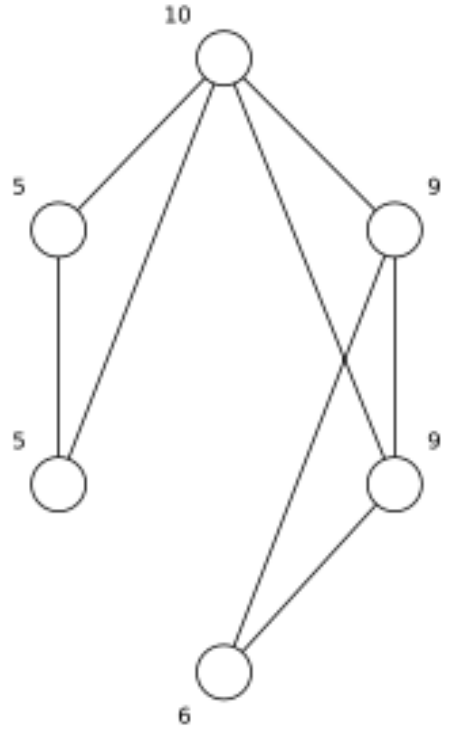
G_2

Weisfeiler-Lehman Test of Isomorphism

Are the label sets of G_1 and G_2 identical?



G_1



G_2

No!!!

Graphs are not isomorphic

Weisfeiler-Lehman Test of Isomorphism

Let $G^1, G^2, \ldots, G^h$ be the graphs emerging from graph G at the iteration $1, 2, \ldots, h$ of the Weisfeiler-Lehman algorithm

Then, the Weisfeiler-Lehman kernel is defined as:

$$k^h_{WL}(G_1, G_2) = k(G_1, G_2) + k(G^1_1, G^1_2) + k(G^2_1, G^2_2) + \ldots + k(G^h_1, G^h_2)$$

where $k(\cdot, \cdot)$ is a base kernel (e.g. subtree kernel, shortest path kernel, ...)

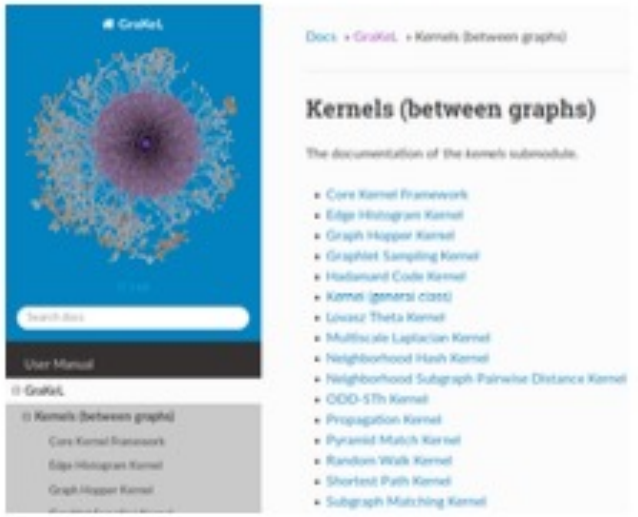
At each iteration of the Weisfeiler-Lehman algorithm:

- run a graph kernel for labeled graphs
- the new kernel values are added to the ones of the previous iteration

[Shervashidze et al., JMLR 12.Sep (2011)]

Grakel Python Library

- Python library for graph kernels
- Contains implementations of a large number of graph kernels
- Compatible with scikit-learn
- Project repository:
  `https://github.com/ysig/GraKeL`



[Siglidis et al., JMLR 2021]

[Nikolentzos et al., JAIR 2021]

Grakel – Evaluation (Node-Labeled Graphs)

Kernels	DATASETS			
	MUTAG	ENZYMES	NCI1	PTC-MR
Vertex Histogram	71.87 (± 1.83)	16.87 (± 1.56)	56.09 (± 0.35)	58.09 (± 0.62)
Random Walk	82.24 (± 2.87)	12.90 (± 1.42)	TIMEOUT	51.26 (± 2.30)
Shortest Path	82.54 (± 1.00)	40.13 (± 1.34)	72.25 (± 0.28)	59.26 (± 2.34)
WL Subtree	84.00 (± 1.25)	53.15 (± 1.22)	85.03 (± 0.20)	63.28 (± 1.34)
WL Shortest Path	82.29 (± 1.93)	28.23 (± 1.00)	61.43 (± 0.32)	55.51 (± 1.68)
WL Pyramid Match	88.60 (± 0.95)	57.72 (± 0.84)	85.31 (± 0.42)	64.52 (± 1.36)
Neighborhood Hash	87.74 (± 1.17)	43.43 (± 1.45)	74.81 (± 0.37)	60.50 (± 2.10)
Neighborhood Subgraph Pairwise Distance	82.46 (± 1.55)	41.97 (± 1.66)	74.36 (± 0.31)	60.04 (± 1.15)
Ordered DAGs Decomposition	79.01 (± 2.04)	31.87 (± 1.35)	75.03 (± 0.45)	59.08 (± 1.85)
Pyramid Match	84.72 (± 1.67)	42.67 (± 1.78)	73.11 (± 0.49)	57.99 (± 2.45)
GraphHopper	82.11 (± 2.13)	36.47 (± 2.13)	71.36 (± 0.13)	55.64 (± 2.03)
Subgraph Matching	84.04 (± 1.55)	35.68 (± 0.80)	TIMEOUT	57.91 (± 1.73)
Propagation	77.23 (± 1.22)	44.48 (± 1.63)	82.12 (± 0.22)	59.30 (± 1.24)
Multiscale Laplacian	86.11 (± 1.60)	53.08 (± 1.53)	79.40 (± 0.47)	59.95 (± 1.71)
CORE WL	85.90 (± 1.44)	52.37 (± 1.29)	85.12 (± 0.21)	63.03 (± 1.67)
CORE Shortest Path	85.13 (± 2.46)	41.55 (± 1.66)	73.87 (± 0.19)	58.21 (± 1.87)

Kernels	DATASETS			Avg. Rank
	D&D	PROTEINS	AIDS	
Vertex Histogram	74.83 (± 0.40)	70.93 (± 0.28)	79.78 (± 0.13)	13.7
Random Walk	OUT-OF-MEM	69.31 (± 0.29)	79.52 (± 0.58)	15.0
Shortest Path	78.93 (± 0.53)	75.92 (± 0.35)	99.41 (± 0.12)	6.7
WL Subtree	78.88 (± 0.46)	75.45 (± 0.33)	98.51 (± 0.05)	4.8
WL Shortest Path	75.66 (± 0.42)	71.88 (± 0.22)	99.36 (± 0.02)	11.8
WL Pyramid Match	OUT-OF-MEM	75.63 (± 0.49)	99.37 (± 0.04)	2.1
Neighborhood Hash	76.02 (± 0.94)	75.55 (± 1.00)	99.54 (± 0.02)	5.0
Neighborhood Subgraph Pairwise Distance	78.76 (± 0.56)	73.17 (± 0.76)	98.04 (± 0.20)	8.0
Ordered DAGs Decomposition	75.82 (± 0.54)	70.49 (± 0.64)	90.75 (± 0.30)	11.4
Pyramid Match	76.98 (± 0.84)	71.90 (± 0.79)	99.56 (± 0.08)	8.2
GraphHopper	TIMEOUT	74.19 (± 0.42)	99.57 (± 0.02)	9.6
Subgraph Matching	OUT-OF-MEM	OUT-OF-MEM	91.96 (± 0.18)	11.2
Propagation	78.43 (± 0.55)	72.71 (± 0.62)	96.51 (± 0.38)	8.4
Multiscale Laplacian	78.28 (± 0.99)	73.89 (± 0.93)	98.48 (± 0.12)	6.0
CORE WL	78.91 (± 0.50)	75.46 (± 0.38)	98.70 (± 0.09)	4.1
CORE Shortest Path	79.33 (± 0.65)	76.31 (± 0.40)	99.47 (± 0.05)	5.5

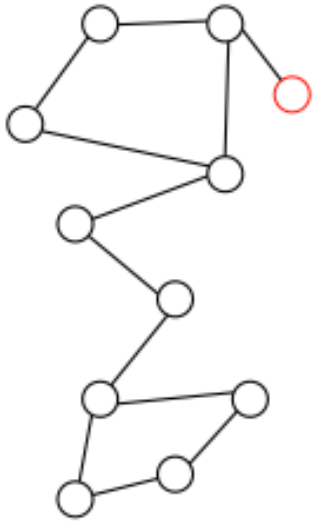
Grakel – Evaluation (Node-Labeled Graphs)

Kernels	DATASETS			
	MUTAG	ENZYMES	NCI1	PTC-MR
Vertex Histogram	0.01s	0.04s	0.84s	0.02s
Random Walk	1m 46.86s	4h 24m 16.26s	TIMEOUT	6m 41.20s
Shortest Path	0.92s	11.03s	1m 9.69s	1.52s
WL Subtree	0.21s	3.81s	7m 5.33s	0.55s
WL Shortest Path	7.02s	1m 27.07s	15m 29.50s	12.55s
WL Pyramid Match	3m 42.07s	1h 5m 37.26s	13h 31m 34.36s	11m 8.16s
Neighborhood Hash	0.40s	11.17s	7m 4.54s	1.31s
Neighborhood Subgraph Pairwise Distance	4.05s	27.02s	6m 9.81s	7.66s
Ordered DAGs Decomposition	1.54s	50.05s	46m 2.13s	4.03s
Pyramid Match	2.59s	31.38s	37m 37.50s	11.35s
GraphHopper	24.70s	15m 38.33s	3h 45m 8.31s	1m 33.90s
Subgraph Matching	1m 57.25s	3h 25m 43.59s	TIMEOUT	4m 19.80s
Propagation	0.48s	12.05s	10m 27.83s	1.81s
Multiscale Laplacian	10m 3.15s	56m 43.76s	5h 30m 56.29s	19m 22.43s
CORE WL	0.55s	12.52s	14m 30.56s	17m 2.27s
CORE Shortest Path	2.69s	48.02s	3m 16.54s	3.97s

Kernels	DATASETS			Avg. Rank
	D&D	PROTEINS	AIDS	
Vertex Histogram	0.24s	0.10s	0.25s	1.0
Random Walk	OUT-OF-MEM	51m 10.11s	1h 51m 56.47s	13.6
Shortest Path	55m 58.79s	1m 18.91s	13.93s	4.4
WL Subtree	5m 52.96s	32.48s	40.49s	2.8
WL Shortest Path	7h 27m 21.90s	8m 3.68s	1m 33.46s	10.1
WL Pyramid Match	OUT-OF-MEM	5h 37m 10.33s	5h 55m 20.37s	14.6
Neighborhood Hash	6m 17.21s	41.81s	33.30s	3.5
Neighborhood Subgraph Pairwise Distance	4h 36m 28.97s	9m 9.80s	1m 12.31s	8.1
Ordered DAGs Decomposition	27m 59.18s	4m 7.81s	2m 5.32s	8.7
Pyramid Match	5m 48.51s	1m 26.82s	2m 48.04s	8.0
GraphHopper	TIMEOUT	3h 43m 1.54s	38m 51.78s	12.1
Subgraph Matching	OUT-OF-MEM	OUT-OF-MEM	4h 26m 46.71s	14.0
Propagation	9m 34.30s	51.20s	1m 43.62s	5.5
Multiscale Laplacian	3h 40m 30.72s	2h 20m 39.57s	1h 11m 58.23s	13.2
CORE WL	17m 2.27s	1m 16.74s	54.79s	7.2
CORE Shortest Path	5h 2m 39.71s	3m 31.97s	40.11s	7.2

Deep Learning for nodes

Traditional Node Representation
Representation: row of adjacency matrix



$$\rightarrow \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \dots & 0 \end{pmatrix}$$

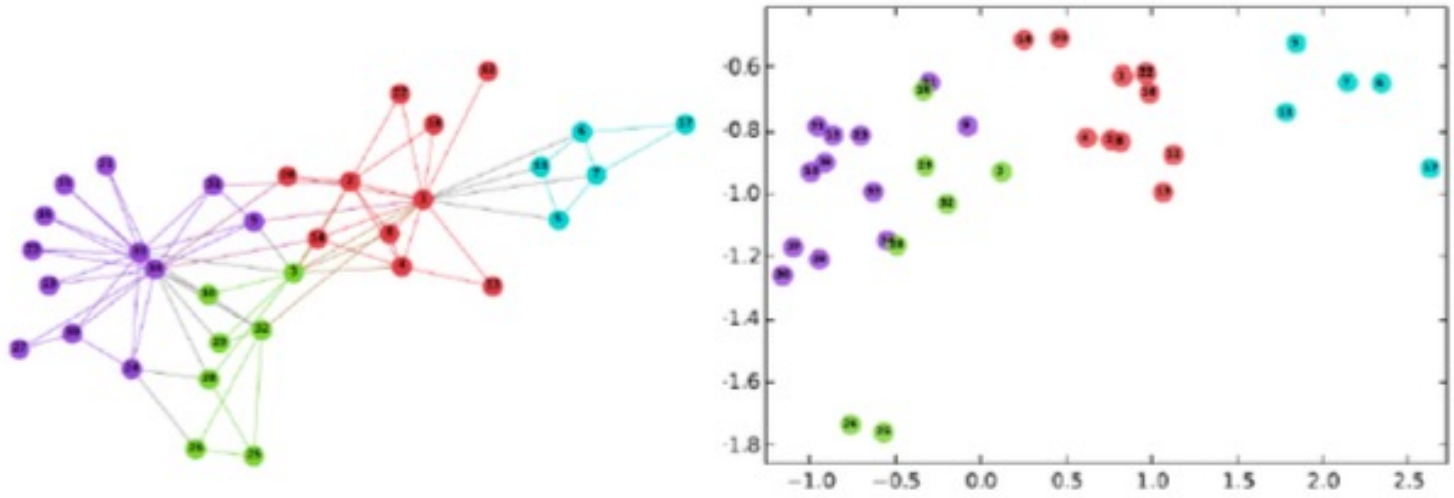
However, such a representation suffers from:

- data sparsity
- high dimensionality

Representation learning for nodes

Map vertices of a graph into a low-dimensional space:

- dimensionality $d \ll |V|$
- similar vertices are embedded close to each other in the low-dimensional space



Input Output

Graph Laplacian Factorization

- Focused mainly on matrix-factorization approaches (e. g., Laplacian eigenmaps)
- Laplacian eigenmaps projects two nodes i and j close to each other when the weight of the edge between the two nodes A_{ij} is high
- Embeddings are obtained by the following objective function:

$$y^* = \arg\min \sum_{i \neq j} (y_i - y_j)^2 A_{ij} = \arg\min y^T L y$$

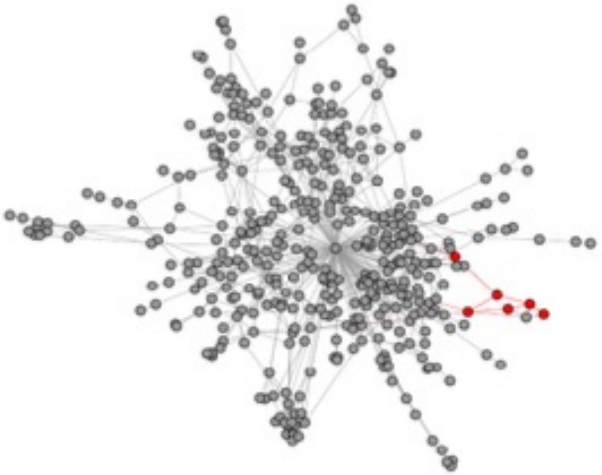
  where L is the graph Laplacian

- The solution is obtained by taking the eigenvectors corresponding to the d smallest eigenvalues of the normalized Laplacian matrix

[1] Belkin and Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In NIPS'02

Deep Walk – Skipgram on Random walks

Inspired by recent advances in language modeling [1]



$\rightarrow$

$v_5 \rightarrow v_8 \rightarrow v_{32} \rightarrow v_{28} \rightarrow v_6 \rightarrow v_{10} \rightarrow v_9$

$v_3 \rightarrow v_5 \rightarrow v_{28} \rightarrow v_8 \rightarrow v_9 \rightarrow v_{10} \rightarrow v_{25}$

$v_{20} \rightarrow v_{10} \rightarrow v_{12} \rightarrow v_6 \rightarrow v_8 \rightarrow v_4 \rightarrow v_5$

$v_{23} \rightarrow v_5 \rightarrow v_{32} \rightarrow v_{10} \rightarrow v_8 \rightarrow v_3 \rightarrow v_1$

$v_4 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5 \rightarrow v_1 \rightarrow v_{12} \rightarrow v_{10}$

$\vdots$

- Simulates a series of short random walks

[1] Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality. In NIPS'13

[2] Perozzi et al. DeepWalk: Online Learning of Social Representations. In KDD'14

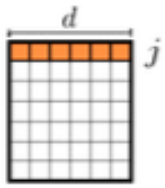
Deep Walk – Skipgram on Random walks

Skipgram is a recently-proposed language model that:

- uses one word to predict the context
- context is composed of words appearing to both the right and left of the given word
- removes the ordering constraint on the problem (i. e. does not take into account the offset of context words from the given word)

In our setting:

$$\mathcal{W}_{v_4} = \begin{matrix} 4 \\ u_k \\ \\ \\ \\ \\ \end{matrix} \begin{bmatrix} 3 \\ 1 \\ 5 \\ 1 \\ \vdots \end{bmatrix} \begin{matrix} \\ v_j \\ \\ \\ \\ \\ \end{matrix}$$

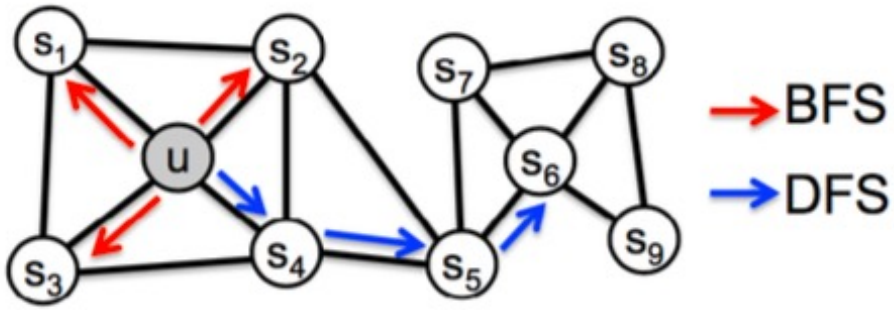


- Slide a window of length $2w + 1$ over the random walk
- Use the representation of central vertex to predict its neighbors

Node2vex – Biased walks

- node2vec is also a random walk based method
- DeepWalk uses a *rigid* search strategy, node2vec simulates biased random walks
- explore diverse neighborhoods of a given vertex
- allow it to learn representations that organize vertices based on - their network roles - the communities they belong to

The *breadth-first sampling* (BFS) and *depth-first sampling* (DFS) represent extreme scenarios in terms of the search space

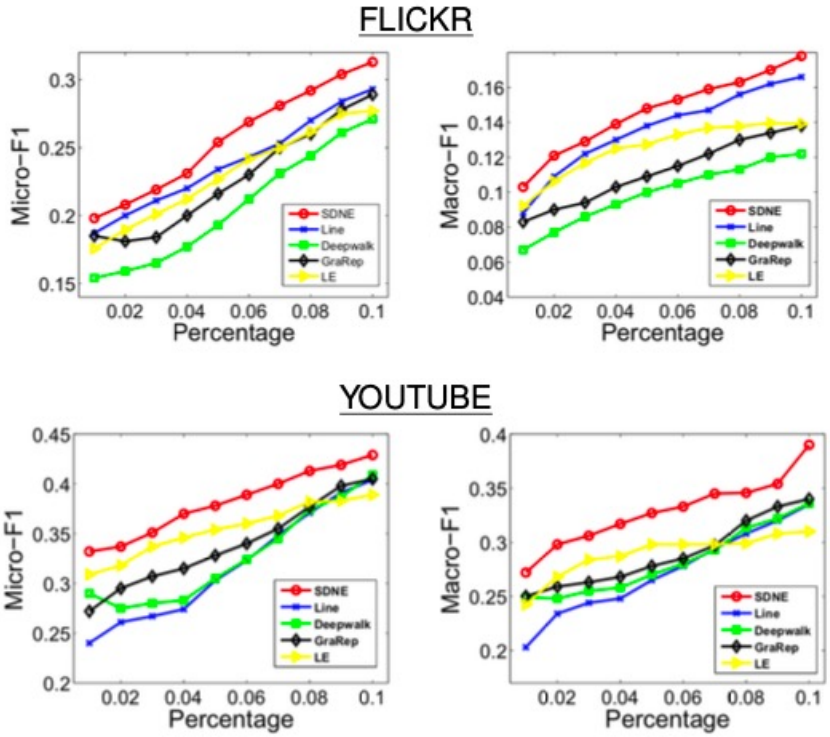


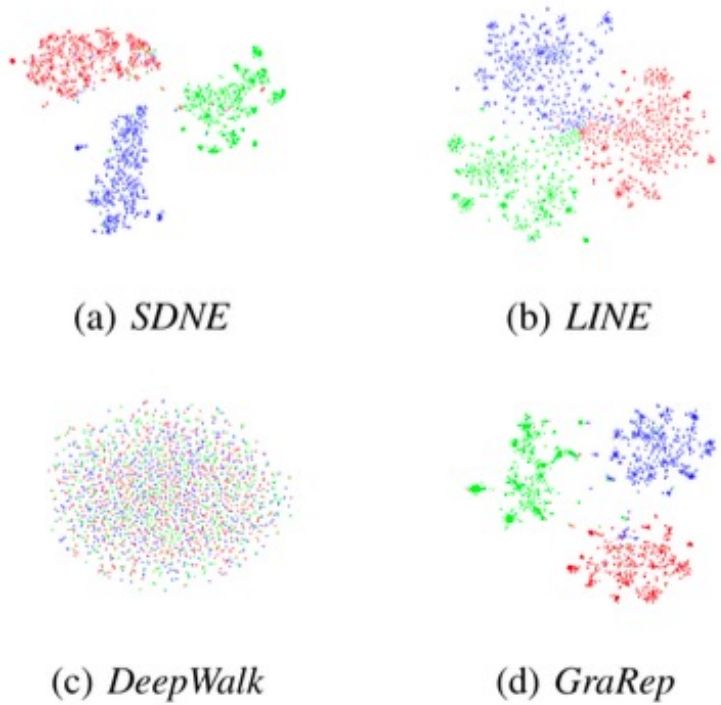
Goal: Given a source node u, sample its neighborhood $\mathcal{N}(u)$ where $|\mathcal{N}(u)| = k$

[1] Grover and Leskovec. node2vec: Scalable Feature Learning for Networks. In KDD'16

More Algorithms

- DeepWalk
- GraRep
- SDNE
- LINE
- Laplacian Eigenmaps (LE)





Visualization of 20-NEWSGROUP

Node Classification

[Wang et al., KDD'16]

THANK YOU!

References (modularity)

- M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review E 69(02), 2004.
- M.E.J. Newman. Modularity and community structure in networks. PNAS, 103(23), 2006.
- S.E. Schaeffer. Graph clustering. Computer Science Review 1(1), 2007.
- S. Fortunato. Community detection in graphs. Physics Reports 486 (3-5), 2010.
- M. Coscia, F. Giannotti, and D. Pedreschi. A classification for community discovery methods in complex networks. Statistical Analysis and Data Mining 4 (5), 2011.
- A. Arenas, J. Duch, A. Fernandez, and S. Gomez. Size reduction of complex networks preserving modularity. New J. Phys., 9(176), 2007.
- M. Girvan and M.E.J. Newman. Community structure in social and biological networks. PNAS 99(12), 2002.
- U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On Modularity Clustering. IEEE TKDE 20(2), 2008.
- M.E.J. Newman. Fast algorithm for detecting community structure in networks. Phys. Rev. E 69, 2004.
- A. Clauset, M.E.J. Newman, and C. Moore. Finding community structure in very large networks. Phys. Rev. E 70, 2004.

References (modularity)

- M.E.J. Newman. Finding community structure in networks using the eigenvectors of matrices. Phys. Rev. E 74, 2006.
- R. Guimera, M. Sales-Pardo, L.A.N. Amaral. Modularity from Fluctuations in Random Graphs and Complex Networks. Phys. Rev. E 70, 2004.
- J. Duch and A. Arenas. Community detection in complex networks using Extremal Optimization. Phys. Rev. E 72, 2005.
- A. Arenas, J. Duch, A. Fernandez, and S. Gomez. Size reduction of complex networks preserving modularity. New Journal of Physics 9(6), 2007.
- E.A. Leicht and M.E.J. Newman. Community structure in directed networks. Phys. Rev. Lett. 100, 2008.
- V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. J. Stat. Mech. 03, 2009.
- S. Muff, F. Rao, A. Caflisch. Local modularity measure for network clusterizations. Phys. Rev. E, 72, 2005.
- S. Fortunato and M. Barthelemy. Resolution limit in community detection. PNAS 104(1), 2007.

References (community evaluation measures)

- M.E.J. Newman. The structure and function of complex networks. SIAM REVIEW 45, 2003.
- M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review E 69(02), 2004.
- S.E. Schaeffer. Graph clustering. Computer Science Review 1(1), 2007.
- S. Fortunato. Community detection in graphs. Physics Reports 486 (3-5), 2010.
- L. Danon, J. Duch, A. Arenas, and A. Diaz-guilera. Comparing community structure identification. Journal of Statistical Mechanics: Theory and Experiment 9008 , 2005.
- M. Coscia, F. Giannotti, and D. Pedreschi. A classification for community discovery methods in complex networks. Statistical Analysis and Data Mining 4 (5), 2011.
- J. Leskovec, K.J. Lang, and M.W. Mahoney. Empirical comparison of algorithms for network community detection. In: WWW, 2010.
- F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. PNAS, 101(9), 2004.
- J. Yang and J. Leskovec. Defining and Evaluating Network Communities based on Ground-Truth. In: ICDM, 2012.
- Fan Chung. Spectral Graph Theory. CBMS Lecture Notes 92, AMS Publications, 1997.