

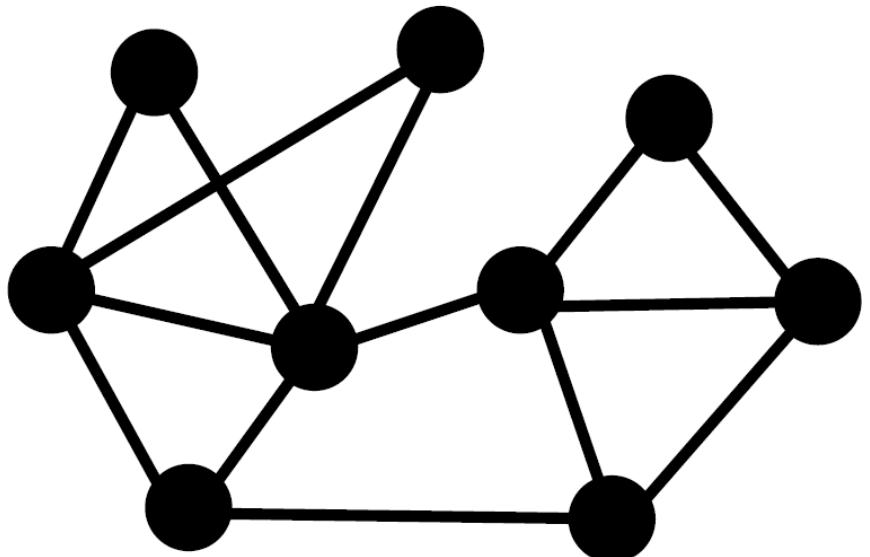
DS 503: Advanced Data Analytics

# Lecture 16: Graph Data

Gagan Raj Gupta

# Why Graphs

Graphs are a general language for describing and analyzing entities with relations/interactions



$G = (V, E)$   
V: Set of Vertices/Nodes  
E: Set of Edges  $(u, v) \ u, v \in V$

Relationship.

Heterogenous.  $\rightarrow$  Node/Edge multiple

# Examples



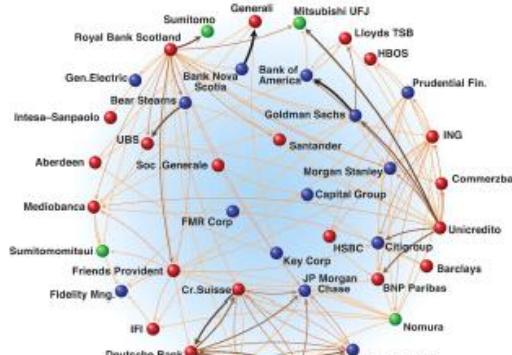
Milind

[Image credit: Medium](#)

## Social Networks



## Citation Networks



[Image credit: Science](#)

## Economic Networks



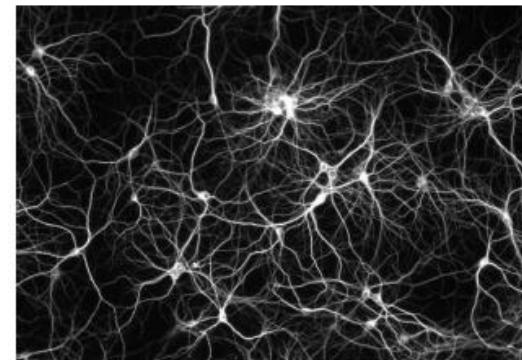
[Image credit: Lumen Learning](#)

## Communication Networks



[Image credit: Missoula Current News](#)

## Internet



[Image credit: The Conversation](#)

## Networks of Neurons

# Examples

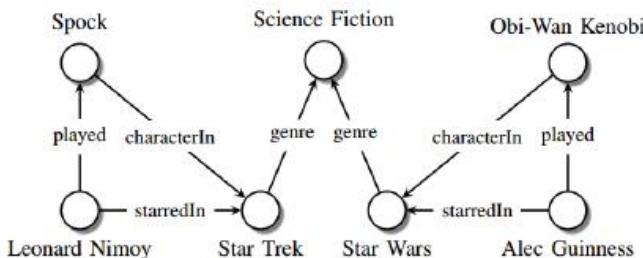


Image credit: [Maximilian Nickel et al](#)

## Knowledge Graphs

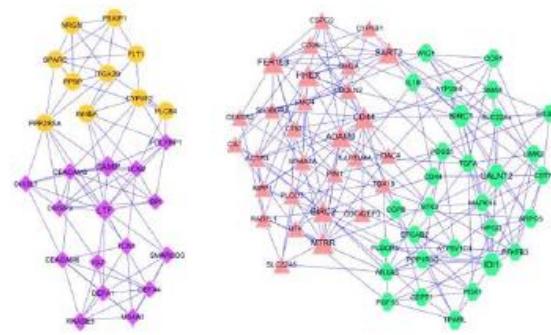


Image credit: [ese.wustl.edu](#)

## Regulatory Networks

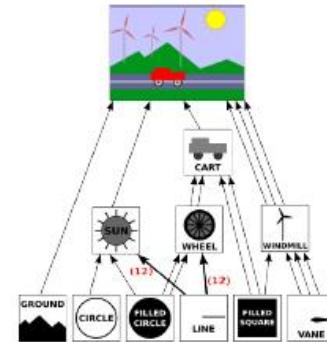


Image credit: [math.hws.edu](#)

## Scene Graphs

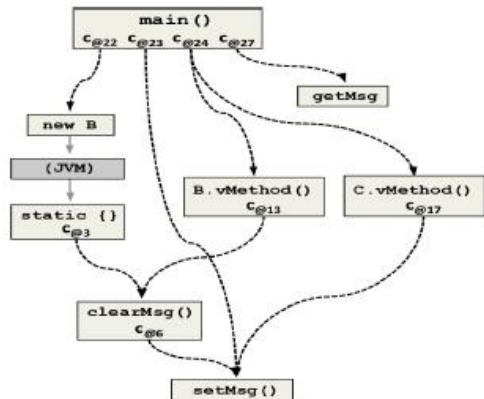


Image credit: [ResearchGate](#)

## Code Graphs

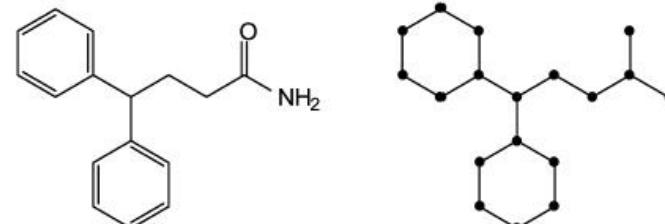


Image credit: [MDPI](#)

## Molecules

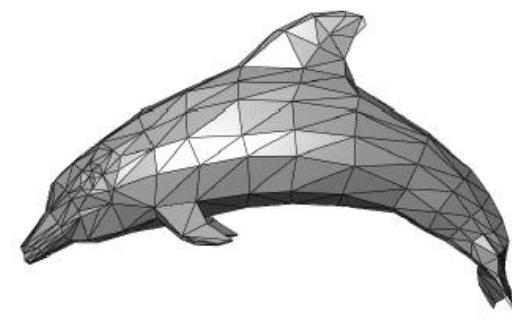
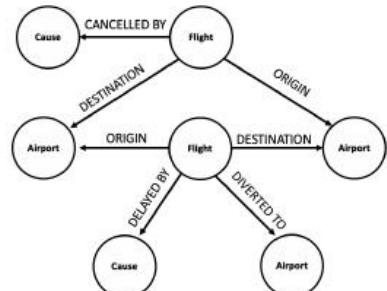


Image credit: [Wikipedia](#)

## 3D Shapes

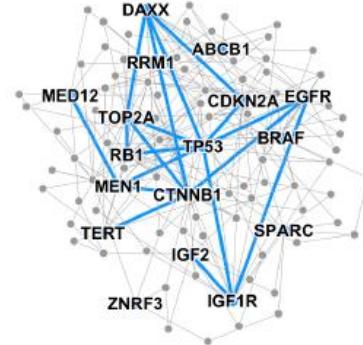
# Examples



## Event Graphs



## Computer Networks



## Disease Pathways

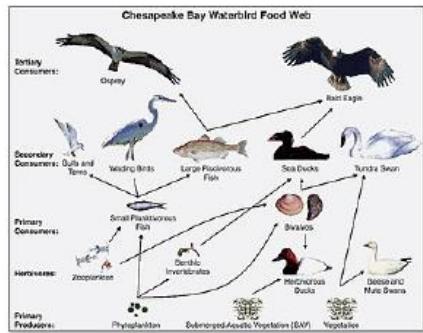


Image credit: [Wikipedia](#)

## Food Webs



Image credit: [Pinterest](#)

## Particle Networks



Image credit: [visitlondon.com](#)

## Underground Networks

# Networks and Graphs

- **Graphs (as a representation):**
- **Information/knowledge** is organized and linked
- **Software** can be represented as a graph
- **Similarity networks:** Connect similar data points
- **Relational structures:** Molecules, Scene graphs, 3D shapes, Particle-based physics simulations

## Networks (also known as Natural Graphs):

- **Social networks:**
  - **Society** is a collection of 7+ billion individuals
- **Communication and transactions:**
  - Electronic devices, phone calls, financial transactions
- **Biomedicine:**
  - Interactions between **genes/proteins** regulate life
- **Brain connections:**

**Sometimes the distinction between networks & graphs is blurred**

# Graphs and Relational Data

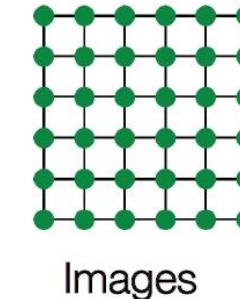
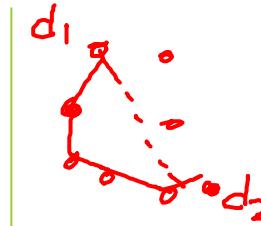
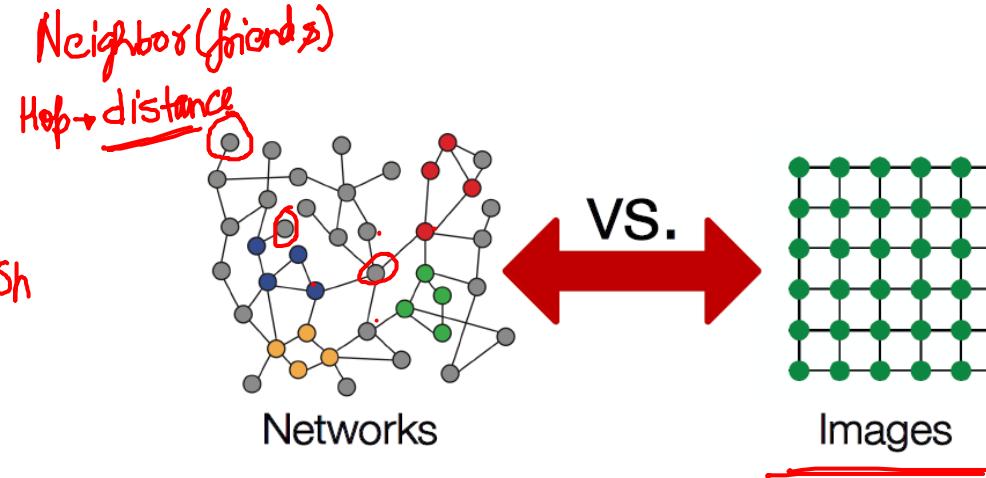
## Main question:

How do we take advantage of relational structure for better prediction?

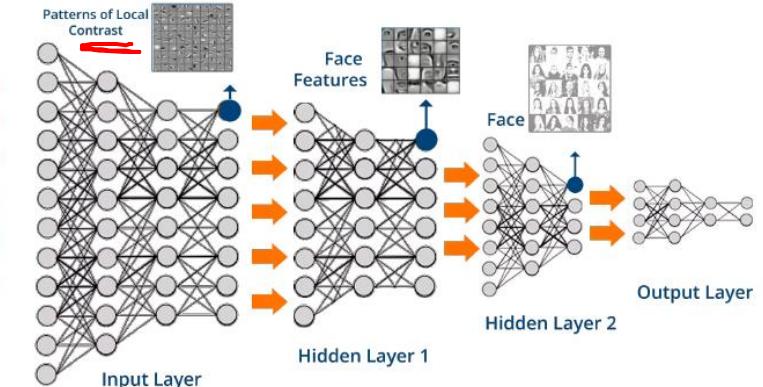
- Complex domains have a rich relational structure, which can be represented as a **relational graph**
- **By explicitly modeling relationships we achieve better performance!**

CNN → Filters { weights }  
of Filter Matrix

# Graph structured data



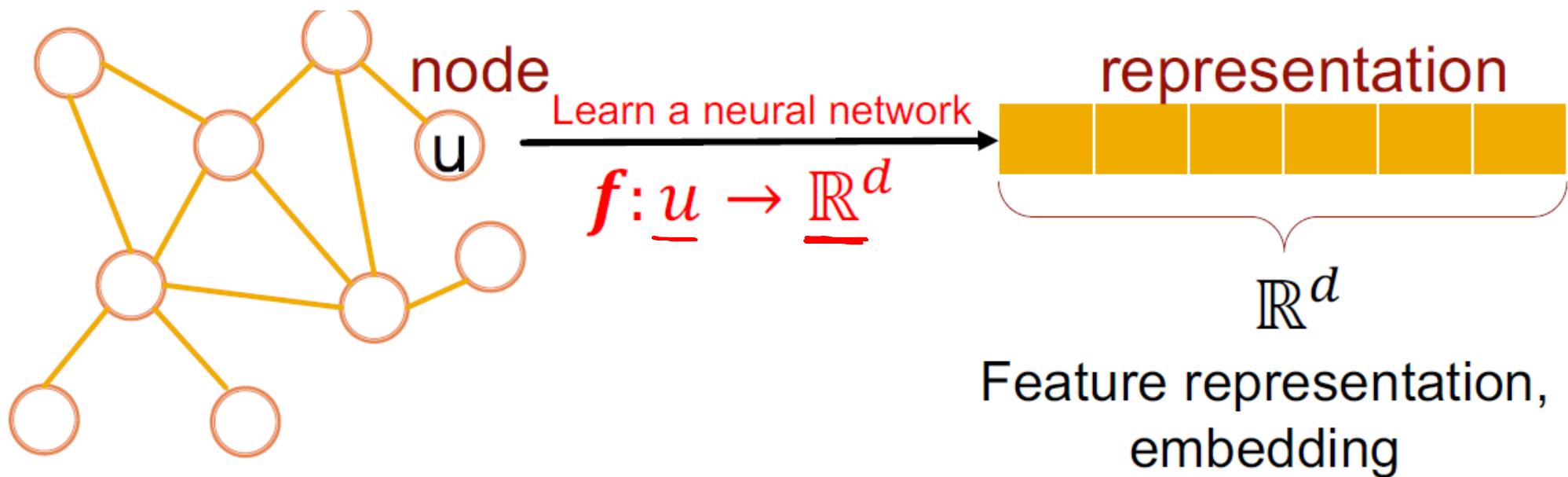
## Modern ML Toolbox



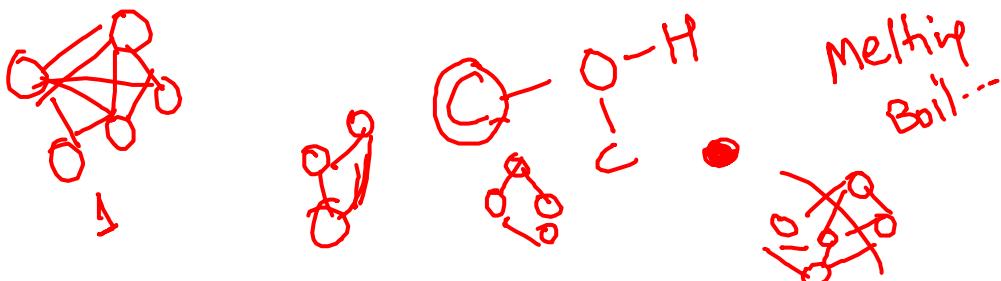
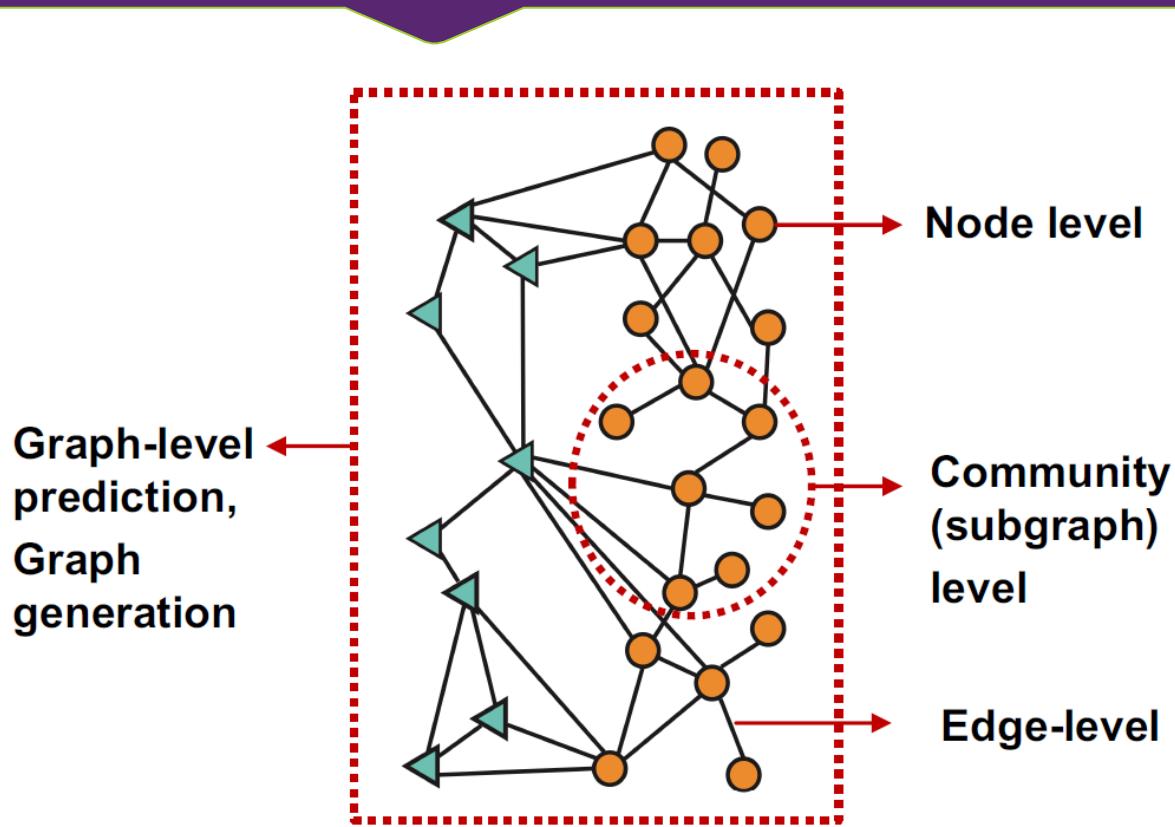
- Modern deep learning toolbox is designed for sequences and grids.
- Graph data  $\{V, E\}$  is far more complex with arbitrary size and complex topological structure (i.e. no spatial locality like grids).
- There is no fixed ordering or reference point. Even simple looking problem of comparing two graphs for equality is NP hard.
- Graph data is often dynamic and have multi-modal features.
- Eg. 3D meshes, social networks, telecommunication, networks, biological networks or brain connectomes.

# Representation Learning

- (Supervised) Machine Learning Lifecycle: Retrain for every new feature. **Every single time!**
- Representation Learning – Automatically learn the features
- Map nodes to d-dimensional embeddings such that **similar nodes in the network** are embedded close together



# Graph ML Tasks



- **Node classification:** Predict a property of a node  
○ Example: Categorize online users / items
- **Link prediction:** Predict whether there are missing links between two nodes  
○ Example: Knowledge graph completion
- **Graph classification:** Categorize different graphs  
○ Example: Molecule property prediction
- **Clustering:** Detect if nodes form a community  
○ Example: Social circle detection
- **Other tasks:**
  - **Graph generation:** Drug discovery
  - **Graph evolution:** Physical simulation

# Defining a graph

- How to build a graph:

- What are nodes?

- What are edges?

- Choice of the proper network representation of a given domain/problem determines our ability to use networks successfully:

- In some cases there is a unique, unambiguous representation

- In other cases, the representation is by no means unique

- The way you assign links will determine the nature of the question you can study

# Graphs from Data Matrix

Many datasets that are not in the form of a graph can still be converted into one.

Let  $\mathbf{D} = \{\mathbf{x}_i\}_{i=1}^n$  (with  $\mathbf{x}_i \in \mathbb{R}^d$ ), be a dataset. Define a weighted graph  $G = (\mathcal{V}, \mathcal{E})$ , with edge weight

$$w_{ij} = \underline{\text{sim}}(\mathbf{x}_i, \mathbf{x}_j)$$

where  $\text{sim}(\mathbf{x}_i, \mathbf{x}_j)$  denotes the similarity between points  $\mathbf{x}_i$  and  $\mathbf{x}_j$ .

For instance, using the Gaussian similarity

$$w_{ij} = \underline{\text{sim}}(\mathbf{x}_i, \mathbf{x}_j) = \underline{\exp} \left\{ -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2} \right\}$$

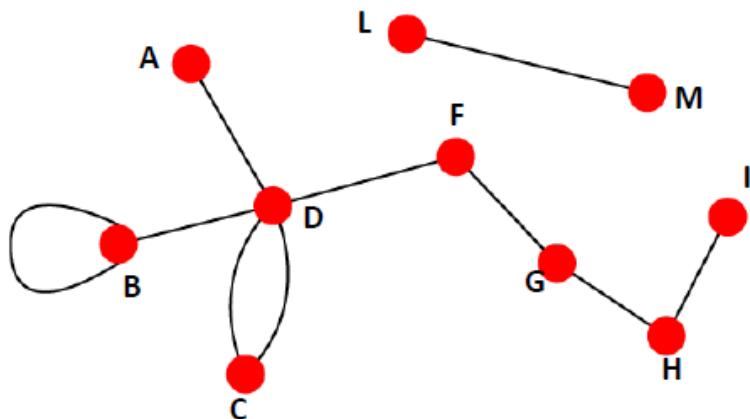
where  $\sigma$  is the spread parameter.

We can add a constraint that an edge exists between i and j only if  $w_{ij}$  exceeds a threshold

# Directed vs Undirected Graphs

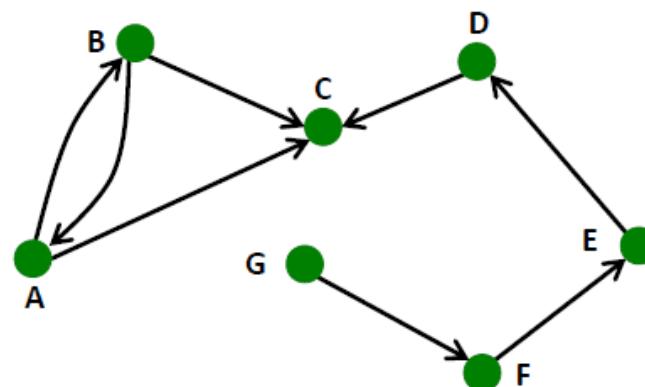
## Undirected

- **Links:** undirected  
(symmetrical, reciprocal)



## Directed

- **Links:** directed  
(arcs)



## Examples:

- Collaborations
- Friendship on Facebook

## Examples:

- Phone calls
- Following on Twitter

# Node Degree



**Node degree,  $k_i$ :** the number of edges adjacent to node  $i$

$$\underline{k_A = 4}$$

$$E = 10$$
$$N = 6$$

**Avg. degree:**  $\bar{k} = \langle k \rangle = \frac{1}{N} \sum_{i=1}^N k_i = \frac{2E}{N}$

In directed networks we define an **in-degree** and **out-degree**.  
The (total) degree of a node is the sum of in- and out-degrees.

$$\underline{k_C^{in} = 2} \quad \underline{k_C^{out} = 1} \quad \underline{k_C = 3}$$

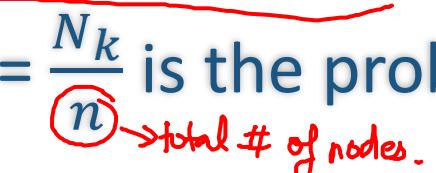
$$\bar{k} = \frac{E}{N}$$

$$\bar{k}^{in} = \bar{k}^{out}$$

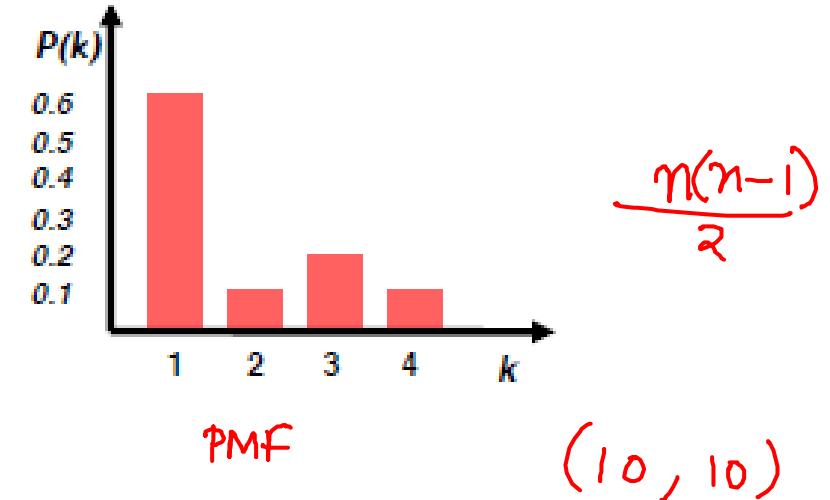
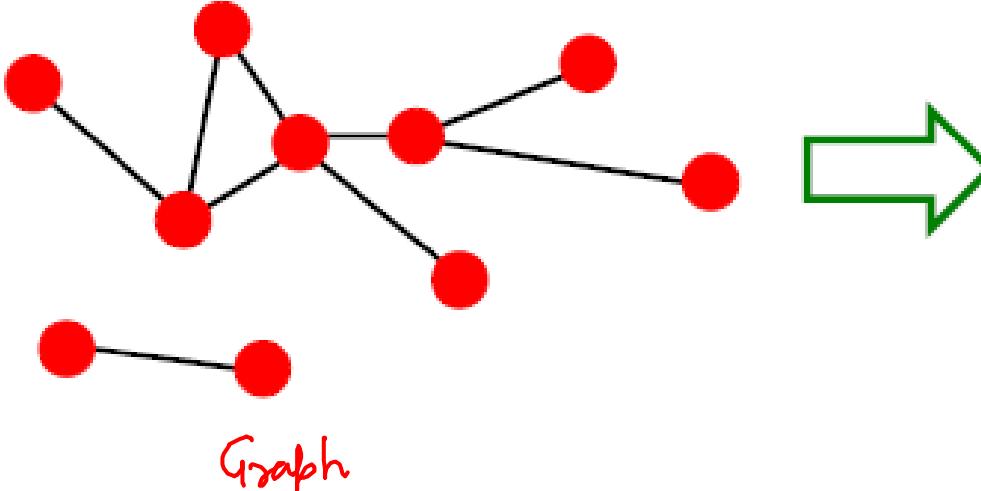
**Source:** Node with  $k^{in} = 0$   
**Sink:** Node with  $k^{out} = 0$

# Degree Distribution

- The degree of a node  $v_i \in V$  is denoted as  $d(v_i)$  or  $d_i$  or  $k_i$ .
- The degree sequence of a graph is the list of the degrees of the nodes sorted in non-increasing order.
- Let  $\underline{N_k}$  denote the number of vertices with degree k. The degree frequency distribution of a graph is given as  $(\underline{N_0}, \underline{N_1}, \dots, \underline{N_t})$  where t is the maximum degree for a node in G.  

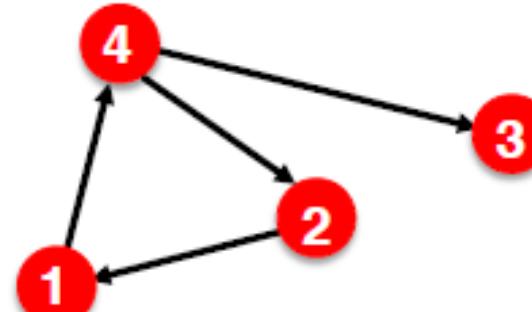
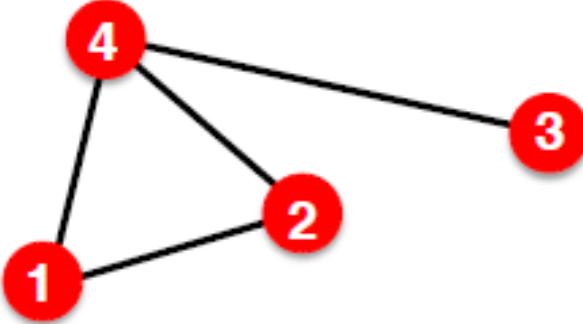
- Let X be a random variable denoting the degree of a node.
- The degree distribution of a graph gives the probability mass function f for X, given as  $f(0), f(1), \dots, f(t)$  where  $f(k) = \frac{\underline{N_k}}{\underline{n}}$  is the probability that a randomly chosen node has degree k.  


# Degree distribution example



- The degree sequence of the graph is:  $(4, 3, 3, 2, 1, 1, 1, 1, 1, 1)$
- The degree frequency distribution is:  $(N_0, N_1, N_2, N_3, N_4) = (0, 6, 1, 2, 1)$
- The degree distribution is given as:  $(f(0), f(1), f(2), f(3), f(4)) = (0, 0.125, 0.375, 0.125, 0.375)$
- Note: For directed graphs, we can have different in- and out- degree distributions

# Representing Graphs: Adjacency Matrix



Some graphs may allow self-loops  
Eg: Proteins, Hyperlinks

$A_{ij} = 1$  if there is a link from node  $i$  to node  $j$

$A_{ij} = 0$  otherwise

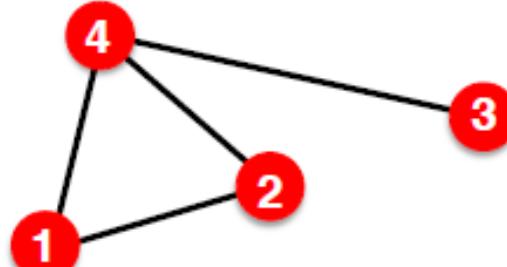
$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 \end{pmatrix}$$

Note that for a directed graph (right) the matrix is not symmetric.

# Adjacency Matrix

Undirected



$$A_{ij} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\begin{aligned} A_{ij} &= A_{ji} \\ A_{ii} &= 0 \end{aligned}$$

$$k_i = \sum_{j=1}^N A_{ij}$$

Row Sum=Degree of i

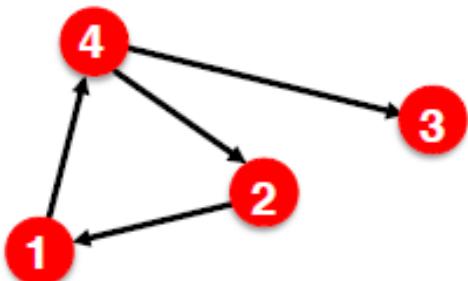
$$k_j = \sum_{i=1}^N A_{ij}$$

Col Sum=Degree of j

$$L = \frac{1}{2} \sum_{i=1}^N k_i = \frac{1}{2} \sum_{i,j} A_{ij}$$

Total Number of Edges

Directed



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

$$\begin{aligned} A_{ij} &\neq A_{ji} \\ A_{ii} &= 0 \end{aligned}$$

$$k_i^{out} = \sum_{j=1}^N A_{ij}$$

Row Sum=Out-Degree of i

$$k_j^{in} = \sum_{i=1}^N A_{ij}$$

Col Sum=In-Degree of j

$$L = \sum_{i=1}^N k_i^{in} = \sum_{j=1}^N k_j^{out} = \sum_{i,j} A_{ij}$$

Total Number of Edges

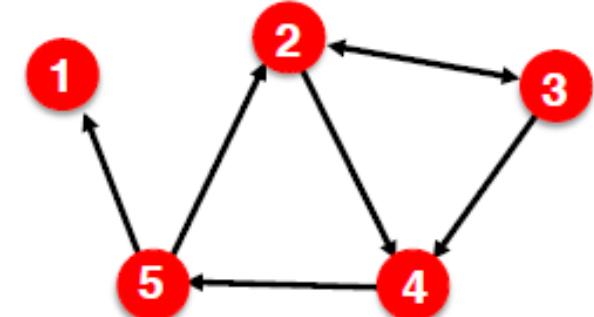
# Adjacency Matrices are Sparse for Real World Graphs

- Most real world networks are sparse.  $E \ll E_{max} = \frac{N}{2}(N - 1)$  or  $\bar{k} \ll N - 1$
- Consequence: Adjacency matrix is filled with zeros!
- Density of the matrix  $\frac{E}{N^2}$  : (WWW=  $1.51 \times 10^{-5}$ , MSN IM= $2.27 \times 10^{-8}$ )

| NETWORK               | NODES                      | LINKS                | DIRECTED/<br>UNDIRECTED | N       | L          | $\langle k \rangle$ |
|-----------------------|----------------------------|----------------------|-------------------------|---------|------------|---------------------|
| Internet              | Routers                    | Internet connections | Undirected              | 192,244 | 609,066    | 6.33                |
| WWW                   | Webpages                   | Links                | Directed                | 325,729 | 1,497,134  | 4.60                |
| Power Grid            | Power plants, transformers | Cables               | Undirected              | 4,941   | 6,594      | 2.67                |
| Phone Calls           | Subscribers                | Calls                | Directed                | 36,595  | 91,826     | 2.51                |
| Email                 | Email Addresses            | Emails               | Directed                | 57,194  | 103,731    | 1.81                |
| Science Collaboration | Scientists                 | Co-authorship        | Undirected              | 23,133  | 93,439     | 8.08                |
| Actor Network         | Actors                     | Co-acting            | Undirected              | 702,388 | 29,397,908 | 83.71               |
| Citation Network      | Paper                      | Citations            | Directed                | 449,673 | 4,689,479  | 10.43               |
| E. Coli Metabolism    | Metabolites                | Chemical reactions   | Directed                | 1,039   | 5,802      | 5.58                |
| Protein Interactions  | Proteins                   | Binding interactions | Undirected              | 2,018   | 2,930      | 2.90                |

# Representing Graphs : Edge List

- For large sparse graphs, it is more efficient to represent the graphs as a list of edges
- Allows us to quickly retrieve all the neighbors of a given node
  - 1:
  - 2: 3,4
  - 3: 2,4
  - 4: 5
  - 5: 1,2



# Node and Edge Attributes

## ○ Possible options:

- Weight (e.g., frequency of communication)
- Ranking (best friend, second best friend...)
- Type (friend, relative, co-worker)
- Sign: Friend vs. Foe, Trust vs. Distrust
- Properties depending on the structure of the rest of the graph:  
Number of common friends

Total time, Total # of Packets

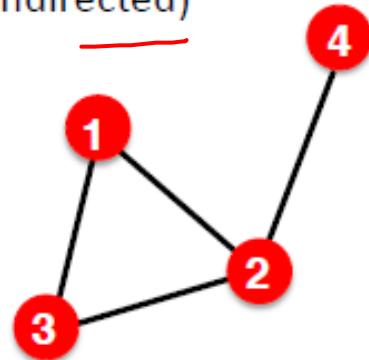
Weak +, -  
Strong +, -  
No correlation

| Correlation



# More Types of Graphs

## ■ Unweighted (undirected)



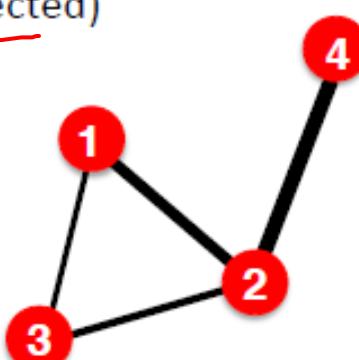
$$A_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$A_{ii} = 0 \quad A_{ij} = A_{ji}$$

$$E = \frac{1}{2} \sum_{i,j=1}^N A_{ij} \quad \bar{k} = \frac{2E}{N}$$

Examples: Friendship, Hyperlink

## ■ Weighted (undirected)



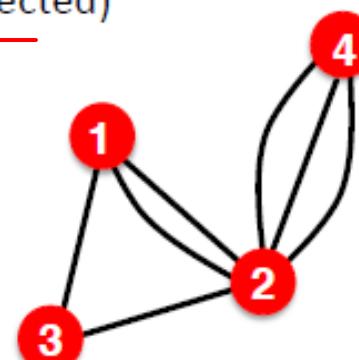
$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

$$A_{ii} = 0 \quad A_{ij} = A_{ji}$$

$$E = \frac{1}{2} \sum_{i,j=1}^N \text{nonzero}(A_{ij}) \quad \bar{k} = \frac{2E}{N}$$

Examples: Collaboration, Internet, Roads

## ■ Multigraph (undirected)



$$A_{ij} = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 3 \\ 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{pmatrix}$$

$$A_{ii} = 0 \quad A_{ij} = A_{ji}$$

$$E = \frac{1}{2} \sum_{i,j=1}^N \text{nonzero}(A_{ij}) \quad \bar{k} = \frac{2E}{N}$$

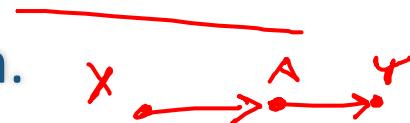
Examples: Communication, Collaboration

IQ 2.3: Calculate the formula for E if the graph has self-loops

# Path and Distance

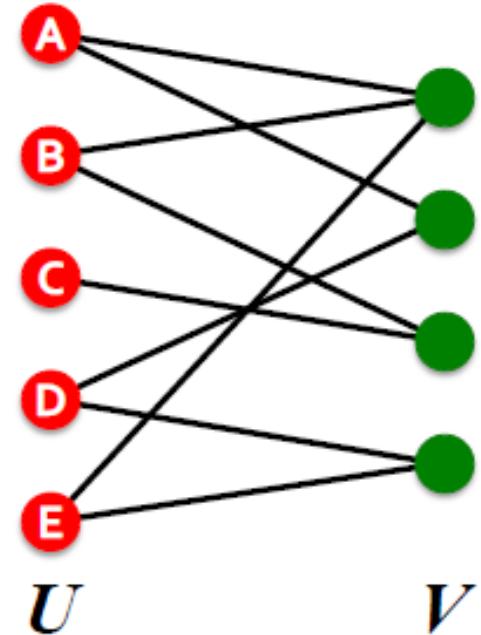


- **Walk:** in a G between nodes x and y is an ordered sequence of vertices, starting at x and ending at y,  $x = v_0, v_1, \dots, v_{t-1}, v_t = y$ , such that there is an edge between every pair of consecutive vertices, that is,  $(v_{i-1}, v_i) \in E, \forall i = 1, 2, \dots, t$ 
  - Length of the walk,  $t = \text{hops} = \text{number of edges along the walk}$
- **Path:** is a walk with distinct vertices (with the exception of the start and end vertices, which are called cycles).
  - A path of minimum  $\sum w_i$  length between nodes x and y is called a shortest path
  - Length of the shortest path = distance between x and y, denoted as  $d(x, y)$  or  $h_{xy}$ .
  - If no path exists between the two nodes, the distance is assumed to be  $d(x, y) = \infty$ .
- Two nodes x and y are connected if there exists a path between them.
- In directed graphs, paths need to follow edge directions:  $d(x, y) \neq d(y, x)$



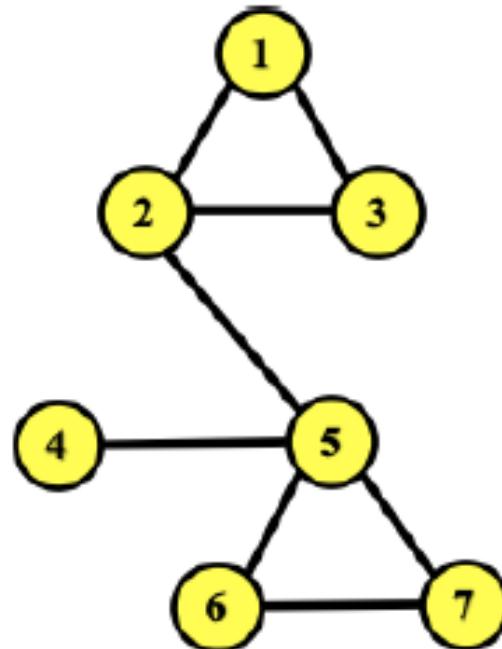
# Bipartite Graph

- **Bipartite graph** is a graph whose nodes can be divided into two disjoint sets  $U$  and  $V$  such that every link connects a node in  $U$  to one in  $V$ ; that is,  $U$  and  $V$  are **independent sets**
- **Examples:**
  - Authors-to-Papers (they authored)
  - Actors-to-Movies (they appeared in)
  - Users-to-Movies (they rated)
  - Recipes-to-Ingredients (they contain)
- **“Folded” networks:**
  - Author collaboration networks
  - Movie co-rating networks



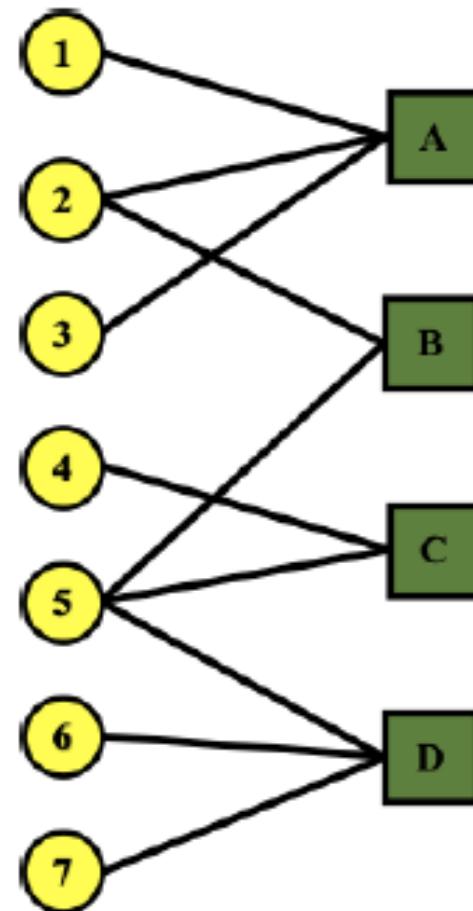
# Folded/Projected Bipartite Graphs

Projection U

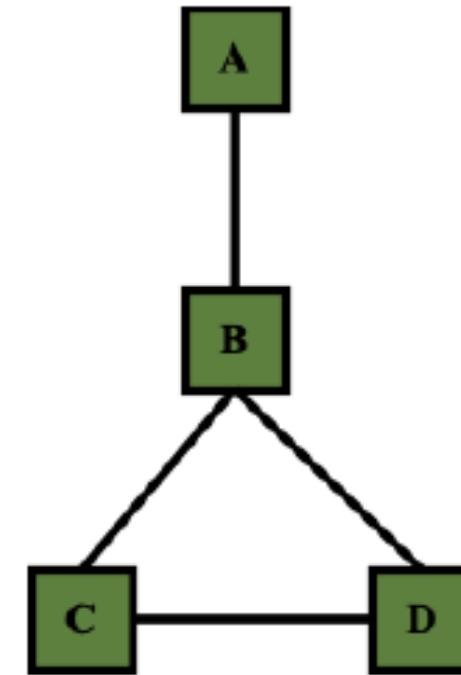


1 is connected to 2 via A  
1 is connected to 5 via A,2,B

U V



Projection V

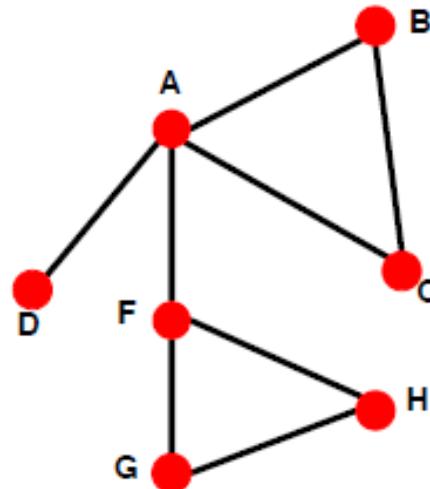


A is connected to B via 2  
A is connected to D via 2,B,5

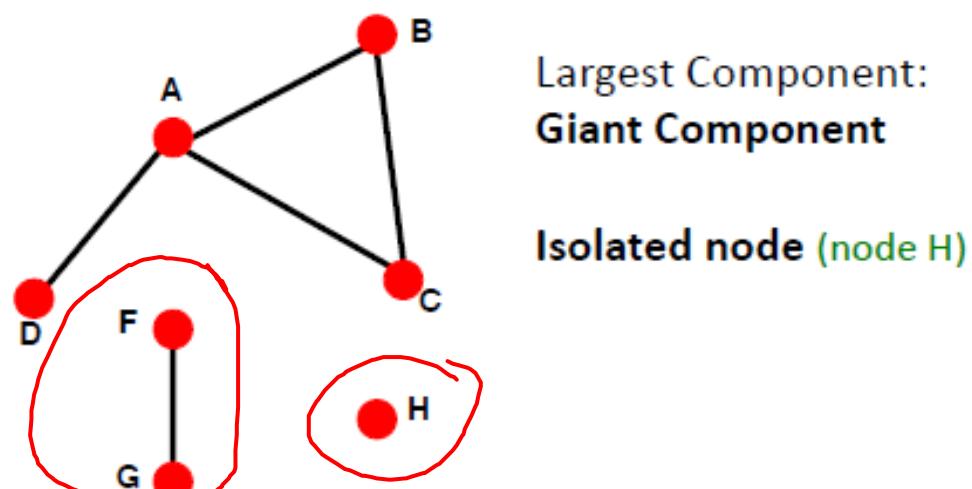
# Connectivity of Undirected Graphs

## ○ Connected (undirected) graph:

- Any two vertices can be joined by a path
- A disconnected graph is made up by two or more connected components



Connected graph



Dis-connected graph

Largest Component:  
Giant Component

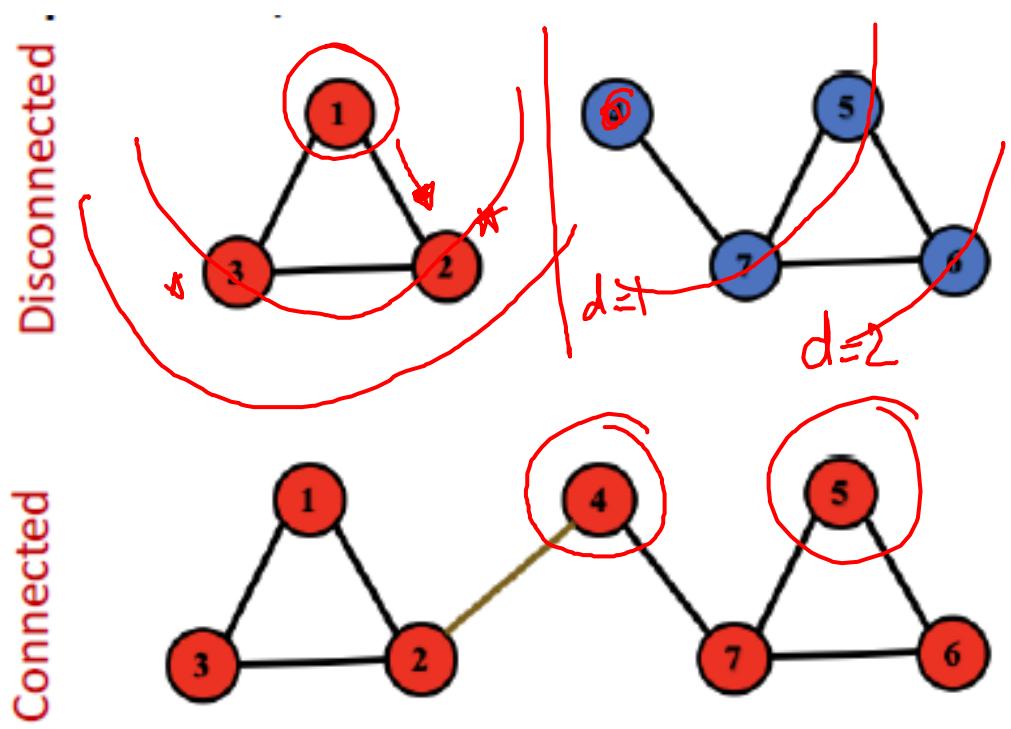
Isolated node (node H)

## Size of the largest connected component (Giant component)

- Largest set where any two vertices can be joined by a path

# Connectivity: Example

The adjacency matrix of a network with several components can be written in a block-diagonal form, so that nonzero elements are confined to squares, with all other elements being zero:


$$\left( \begin{array}{ccc|ccccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array} \right)$$
$$\left( \begin{array}{ccc|ccccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array} \right)$$

Connected Components can be determined using BFS. Start from a random node and perform BFS. Label node visits. All nodes that can be visited are in the connected component. Otherwise, find an unvisited node and start BFS again.

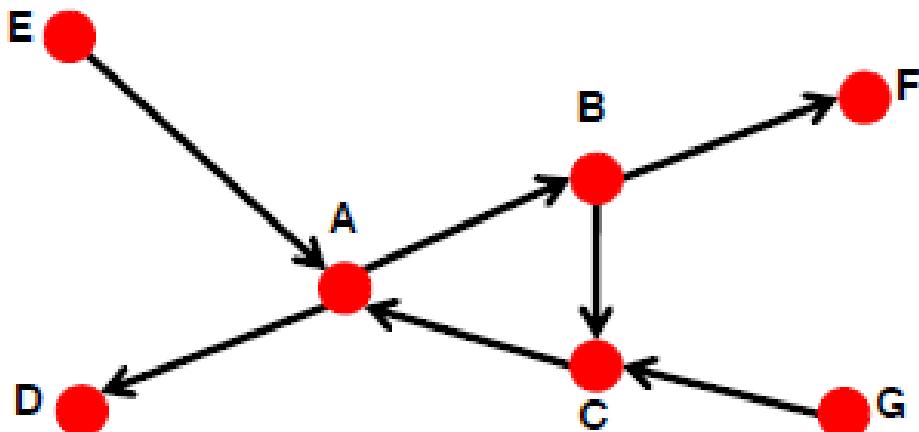
# Connectivity of Directed Graphs

## ○ Strongly connected directed graph

- has a path from each node to every other node and vice versa (e.g., A-B path and B-A path)

## ○ Weakly connected directed graph

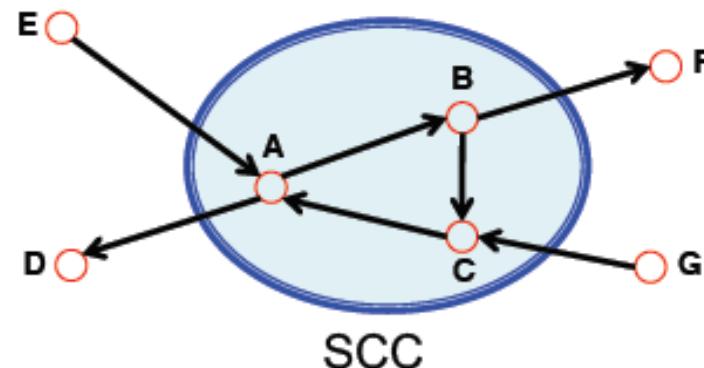
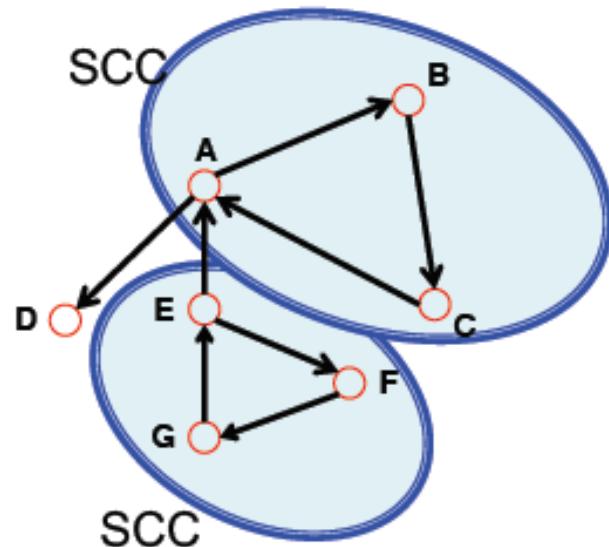
- is connected if we disregard the edge directions



Graph on the left is connected but not strongly connected  
(e.g., there is no way to get from F to G by following the edge directions).

# Connectivity of Directed Graphs

- Strongly connected components (SCCs) can be identified, but not every node is part of a nontrivial strongly connected component.
- In-component: nodes that can reach the SCC,
- Out-component: nodes that can be reached from the SCC.



# Network Diameter

- **Eccentricity of a node  $v_i$ :**  $e(v_i)$  = Max distance (shortest path) from any other node  $v_j$  in the graph
- **Radius of a connected graph:** Minimum eccentricity of any node,  $\min_i e(v_i)$
- **Diameter:** The maximum distance between any pair of nodes in a graph,  $\max_i e(v_i)$ 
  - Effective diameter: min number of hops in which a large fraction (e.g. 90%) of all connected pair of nodes can reach each other
- **Average path length** for a connected graph or a strongly connected directed graph
  - $\bar{h} = \frac{1}{2E_{max}} \sum_{i,j \neq i} h_{ij}$ 
    - $h_{ij}$  is the distance from node i to j
    - $E_{max}$  is the max number of edges (total number of node pairs) =  $\frac{n(n-1)}{2}$
  - Many times we compute the average only over the connected pairs of nodes (that is, we ignore “infinite” length paths)
  - Note that this measure also applies to (strongly) connected components of a graph

# Centrality Analysis

A centrality is a function  $c: V \rightarrow \mathbb{R}$ , that induces a ranking on  $V$ .

**Degree Centrality:** The simplest notion of centrality is the degree  $d_i$  of a vertex  $v_i$  – the higher the degree, the more important or central the vertex.

**Eccentricity Centrality:** Eccentricity centrality is defined as:

$$c(v_i) = \frac{1}{e(v_i)} = \frac{1}{\max_j \{d(v_i, v_j)\}}$$

The less eccentric a node is, the more central it is.

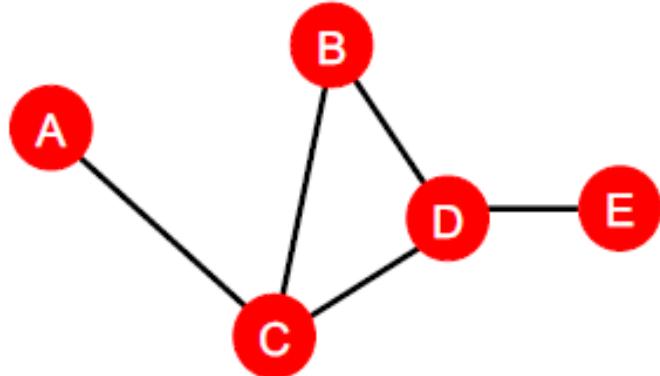
**Closeness Centrality:** closeness centrality uses the sum of all the distances to rank how central a node is

$$c(v_i) = \frac{1}{\sum_j d(v_i, v_j)}$$

# Betweenness Centrality

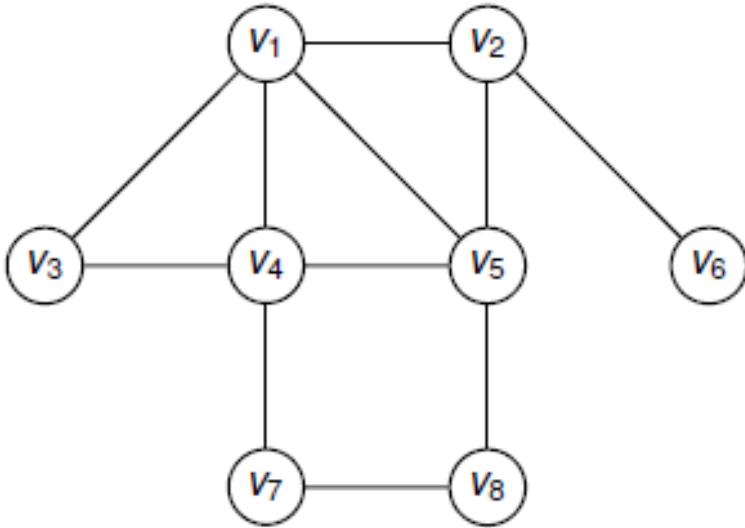
- A node is important if it lies on many shortest paths between other nodes.

- $c_v = \sum_{s \neq v \neq t} \frac{\# \text{ shortest paths between } s \text{ and } t \text{ that contain } v}{\# \text{ shortest paths between } s \text{ and } t}$



$$\begin{aligned}c_A &= c_B = c_E = 0 \\c_C &= 3 \\(\text{A-C-B}, \text{A-C-D}, \text{A-C-D-E}) \\c_D &= 3 \\(\text{A-C-D-E}, \text{B-D-E}, \text{C-D-E})\end{aligned}$$

# Example



| Centrality                        | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ | $v_8$ |
|-----------------------------------|-------|-------|-------|-------|-------|-------|-------|-------|
| Degree                            | 4     | 3     | 2     | 4     | 4     | 1     | 2     | 2     |
| Eccentricity<br>$e(v_i)$          | 0.5   | 0.33  | 0.33  | 0.33  | 0.5   | 0.25  | 0.25  | 0.33  |
| Closeness<br>$\sum_j d(v_i, v_j)$ | 0.100 | 0.083 | 0.071 | 0.091 | 0.100 | 0.056 | 0.067 | 0.071 |
| Betweenness                       | 4.5   | 6     | 0     | 5     | 6.5   | 0     | 0.83  | 1.17  |

These can serve as node features in machine learning tasks

# Prestige or Eigenvector Centrality

- A node  $v$  is important if **surrounded by important neighboring nodes**  $u \in N(v)$ .
- We model the centrality of node  $v$  as **the sum of the centrality of neighboring nodes**:  
$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u$$
- Notice that the above equation models centrality in a **recursive manner**. How do we solve it?
- By rewriting the equation in matrix form,  $\lambda c = A c$ , where  $A$  is adjacency matrix and  $c$  is the centrality vector
- We see that centrality is the **eigenvector!**
- The largest eigenvalue  $\lambda_{max}$  is always positive and unique (by Perron-Frobenius Theorem).
- The leading eigenvector  $c_{max}$  is used for centrality.

# Clustering Coefficient

## ○ Clustering coefficient (for undirected graphs):

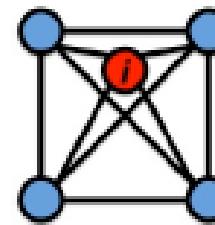
- How connected are  $i$ 's neighbors to each other?

- Node  $i$  with degree  $k_i$

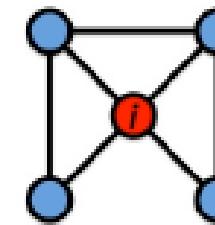
- $C_i \in [0,1]$

- $C_i = \frac{2e_i}{k_i(k_i-1)}$

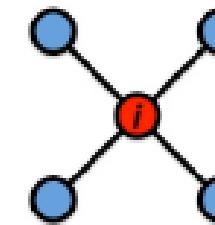
where  $e_i$  is the number of edges between the neighbors of node  $i$



$$C_i = 1$$



$$C_i = 1/2$$

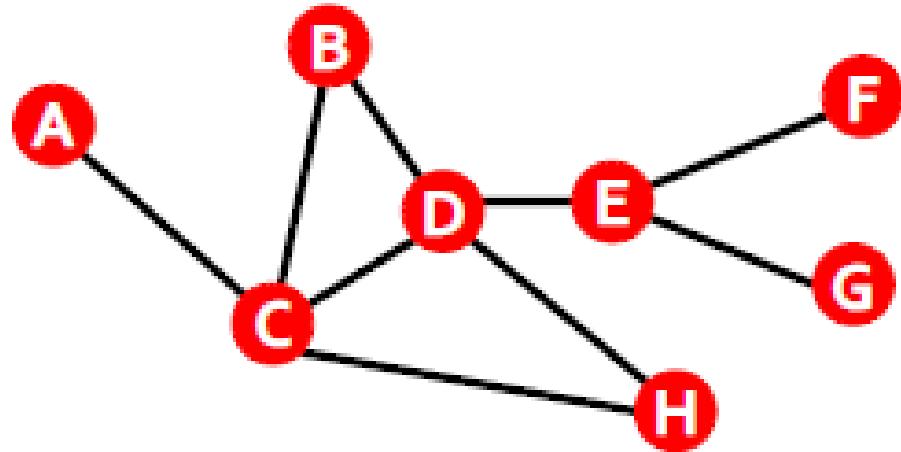


$$C_i = 0$$

- Average clustering coefficient of the Graph:  $C = \frac{1}{N} \sum_{i=1}^N C_i$

- Clustering coefficient is undefined (or defined to be 0) for nodes with degree 0 or 1

# Clustering Coefficient Example



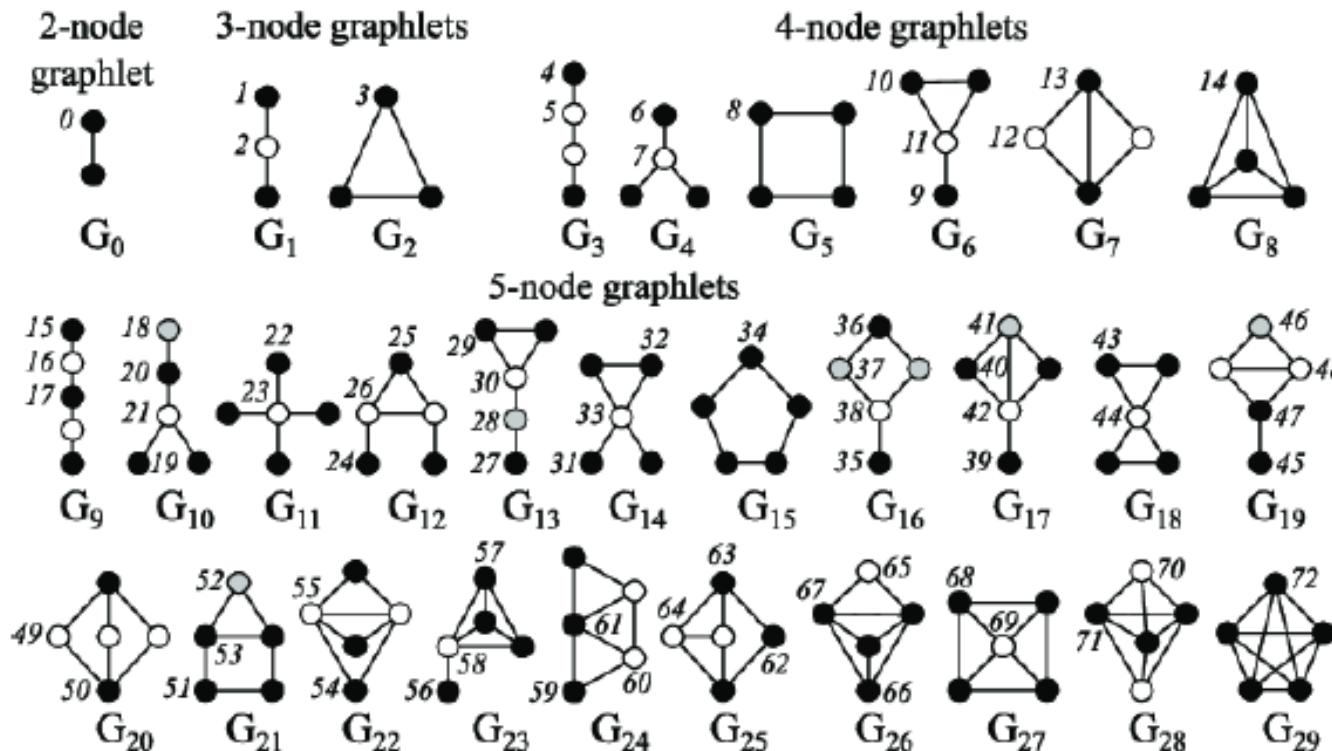
$$k_B=2, \ e_B=1, \ C_B=2/2 = 1$$

$$k_D=4, \ e_D=2, \ C_D=4/12 = 1/3$$

**LQ 2.4:** Compute clustering coefficient for every node and determine the average clustering coefficient for the graph.

# Graphlets

- **Observation:** Clustering coefficient counts the #(triangles) in the ego-network
- We can generalize the above by counting #(pre-specified subgraphs, i.e., **graphlets**).
- Graphlets are rooted connected non-isomorphic subgraphs:



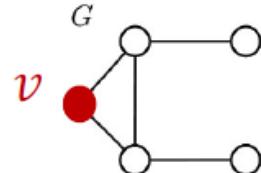
**Graphlet Degree Vector (GDV):**  
Graphlet-base features for nodes

- **Degree** counts **#(edges)** that a node touches
- **Clustering coefficient** counts **#(triangles)** that a node touches
- **GDV** counts **#(graphlets)** rooted at a given node

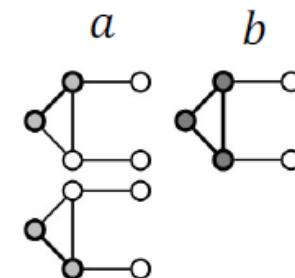
# Graphlets

- Considering graphlets on 2 to 5 nodes we get:
- Vector of 73 coordinates** is a signature of a node that describes the topology of node's neighborhood
- Captures its interconnectivities out to a **distance of 4 hops**
- Graphlet degree vector provides a measure of a **node's local network topology**:
- Comparing vectors of two nodes provides a more detailed measure of local topological similarity than node degrees or clustering coefficient.

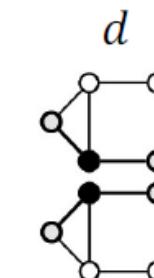
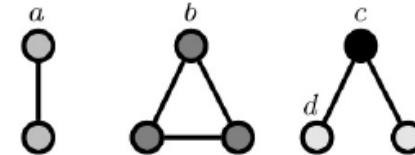
## Example:



Graphlet instances:



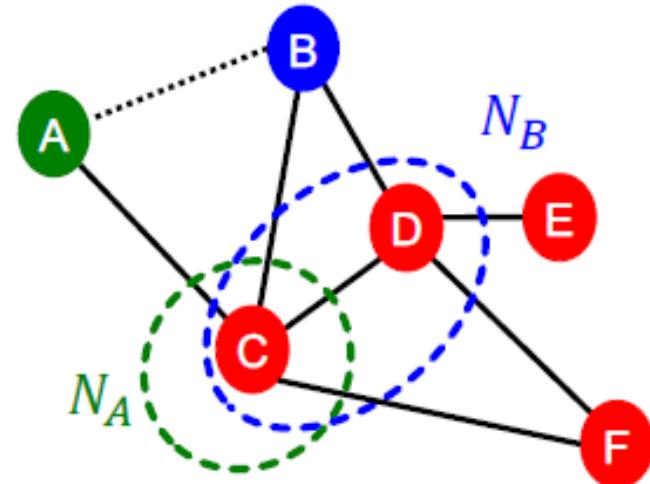
List of graphlets



GDV of node  $v$ :  
 $a, b, c, d$   
[2,1,0,2]

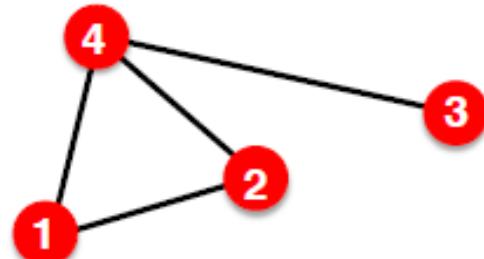
# Link Feature: Local Neighborhood Overlap

- Captures # neighboring nodes shared between two nodes  $u$  and  $v$
- Number of Common neighbors:  $|N(u) \cap N(v)|$ 
  - Example:  $|N(A) \cap N(B)| = |\{C\}| = 1$
- Jaccard's coefficient:  $\frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$ 
  - Example:  $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{1}{2}$
- Adamic-Adar index:  $\sum_{s \in N(u) \cap N(v)} \frac{1}{\log(k_s)}$ 
  - Example:  $\frac{1}{\log(k_C)} = \frac{1}{\log(4)}$



# Global Neighborhood Overlap

- Katz Index: use the paths of all lengths between a given pair of nodes
  - These can be computed using powers of the graph adjacency matrix
- Recall:  $A_{uv} = 1$  if  $u \in N(v)$
- Let  $P_{uv}^K = \#$  of paths of length K between u and v
  - $P^K = A^K$



Node 1's neighbors      #paths of length 1 between  
Power of      Node 1's neighbors and Node 2       $P_{12}^{(2)} = A_{12}^2$   
adjacency

$$A^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

# Global Neighborhood Overlap

- **Katz index** between  $v_1$  and  $v_2$  is calculated as

**Sum over all path lengths**

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \beta^l A_{v_1 v_2}^l$$

#paths of length  $l$   
between  $v_1$  and  $v_2$

$0 < \beta < 1$ : discount factor

- Katz index matrix is computed in closed-form:

$$\begin{aligned} S &= \sum_{i=1}^{\infty} \beta^i A^i = \underbrace{(I - \beta A)^{-1}}_{= \sum_{i=0}^{\infty} \beta^i A^i} - I, \\ &\quad \text{by geometric series of matrices} \end{aligned}$$

# Summary: Key properties of the network

| Property                 | Notation |
|--------------------------|----------|
| Degree Distribution      | $f(k)$   |
| Path Length              | $H$      |
| Clustering Coefficient   | $C$      |
| Connected Components     | $S$      |
| Node centrality measures | $c$      |
| Graphlet count vector    | GDV      |
| Katz Index matrix        | $S$      |

Node features can be useful for predicting influential nodes in a graph for:

- Predicting celebrity users in a social network
- Targeted promotions/advertising
- Recommendations
- Predicting protein functionality in a protein-protein interaction network

# Spectrum Kernel for Strings

Given alphabet  $\Sigma$ , the  $l$ -spectrum feature map is the mapping  $\phi: \Sigma^* \rightarrow \mathbb{R}^{|\Sigma|^l}$  from the set of substrings over  $\Sigma$  to the  $|\Sigma|^l$ -dimensional space representing the number of occurrences of all possible substrings of length  $l$ , defined as

$$\phi(\mathbf{x}) = \left( \dots, \#(\alpha), \dots \right)_{\alpha \in \Sigma^l}^T$$

where  $\#(\alpha)$  is the number of occurrences of the  $l$ -length string  $\alpha$  in  $\mathbf{x}$ .

The (full) spectrum map considers all lengths from  $l = 0$  to  $l = \infty$ , leading to an infinite dimensional feature map  $\phi: \Sigma^* \rightarrow \mathbb{R}^\infty$ :

$$\phi(\mathbf{x}) = \left( \dots, \#(\alpha), \dots \right)_{\alpha \in \Sigma^*}^T$$

where  $\#(\alpha)$  is the number of occurrences of the string  $\alpha$  in  $\mathbf{x}$ .

The ( $l$ )-spectrum kernel between two strings  $\mathbf{x}_i, \mathbf{x}_j$  is simply the dot product between their ( $l$ )-spectrum maps:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

The (full) spectrum kernel can be computed efficiently via suffix trees in  $O(n + m)$  time for two strings of length  $n$  and  $m$ .

Example: If  $\mathbf{x} = \text{ACAGCAGTA}$  and  $\mathbf{y} = \text{AGCAAGCGAG}$

$$\phi(\mathbf{x}) = (\text{ACA : } 1, \text{AGC : } 1, \text{AGT : } 1, \text{CAG : } 2, \text{GCA : } 1, \text{GTA : } 1)$$

$$\phi(\mathbf{y}) = (\text{AAG : } 1, \text{AGC : } 2, \text{CAA : } 1, \text{CGA : } 1, \text{GAG : } 1, \text{GCA : } 1, \text{GCG : } 1)$$

$$K(\mathbf{x}, \mathbf{y}) = 3$$

Note: The max substring is the length of the string

Note: Only the counts of common substrings is required

# Bag of Words (BoW) models

- BoW simply uses the word counts as features for documents (no ordering considered).
- Naïve extension to a graph: **Regard nodes as words.**
- Since both graphs have **4 red nodes**, we get the same feature vector for two different graphs...

$$\phi(\text{graph 1}) = \phi(\text{graph 2})$$

- Using Bag of node degrees:

Deg1: ● Deg2: ● Deg3: ●

$$\phi(\text{graph 1}) = \text{count}(\text{graph 1}) = [1, 2, 1]$$

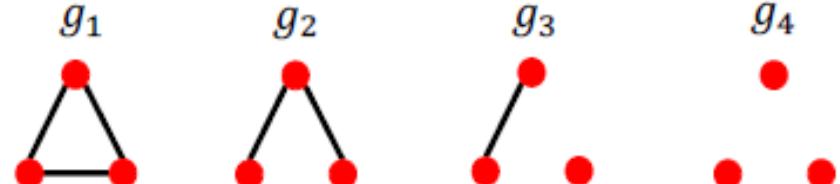
Obtains different features for different graphs!

$$\phi(\text{graph 2}) = \text{count}(\text{graph 2}) = [0, 2, 2]$$

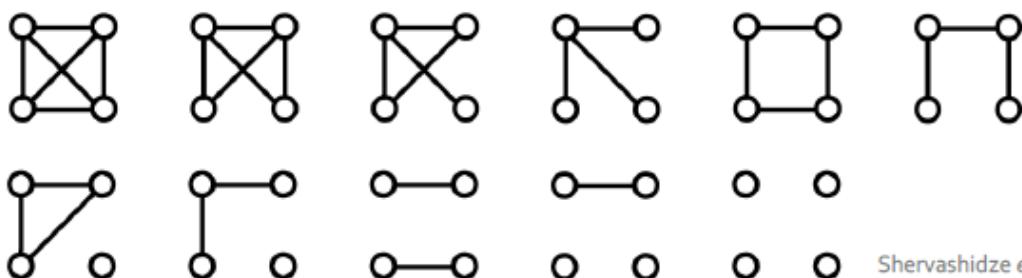
# Graphlet Features of a Graph

- **Key idea:** Count the number of different graphlets in a graph.
- **Note:** Def. of graphlets here is slightly different from node-level features.
- The two differences are:
  - Nodes in graphlets here do **not need to be connected** (allows for isolated nodes)
  - The graphlets here are not rooted.
- Let  $G_k = (g_1, g_2 \dots, g_{n_k})$  be a list of graphlets of size  $k$ .

- For  $k = 3$ , there are 4 graphlets.



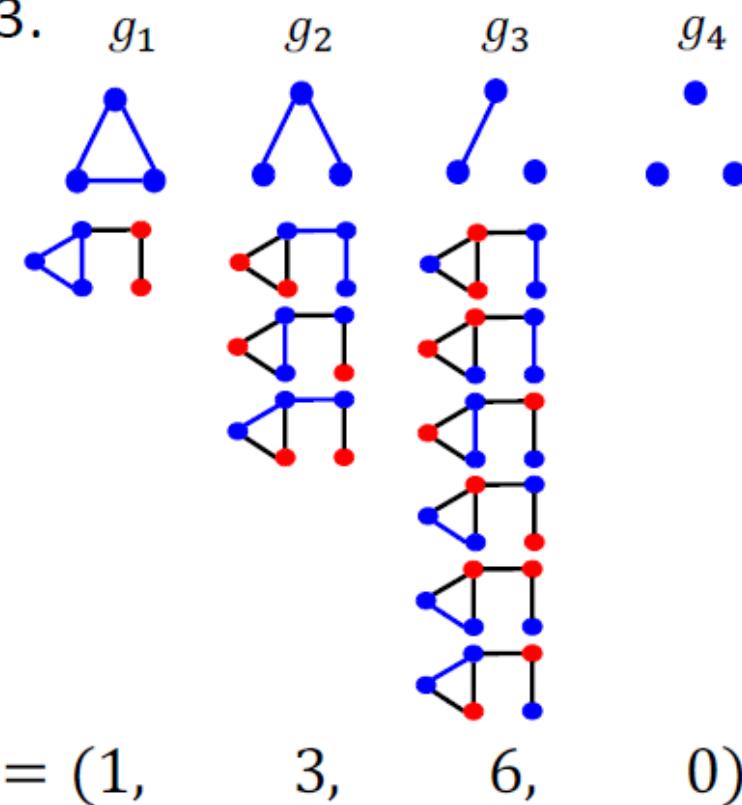
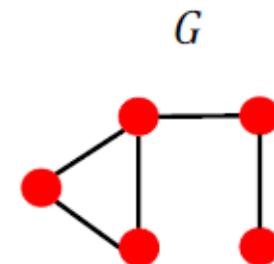
- For  $k = 4$ , there are 11 graphlets.



# Graphlet Features

- Given a graph  $G$ , and graphlet list  $G_k = (g_1, g_2 \dots, g_{n_k})$ , define the graphlet count vector  $f_G \in R^{n_k}$  as  $(f_G)_i = \#(g_i \subset G)$  for  $i = 1, 2, \dots, n_k$

- Example for  $k = 3$ .



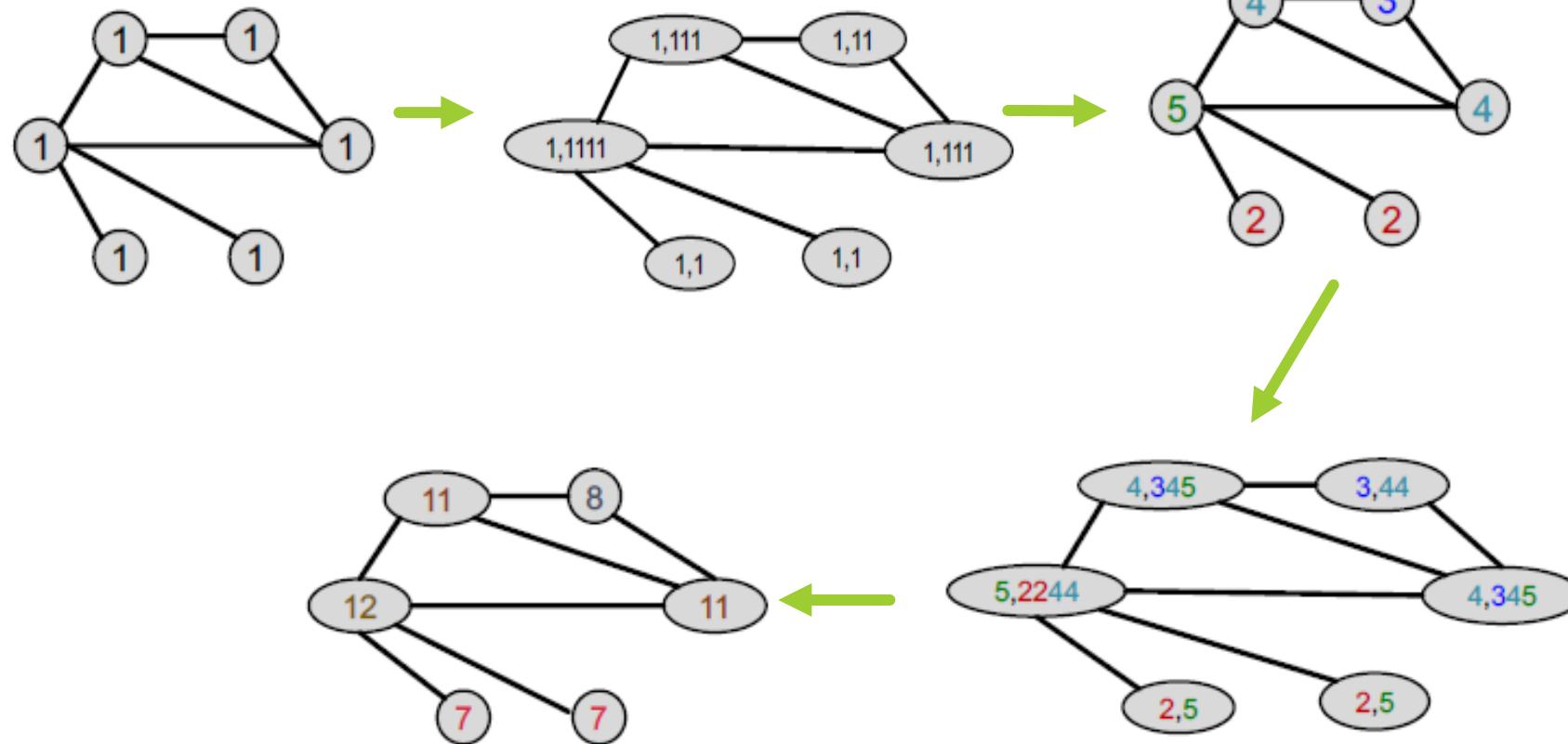
# Graphlet Kernel

- Given two graphs,  $G$  and  $G''$ , graphlet kernel is computed as  $K(G, G') = f_G^T f_{G'}$
- **Problem:** If  $G$  and  $G''$  have different sizes, that will greatly skew the value.
- **Solution:** Normalize each feature vector
  - $h_G = \frac{f_G}{\text{sum}(f_G)}$  and  $K(G, G') = h_G^T h_{G'}$
- **Limitations:** Counting graphlets is expensive!
  - Counting size- $k$  graphlets for a graph of size  $n$  by enumeration takes  $n^k$
  - Unavoidable in worst-case since subgraph isomorphism (judging whether a graph is a sub-graph of another graph) is NP-hard
  - If a graph's node degree is bounded by  $d$ , an  $O(nd^{k-1})$  algorithm exists to count all the graphlets of size  $k$

# Weisfeiler-Lehman (WL) Kernel

- **Goal:** design an efficient graph feature descriptor  $\phi(G)$
- **Idea:** use neighborhood structure to iteratively enrich node vocabulary.
- Generalized version of **Bag of node degrees** (**vector of degree** since node degrees are one-hop neighborhood information).
- **Algorithm to achieve this:** **Color refinement**
- **Given:** A graph  $G$  with a set of nodes  $V$ .
  - Assign an initial color  $c, v$  to each node  $v$ .
  - Iteratively refine node colors by  $c^{k+1}(v) = \text{HASH}(\{c^k(v), \{c^k(u)\}_{u \in N(v)}\})$  where **HASH** maps different inputs to different colors.
- After  $K$  steps of color refinement,  $c^k(v)$  **summarizes the structure of  $K$ -hop neighborhood**

# Example 1



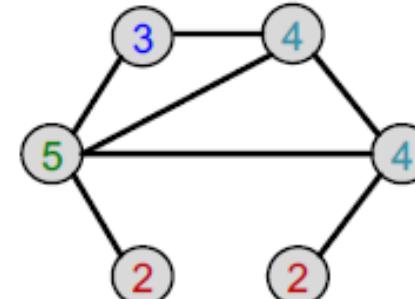
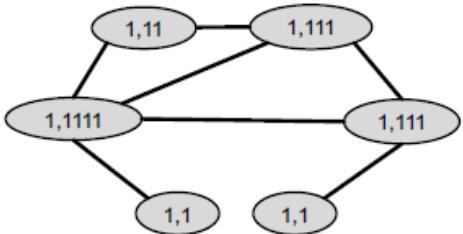
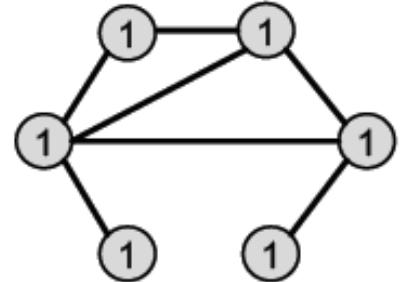
Hash table

|        |   |   |
|--------|---|---|
| 1,1    | → | 2 |
| 1,11   | → | 3 |
| 1,111  | → | 4 |
| 1,1111 | → | 5 |

Hash table

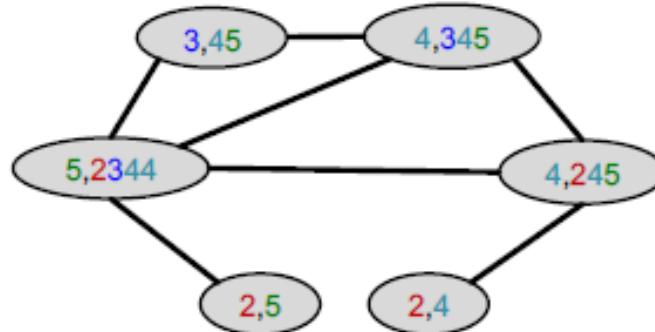
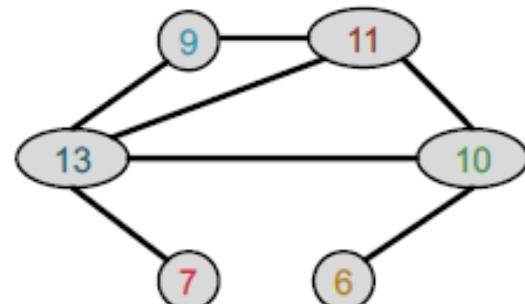
|        |   |    |
|--------|---|----|
| 2,4    | → | 6  |
| 2,5    | → | 7  |
| 3,44   | → | 8  |
| 3,45   | → | 9  |
| 4,245  | → | 10 |
| 4,345  | → | 11 |
| 5,2244 | → | 12 |
| 5,2344 | → | 13 |

## Example 2



Hash table

|        |   |   |
|--------|---|---|
| 1,1    | → | 2 |
| 1,11   | → | 3 |
| 1,111  | → | 4 |
| 1,1111 | → | 5 |

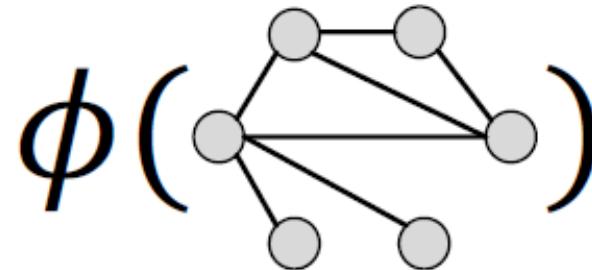


Hash table

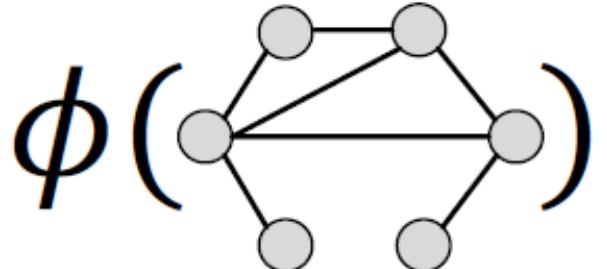
|        |   |    |
|--------|---|----|
| 2,4    | → | 6  |
| 2,5    | → | 7  |
| 3,44   | → | 8  |
| 3,45   | → | 9  |
| 4,245  | → | 10 |
| 4,345  | → | 11 |
| 5,2244 | → | 12 |
| 5,2344 | → | 13 |

# WL Graph Features

After color refinement, WL kernel counts number of nodes with a given color.



Colors  
= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]  
Counts



1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13  
= [6, 2, 1, 2, 1, 1, 1, 0, 1, 1, 1, 0, 1]

The WL kernel value is computed by the inner product of the color count vectors:

$$K\left(\begin{array}{c} \text{graph} \\ , \\ \text{graph} \end{array}\right) = \phi\left(\begin{array}{c} \text{graph} \end{array}\right)^T \phi\left(\begin{array}{c} \text{graph} \end{array}\right) = 49$$

# WL Kernel

- WL kernel is **computationally efficient**
  - The time complexity for color refinement at each step is linear in #(edges), since it involves aggregating neighboring colors.
- When computing a kernel value, only colors appeared in the two graphs need to be tracked.
  - Thus, #(colors) is at most the total number of nodes.
- Counting colors takes linear-time w.r.t. #(nodes).
- In total, time complexity is **linear in #(edges)**.

# Diffusion Kernels on Graph Nodes

Let  $\mathbf{S}$  be some symmetric similarity matrix between nodes of a graph  $G = (V, E)$ . For instance,  $\mathbf{S}$  can be the (weighted) adjacency matrix  $\mathbf{A}$  or the Laplacian matrix  $\mathbf{L} = \mathbf{A} - \Delta$  (or its negation), where  $\Delta$  is the degree matrix for an undirected graph  $G$ , defined as  $\Delta(i, i) = d_i$  and  $\Delta(i, j) = 0$  for all  $i \neq j$ , and  $d_i$  is the degree of node  $i$ .

**Power Kernels:** Summing up the product of the base similarities over all  $l$ -length paths between two nodes, we obtain the  $l$ -length similarity matrix  $\mathbf{S}^{(l)}$ , which is simply the  $l$ th power of  $\mathbf{S}$ , that is,

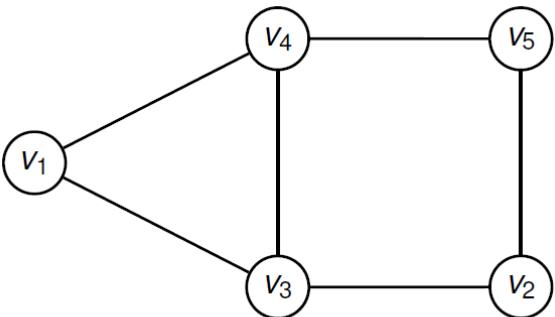
$$\mathbf{S}^{(l)} = \mathbf{S}^l$$

Even path lengths lead to positive semidefinite kernels, but odd path lengths are not guaranteed to do so, unless the base matrix  $\mathbf{S}$  is itself a positive semidefinite matrix.

Power kernel  $\mathbf{K}$  can be obtained via the eigen-decomposition of  $\mathbf{S}^l$ :

$$\mathbf{K} = \mathbf{S}^l = (\mathbf{U}\Lambda\mathbf{U}^T)^l = \mathbf{U}(\Lambda^l)\mathbf{U}^T$$

# Example



Adjacency and degree matrices are given as

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$\Delta = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

$$\mathbf{S} = -\mathbf{L} = \mathbf{A} - \mathbf{D} = \begin{pmatrix} -2 & 0 & 1 & 1 & 0 \\ 0 & -2 & 1 & 0 & 1 \\ 1 & 1 & -3 & 1 & 0 \\ 1 & 0 & 1 & -3 & 1 \\ 0 & 1 & 0 & 1 & -2 \end{pmatrix}$$

The eigenvalues of  $\mathbf{S}$  are as follows:

$$\lambda_1 = 0 \quad \lambda_2 = -1.38 \quad \lambda_3 = -2.38 \quad \lambda_4 = -3.62 \quad \lambda_5 = -4.62$$

and the eigenvectors of  $\mathbf{S}$  are

$$\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 & \mathbf{u}_4 & \mathbf{u}_5 \end{pmatrix}$$
$$\begin{pmatrix} 0.45 & -0.63 & 0.00 & 0.63 & 0.00 \\ 0.45 & 0.51 & -0.60 & 0.20 & -0.37 \\ 0.45 & -0.20 & -0.37 & -0.51 & 0.60 \\ 0.45 & -0.20 & 0.37 & -0.51 & -0.60 \\ 0.45 & 0.51 & 0.60 & 0.20 & 0.37 \end{pmatrix}$$

# Exponential diffusion Kernel

We can obtain a new kernel between nodes of a graph by paths of all possible lengths, but damp the contribution of longer paths

$$\begin{aligned}\mathbf{K} &= \sum_{l=0}^{\infty} \frac{1}{l!} \beta^l \mathbf{S}^l \\ &= \mathbf{I} + \beta \mathbf{S} + \frac{1}{2!} \beta^2 \mathbf{S}^2 + \frac{1}{3!} \beta^3 \mathbf{S}^3 + \dots \\ &= \exp\{\beta \mathbf{S}\}\end{aligned}$$

where  $\beta$  is a damping factor, and  $\exp\{\beta \mathbf{S}\}$  is the matrix exponential. The series on the right hand side above converges for all  $\beta \geq 0$ .

Substituting  $\mathbf{S} = \mathbf{U} \Lambda \mathbf{U}^T$  the kernel can be computed as

$$\begin{aligned}\mathbf{K} &= \mathbf{I} + \beta \mathbf{S} + \frac{1}{2!} \beta^2 \mathbf{S}^2 + \dots \\ &= \mathbf{U} \begin{pmatrix} \exp\{\beta \lambda_1\} & 0 & \cdots & 0 \\ 0 & \exp\{\beta \lambda_2\} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \exp\{\beta \lambda_n\} \end{pmatrix} \mathbf{U}^T\end{aligned}$$

Note:  $\lambda_i$ s are the eigenvalues of  $\mathbf{S}$

## Von Neumann diffusion Kernel

The *von Neumann diffusion kernel* is defined as

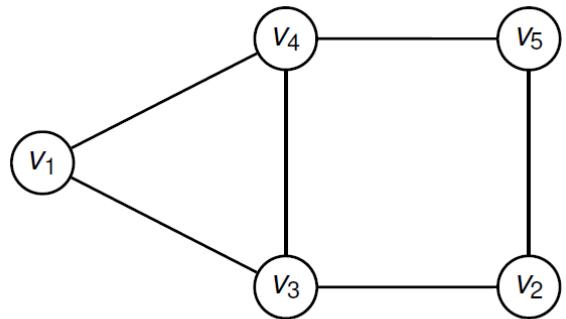
$$\mathbf{K} = \sum_{l=0}^{\infty} \beta^l \mathbf{S}^l$$

where  $\beta \geq 0$ . Expanding and rearranging the terms, we obtain

$$\mathbf{K} = (\mathbf{I} - \beta \mathbf{S})^{-1}$$

The kernel is guaranteed to be positive semidefinite if  $|\beta| < 1/\rho(\mathbf{S})$ , where  $\rho(\mathbf{S}) = \max_i\{|\lambda_i|\}$  is called the *spectral radius* of  $\mathbf{S}$ , defined as the largest eigenvalue of  $\mathbf{S}$  in absolute value.

# Example



Assuming  $\beta = 0.2$ , the exponential diffusion kernel matrix is given as

$$\mathbf{K} = \exp\{0.2\mathbf{S}\} = \mathbf{U} \begin{pmatrix} \exp\{0.2\lambda_1\} & 0 & \cdots & 0 \\ 0 & \exp\{0.2\lambda_2\} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \exp\{0.2\lambda_n\} \end{pmatrix} \mathbf{U}^T$$
$$= \begin{pmatrix} 0.70 & 0.01 & 0.14 & 0.14 & 0.01 \\ 0.01 & 0.70 & 0.13 & 0.03 & 0.14 \\ 0.14 & 0.13 & 0.59 & 0.13 & 0.03 \\ 0.14 & 0.03 & 0.13 & 0.59 & 0.13 \\ 0.01 & 0.14 & 0.03 & 0.13 & 0.70 \end{pmatrix}$$

Assuming  $\beta = 0.2$ , the von Neumann kernel is given as

$$\mathbf{K} = \mathbf{U}(\mathbf{I} - 0.2\mathbf{\Lambda})^{-1}\mathbf{U}^T = \begin{pmatrix} 0.75 & 0.02 & 0.11 & 0.11 & 0.02 \\ 0.02 & 0.74 & 0.10 & 0.03 & 0.11 \\ 0.11 & 0.10 & 0.66 & 0.10 & 0.03 \\ 0.11 & 0.03 & 0.10 & 0.66 & 0.10 \\ 0.02 & 0.11 & 0.03 & 0.10 & 0.74 \end{pmatrix}$$