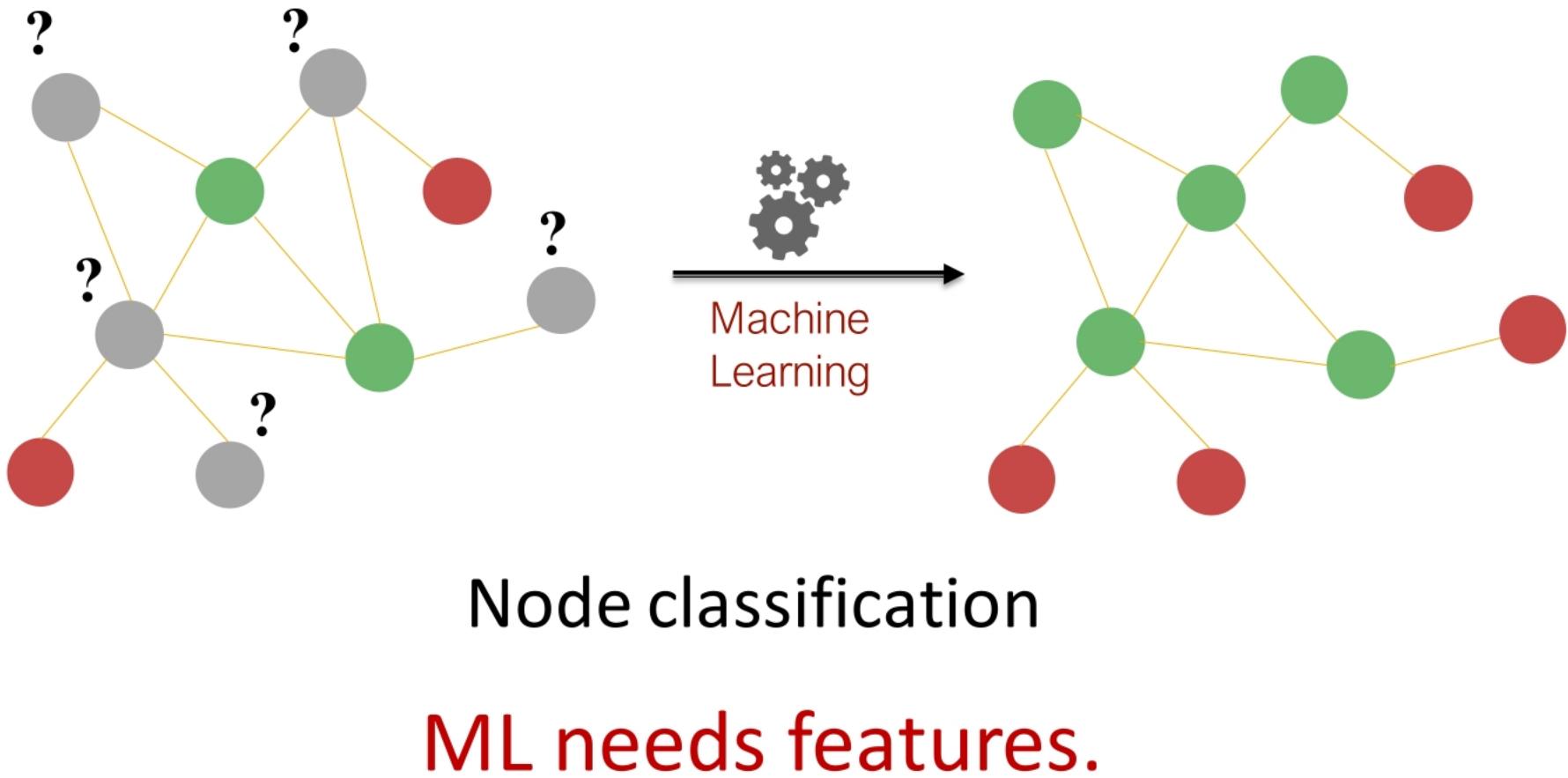


Node-Level Tasks and Features

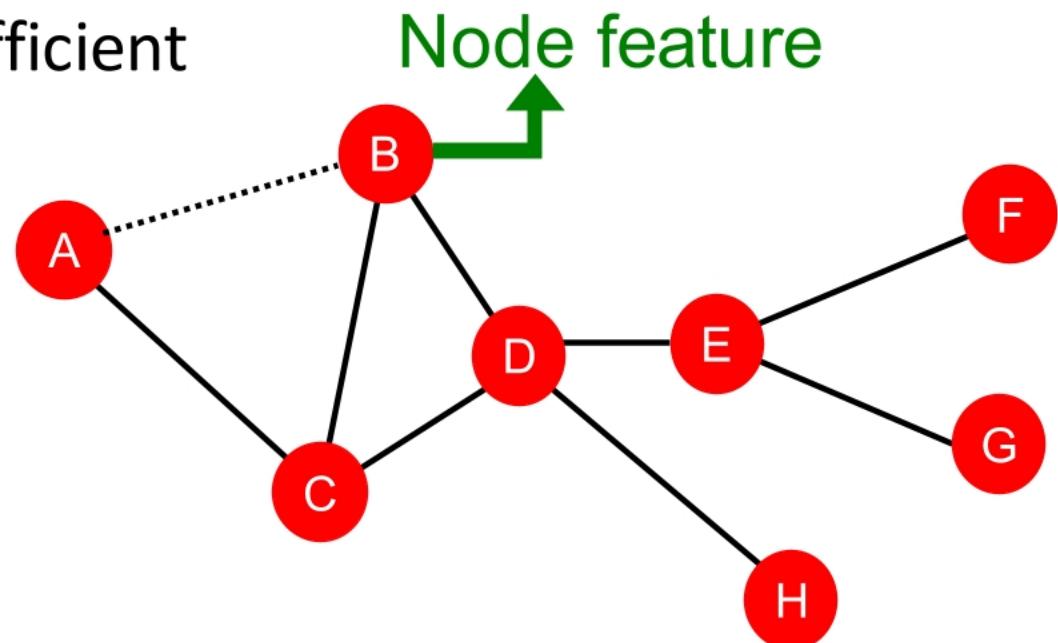
Node-Level Tasks



Node-Level Features: Overview

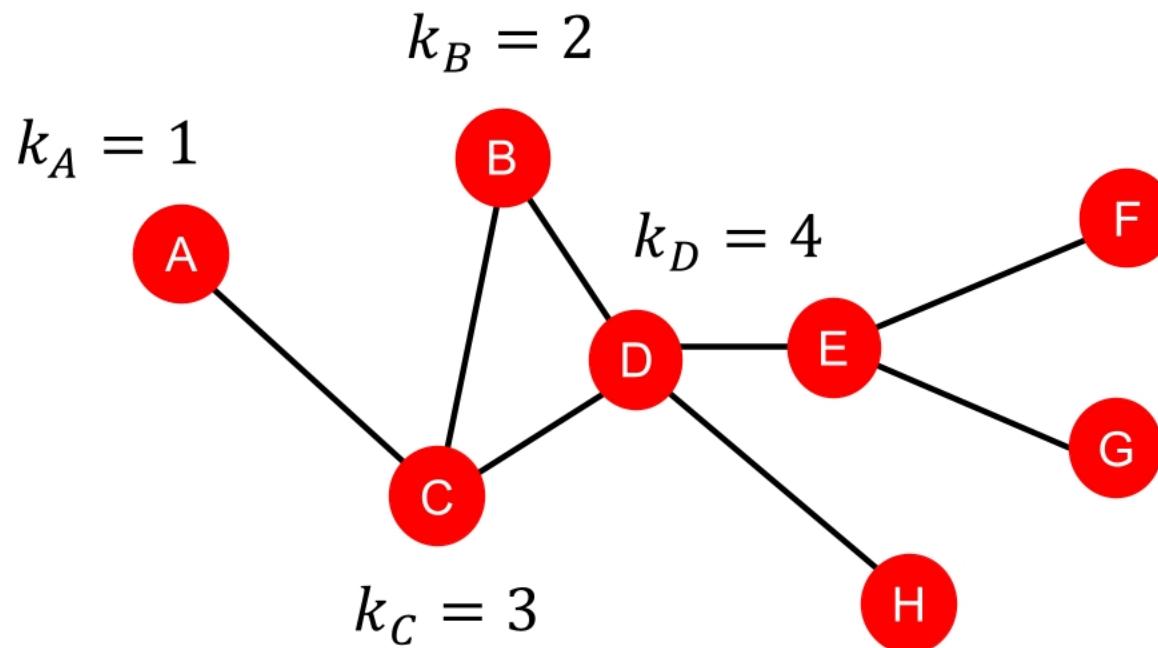
Goal: Characterize the structure and position of a node in the network:

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



Node Features: Node Degree

- The degree k_v of node v is the number of edges (neighboring nodes) the node has.
- Treats all neighboring nodes equally.



Node Features: Node Centrality

- Node degree counts the neighboring nodes without capturing their importance.
- Node centrality c_v , takes the node importance in a graph into account
- **Different ways to model importance:**
 - Eigenvector centrality
 - Betweenness centrality
 - Closeness centrality
 - and many others...

Node Centrality (1)

■ 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**:

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

λ is normalization constant (it will turn out to be the largest eigenvalue of A)

- Notice that the above equation models centrality in a **recursive manner**. **How do we solve it?**

Node Centrality (1)

■ Eigenvector centrality:

- Rewrite the recursive equation in the matrix form.

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

λ is normalization const
(largest eigenvalue of A)

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

- We see that centrality c is the **eigenvector of A !**
- The largest eigenvalue λ_{max} is always positive and unique (by Perron-Frobenius Theorem).
- The eigenvector c_{max} corresponding to λ_{max} is used for centrality.

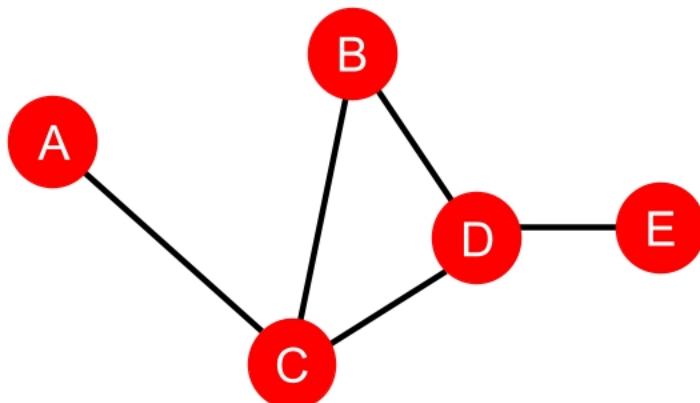
Node Centrality (2)

■ Betweenness centrality:

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

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

- Example:



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

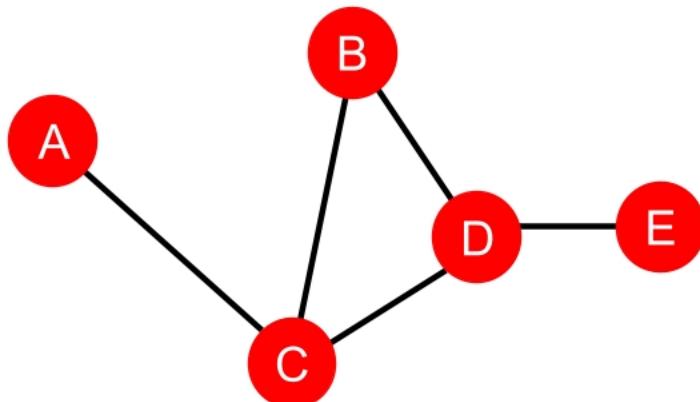
Node Centrality (3)

■ Closeness centrality:

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

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

- Example:



$$c_A = 1/(2 + 1 + 2 + 3) = 1/8$$

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

$$c_D = 1/(2 + 1 + 1 + 1) = 1/5$$

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

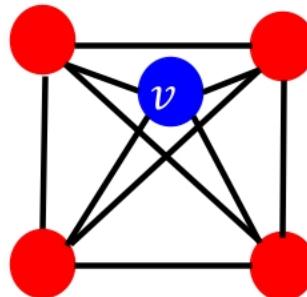
Node Features: Clustering Coefficient

- Measures how connected v 's neighboring nodes are:

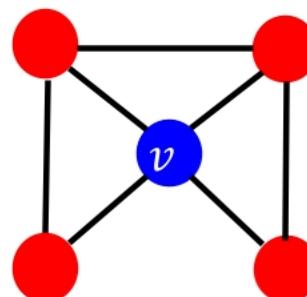
$$e_v = \frac{\text{\#(edges among neighboring nodes)}}{\binom{k_v}{2}} \in [0,1]$$

#(node pairs among k_v neighboring nodes)
In our examples below the denominator is 6 (4 choose 2).

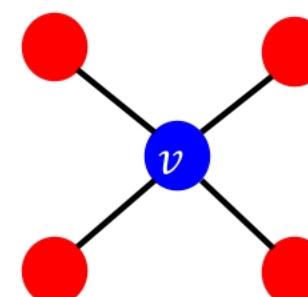
- Examples:



$$e_v = 1$$



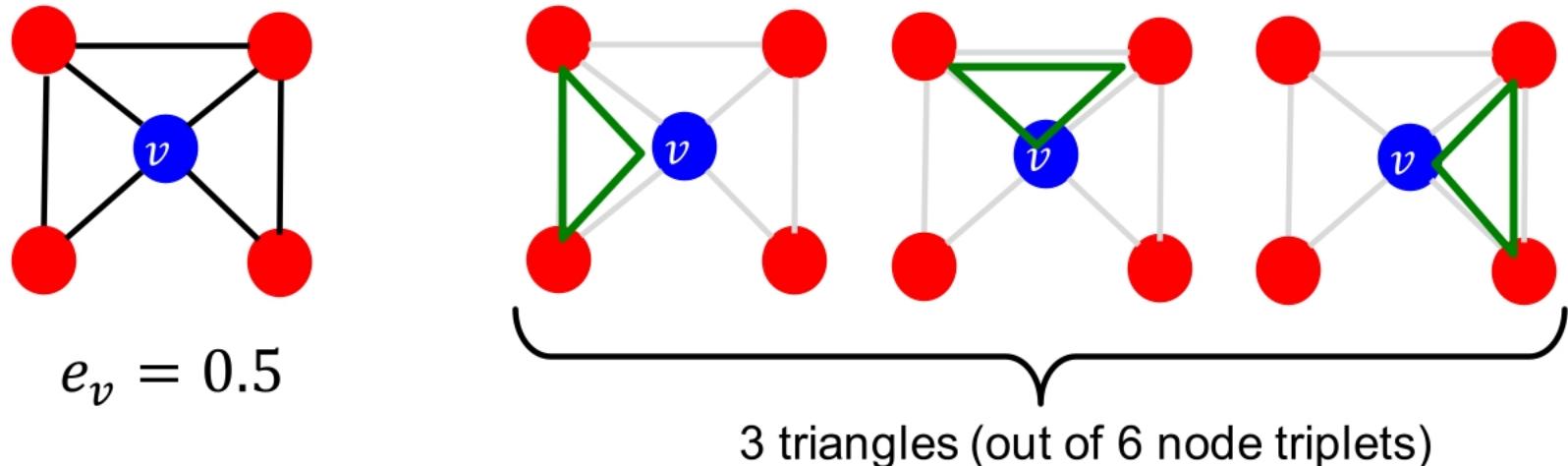
$$e_v = 0.5$$



$$e_v = 0$$

Node Features: Graphlets

- **Observation:** Clustering coefficient counts the #(triangles) in the ego-network



- We can generalize the above by counting #(pre-specified subgraphs, i.e., **graphlets**).

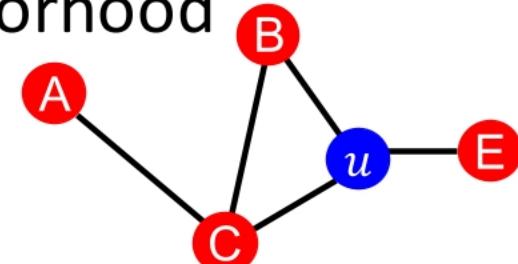
Node Features: Graphlets

- **Goal:** Describe network structure around node u

- **Graphlets** are small subgraphs that describe the structure of node u 's network neighborhood

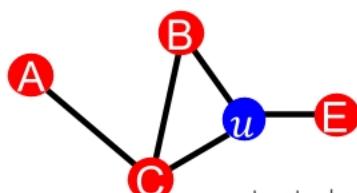
Analogy:

- **Degree** counts **#(edges)** that a node touches
- **Clustering coefficient** counts **#(triangles)** that a node touches.
- **Graphlet Degree Vector (GDV)**: Graphlet-base features for nodes
 - **GDV** counts **#(graphlets)** that a node touches



Node Features: Graphlets

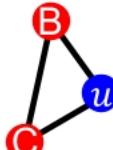
- Considering graphlets of size 2-5 nodes we get:
 - **Vector of 73 coordinates** is a signature of a node that describes the topology of node's neighborhood
- Graphlet degree vector provides a measure of a **node's local network topology**:
 - Comparing vectors of two nodes provides a more detailed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



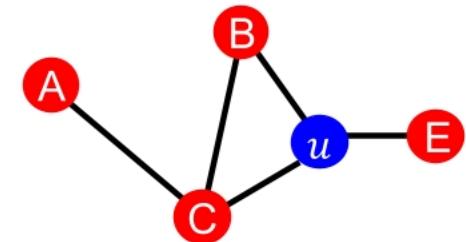
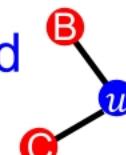
Induced Subgraph & Isomorphism

- Def: Induced subgraph is another graph, formed from a subset of vertices and *all* of the edges connecting the vertices in that subset.

Induced subgraph:

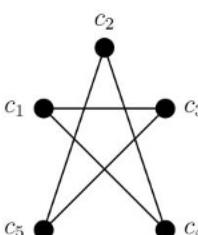
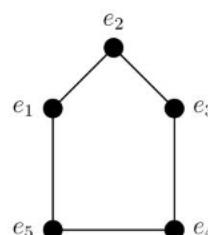


Not induced subgraph:



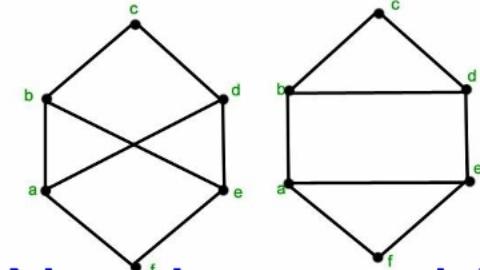
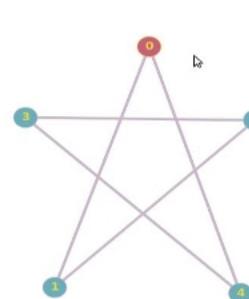
- Def: Graph Isomorphism

- Two graphs which contain the same number of nodes connected in the same way are said to be isomorphic.



Isomorphic

Node mapping: (e2,c2), (e1, c5),
(e3,c4), (e5,c3), (e4,c1)



Non-Isomorphic

The right graph has cycles of length 3 but the left graph does not, so the graphs cannot be isomorphic.

Source: Mathoverflow

Node Features: Graphlets

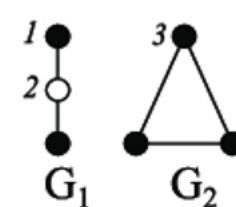
Graphlets: Rooted connected induced non-isomorphic subgraphs:

Take some nodes and all the edges between them.

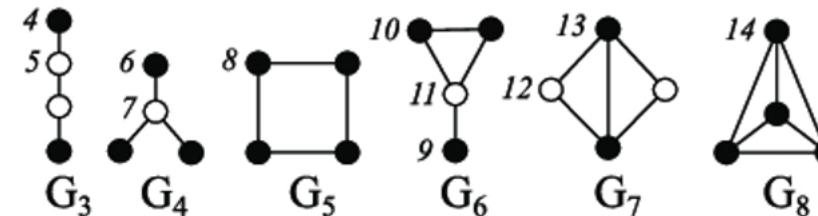
2-node graphlet



3-node graphlets

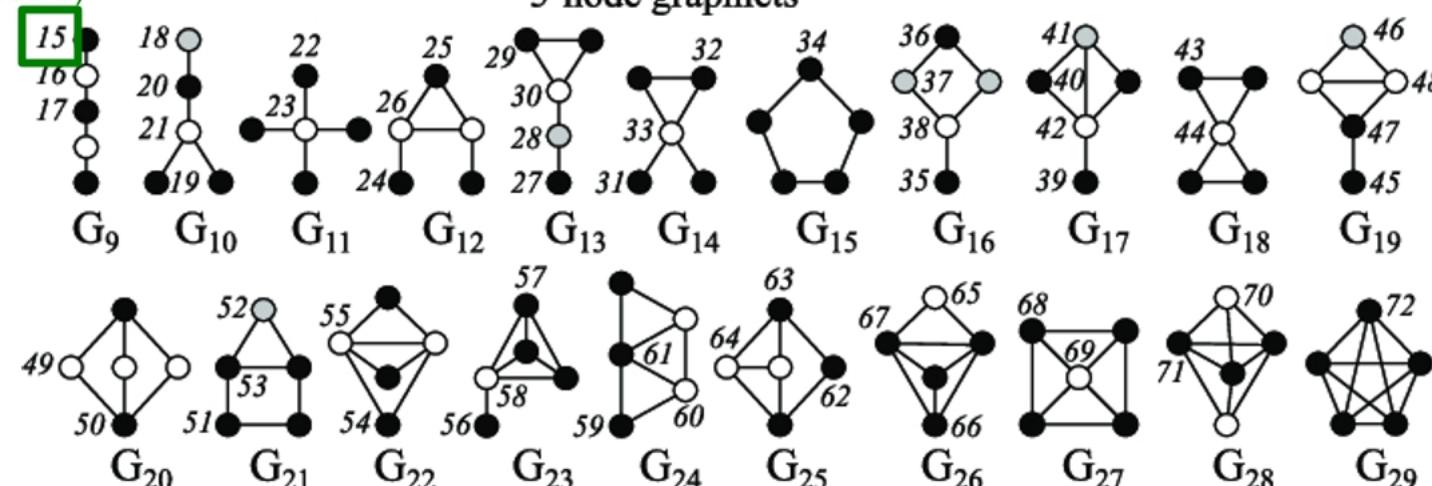


4-node graphlets



u has graphlets: 0, 1, 2, 3, 5, 10, 11, ...

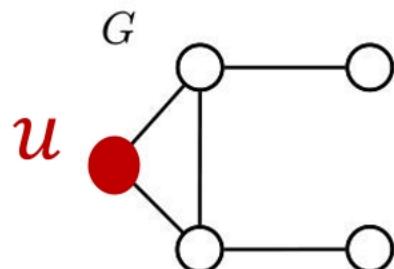
Graphlet id (Root/“position” of node u)



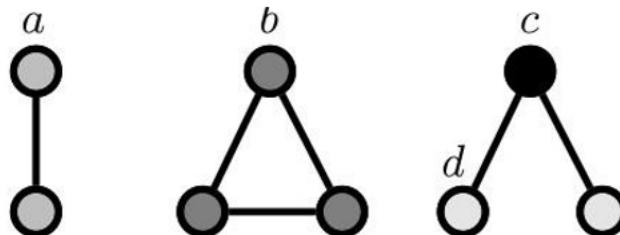
There are 73 different graphlets on up to 5 nodes

Node Features: Graphlets

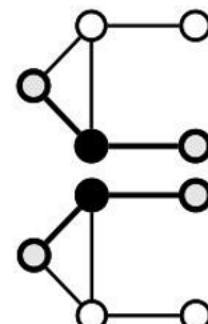
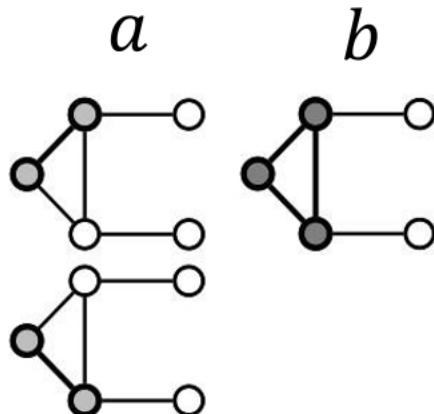
- **Graphlet Degree Vector (GDV):** A count vector of graphlets rooted at a given node.
- **Example:**



Possible graphlets up to size 3



Graphlet instances of node u :



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

Node-Level Feature: Summary

- We have introduced different ways to obtain node features.
- They can be categorized as:
 - Importance-based features:
 - Node degree
 - Different node centrality measures
 - Structure-based features:
 - Node degree
 - Clustering coefficient
 - Graphlet count vector

Node-Level Feature: Summary

- **Importance-based features**: capture the importance of a node in a graph
 - Node degree:
 - Simply counts the number of neighboring nodes
 - Node centrality:
 - Models **importance of neighboring nodes** in a graph
 - Different modeling choices: eigenvector centrality, betweenness centrality, closeness centrality
- Useful for predicting influential nodes in a graph
 - **Example**: predicting celebrity users in a social network

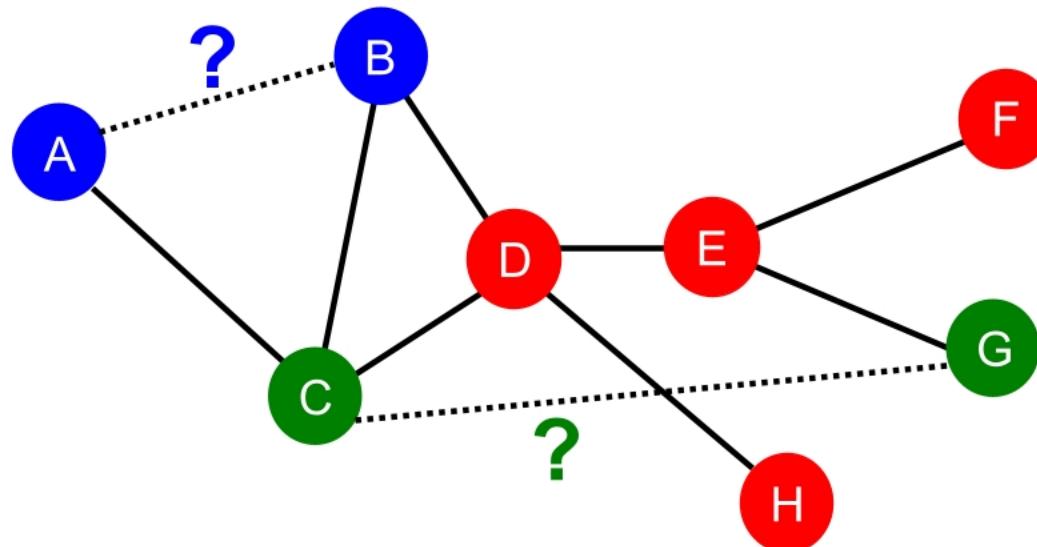
Node-Level Feature: Summary

- **Structure-based features:** Capture topological properties of local neighborhood around a node.
 - **Node degree:**
 - Counts the number of neighboring nodes
 - **Clustering coefficient:**
 - Measures how connected neighboring nodes are
 - **Graphlet degree vector:**
 - Counts the occurrences of different graphlets
- **Useful for predicting a particular role a node plays in a graph:**
 - **Example:** Predicting protein functionality in a protein-protein interaction network.

Link Prediction Task and Features

Link-Level Prediction Task: Recap

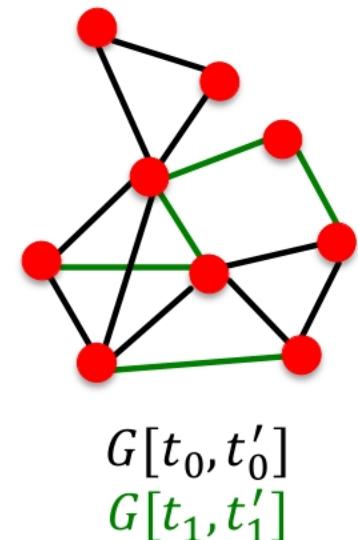
- The task is to predict **new links** based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- **The key is to design features for a pair of nodes.**



Link Prediction as a Task

Two formulations of the link prediction task:

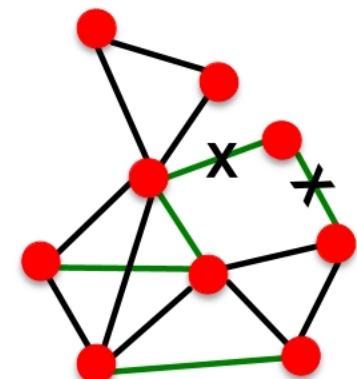
- 1) Links missing at random:
 - Remove a random set of links and then aim to predict them
- 2) Links over time:
 - Given $G[t_0, t'_0]$ a graph defined by edges up to time t'_0 , **output a ranked list L** of edges (not in $G[t_0, t'_0]$) that are predicted to appear in time $G[t_1, t'_1]$
 - **Evaluation:**
 - $n = |E_{new}|$: # new edges that appear during the test period $[t_1, t'_1]$
 - Take top n elements of L and count correct edges



Link Prediction via Proximity

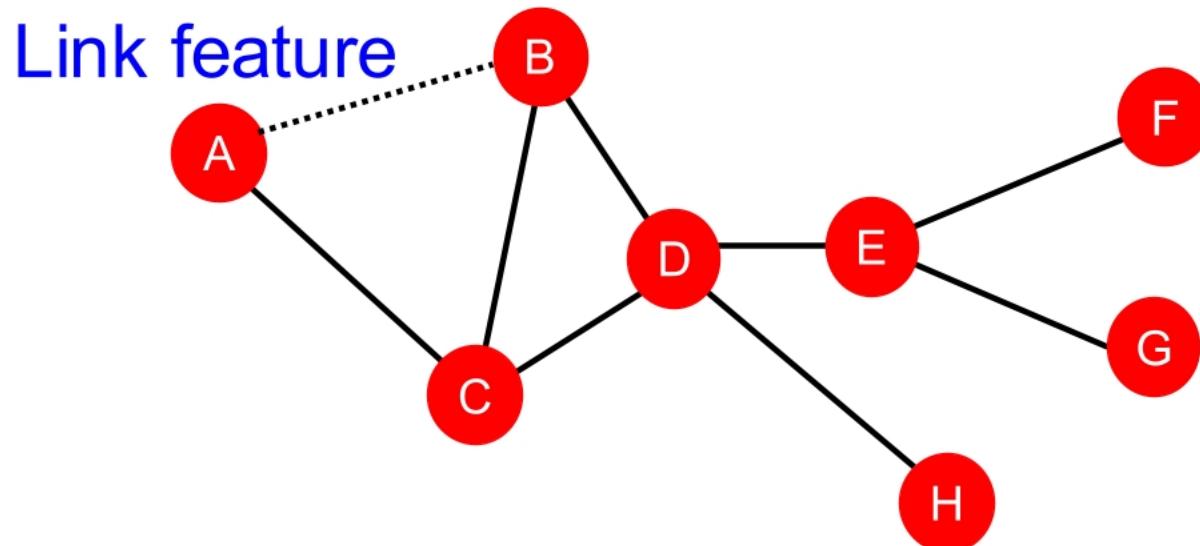
■ Methodology:

- For each pair of nodes (x,y) compute score $c(x,y)$
 - For example, $c(x,y)$ could be the # of common neighbors of x and y
- Sort pairs (x,y) by the decreasing score $c(x,y)$
- **Predict top n pairs as new links**
- **See which of these links actually appear in $G[t_1, t'_1]$**



Link-Level Features: Overview

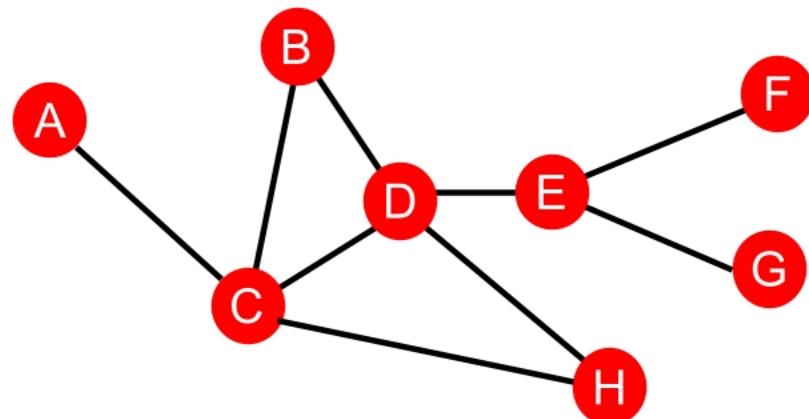
- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap



Distance-Based Features

Shortest-path distance between two nodes

- Example:



$$S_{BH} = S_{BE} = S_{AB} = 2$$

$$S_{BG} = S_{BF} = 3$$

- However, this does not capture the degree of neighborhood overlap:
 - Node pair (B, H) has 2 shared neighboring nodes, while pairs (B, E) and (A, B) only have 1 such node.

Local Neighborhood Overlap

Captures # neighboring nodes shared between two nodes v_1 and v_2 :

- Common neighbors: $|N(v_1) \cap N(v_2)|$

- Example: $|N(A) \cap N(B)| = |\{C\}| = 1$

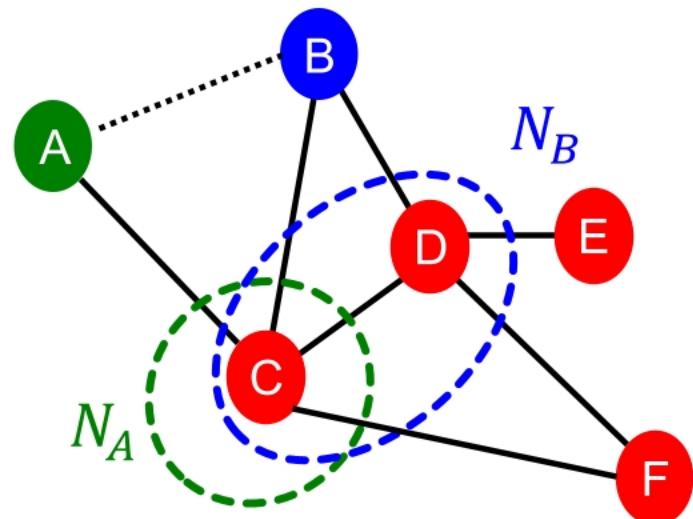
- Jaccard's coefficient: $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$

- Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{A, B, C, D\}|} = \frac{1}{2}$

- Adamic-Adar index:

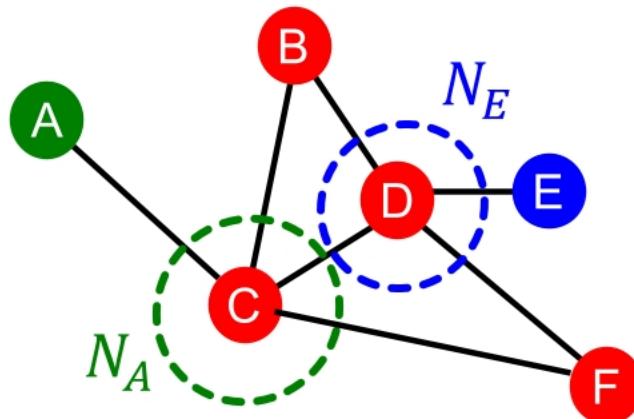
$$\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$$

- Example: $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$



Global Neighborhood Overlap

- **Limitation of local neighborhood features:**
 - Metric is always zero if the two nodes do not have any neighbors in common.



$$N_A \cap N_E = \emptyset$$
$$|N_A \cap N_E| = 0$$

- However, the two nodes may still potentially be connected in the future.
- **Global neighborhood overlap** metrics resolve the limitation by considering the entire graph.

Global Neighborhood Overlap

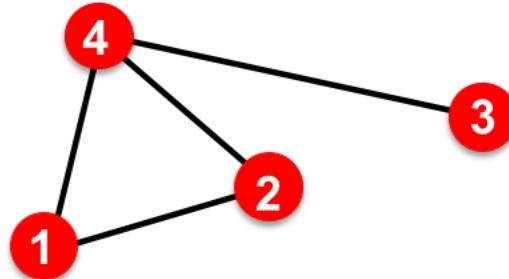
- **Katz index:** count the number of walks of all lengths between a given pair of nodes.
- **Q: How to compute #walks between two nodes?**
- Use **powers of the graph adjacency matrix!**

Intuition: Powers of Adj Matrices

■ Computing #walks between two nodes

- Recall: $A_{uv} = 1$ if $u \in N(v)$
- Let $P_{uv}^{(K)} = \# \text{walks of length } K \text{ between } u \text{ and } v$
- We will show $P^{(K)} = A^k$
- $P_{uv}^{(1)} = \# \text{walks of length 1 (direct neighborhood)} \text{ between } u \text{ and } v = A_{uv}$

$$P_{12}^{(1)} = A_{12}$$



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

Intuition: Powers of Adj Matrices

- How to compute $P_{uv}^{(2)}$?
 - Step 1: Compute #walks of length 1 between each of u 's neighbor and v
 - Step 2: Sum up these #walks across u 's neighbors
 - $P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^2$

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

$$A^2 = \begin{matrix} \text{Power of} \\ \text{adjacency} \end{matrix} \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

Diagram illustrating the computation of A^2 . The first matrix shows Node 1's neighbors (highlighted in blue) as columns 1 and 4. The second matrix shows the #walks of length 1 between Node 1's neighbors and Node 2 (highlighted in green). The result is A_{12}^2 , where the 12th entry is highlighted in red.

Global Neighborhood Overlap

- **Katz index:** count the number of walks of all lengths between a pair of nodes.
- How to compute #walks between two nodes?
- Use **adjacency matrix powers!**
 - A_{uv} specifies #walks of length 1 (direct neighborhood) between u and v .
 - A_{uv}^2 specifies #walks of **length 2** (neighbor of neighbor) between u and v .
 - And, A_{uv}^l specifies #walks of **length l** .

Global Neighborhood Overlap

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

Sum over all walk lengths

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

#walks of length l
between v_1 and v_2

$0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

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

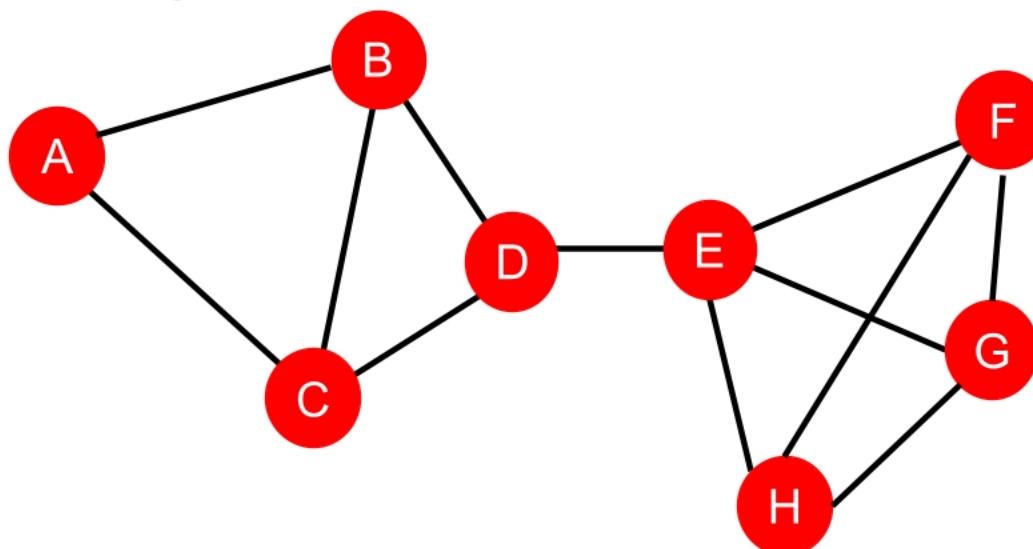
Link-Level Features: Summary

- **Distance-based features:**
 - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.
- **Local neighborhood overlap:**
 - Captures how many neighboring nodes are shared by two nodes.
 - Becomes zero when no neighbor nodes are shared.
- **Global neighborhood overlap:**
 - Uses global graph structure to score two nodes.
 - Katz index counts #walks of all lengths between two nodes.

Graph-Level Features and Graph Kernels

Graph-Level Features

- **Goal:** We want features that characterize the structure of an entire graph.
- **For example:**



Background: Kernel Methods

- **Kernel methods** are widely-used for traditional ML for graph-level prediction.
- **Idea: Design kernels instead of feature vectors.**
- **A quick introduction to Kernels:**
 - Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
 - Kernel matrix $\mathbf{K} = (K(G, G'))_{G, G'}$, must always be positive semidefinite (i.e., has positive eigenvalues)
 - There exists a feature representation $\phi(\cdot)$ such that $K(G, G') = \phi(G)^T \phi(G')$
 - Once the kernel is defined, off-the-shelf ML model, such as **kernel SVM**, can be used to make predictions.

Graph-Level Features: Overview

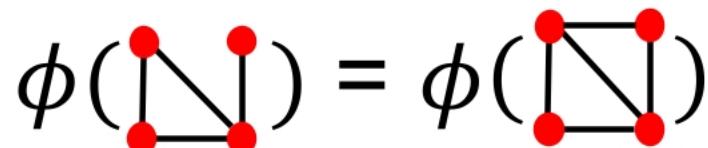
- **Graph Kernels:** Measure similarity between two graphs:
 - Graphlet Kernel [1]
 - Weisfeiler-Lehman Kernel [2]
 - Other kernels are also proposed in the literature (beyond the scope of this lecture)
 - Random-walk kernel
 - Shortest-path graph kernel
 - And many more...

[1] Shervashidze, Nino, et al. "Efficient graphlet kernels for large graph comparison." Artificial Intelligence and Statistics. 2009.

[2] Shervashidze, Nino, et al. "Weisfeiler-lehman graph kernels." Journal of Machine Learning Research 12.9 (2011).

Graph Kernel: Key Idea

- **Goal:** Design graph feature vector $\phi(G)$
- **Key idea:** Bag-of-Words (BoW) for a graph
 - **Recall:** 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...

$$\phi(\text{graph 1}) = \phi(\text{graph 2})$$
The equation shows two graphs side-by-side, separated by an equals sign. Both graphs consist of four red circular nodes. Graph 1 has three nodes in a horizontal row at the bottom, with the fourth node positioned above the middle node of the row and connected to both of them. Graph 2 has its four nodes arranged in a square-like cycle, where each node is connected to its two neighbors. This visual comparison serves as a counterexample to the claim that nodes are regarded as words, as the two graphs represent different structures despite having the same number of nodes.

Graph Kernel: Key Idea

What if we use Bag of node degrees?

Deg1: ● Deg2: ● Deg3: ●

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

Obtains different features
for different graphs!

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

- Both Graphlet Kernel and Weisfeiler-Lehman (WL) Kernel use **Bag-of-*** representation of graph, where * is more sophisticated than node degrees!

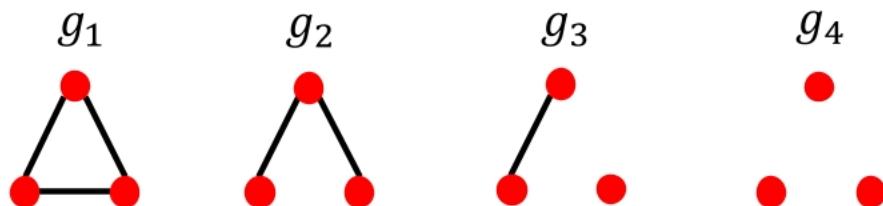
Graphlet Features

- **Key idea:** Count the number of different graphlets in a graph.
- **Note:** Definition 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.
 - Examples in the next slide illustrate this.

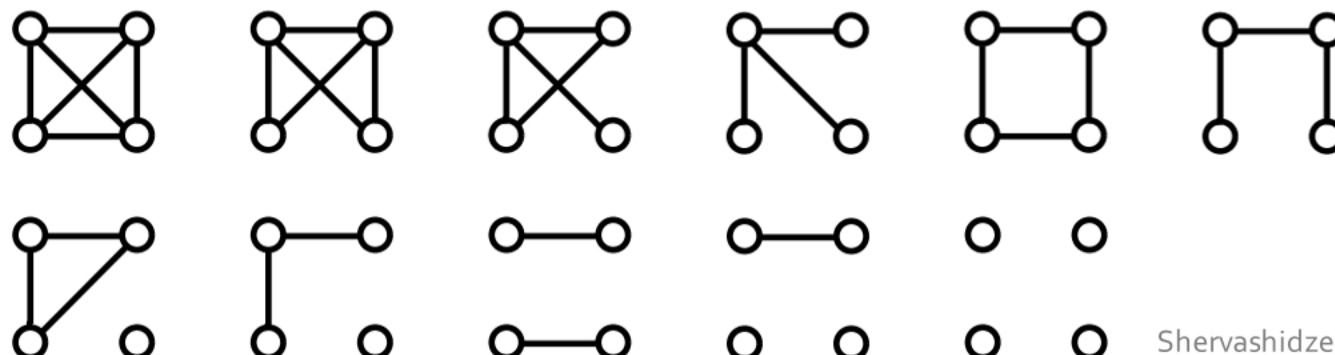
Graphlet Features

Let $\mathcal{G}_k = (g_1, g_2, \dots, g_{n_k})$ be a list of graphlets of size k .

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



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



Shervashidze et al., AISTATS 2011

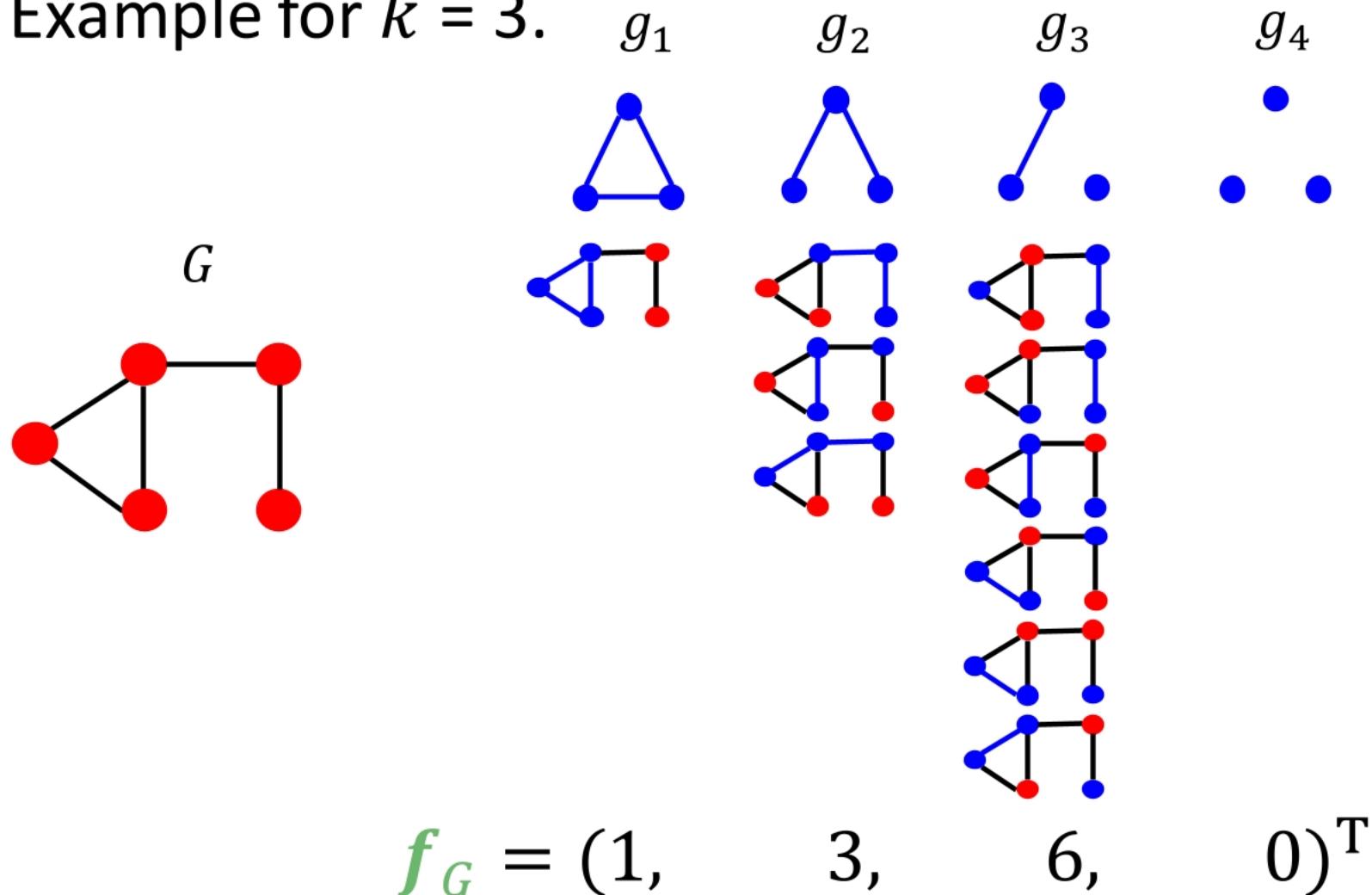
Graphlet Features

- Given graph G , and a graphlet list $\mathcal{G}_k = (g_1, g_2, \dots, g_{n_k})$, define the graphlet count vector $f_G \in \mathbb{R}^{n_k}$ as

$$(f_G)_i = \#(g_i \subseteq G) \text{ for } i = 1, 2, \dots, n_k.$$

Graphlet Features

- Example for $k = 3$.



Graphlet Kernel

- 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

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

Graphlet Kernel

Limitations: Counting graphlets is **expensive!**

- Counting size- k graphlets for a graph with size n by enumeration takes n^k .
- This is unavoidable in the worst-case 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?

Weisfeiler-Lehman 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** since node degrees are one-hop neighborhood information.
- **Algorithm to achieve this:**

Color refinement

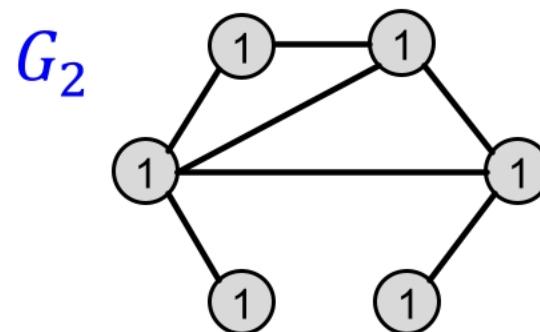
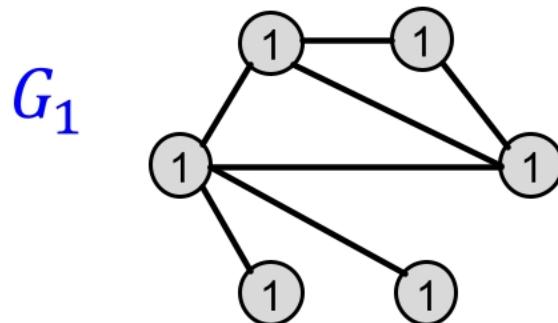
Color Refinement

- **Given:** A graph G with a set of nodes V .
 - Assign an initial color $c^{(0)}(v)$ to each node v .
 - Iteratively refine node colors 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 colors.
- After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K -hop neighborhood

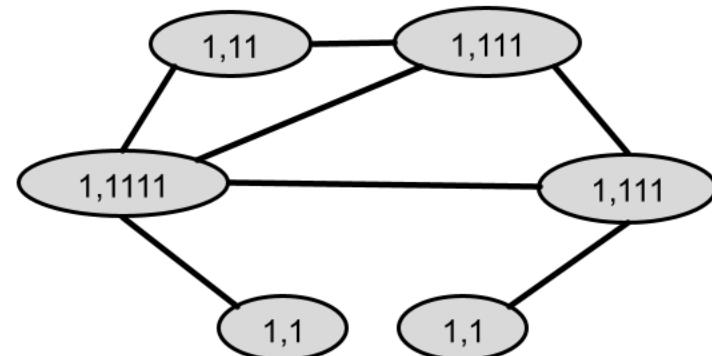
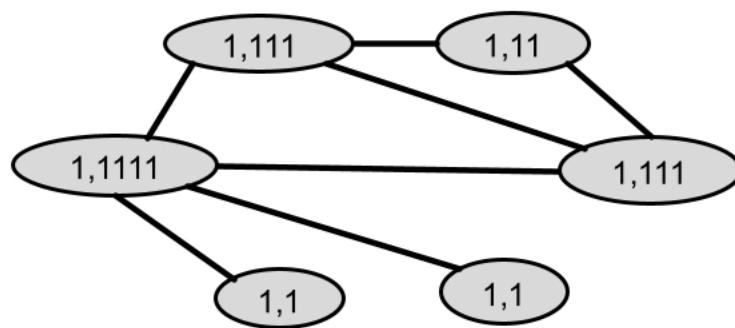
Color Refinement (1)

Example of color refinement given two graphs

- Assign initial colors



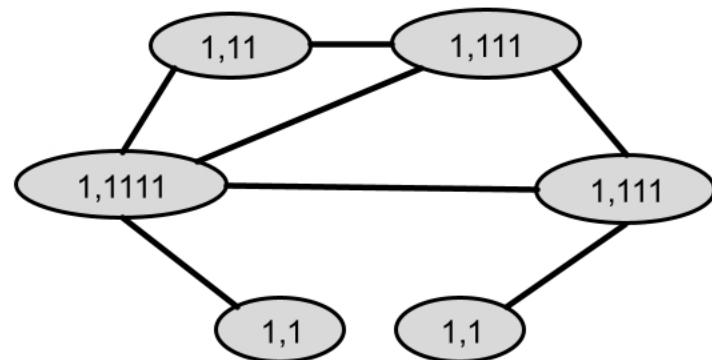
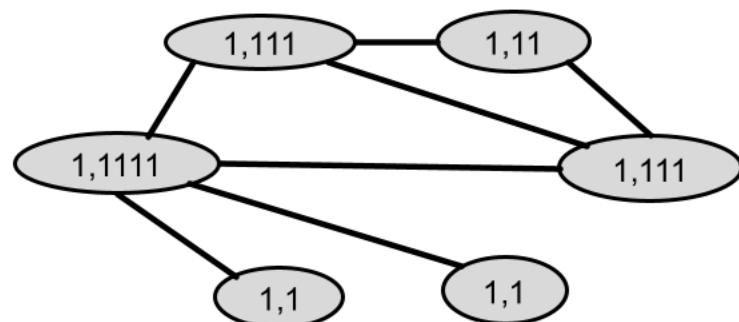
- Aggregate neighboring colors



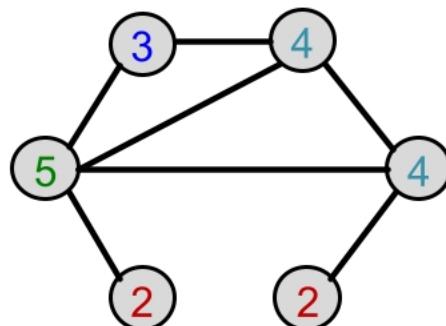
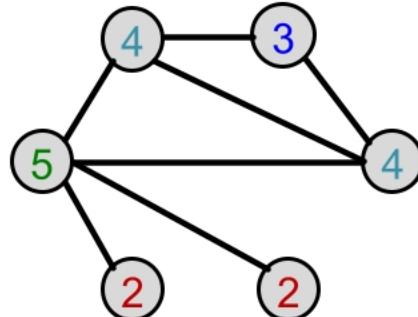
Color Refinement (2)

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors



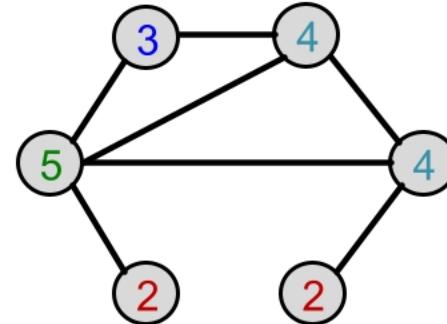
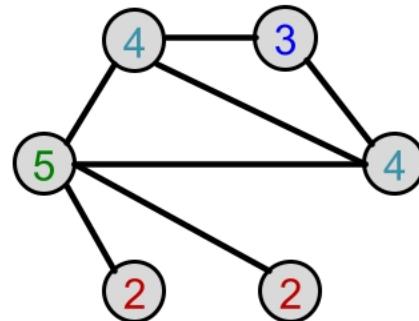
Hash table

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

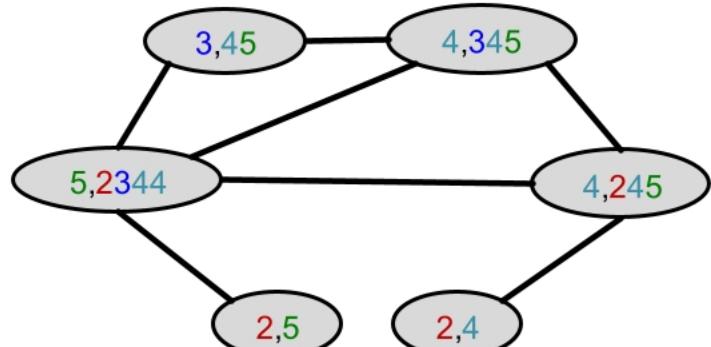
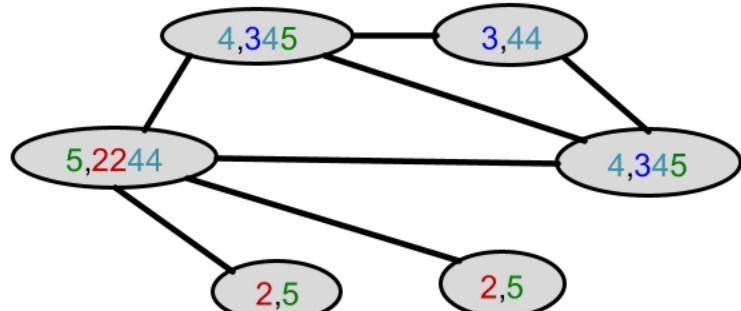
Color Refinement (3)

Example of color refinement given two graphs

- Aggregated colors



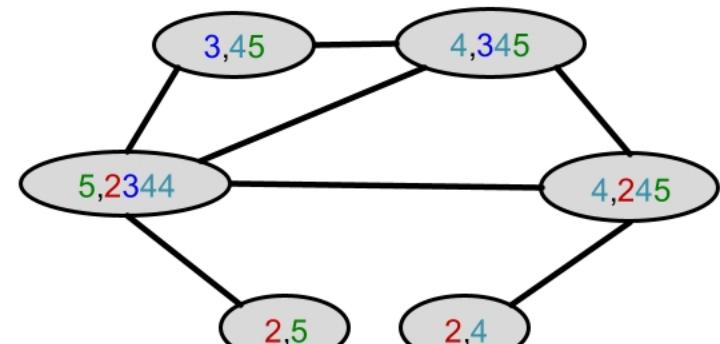
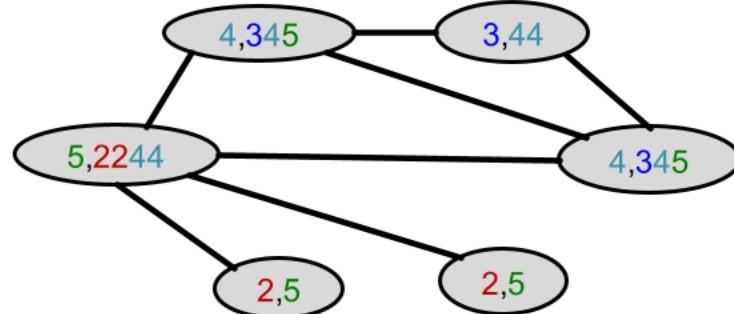
- Hash aggregated colors



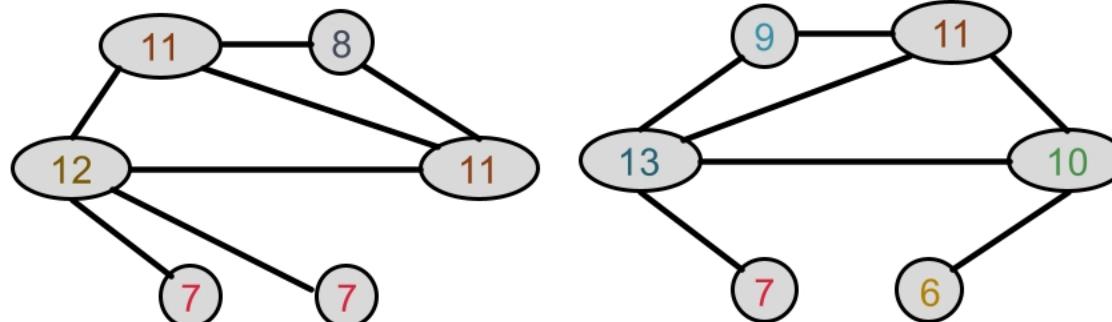
Color Refinement (4)

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors

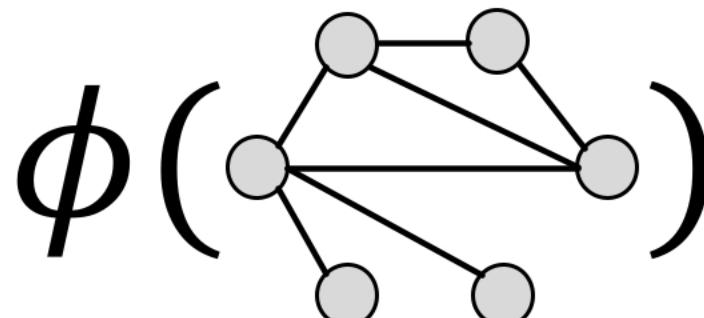


Hash table

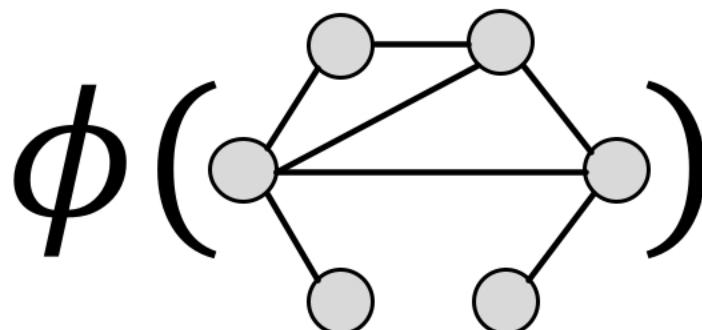
2,4	-->	6
2,5	-->	7
3,44	-->	8
3,45	-->	9
4,245	-->	10
4,345	-->	11
5,2244	-->	12
5,2344	-->	13

Weisfeiler-Lehman 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
= [6,2,1,2,1,0,2,1,0,0, 0, 2, 1]
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]

Weisfeiler-Lehman Kernel

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

$$\begin{aligned} K(&\text{graph}_1, \text{graph}_2) \\ &= \phi(\text{graph}_1)^T \phi(\text{graph}_2) \\ &= 49 \end{aligned}$$

Weisfeiler-Lehman 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)**.

Graph-Level Features: Summary

■ Graphlet Kernel

- Graph is represented as **Bag-of-graphlets**
- **Computationally expensive**

■ Weisfeiler-Lehman Kernel

- Apply K -step color refinement algorithm to enrich node colors
 - Different colors capture different K -hop neighborhood structures
- Graph is represented as **Bag-of-colors**
- **Computationally efficient**
- Closely related to Graph Neural Networks (as we will see!)