

Graph-level Tasks

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr



네트워크 과학 연구실
NETWORK SCIENCE LAB



가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA



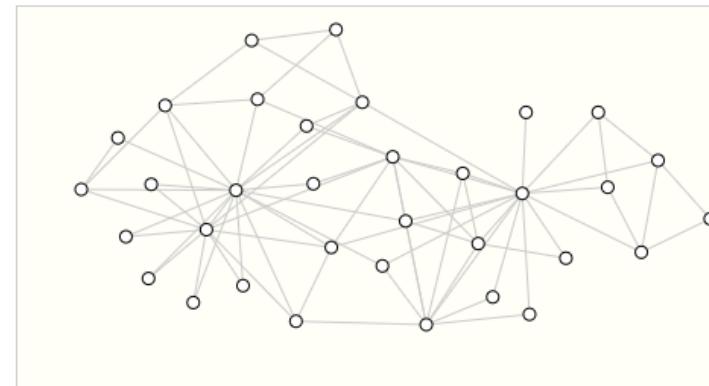
Contents



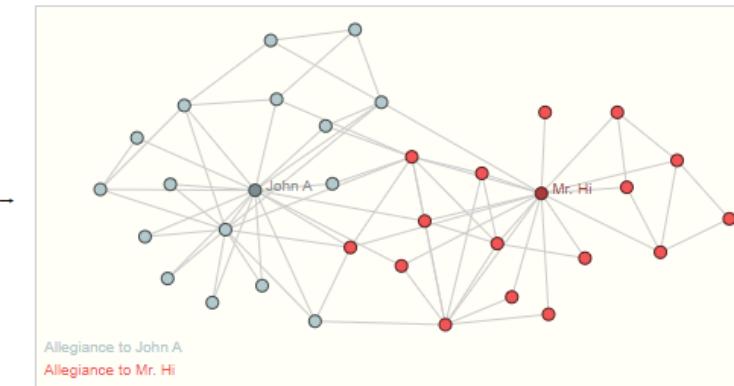
- Graph-level Overview
- Feature Extraction Methods
 - Local features: Graph Size, Density, Connectivity, Average Degree, Centrality Measurement, Diameter, Average Clustering Coefficient, Average Shortest Path Length
 - Global features: Motif Counts, Subgraph, Kernel-based, Graph Spectrum, Community Structure
- Graph-level Tasks:
 - Graph Regression/Classification/Generation/Clustering/Similarity/Embedding
- Applications of Graph-level Representation Learning

Machine Learning Tasks on Graph

- **Node Classification:** given a graph with labels on some nodes, provide a high-quality labeling for the rest of the nodes.

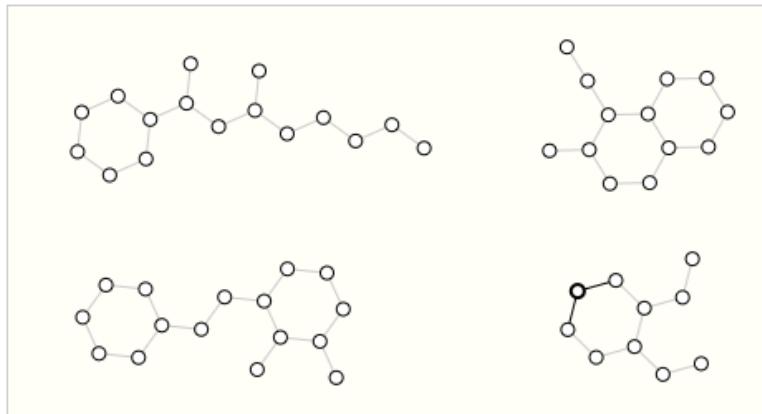


Input: graph with unlabeled nodes

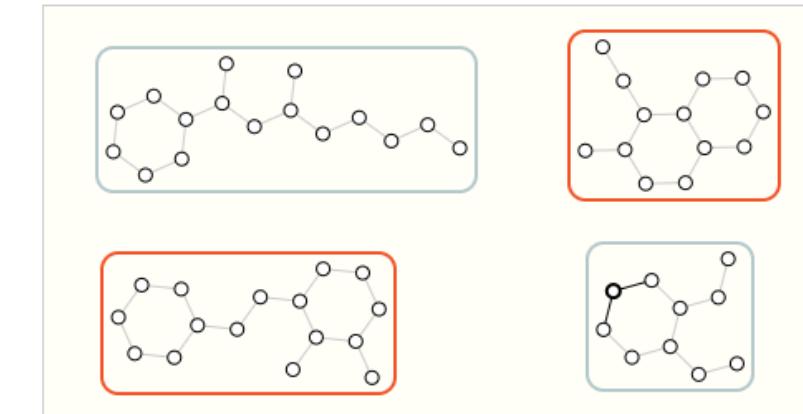


Output: graph node labels

- **Graph Classification:** given a set of graphs with known class labels for some of them, decide to which class the rest of the graphs belong



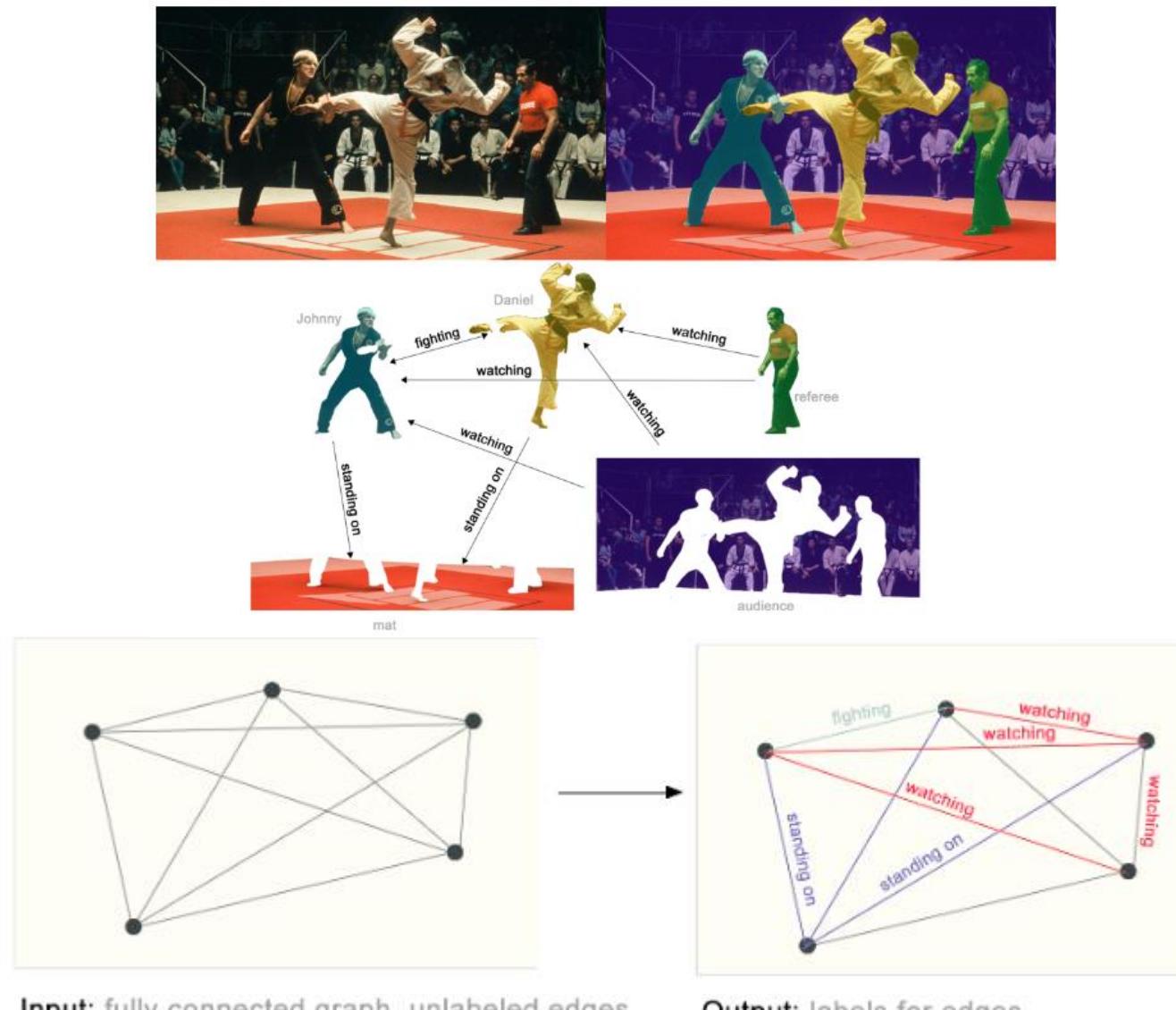
Input: graphs



Output: labels for each graph, (e.g., "does the graph contain two rings?")

Machine Learning Tasks on Graph

- Edge/Link Prediction: given a pair of nodes, predict if they should be linked with an edge

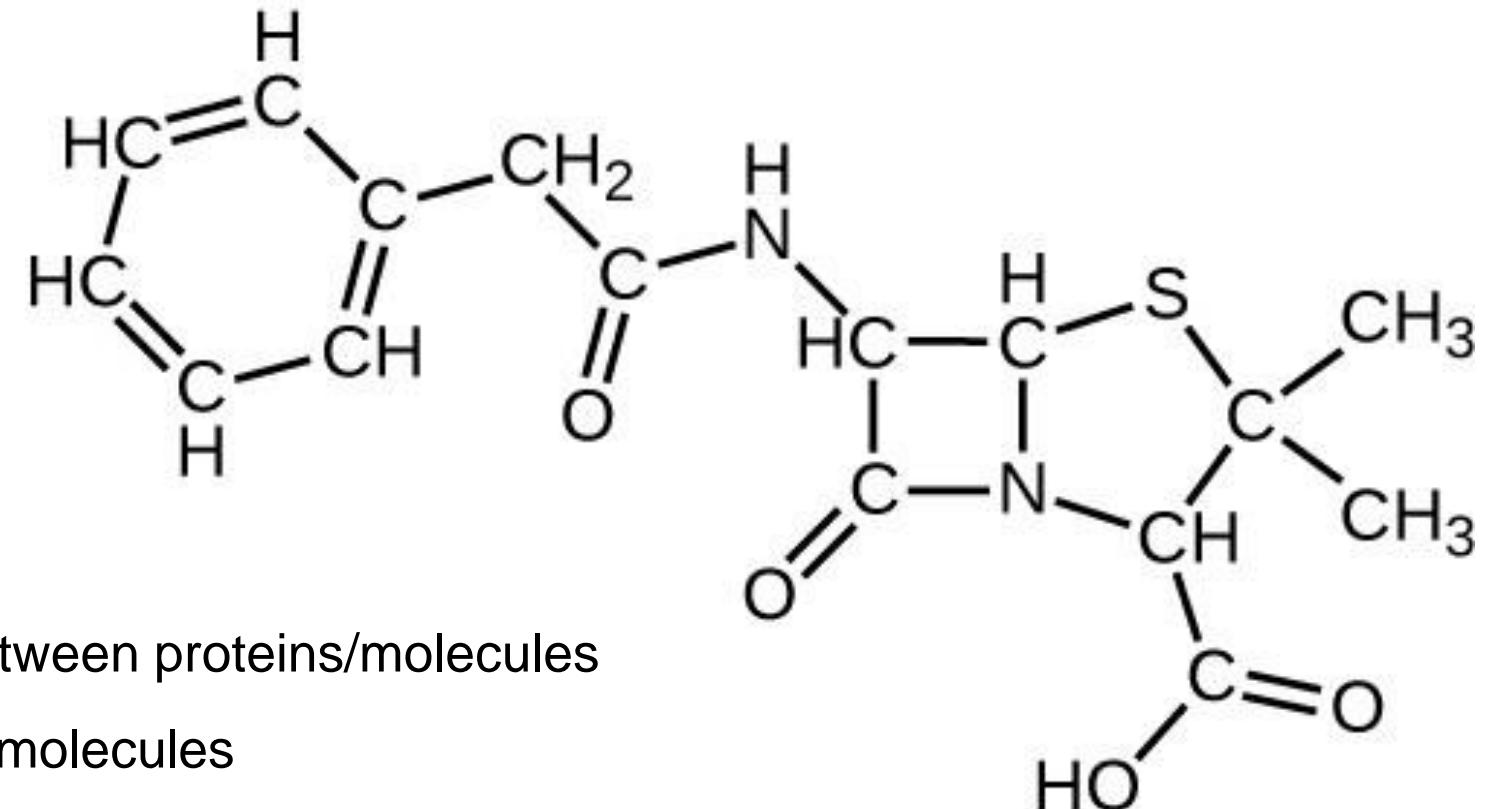


➤ Intuition:

- Node-level task involve making prediction or performing computation at the level of individual nodes within a graph
 - Each node in the graph is treated as a **separate entity**, and the task focuses on understanding or predicting properties or behaviours specific to each node
- Graph-level task involve making prediction or performing computation based on the entire graph structure
 - The entire graph structure is considered as **a whole**, and the task typically involves understanding or predicting global properties or behaviours of the entire graph

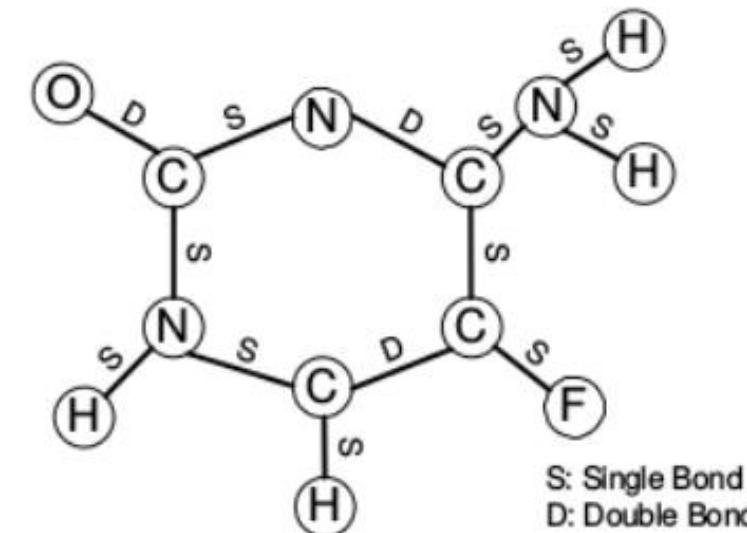
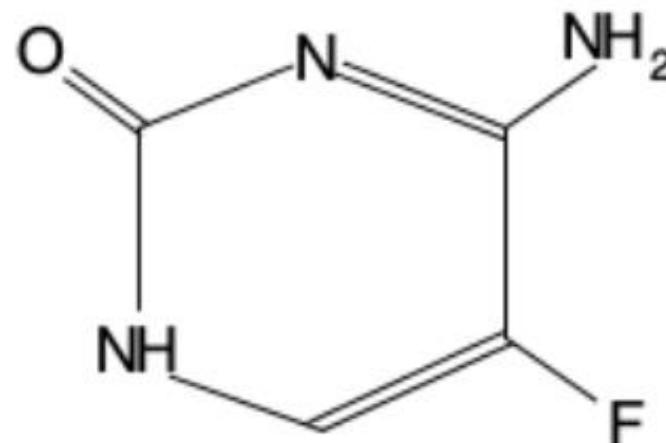
- For each protein, create a graph that contains information about its:

- Structure
- Sequence
- Chemical properties



- Use graph features to:
 - Measure structural similarity between proteins/molecules
 - Predict the function of proteins/molecules

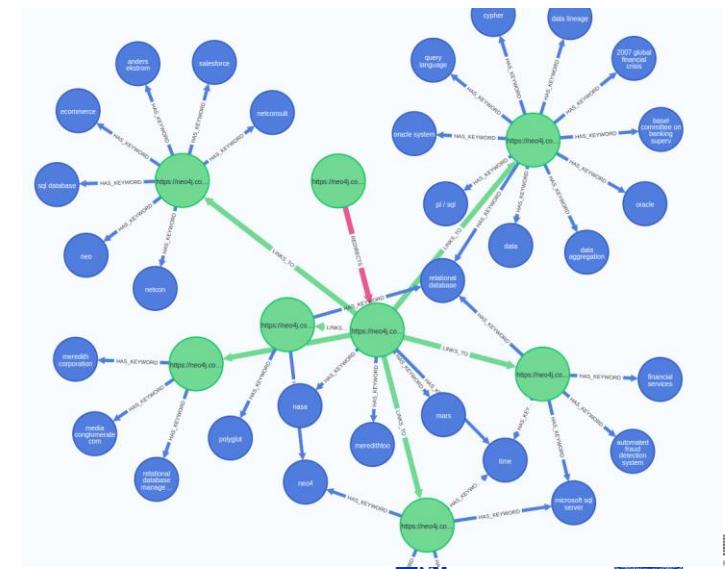
- Represent each chemical compound as a graph



- 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

Motivation – Anomaly Detection for the Web Graph

- Search engines create snapshots of the web → **Web graphs**
- These are necessary for
 - Monitoring the evolution of the web
 - Computing global properties such as PageRank
- Identify anomalies in a single snapshot by comparing it with previous snapshots
- Employed similarity measures:
 - Node/edge overlap
 - Node ranking
 - Node/edge vector similarity
 - etc



- Given a computer program, create its control flow graph:

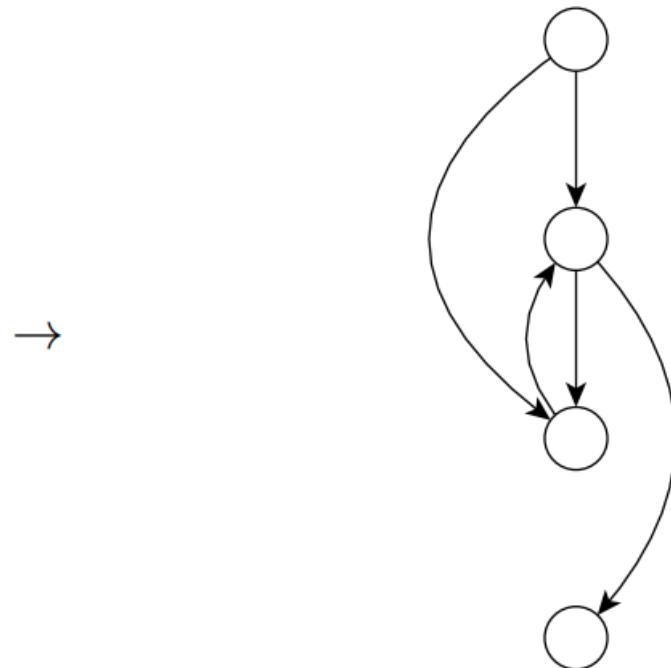
```
processed_pages.append(processed_page)
visited += 1
links = extract_links(html_code)
for link in links:
    if link not in visited_links:
        links_to_visit.append(link)

return create_vocabulary(processed_pages)

def parse_page(html_code):
    punct = re.compile(r'([^\w\s-])')
    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

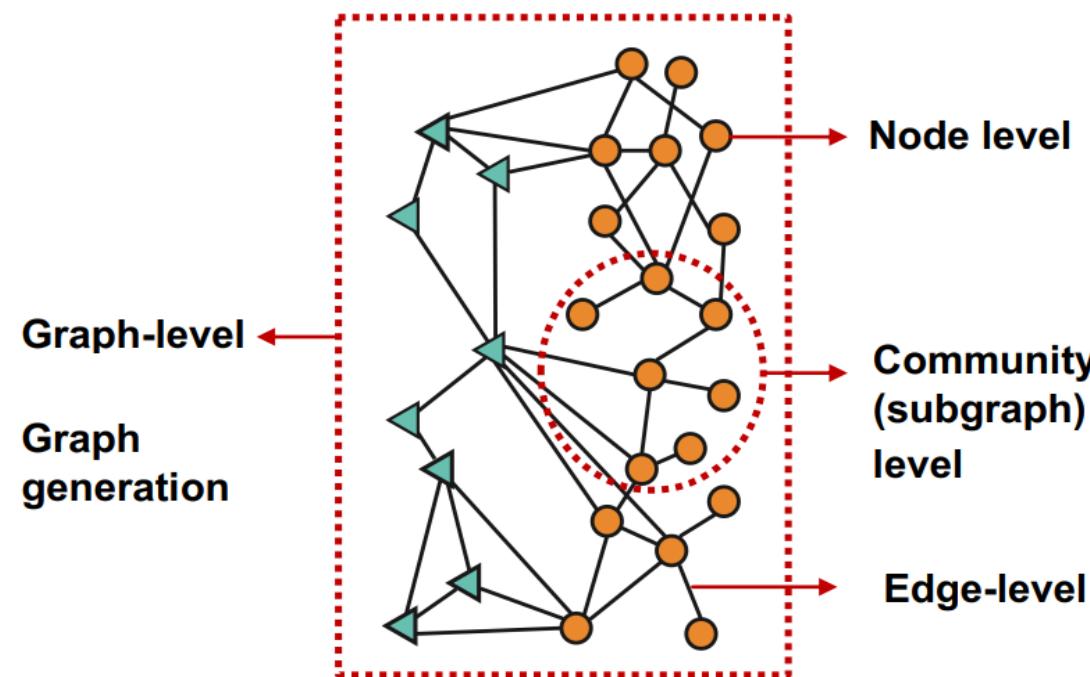
    return vocabulary
```



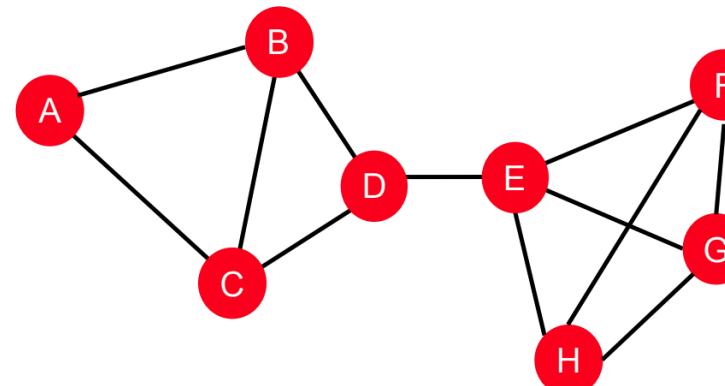
- Compare the control flow graph of the problem against the set of control flow graphs of known malware
 - If it contains a subgraph isomorphic to these graphs → malicious code inside the program

➤ Three main types of graph components:

1. Node-level features
2. Edge-level features
3. Graph-level features



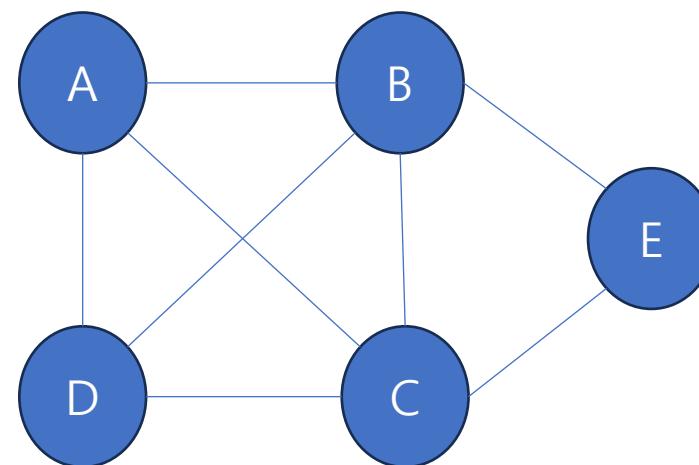
- **Goal:** We want features that characterize the structure of an entire graph
 - Similar graphs has similar features
- **Idea:** Extract structural information within graphs as graph features
- **Methods:**
 - Local features: Graph Size, Density, Connectivity, Degree, Centrality, Average Clustering Coefficient, Diameter, Average Shortest Path Length
 - Global features: Motif/Subgraph, Kernel, Spectral, Community



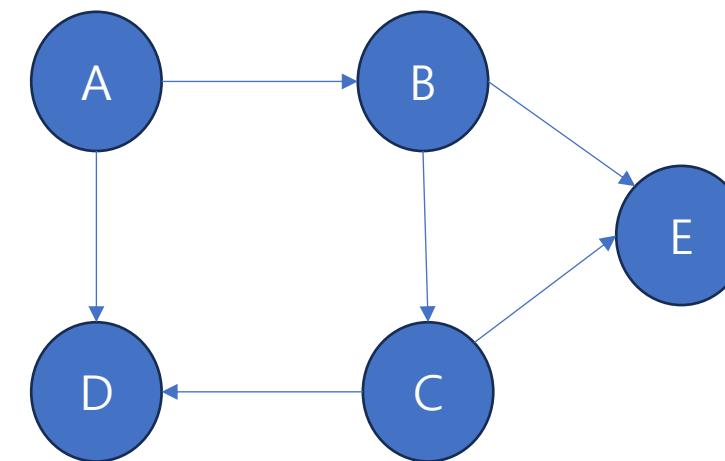
$f(\text{graph}) = ?$

- Graph size provide fundamental information about the graph's scale and complexity:

- Order of a graph is the number of vertices in the graph
 - Denotes as $|V|$
- Size of a graph is the number of edges in the graph
 - Denotes as $|E|$



- $|V| = 5, |E| = 8$



- $|V| = 5, |E| = 6$

Local Features: Graph Density

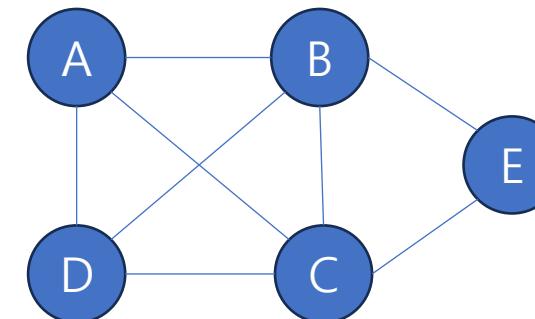
- Graph Density represents the ratio between the edges present in a graph and the maximum number of edges that the graph can contain

- Edge connectivity is key to find dense

- In undirected graph:

- Calculate the maximum number of edges:

$$Max_{EG} = \frac{|V|(|V| - 1)}{2}$$



$$Den_G = \frac{2 * 8}{5(5 - 1)} = 0.8$$

- Edges are divided by the maximum number of edges that the graph can contain. Density graph is defined

$$Den_G = \frac{\frac{|E|}{2}}{\frac{|V|(|V| - 1)}{2}} = \frac{2|E|}{|V|(|V| - 1)}$$

Where $|V|$ is number of nodes

$|E|$ is number of edges

Local Features: Graph Density

- Similarly, in directed graph: replace each undirected edge with two bidirectional edges in a directed graph
- Maximum number of edges:

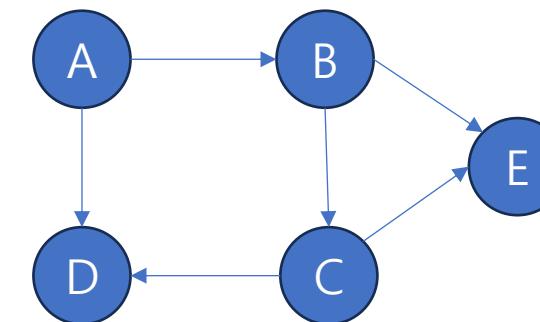
$$Max_{EG} = |V|(|V| - 1)$$

- Edges are divided by the maximum number of edges that the graph can contain. Density graph is defined

$$Den_G = \frac{|E|}{|V|(|V| - 1)}$$

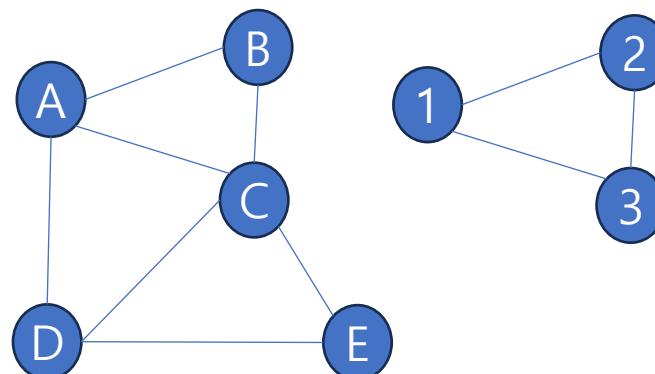
Where $|V|$ is number of nodes

$|E|$ is number of edges

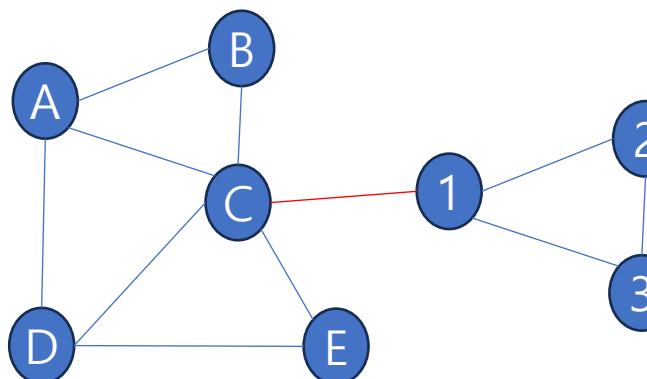


$$Den_{DG} = \frac{6}{5(5 - 1)} = 0.3$$

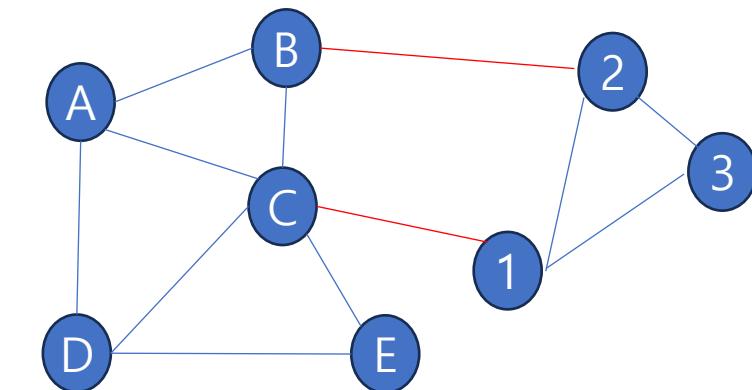
- Graph Connectivity: The minimum number of elements (**nodes or edges**) that need to be **removed** to separate the remaining nodes into two or more isolated subgraphs



Unconnected graph

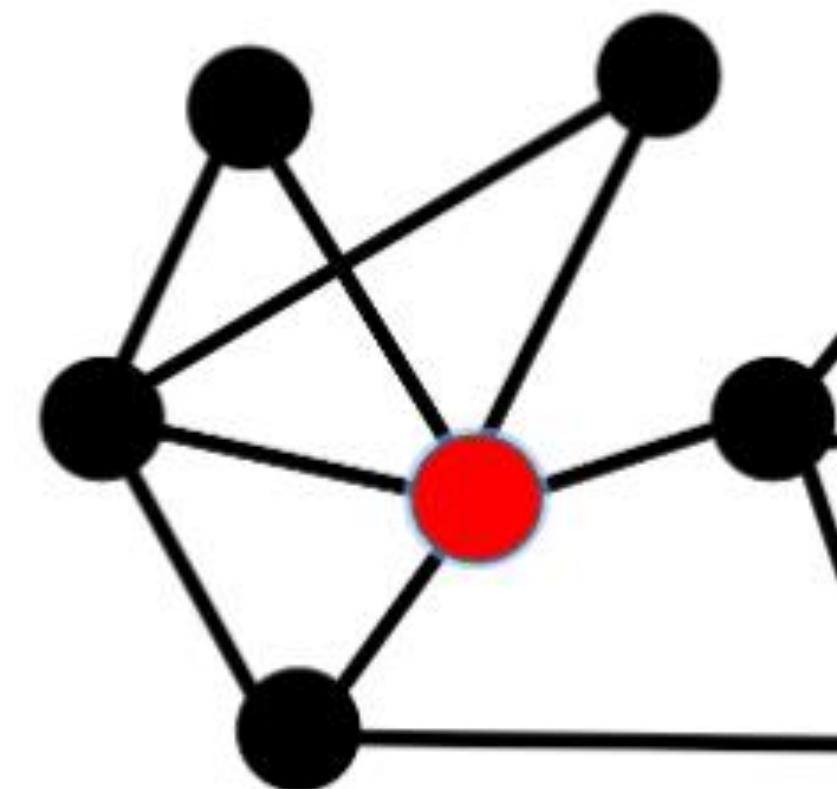


1- connected graph



2- connected graph

- Node degree counts the neighbouring nodes without capturing their importance
- Node centrality takes the node importance in a graph into account
- Different ways to evaluate importance:
 - Eigenvector Centrality
 - Betweenness Centrality
 - Closeness Centrality
 -

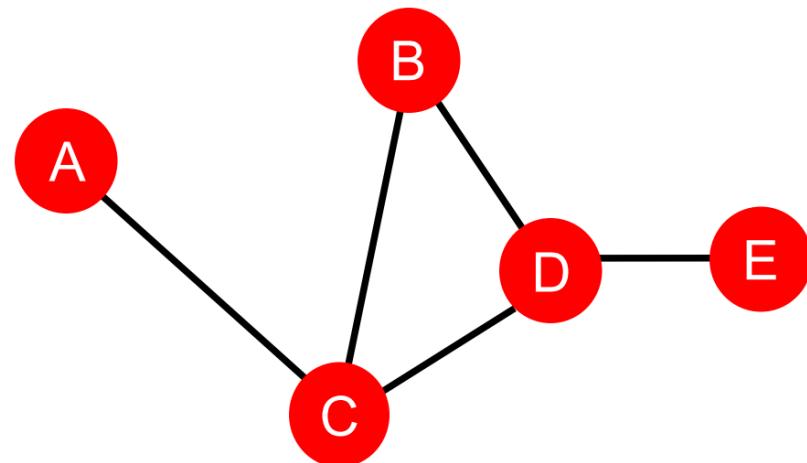


➤ Betweenness Centrality

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

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

- A graph betweenness centrality is **average** of all the nodes' betweenness centrality
- For example:



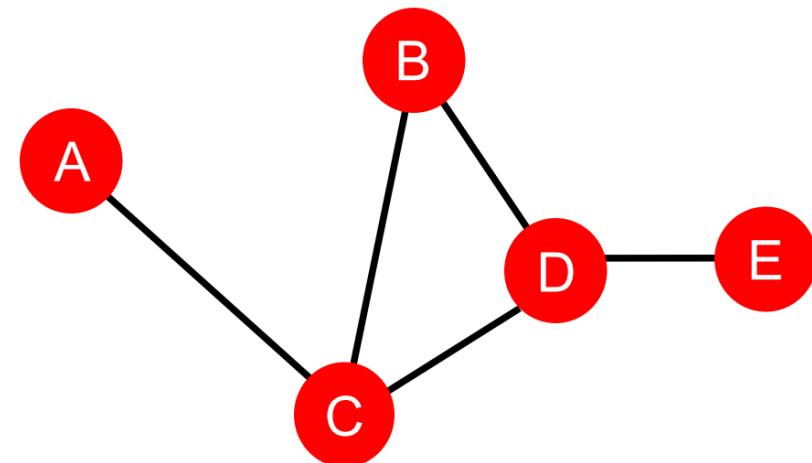
- $C_A = C_B = C_E = 0$
- $C_C = 3$ (A-C-B, A-C-D, A-C-D-E)
- $C_D = 3$ (A-C-D-E, B-D-E, C-D-E)
- $C_G = \frac{0+0+3+3+0}{5} = 1.2$

➤ Closeness Centrality

- A node is important if it has **small shortest path lengths** to all other nodes

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

- A graph closeness centrality is **average** of all the nodes' closeness centrality
- For example:



$$\cdot C_A = 1/(2 + 1 + 2 + 3) = 1/8$$

(A-C-B, A-C, A-C-D, A-C-D-E)

$$\cdot C_D = 1/(2 + 1 + 1 + 1) = 1/5$$

(D-C-A, D-B, D-C, D-E)

$$\cdot C_G = (\frac{1}{8} + \frac{1}{5} + \frac{1}{6} + \frac{1}{5} + \frac{1}{8})/5 \approx 0.1633$$

➤ Eigenvector Centrality

- A node v is important if surrounded by important neighbouring nodes $u \in N(v)$
- We model the centrality of node v as the sum of the centrality of neighbouring nodes recursively:

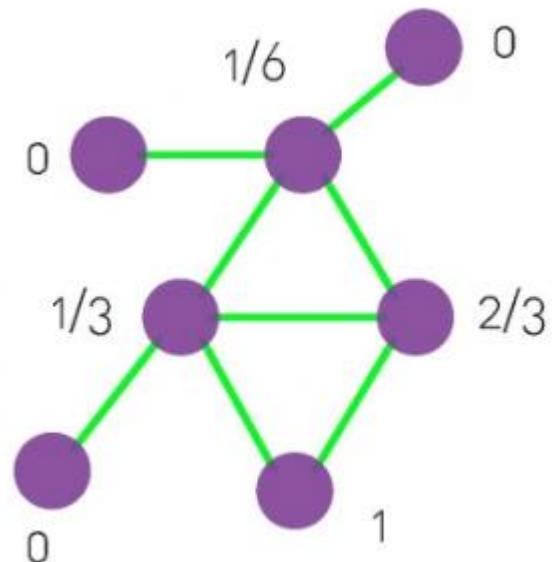
$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u \quad \longleftrightarrow \quad \lambda c = Ac$$

- A : Adjacency matrix
- $A_{uv} = 1$ if $u \in N(v)$
- c : Centrality vector
- λ : Eigenvalue

- Obtain the graph-level feature by **averaging** of all the nodes' eigenvector centrality

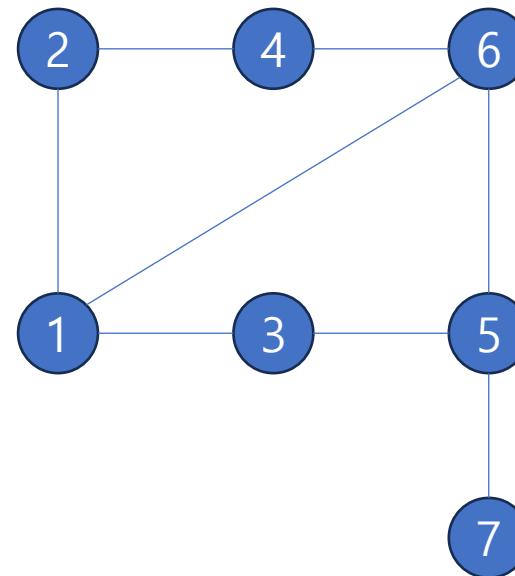
- The degree of clustering of a whole network is captured by the average clustering coefficient, namely $\langle C \rangle$, representing the average of all the local clustering coefficient C_i over all nodes $i = 1, \dots, N$

$$\langle C \rangle = \frac{1}{N} \sum_{i=0}^N C_i$$



$$\langle C \rangle = \frac{1}{7} * \left(0 + \frac{1}{6} + \frac{1}{3} + \frac{2}{3} + 1 + 0 + 0 \right) = 0.333$$

- The diameter of a graph is the length of the shortest path between the most distanced nodes
- d measures the extent of a graph and the topological length between two nodes



Diameter of this graph is 4

- The average number of steps along the shortest paths for all possible pairs of network nodes
- A measure of the efficiency of information in a graph
 - i.e. The average number of clicks which will lead you from one website to another,
 - Or the number of people you will have to communicate through, on an average, to contact a complete stranger

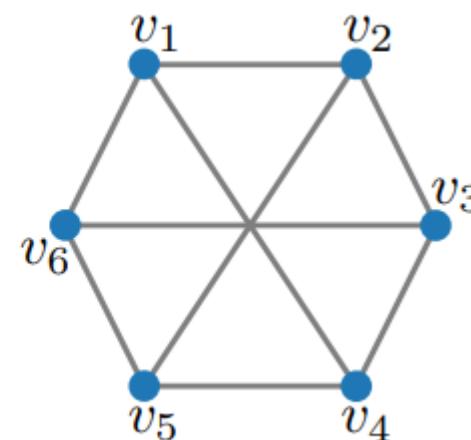
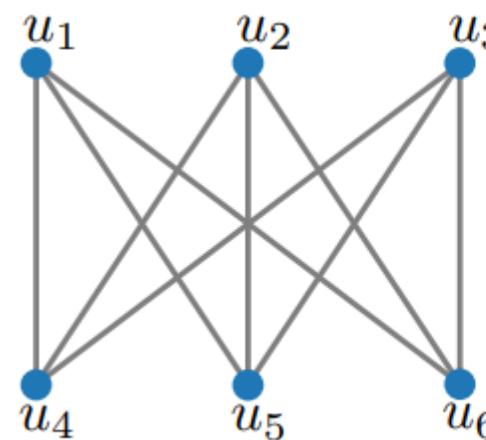
$$a = \sum_{\substack{s,t \in V \\ s \neq t}} \frac{d(s,t)}{n(n-1)}$$

Where V is the set of nodes in G , $d(s, t)$ is the shortest path from s to t , and n is the number of nodes in G

- For graphs $G = (V^G, E^G)$ and $G' = (V^{G'}, E^{G'})$, if we can define a bijection $\psi: V^G \rightarrow V^{G'}$, such that:

$$\forall v_i^G, v_j^G \in V^G, (v_i^G, v_j^G) \in E^G \text{ iff } (\psi(v_i^G), \psi(v_j^G)) \in E^{G'}$$

- The graphs G and G' are said to be isomorphic and denoted as $G \simeq G'$



| V^G | $V^{G'}$ |
|-----------------------|----------|
| $v_1 \rightarrow u_1$ | |
| $v_2 \rightarrow u_4$ | |
| $v_3 \rightarrow u_2$ | |
| $v_4 \rightarrow u_5$ | |
| $v_5 \rightarrow u_3$ | |
| $v_6 \rightarrow u_6$ | |

$\psi :$

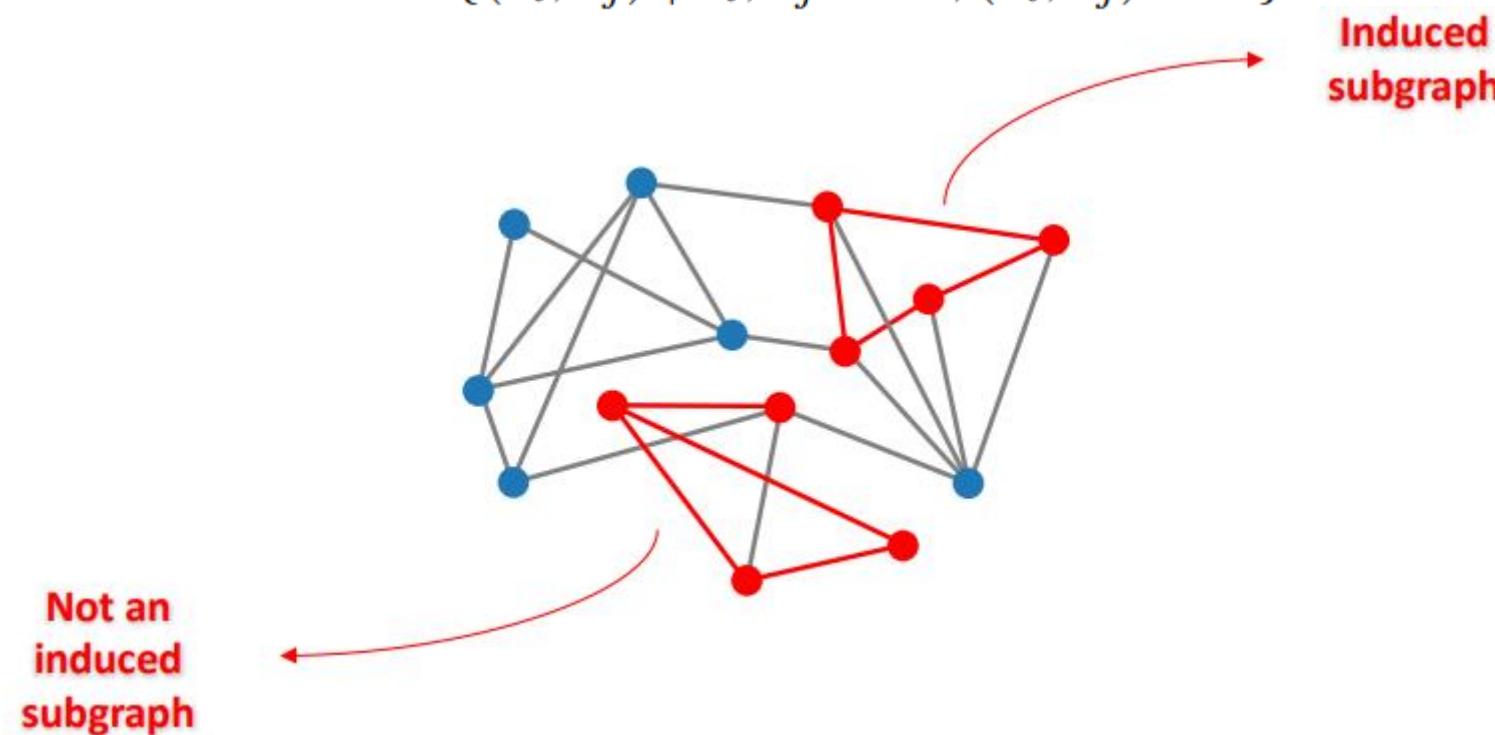
- An induced subgraph is a subgraph $G' = (V', E')$ consist of a subset of the nodes in the graph $G = (V, E)$ and all edges that connect them

$$E' = \{(v_i, v_j) \mid v_i, v_j \in V', (v_i, v_j) \in E\}$$



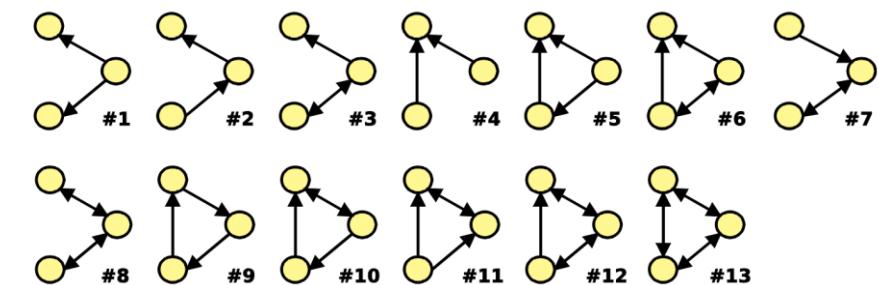
- An induced subgraph is a subgraph $G' = (V', E')$ consist of a subset of the nodes in the graph $G = (V, E)$ and all edges that connect them

$$E' = \{(v_i, v_j) \mid v_i, v_j \in V', (v_i, v_j) \in E\}$$



- A measure of similarity for finding graph similarity

- Given a network, most of the time, some subgraphs are “overrepresented”
 - A connected graph that has many occurrences in a network is called a **motif** of the network
 - Assume set of occurrences G' in G is $occ_G(H)$
 - cardinality of $occ_G(H)$ in G is **frequent**
 - How to know if G' is **frequent** in G ?
- Compute the probability that $occ_N(G') \geq occ_G(G')$ for a random network N
- G' is said to be frequent in G if this probability is small enough
- To compute this probability, we need to have a distribution over networks



- Subgraph Isomorphism:

- Input: Two graphs G' and G
- Question: Does G' has at least one occurrence in G ?
- Determining if a graph contains a clique of size k is already NP-complete

- Occurrences counting:

- Input: Two graphs G' and G
- Question: Determine occurrence $occ_G(H)$
- This problem is #P-complete



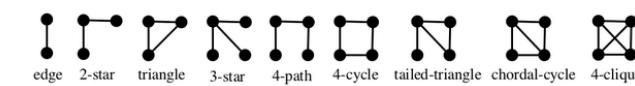
(a) Initial graph



(b) Weighted 4-clique graph



(c) Weighted 4-path graph



(d) Various graph motifs

➤ A quick introduction to Kernels:

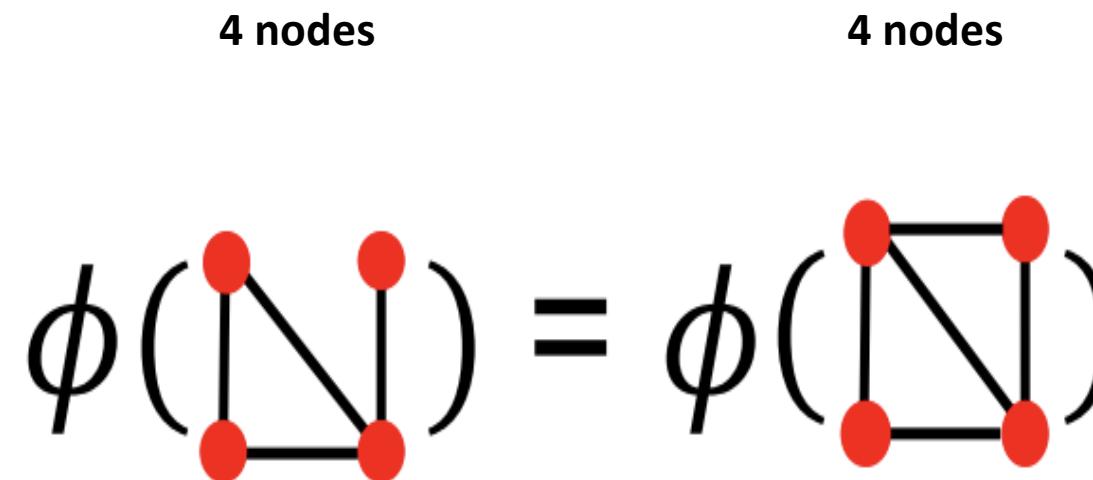
- Kernel $K(G, G') \in \mathbb{R}$ measures similarity between two graphs (data)
- There exists a feature representation $\phi(\cdot)$ such that:

$$K(G, G') = \phi(G)^T \phi(G')$$

- Kernel enables vectors to be operated in higher dimension
- Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions

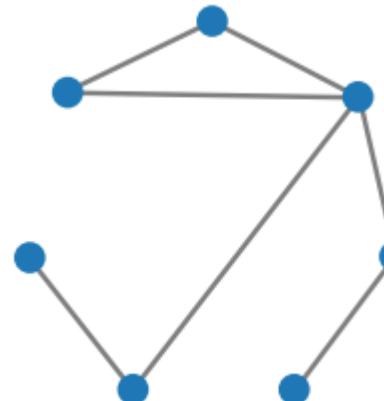
$$K = (\quad \begin{array}{c} \text{graph 1} \\ \text{(red nodes)} \end{array}, \quad \begin{array}{c} \text{graph 2} \\ \text{(red nodes)} \end{array}) \rightarrow \varphi \left(\begin{array}{c} \text{graph 2} \\ \text{(red nodes)} \end{array} \right)$$

- **Goal:** Design graph kernel $\phi(G)$
- **Key Idea:** Bag-of-Words (BoW) for a graph
 - In NLP, BoW counts the word's frequency in a document as feature
 - Simplest way on graph: **Regard nodes as words**
 - For example: in both graph have 4 nodes, we found the features of 2 graphs are same



- Problem: Bag of node counts doesn't work well...
- What if we use **Bag of Node Degrees?**
 - Characterizes a graph G based on frequency of different degree values in that graph.
 - For bag of degrees, the set of nodes σ_n is defined as

$$\sigma_n(G) = \{v_i | v_i \in V^G, d_i = n\}$$



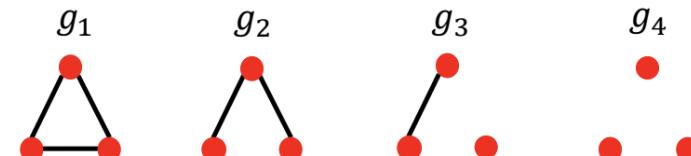
| $d = 1$ | $d = 2$ | $d = 3$ | $d = 4$ |
|---------|---------|---------|---------|
| 2 | 4 | 0 | 1 |

$$\phi_{BN}(G) = (2, 4, 0, 1)$$

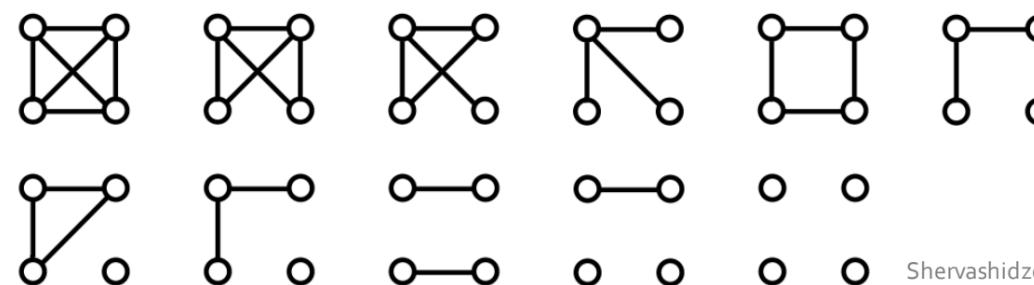
- Does not reflect the global structure of the graph

- **Key Idea:** Count the number of different graphlets in a graph
- Graphlet Differences:
 - Nodes do not need to be connected
 - Graphlets are not rooted
- Let $G_k = (g_1, g_2, g_3, g_4)$ be a list of graphlet size k

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

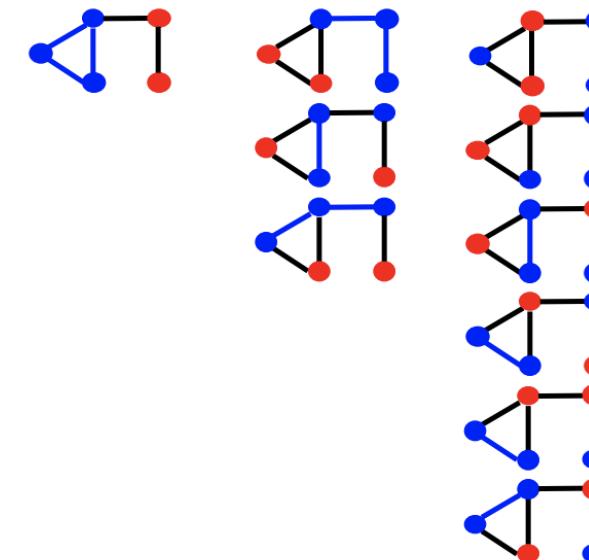
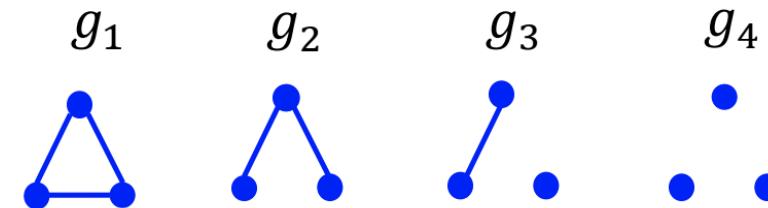
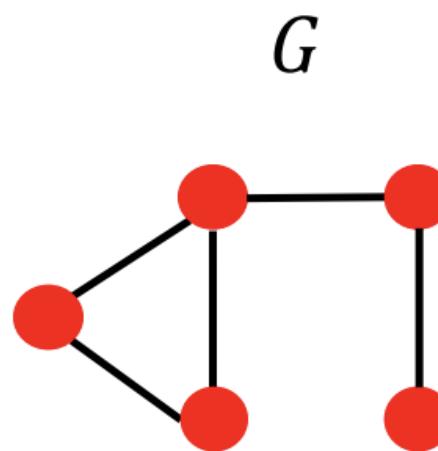


- For $k = 4$, there are 11 graphlets.



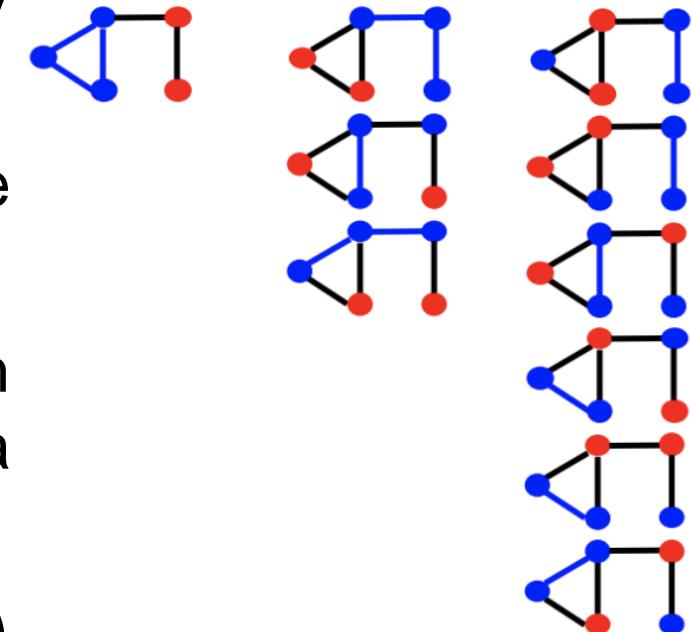
Shervashidze et al., AISTATS 2011

➤ Example for $k = 3$:



$$\mathbf{f}_G = (1, 3, 6, 0)^T$$

- Counting graphlets is expensive!
 - Counting size k graphlets for graph with n nodes by enumeration takes n^k time.
 - If our graph changes with time, we must recalculate the features again.
 - Worst-case scenario: unavoidable since subgraph isomorphism test (judging whether a graph is a subgraph 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 .
- **Can we design a more efficient graph kernel?**



- **Goal:** Design an efficient graph feature descriptor $\phi(G)$
- **Key Idea:**
 - Iteratively use neighbourhood structure to describe node's neighbouring topology
 - Generalized version of **Bag of node degrees** (node degree only contains one-hop neighbourhood information)

→ **Colour Refinement Algorithm**

- For a graph G with nodes V :
- Assign initial colour $c^{(0)}(v)$ to each node v
- Iteratively refine node colours by:

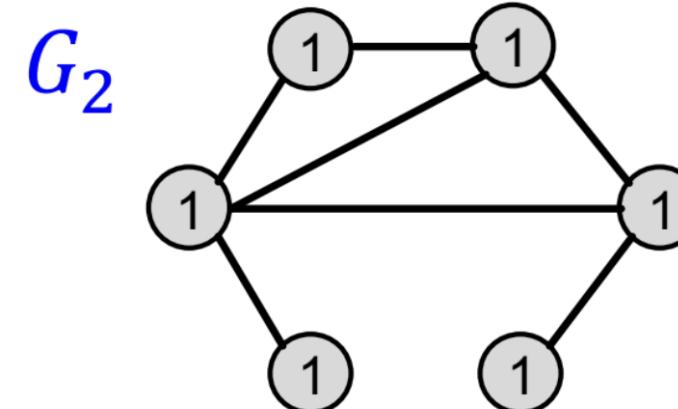
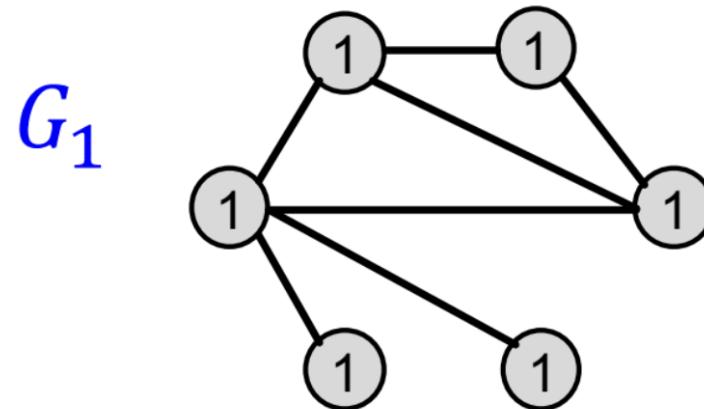
$$c^{(k+1)}(v) = \text{HASH} \left(\left\{ c^{(k)}(v), \left\{ c^{(k)}(u) \right\}_{u \in N(v)} \right\} \right)$$

where HASH maps different inputs to different colours

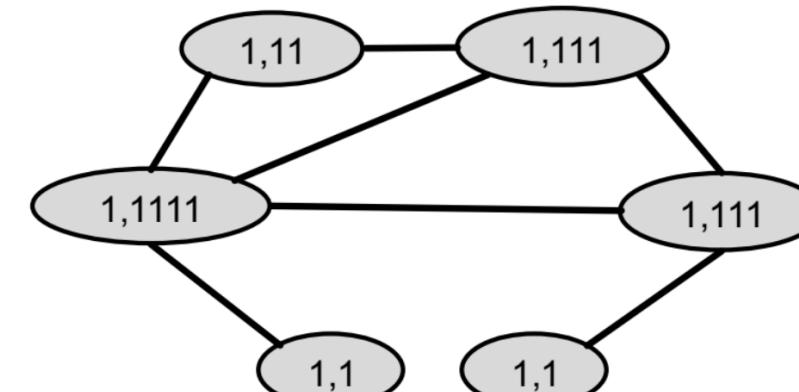
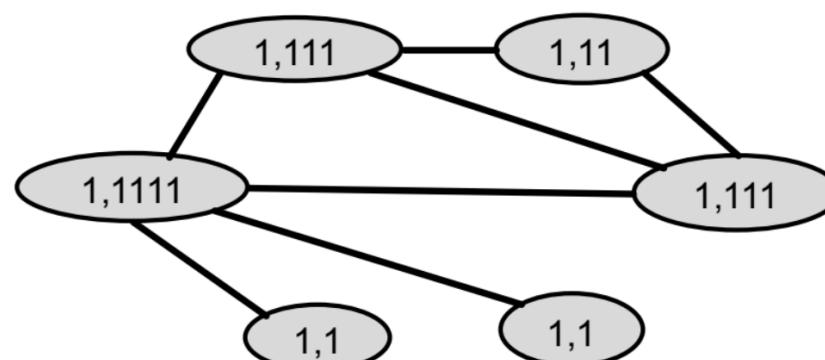
- After K steps of iteration, $c^{(K)}(v)$ represents the structure of the K -hop neighbourhood

➤ Example with two graphs:

➤ Assign initial colour:

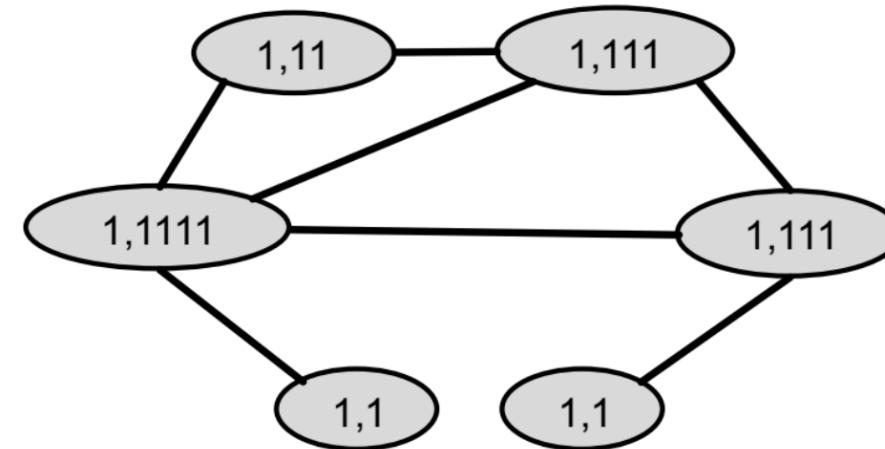
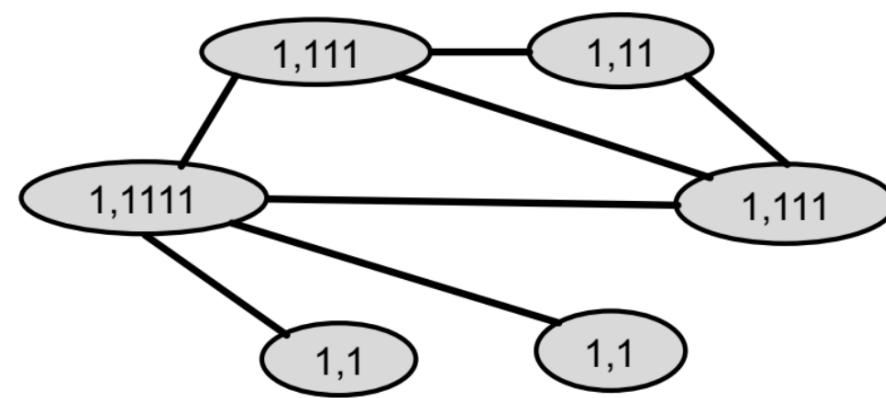


➤ Aggregate neighboring colours:

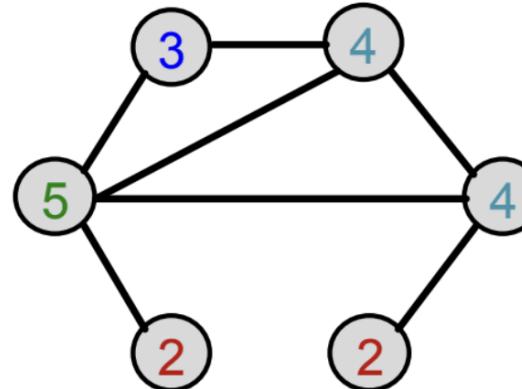
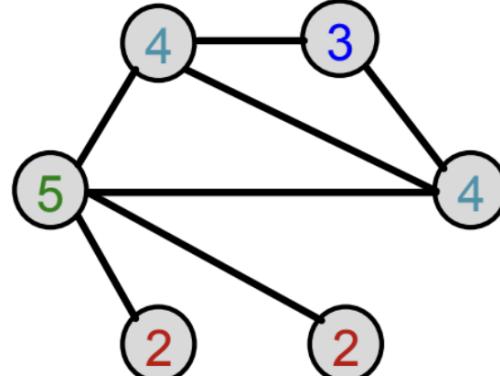


- Example with two graphs:

- Aggregated colours:



- Hash aggregated colours:

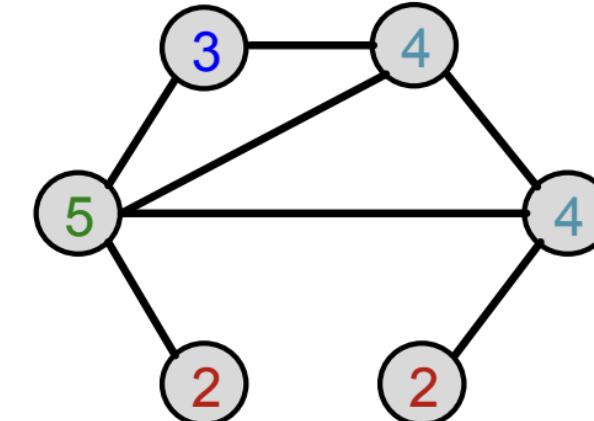
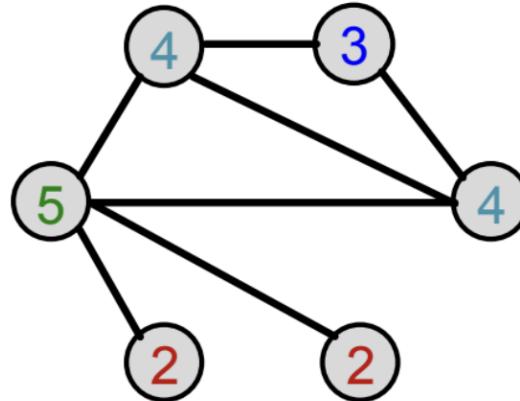


Hash table

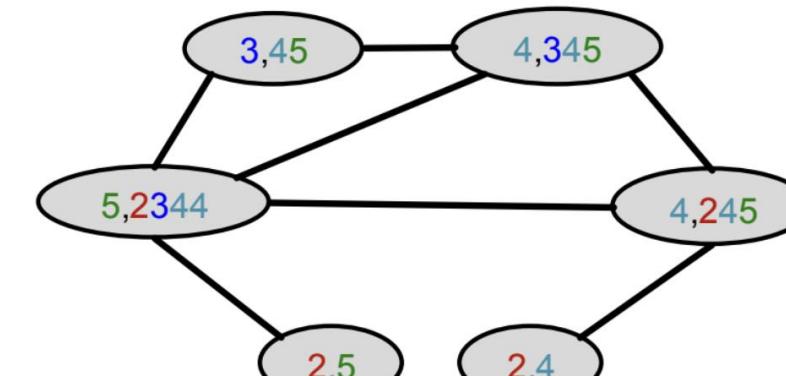
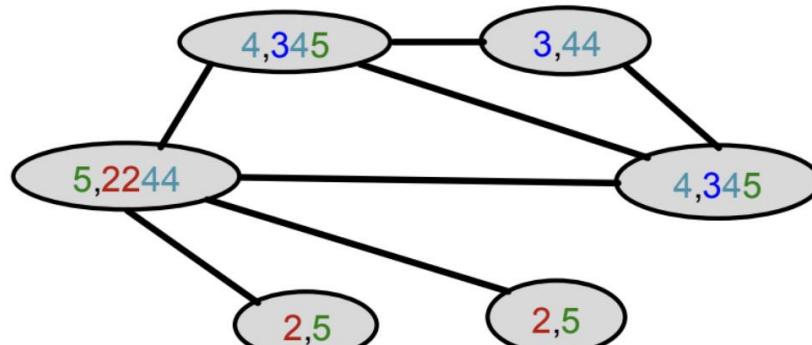
| | | |
|--------|-----|---|
| 1,1 | --> | 2 |
| 1,11 | --> | 3 |
| 1,111 | --> | 4 |
| 1,1111 | --> | 5 |

- Example with two graphs:

- Aggregated colours:

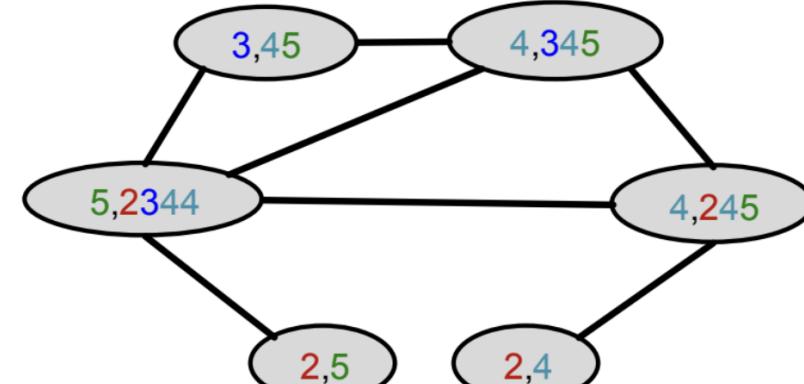
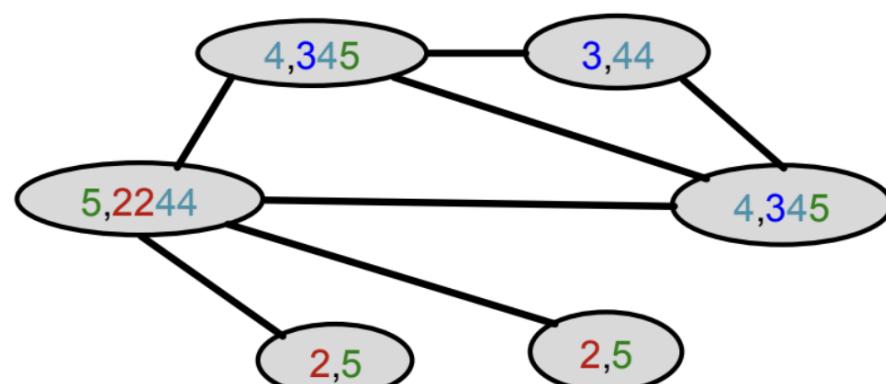


- Hash aggregated colours:

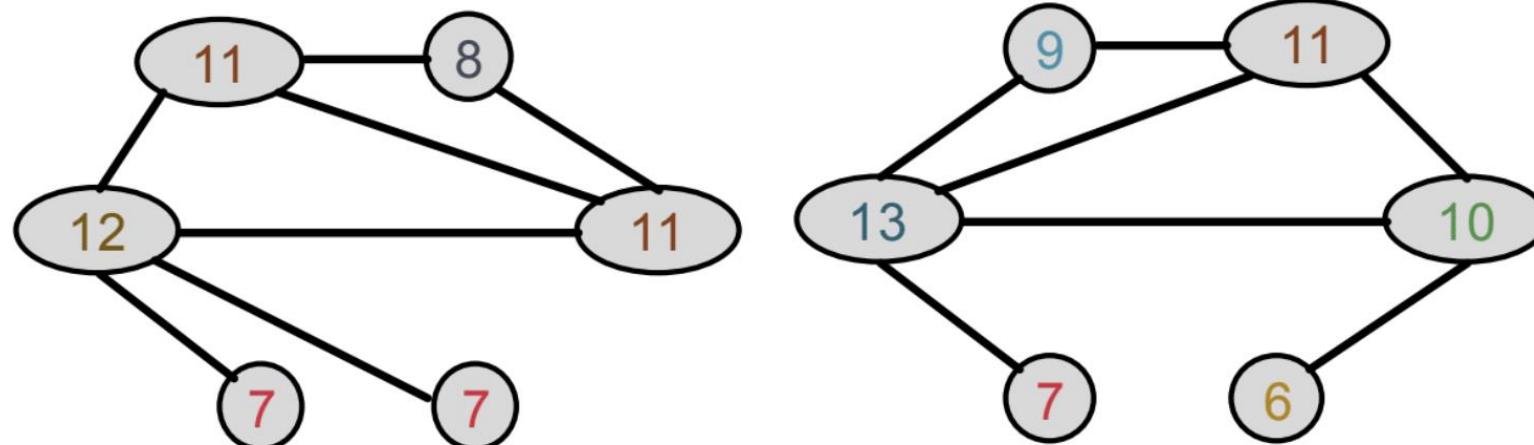


- Example with two graphs:

- Aggregated colours:



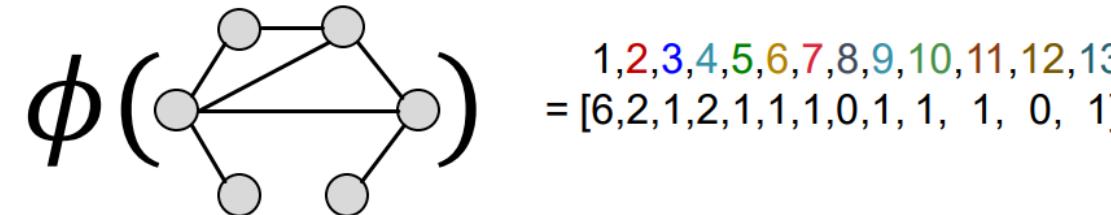
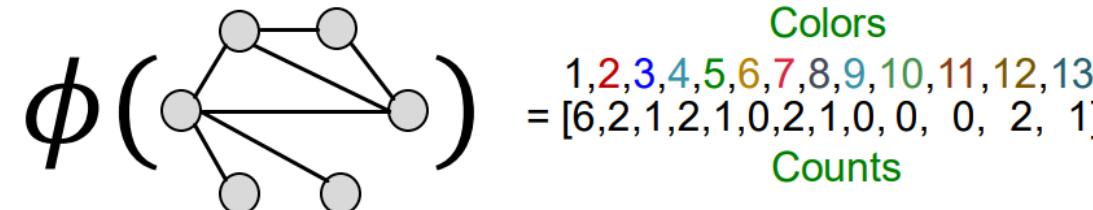
- Hash aggregated colours:



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 |

- After colour refinement, WL Kernel counts # of nodes within given colours:

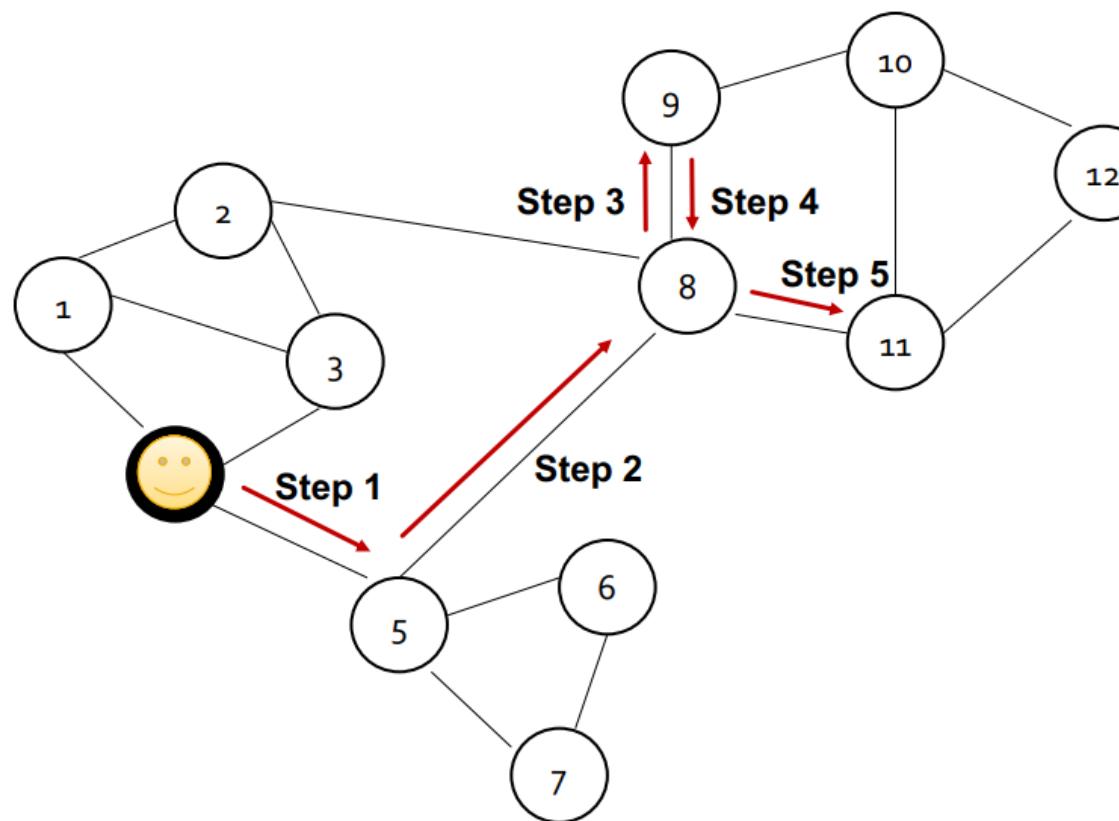


- The WL kernel value is computed by the inner product of the colour count vectors:

$$\begin{aligned} K(\text{graph}, \text{graph}) &= \phi(\text{graph})^T \phi(\text{graph}) \\ &= 49 \end{aligned}$$

- WL kernel benefits:
 - Computationally efficient (colour refinement need #(edges) steps)
 - #(colours) depends on total number of nodes - only colours appeared in the two graphs need to be tracked
 - Time complexity is linear in #(edges)
 - Can be used to check graph isomorphism

- For a graph and a starting point, select a neighbor of it at random, and move to this neighbor
- Select a neighbor of this point at random, and move to it, etc.
- The (random) sequence of points visited this way is a random walk on the graph



- Compute all-pairs-shortest-paths for G and G' via Floyd-Warshall
- Define a kernel by comparing all pairs of shortest path lengths from G and G' :

$$k(G, G') = \sum_{v_i, v_j \in G} \sum_{v'_k, v'_l \in G'} k_{length}(d(v_i, v_j), d(v'_k, v'_l))$$

where $d(v_i, v_j)$ is the length of the shortest path between node v_i and v_j

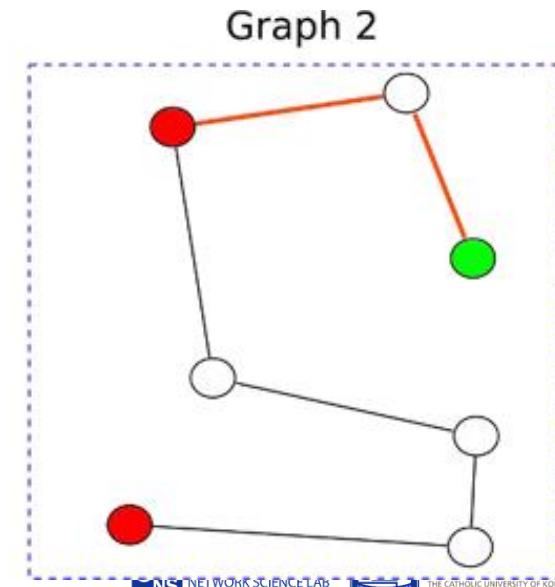
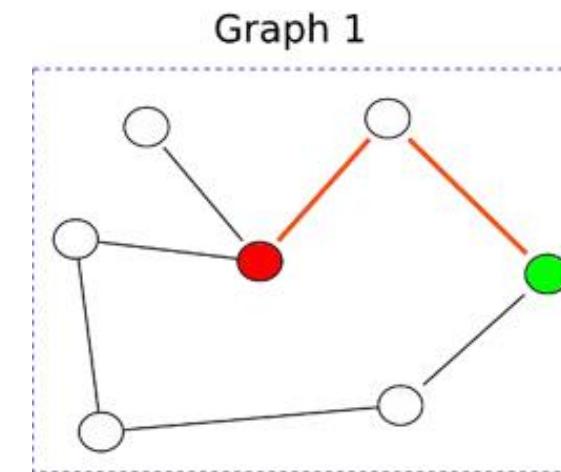
- k_{length} is a kernel that compares the lengths of two shortest paths:

- A linear kernel

$$k(d(v_i, v_j), d(v'_k, v'_l)) = d(v_i, v_j) * d(v'_k, v'_l)$$

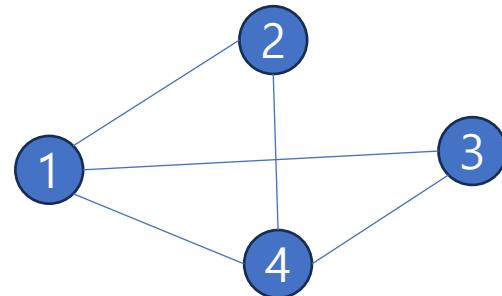
- A delta kernel

$$k(d(v_i, v_j), d(v'_k, v'_l)) = \begin{cases} 1, & \text{if } d(v_i, v_j) = d(v'_k, v'_l) \\ 0, & \text{otherwise} \end{cases}$$



- RWR [Kashima+ '03, Gaertner+ '03, Vishwanathan '10]
- Shortest path kernels [Borgwardt & Kriegel '05]
- Cyclic path kernels [Horvath+ '04]
- Depth-first search kernels [Swamidass+ '05]
- Subtree kernels [Shervashidze+ '09 NIPS, JMLR'11 , Ralaivola+'05]
- Graphlet / Subgraph kernels [Shervashidze+ '09, Thoma+ '10]
- All-paths kernels [Airola+ '08]
- ...

➤ Graph and Associated Matrices



$$G = (V, E)$$

$$|V| = 4$$

$$|E| = 5$$

➤ Laplacian matrix

$$\begin{aligned} L_G &= D_G - A_G \\ &= B_G B_G^T \end{aligned}$$

➤ Adjacency matrix

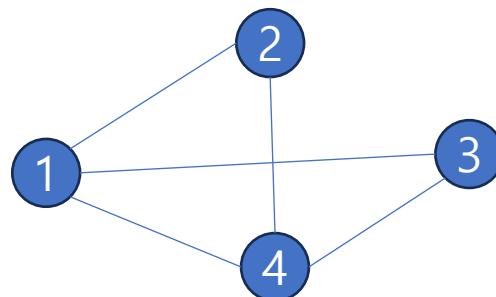
$$A_G = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

➤ Degree matrix

$$D_G = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

➤ Incidence matrix

$$B_G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & -1 \end{bmatrix}$$



$$L_G = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & 0 & -1 \\ -1 & 0 & 2 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

Eigenvalue

$$\lambda = \{0, 2, 4, 4\}$$

Eigenvector

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ -1 \\ 3 \end{bmatrix}, \begin{bmatrix} -2 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

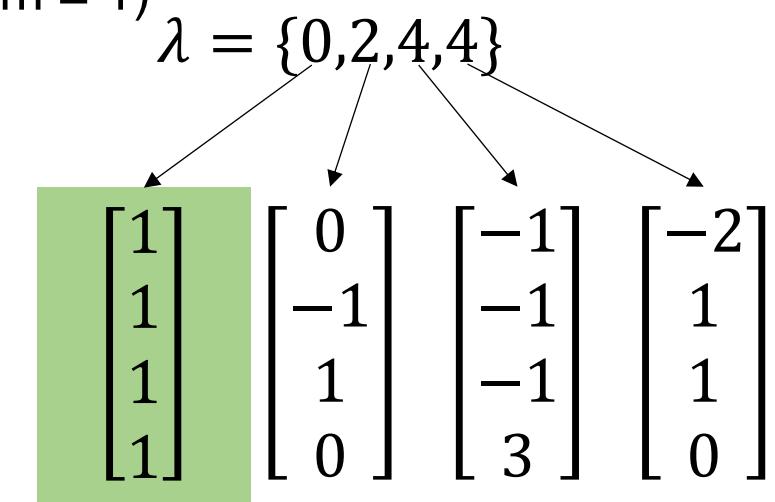
- Symmetric \rightarrow real eigenvalues; eigenvectors are mutually orthogonal
- Orthogonally diagonalizable \rightarrow an eigenvalue with multiplicity k has k -dimensional eigenvectors

- For entries $x = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$

$$x^T L_G x = x^T B_G B_G^T x = (x^T B_G)(B_G^T x) = \sum_{(i,j) \in E} (x_i - x_j)^2 \geq 0$$

- Meaning that: positive semidefinite \rightarrow eigenvalues in Laplacian is non-negative
- Row sum = 0 \rightarrow singular
 - At least one eigenvalue = 0, unity eigenvector (since row sum = 1)
- Orthogonal eigenvector u = eigenvector of non-zero eigenvalue

$$\sum_{i=1}^n u_i = 0$$



➤ Which graphs are determined by their spectrum?

- Complete Graphs
- Graphs with one edge
- Graphs missing 1 edge
- Regular graphs with degree 2
- Regular graphs of degree $n - 3$

| | Eigenvalues |
|----------|--|
| Complete | $\{0, n^{n-1}\}$ |
| Line | $2 - 2 \cos\left(\frac{\pi k}{n}\right), k = 1, \dots, n$ |
| Ring | $2 - 2 \cos\left(\frac{2\pi k}{n}\right), k = 1, \dots, \frac{n}{2}$ |
| Star | $\{0, 1^{n-2}, 2\}$ |

- Reminder: Communities in networks is defined with respect to the whole graph.

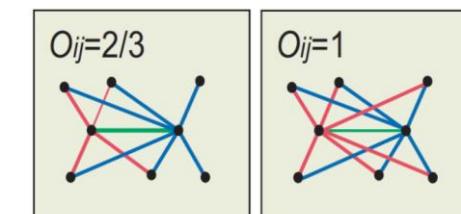
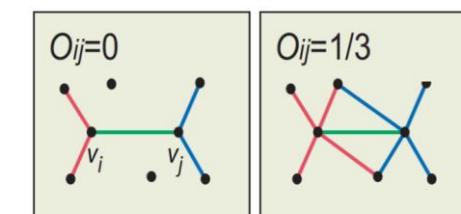
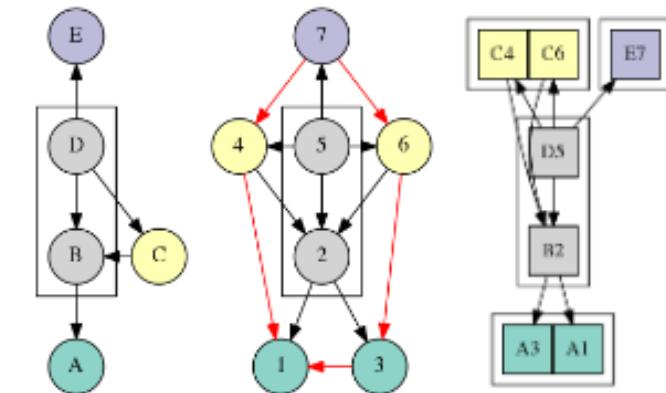
- identifying sets of nodes that are densely connected with each other.

- A graph has a community structure if it is different from a random graph.

- Common methods:

- Structural equivalence: share the same neighbors and are not adjacent
- Edge Overlap between the number of neighbor $N(i)$ and $N(j)$ of node i and j

$$O_{ij} = \frac{|(N(i) \cap N(j)) \setminus \{i, j\}|}{|(N(i) \cup N(j)) \setminus \{i, j\}|}$$



➤ Local:

- Fundamental: Graph Size, Density, Connectivity, Node Degree
- Graph Centrality: Betweenness, Closeness, Eigenvector
- Average Clustering Coefficient
- Diameter: Length of the shortest path between the most distanced nodes
- Average Shortest Path Length: A average number of steps along the shortest paths for all possible pairs of nodes

➤ Global:

- Subgraph/Motif counts
- Kernel-based: Graphlet, Weisfeiler-Lehman, Random Walk, Shortest Path, etc.
- Spectral: Laplacian matrix
- Community Structure

➤ There are many graph-level tasks:

- Graph regression
- Graph classification
- Graph generation
- Graph clustering
- Graph similarity
- Graph embeddings

➤ Graph regression:

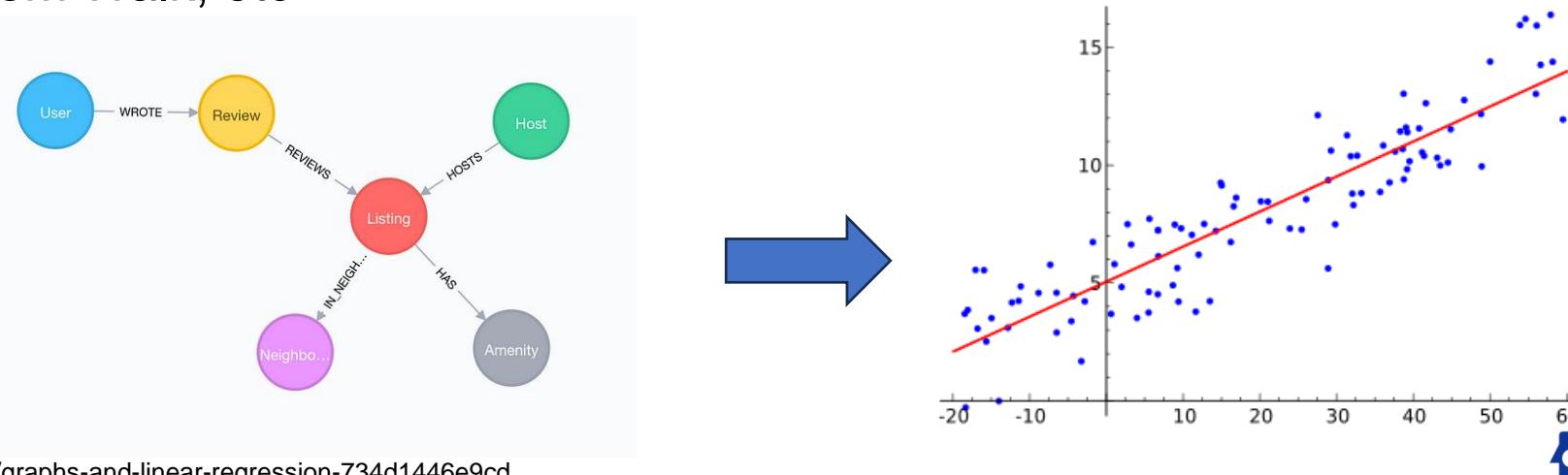
- Given a set of graphs $G = \{G_1, G_2, \dots, G_n\}$ and their associated target value $y = \{y_1, y_2, \dots, y_n\}$
- The goal is to learn a function f that map each graph G_i to a continuous target value $\hat{y}_i = f(G_i)$ such that the predicted values \hat{y}_i are close to the true values

$$\hat{y}_i = f(G_i) + \epsilon_i$$

Where ϵ_i represents the error or residual between the predicted and true values

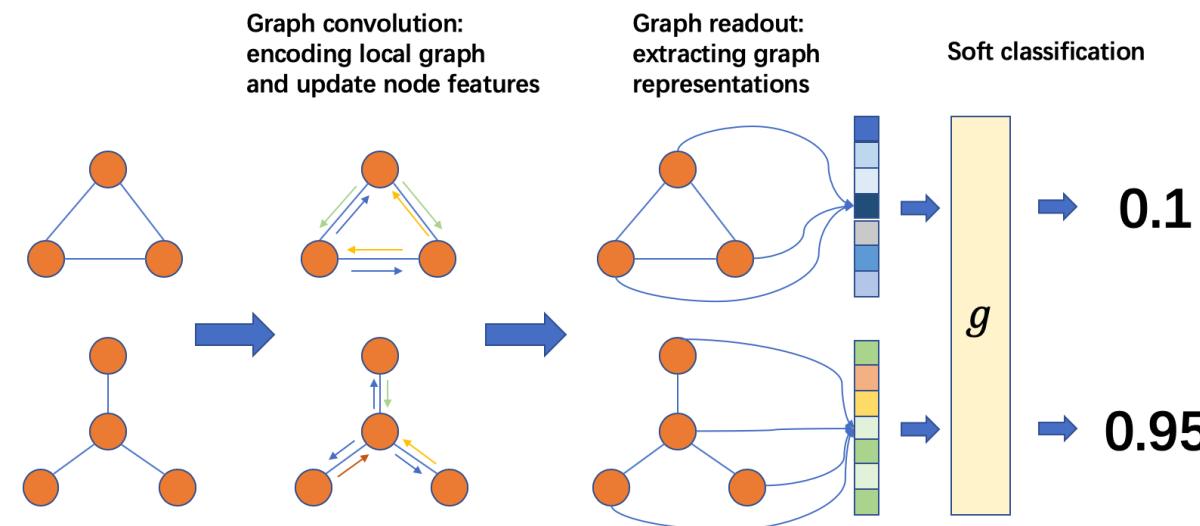
➤ Conventional methods

- Kernel method: Subgraph Matching, Weisfeiler-Lehman Subtree Kernel, Shortest path, Random Walk, etc.



➤ Graph classification:

- Given a set of attributed graphs $D = \{(G_1, l_1), (G_2, l_2), \dots, (G_n, l_n)\}$, the goal is to learn a function $f : G \rightarrow L$, where G is the input space of graphs and L is the set of graph labels
- Each graph $G_i = (A_{G_i}, D_{G_i})$ is comprised of an adjacency matrix A_{G_i} , an attribute matrix D_{G_i} and corresponding label l_i
- Conventional methods
 - Kernel method: Subgraph matching, Weisfeiler-Lehman Subtree Kernel, Shortest path, Random walk, etc.



- Counts the number of matchings between subgraphs of bounded size in two graphs
- Given two graphs $G = (V, E)$ and $G' = (V', E')$:
 - $B(G, G')$ denote the set of all bijections between sets $S \subseteq V$ and $S' \subseteq V'$
 - $\lambda: B(G, G') \rightarrow \mathbb{R}^+$ be a weight function
 - The subgraph matching kernel is defined as

$$k(G, G') = \sum_{\phi \in B(G, G')} \lambda(\phi) \prod_{v \in S} \kappa_V(v, \phi(v)) \prod_{e \in S \times S} \kappa_E(e, \psi(e))$$

Where $S = \text{dom}(\phi)$ and κ_V, κ_E are kernel functions defined on vertices and edges

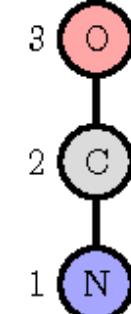
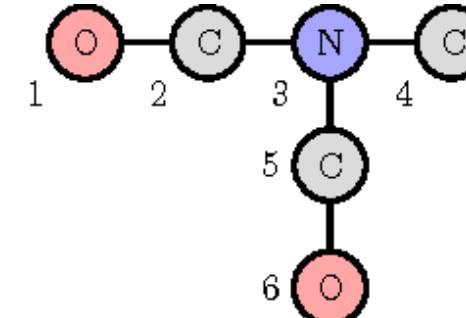
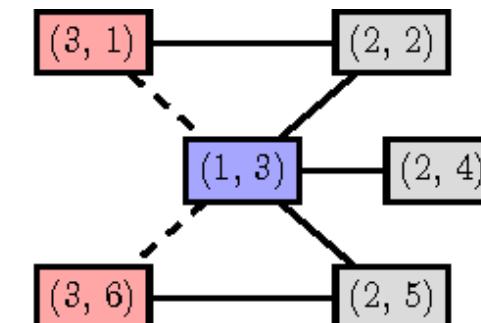
- The instance of the subgraph matching kernel that is obtained if we set the κ_V, κ_E functions as follows

$$\kappa_V(v, v') = \begin{cases} 1, & \text{if } \ell(v) \equiv \ell(v'), \\ 0, & \text{otherwise and} \end{cases}$$

$$\kappa_E(e, e') = \begin{cases} 1, & \text{if } e \in E \wedge e' \in E' \wedge \ell(e) \equiv \ell(e') \text{ or } e \notin E \wedge e' \notin E', \\ 0, & \text{otherwise.} \end{cases}$$

is known as the common subgraph isomorphism kernel

- This kernel counts the number of isomorphic subgraphs contained in two graphs

(a) G_1 (b) G_2 (c) Product graph G_P

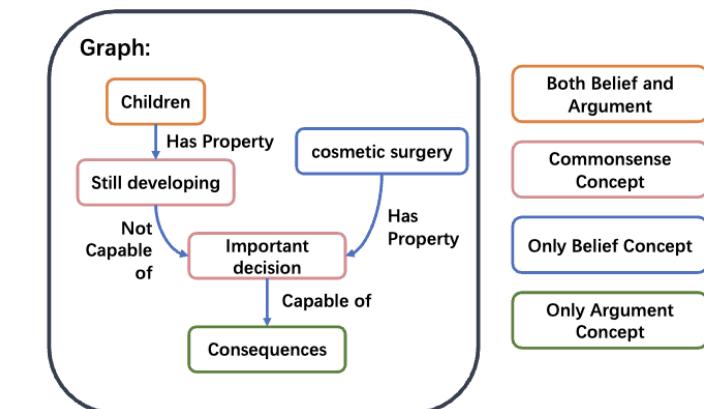
➤ Graph generation:

- A quadruplet graph $G = (V, E, X, \mathbf{E})$, where V is the vertex set, $E \subseteq V \times V$ is the edge set, $X \in \mathbb{R}^{N \times D}$ is the node feature matrix, $\mathbf{E} \in \mathbb{R}^{N \times N \times F}$ is the edge attributes, and D, F are the feature dimensionality
- Given a set of M observed graphs $\mathcal{G} = \{G_i\}_{i=1}^M$, graph generation learns the distribution of these graphs $p(G)$, from which new graphs can be sampled $G_{new} \sim p(\mathcal{G})$
- Conventional methods
 - Preferential Attachment

Belief : Children should be able to consent to cosmetic surgery .
 Argument : Children do not have the mental capacity to understand the Consequences of medical decisions.
 Stance: Counter

$$PA = |N(x)| \cdot |N(y)| = d_x \cdot d_y$$

Where $N(x), N(y)$ are number of neighbors of node x and y



➤ Graph clustering:

- Given a set of graphs, partitioning a set of graphs into different groups that share some form of similarity
- Conventional methods:
 - Hierarchical clustering, K-means clustering, spectral clustering
 - Modularity: To qualify the graph partitions
 - Louvian Algorithm: to find partition of a given network with high modularity score

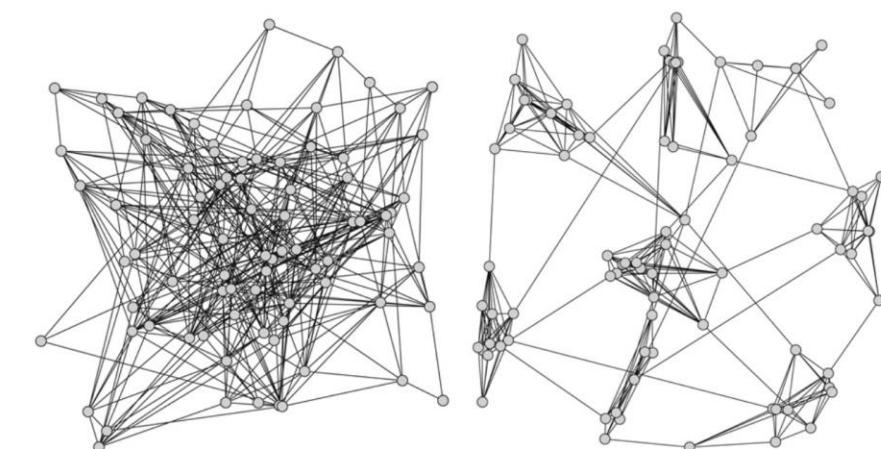
$$Q = \frac{1}{4m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

normalization

adjacency matrix

probability a random edge would go between i and j

$m = \# \text{ edges in graph}$
 $k_i = \text{degree}(i)$

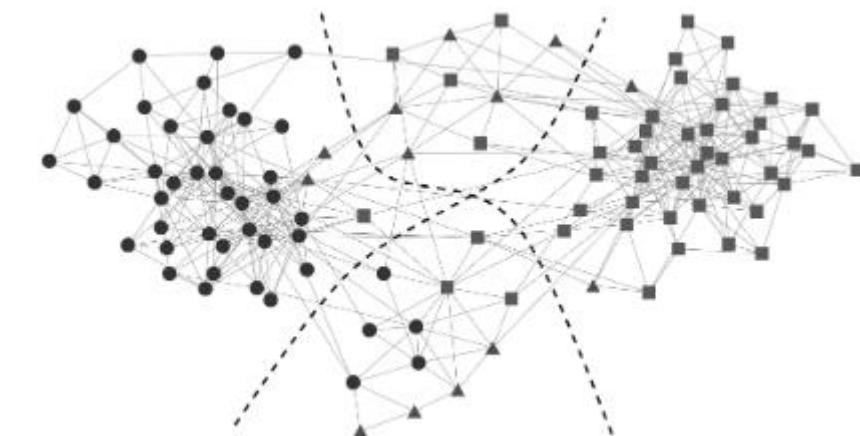


- Normally, modularity only consider to qualify into two graph partition
- To divide network into more than two communities:

$$\begin{aligned}\Delta Q &= \frac{1}{2m} \left[\frac{1}{2} \sum_{i,j \in g} B_{ij}(s_i s_j + 1) - \sum_{i,j \in g} B_{ij} \right] \\ &= \frac{1}{4m} \left[\sum_{i,j \in g} B_{ij} s_i s_j - \sum_{i,j \in g} B_{ij} \right] \\ &= \frac{1}{4m} \sum_{i,j \in g} \left[B_{ij} - \delta_{ij} \sum_{k \in g} B_{ik} \right] s_i s_j \\ &= \frac{1}{4m} \mathbf{s}^T \mathbf{B}^{(g)} \mathbf{s},\end{aligned}$$

Where δ_{ij} is the Kronecker delta-symbol, $s_i^2 = 1$ and B^g is the $n_g \times n_g$ matrix with elements indexed by the labels i, j of nodes within group g and having values

$$B_{ij}^{(g)} = B_{ij} - \delta_{ij} \sum_{k \in g} B_{ik}.$$

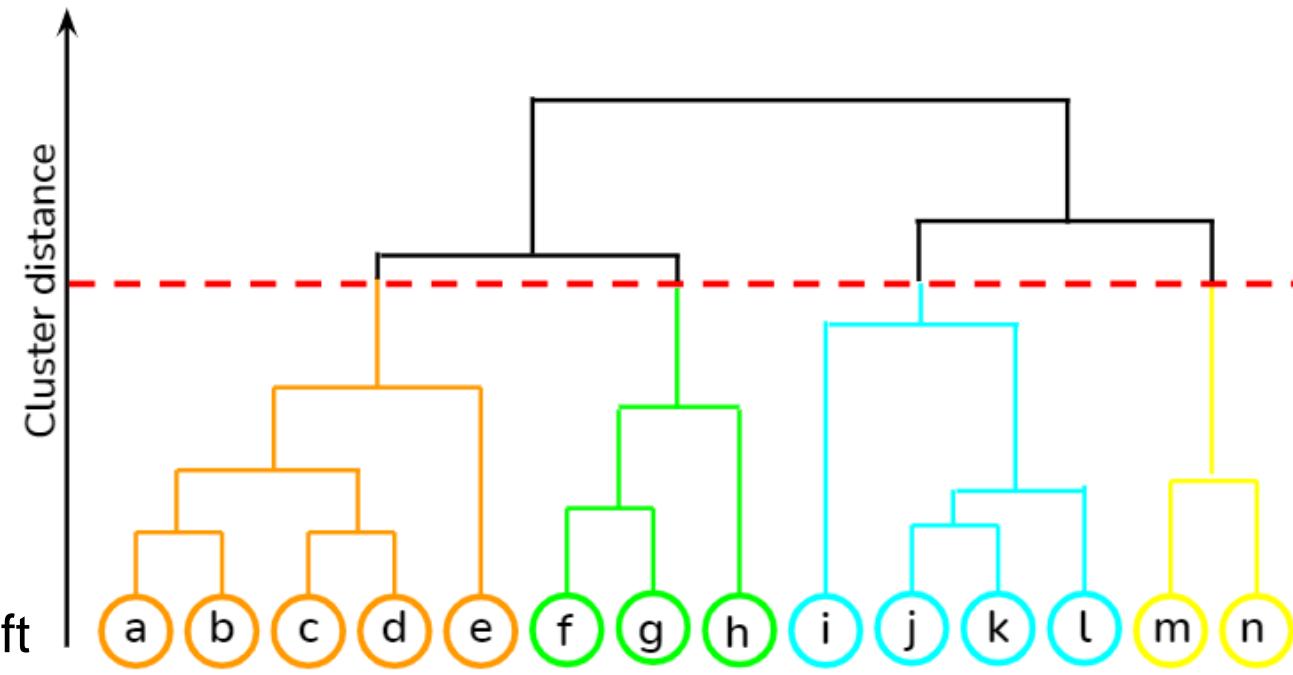


➤ Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Two approaches:

- **Agglomerative** ("bottom up"): each point starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy – **popular one**
- **Divisive** ("top down"): all points start in one cluster, and splits are performed recursively as one moves down the hierarchy:

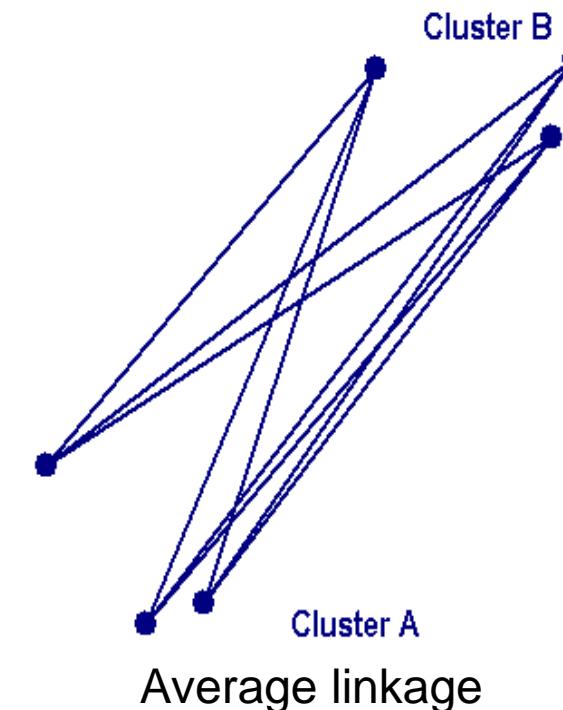
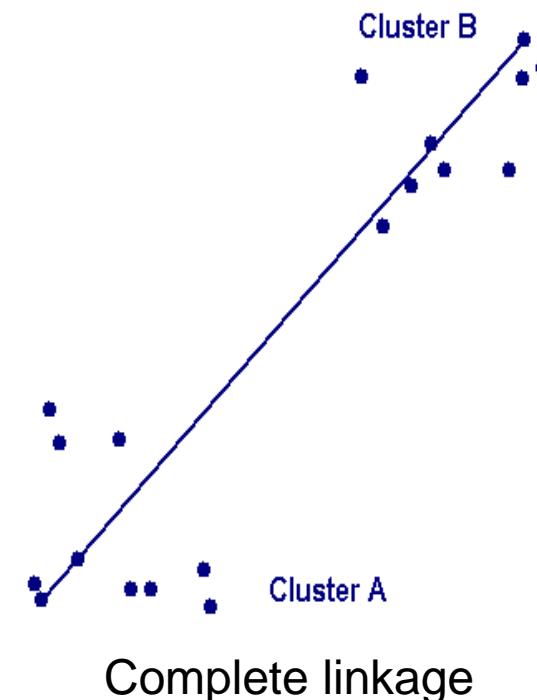
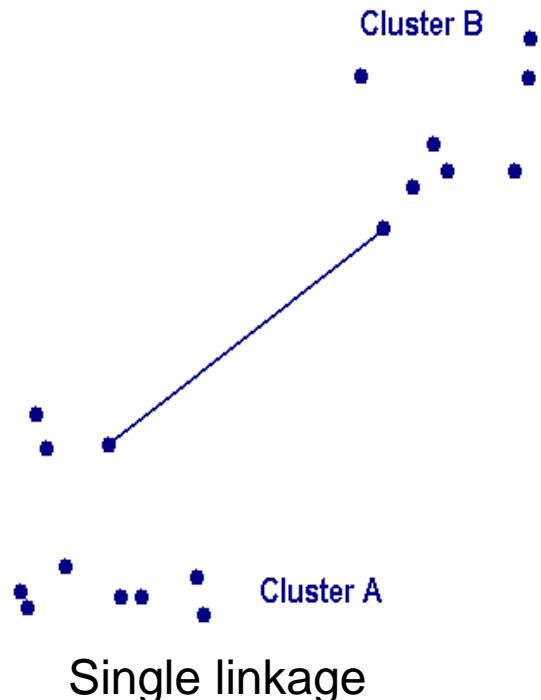
➤ Agglomerative clustering algorithm:

- Maintain a set of clusters
- Initially, each instance in its own cluster
- Repeat:
 - Pick the two **closest** clusters
 - Merge them into a new cluster
 - Stop when there's only one cluster left



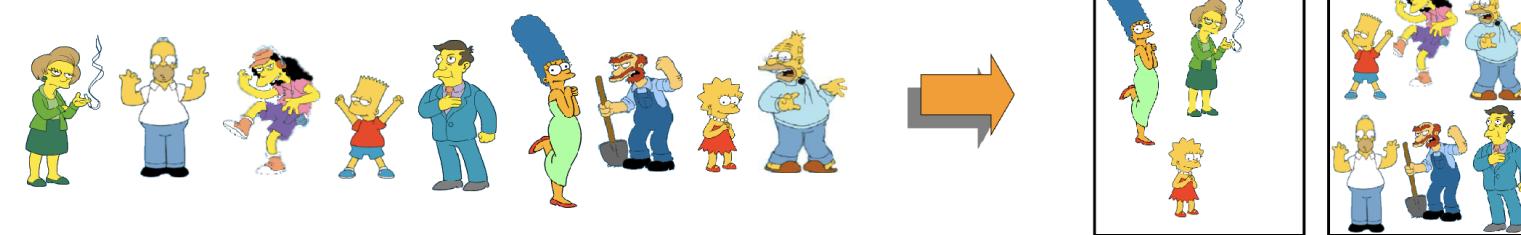
➤ Three Different Distance Measures to define **closest** cluster:

- Minimum distance / Closest distance (single linkage)
- Maximum distance / Farthest distance (complete linkage)
- Group Average (average linkage)



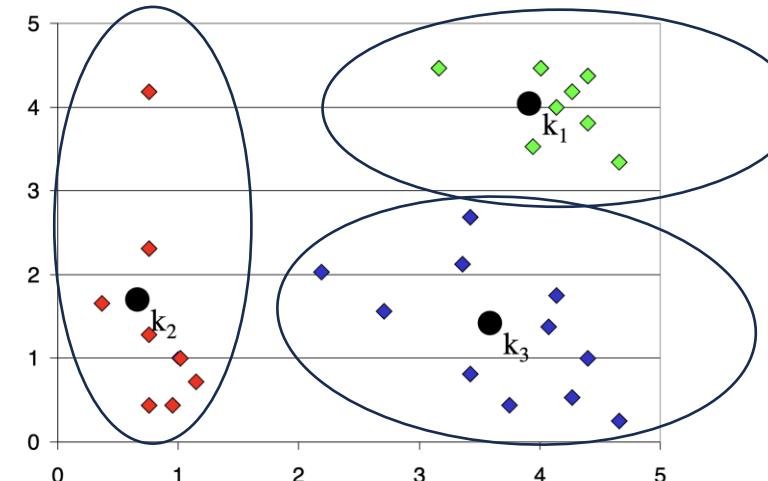
➤ Partition clustering:

- Nonhierarchical, each instance is placed in exactly one of K non-overlapping clusters
- Since the output is only one set of clusters the user has to specify the desired number of clusters K



➤ K-mean clustering algorithm:

- Re-assign and move centers, until ...no objects changed membership.

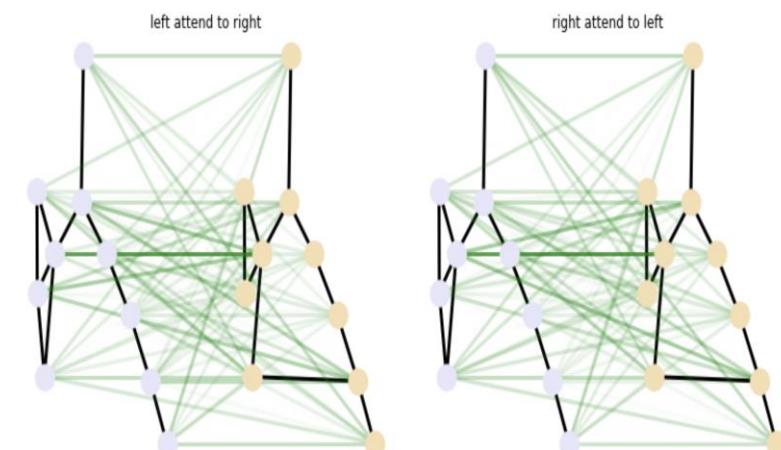


➤ Graph similarity:

- Given two graphs G_1 and G_2 , a graph similarity model can be written as a function $f(G_1, G_2)$ that computes a scalar similarity value
- Output: find the similarity score between graphs (from 0 to 1)
- Conventional methods:
 - Graph matching, isomorphism testing, etc.
 - Graph edit distances

$$GED(g_1, g_2) = \min_{(e_1, \dots, e_k) \in P(g_1, g_2)} \sum_{i=1}^k c(e_i)$$

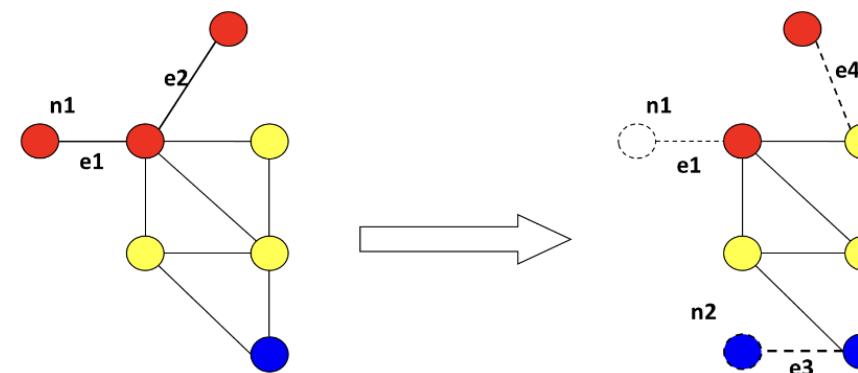
Where $P(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2 and $c(e) \geq 0$ is the cost of each graph edit operation e .



- Compute the minimum cost to transform g_1 into g_2
- Assign cost for operations, e.g.,
 - Node or edge insertion
 - Node or edge deletion
 - Node or edge substitution

$$GED(g_1, g_2) = \min_{(e_1, \dots, e_k) \in P(g_1, g_2)} \sum_{i=1}^k c(e_i)$$

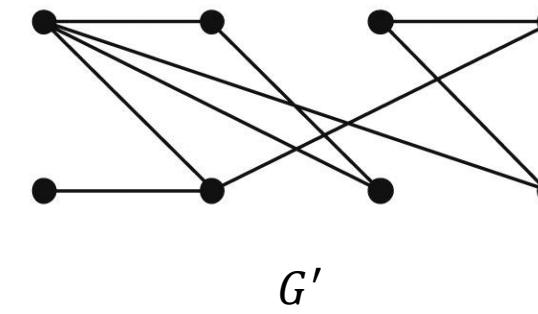
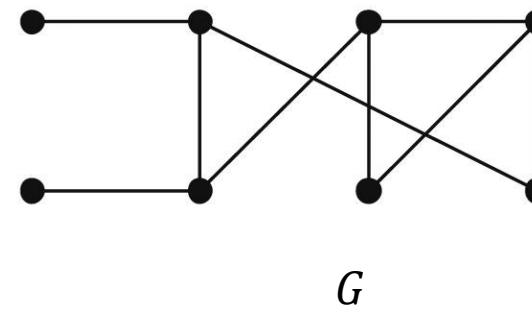
Where $P(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2 and $c(e) \geq 0$ is the cost of each graph edit operation e



- Given graph G and G' of size n with adjacency spectra λ^A and $\lambda^{A'}$
- Adjacency spectral

$$d_A(G, G') \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^n (\lambda_i^A - \lambda_i^{A'})^2},$$

- Smaller distance means that two graph are similarity
- Spectral distances do not require node correspondence
- Distance between the two graphs is defined:



- Given two graphs, G and G' , graphlet kernel is computed as

$$K(G, G') = \mathbf{f}_G^T \mathbf{f}_{G'}$$

- **Problem:** if G and G' have different sizes, that will greatly skew the value
- **Solution:** normalize each feature vector:

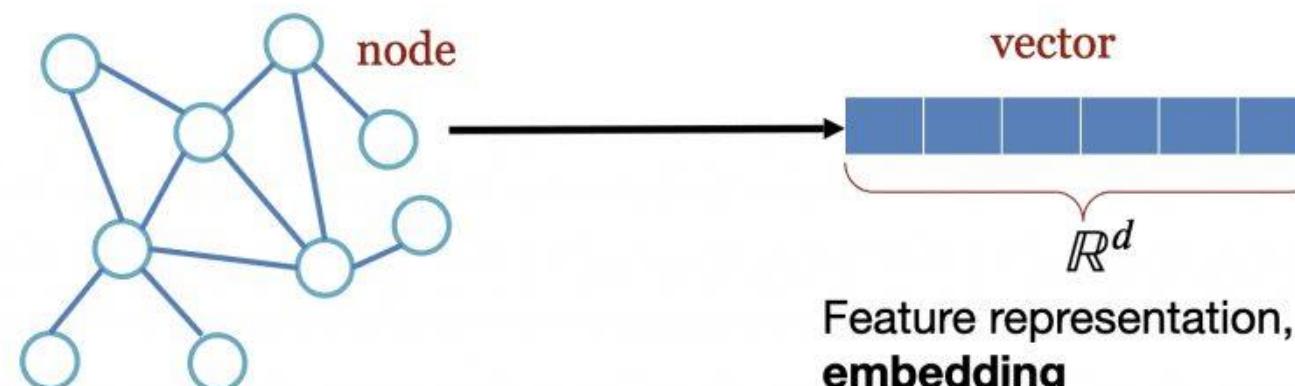
$$\mathbf{h}_G = \frac{\mathbf{f}_G}{\text{Sum}(\mathbf{f}_G)} \quad K(G, G') = \mathbf{h}_G^T \mathbf{h}_{G'}$$

➤ Graph Embedding:

- Given a graph $G = (N, E)$, where V is the vertex set, $E \subseteq V \times V$ is the edge set. Graph embedding aims to learn a low-dimensional vector representation x_i for each node v_i from embedding function Φ maps: $x_i = \Phi(v_i) \in \mathbb{R}^d$

where d is the dimensionality of the embedding space

- Goal: to learn embeddings that preserve important structural properties of the graph, such as node proximity or graph topology, in the low-dimensional space



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

12

- Graph Embedding:

- Conventional methods
 - Euclidean distance and Cosine similarity

$$\cos(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_i A_i B_i}{\sqrt{\sum_i A_i^2} \sqrt{\sum_i B_i^2}}$$

Where A, B denotes two vectors of two graphs

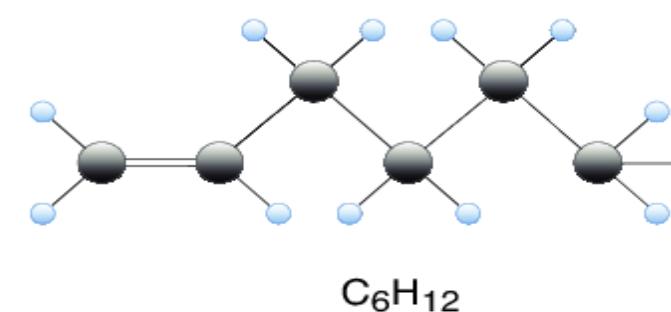
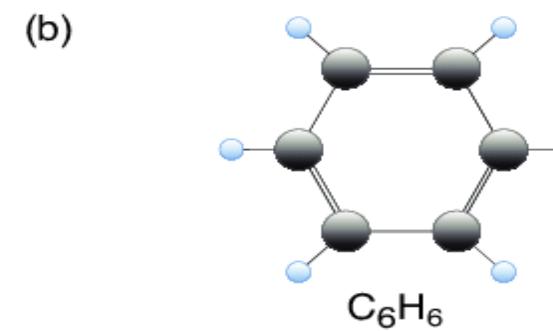
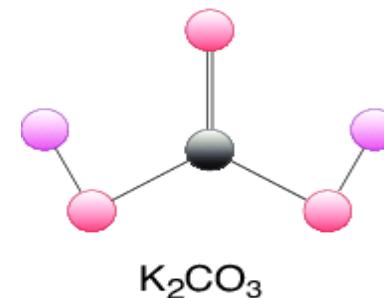
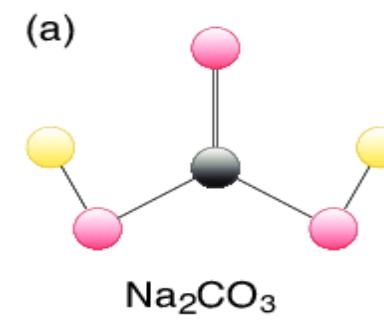
- Laplacian matrix factorization: singular value decomposition, nonnegative matrix factorization
 - Singular value decomposition:

$$A = UDV^T$$

Where A denotes adjacency matrix. U, V are orthonormal and the matrix D is diagonal with positive real entries

➤ Drug Discovery:

- Classifying a graph itself into different categories
- Input: A collection of graphs
- For example, determining if a chemical compound is toxic or non-toxic by looking at its graph structure



- Programming code recommendation: graph classification tasks

- Program-derived graphs

- Contains bugs or not

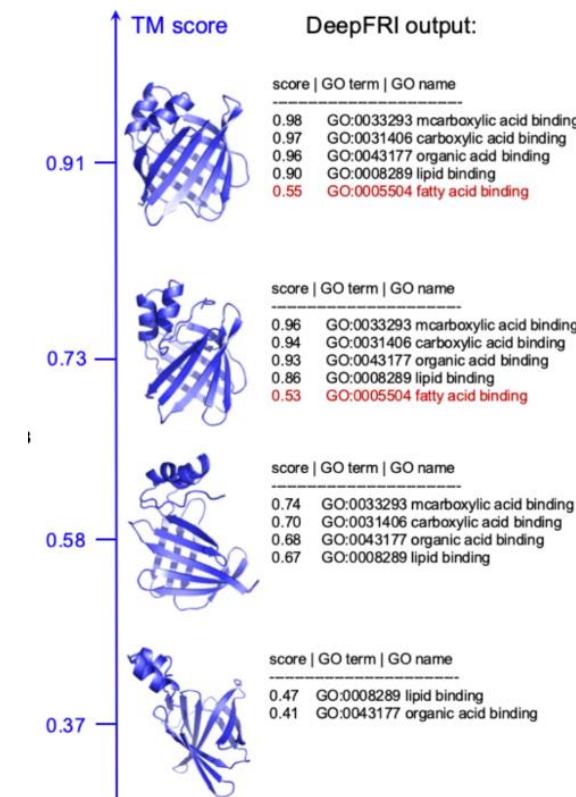
```
function clearEmployeeListOnLinkClick() {
  document.querySelector("a").addEventListener("click",
    function(event) {
      document.querySelector("ul").innerHTML = "";
    }
);
}

if (matches) {
  return {
    episode: Number(matches.groups.episode),
    hosts: matches.groups.hosts.split(/([,\s]+|\sand\s)/).map(el => S(el).trim().s)
  };
}
```

- Drug Discovery: graph regression tasks

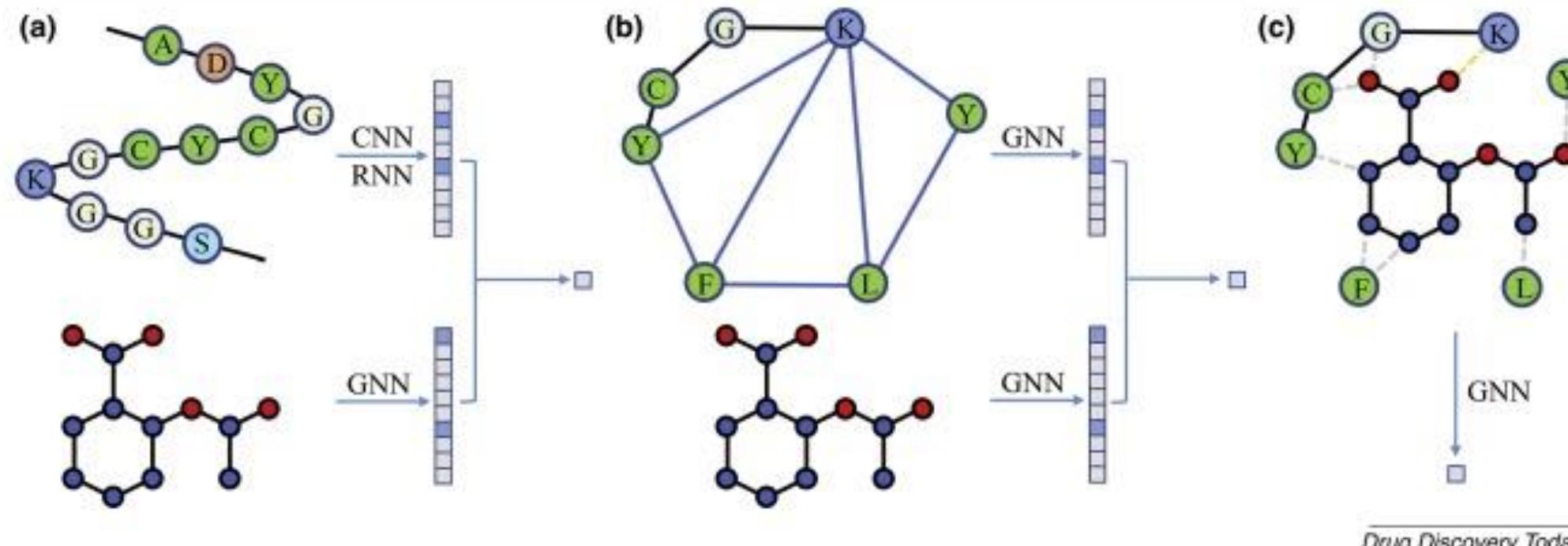
- Proteins

- Predict protein function from molecular

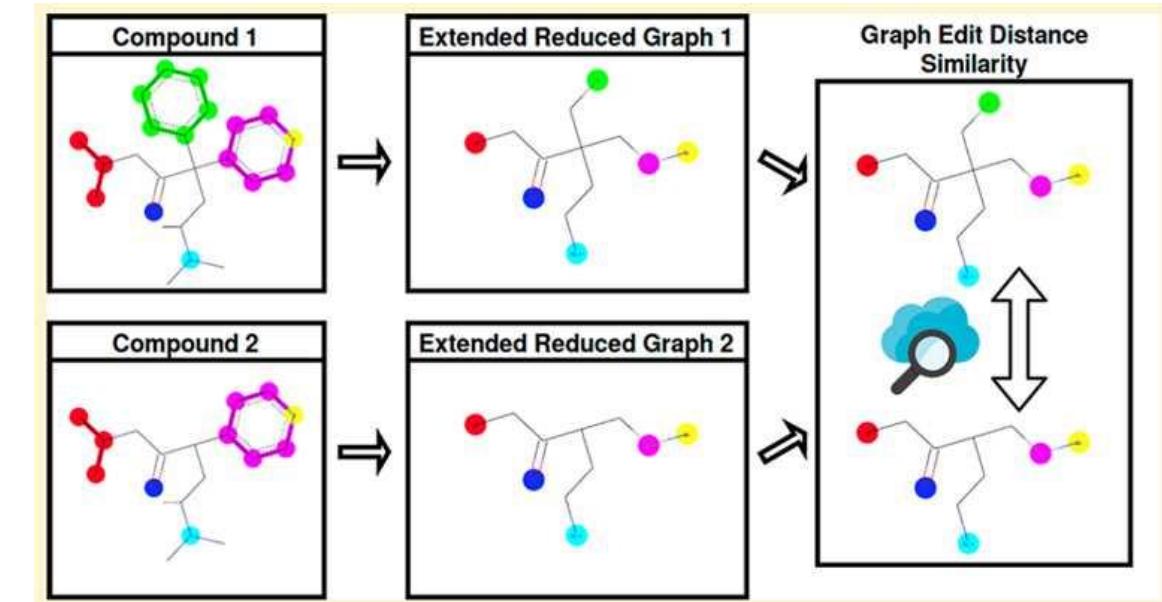


➤ Drug-Discovery:

- Molecular-design for drug-discovery using graph generation:
 - Generate syntactically valid molecules.
 - Generated molecule should possess certain properties.

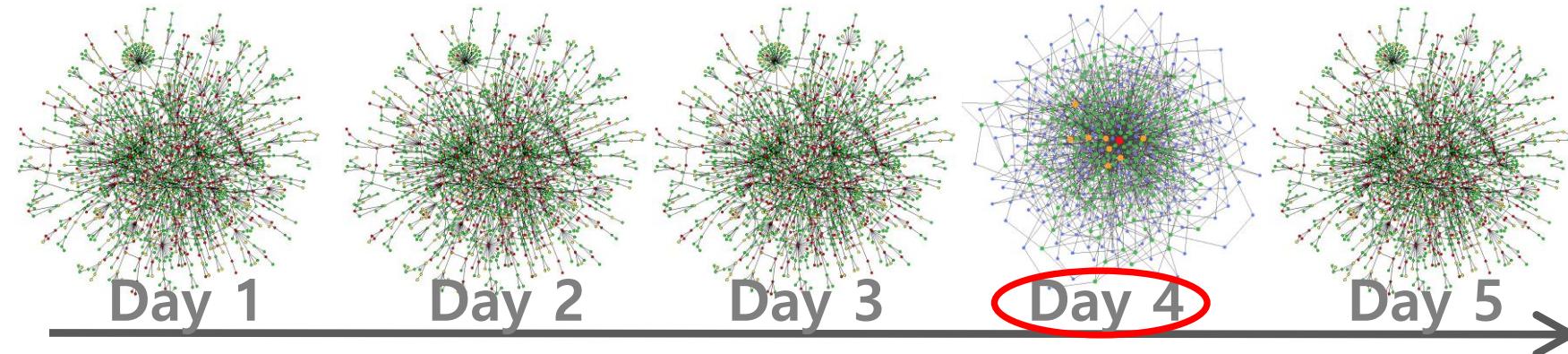


- Drug database:
 - Drug similarity search using graph similarity:
- Program database:
 - Code recommendation using graph similarity



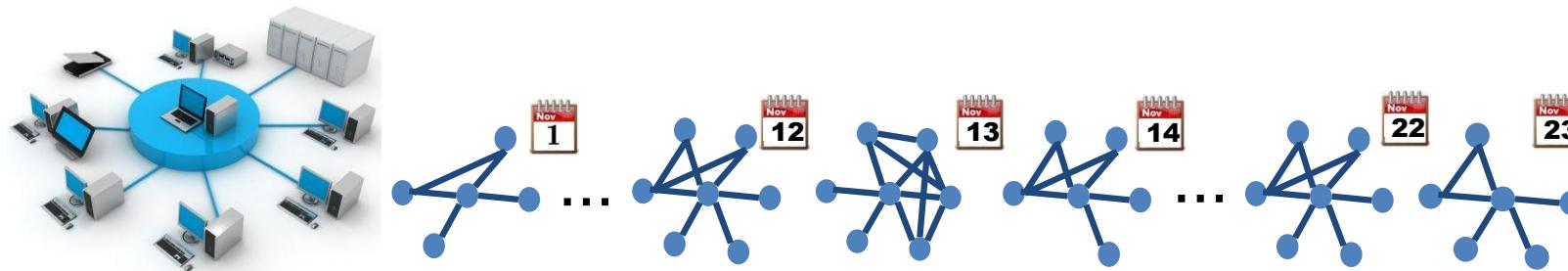
- Network Analysis - Graph similarity:

- Discontinuity Detection



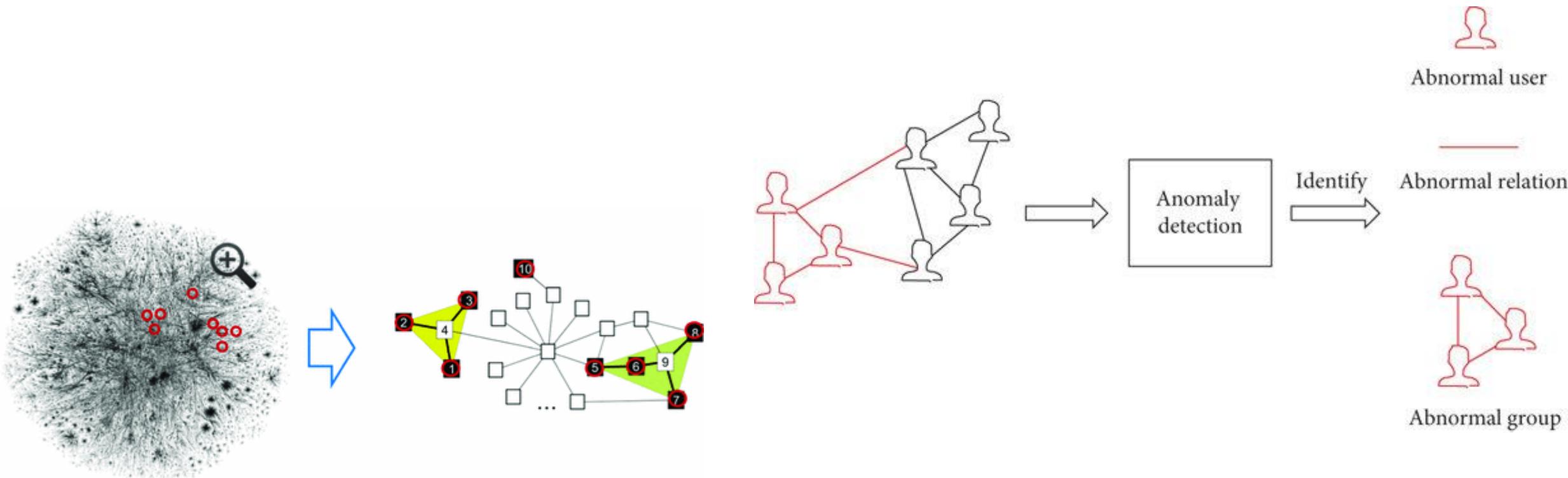
- Intrusion detection:

- Network intrusion



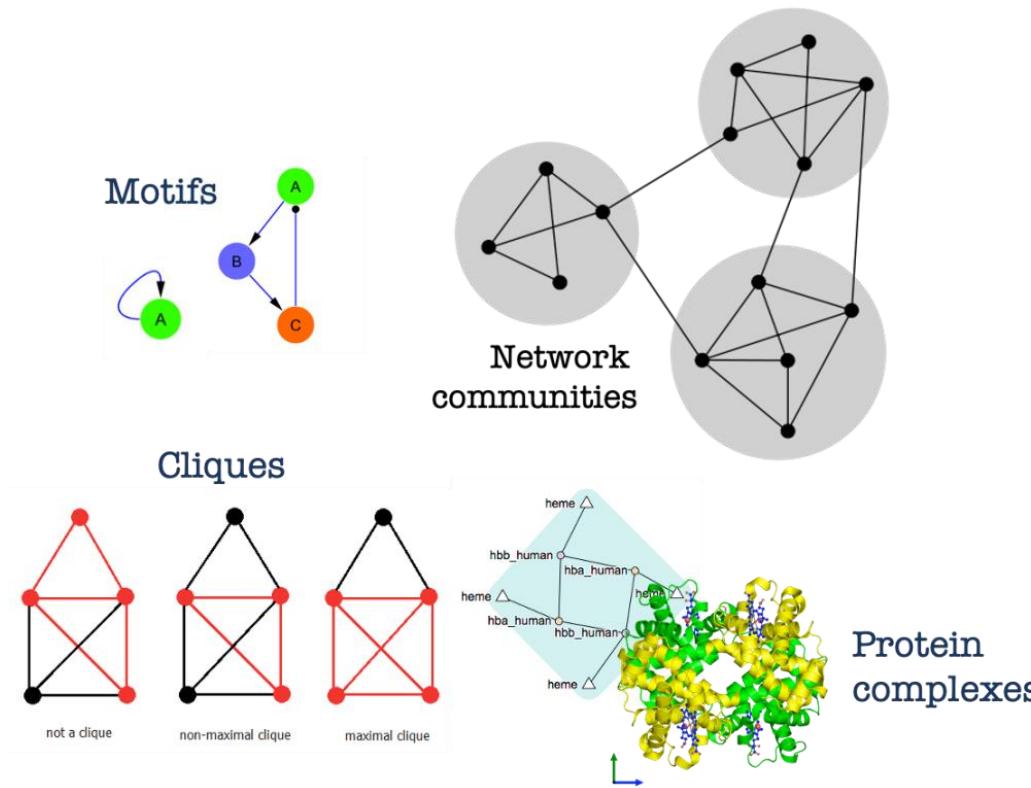
➤ Anomaly detection:

- Abnormal group vs normal group in social network using graph similarity

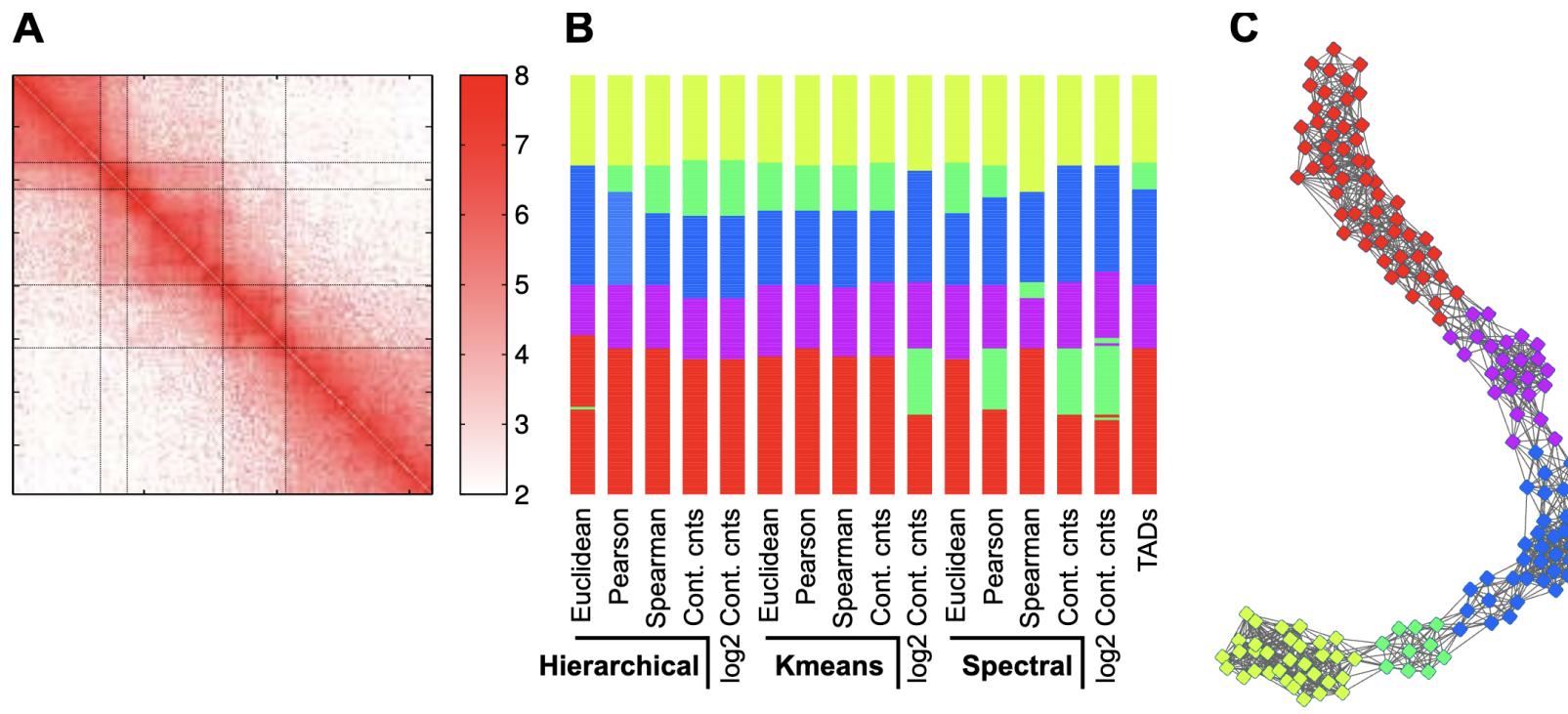


➤ Biological network analysis:

- Identifying functional modules in Protein-protein interaction networks using graph clustering



- Genome analysis in Bioinformatics:
 - Finding higher-order Topologically Associated Domain using graph clustering





네트워크 과학 연구실
NETWORK SCIENCE LAB



가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

