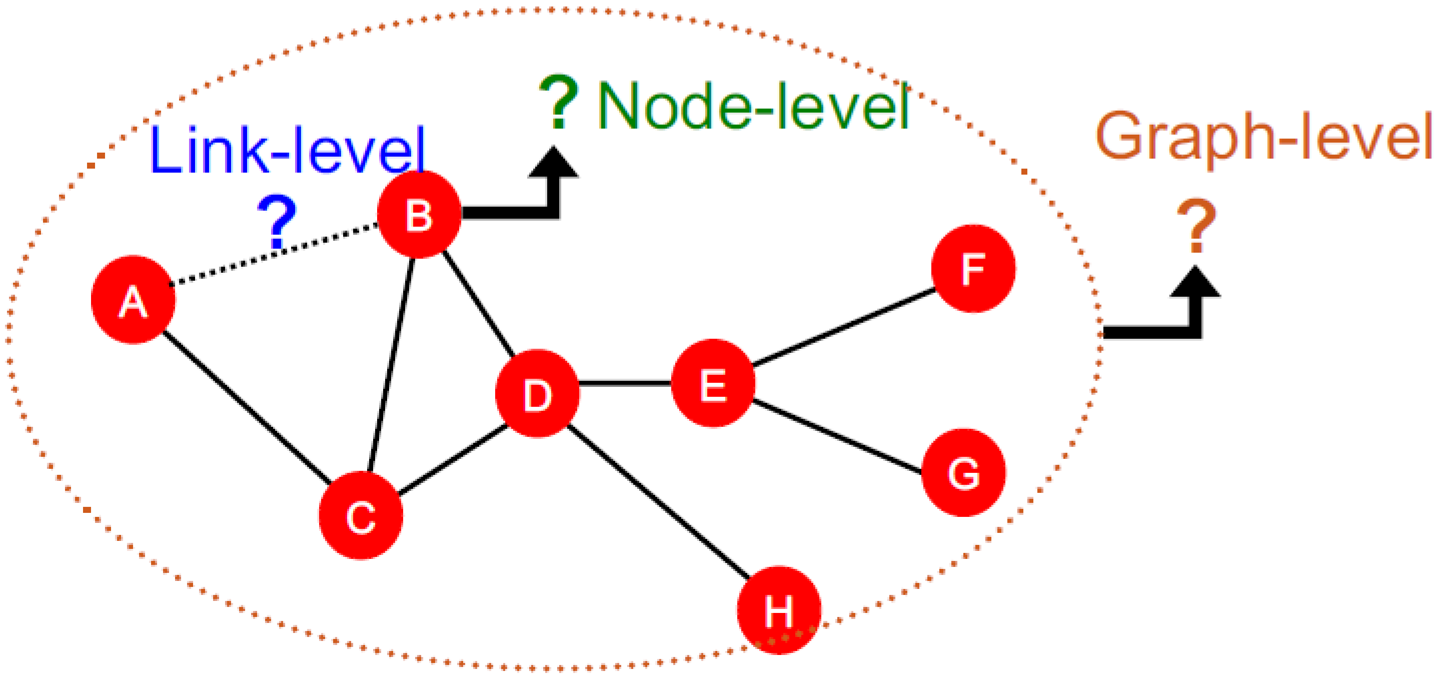
Traditional Graph Machine Learning

Slides Adapted from Leskovec's Lecture
CS224W Machine Learning with Graphs

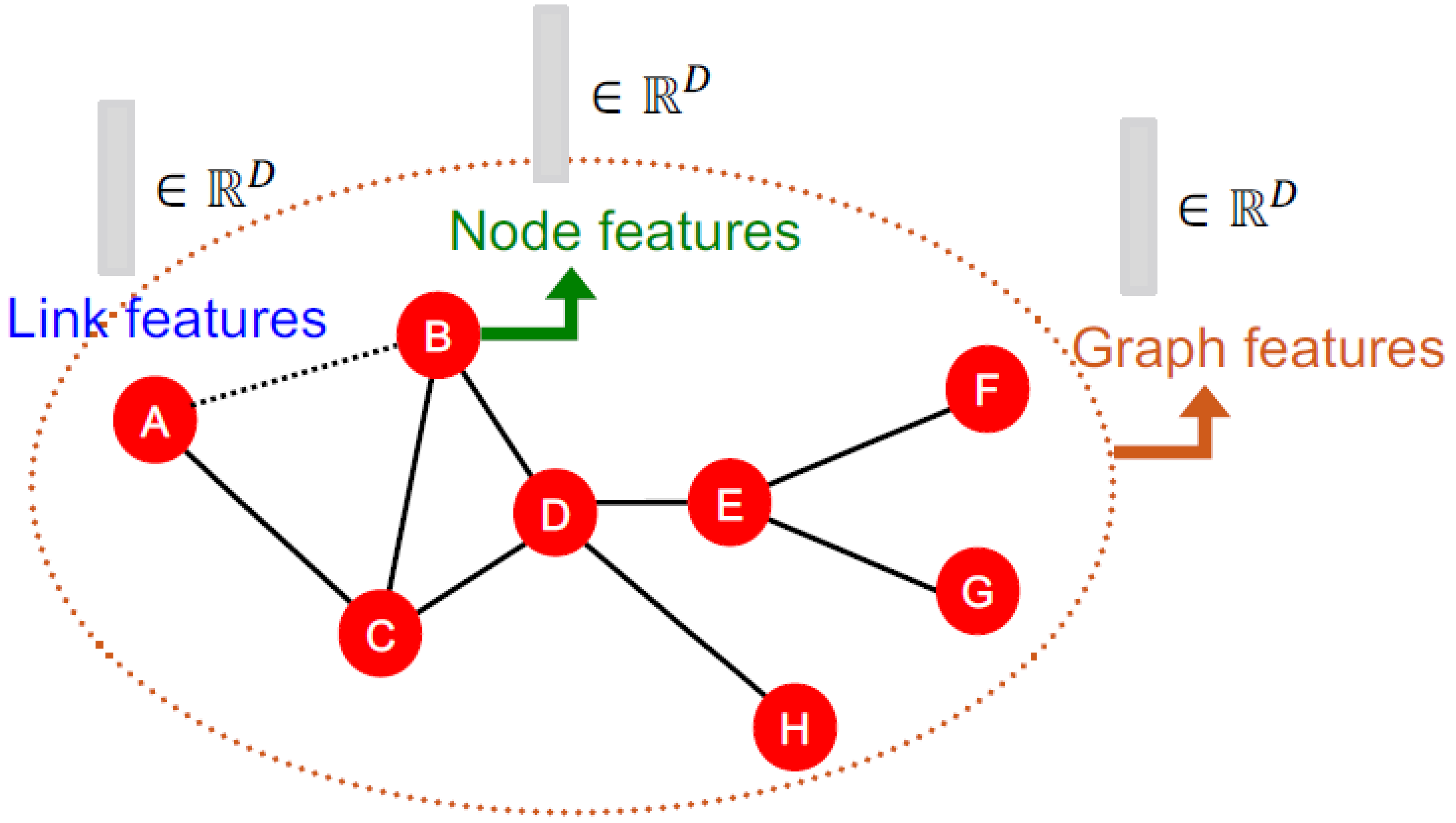
Predictions in Graphs

- Node Level Prediction
- Edge Level Prediction
- Graph Level Prediction



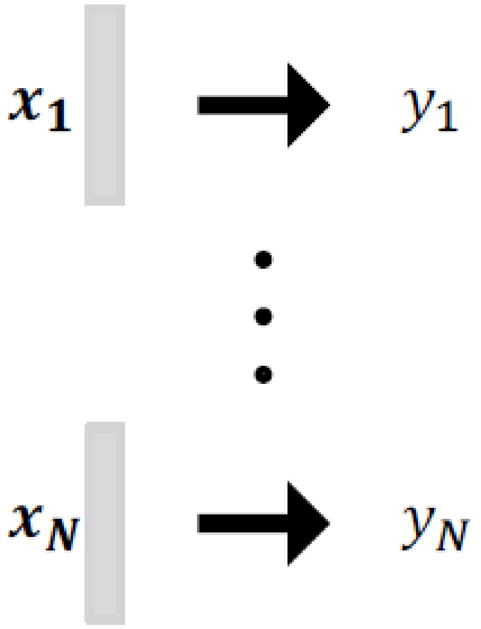
Traditional ML Pipeline

- Design features for nodes/links/graphs
- Obtain features for all training data

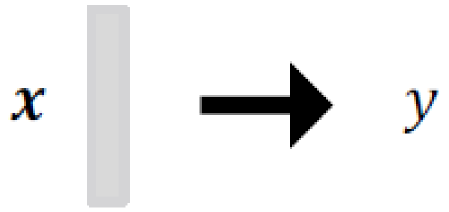


Traditional ML Pipeline

- **Train an ML model:**
  - Random forest
  - SVM
  - Neural network, etc.



- **Apply the model:**
  - Given a new node/link/graph, obtain its features and make a prediction



- Relies on Effective Feature Engineering
- Traditional ML relies on hand-designed features

Machine Learning in Graphs

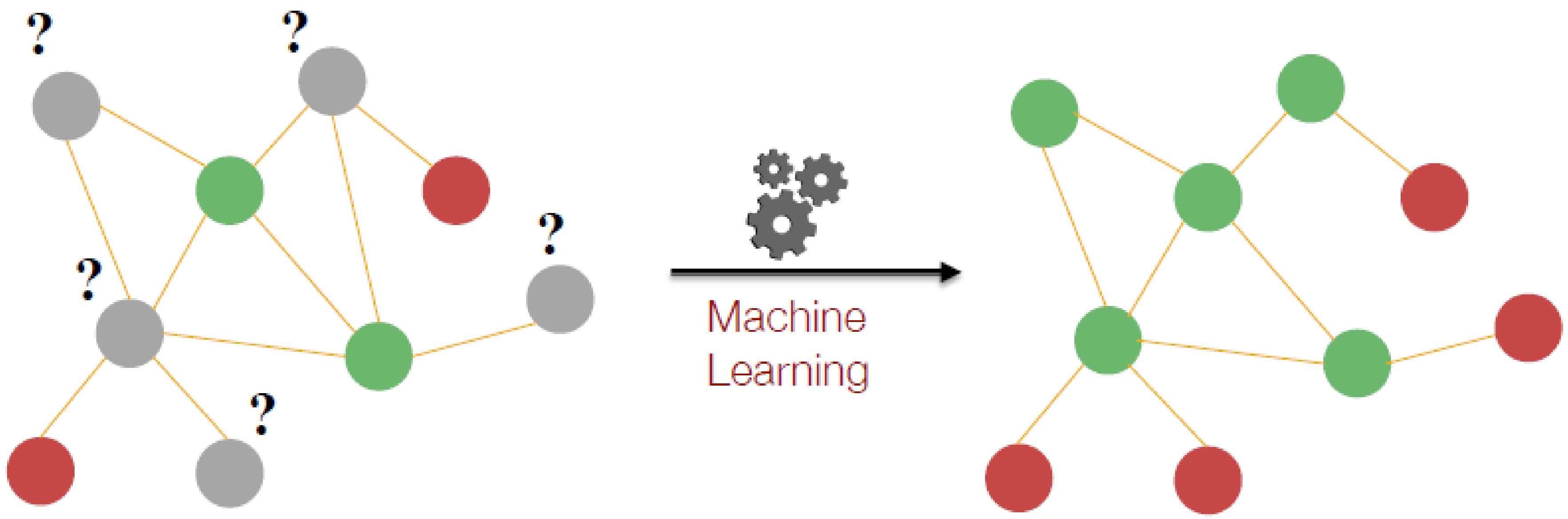
Goal: Make predictions for a set of objects

Design choices:

- **Features:** *d*-dimensional vectors
- **Objects:** Nodes, edges, sets of nodes, entire graphs
- **Objective function:**
  - What task are we aiming to solve?
    - Given: $G = (V, E)$
    - Learn a function: $f : V \to \mathbb{R}$

How do we learn the function?

Node Classification Task



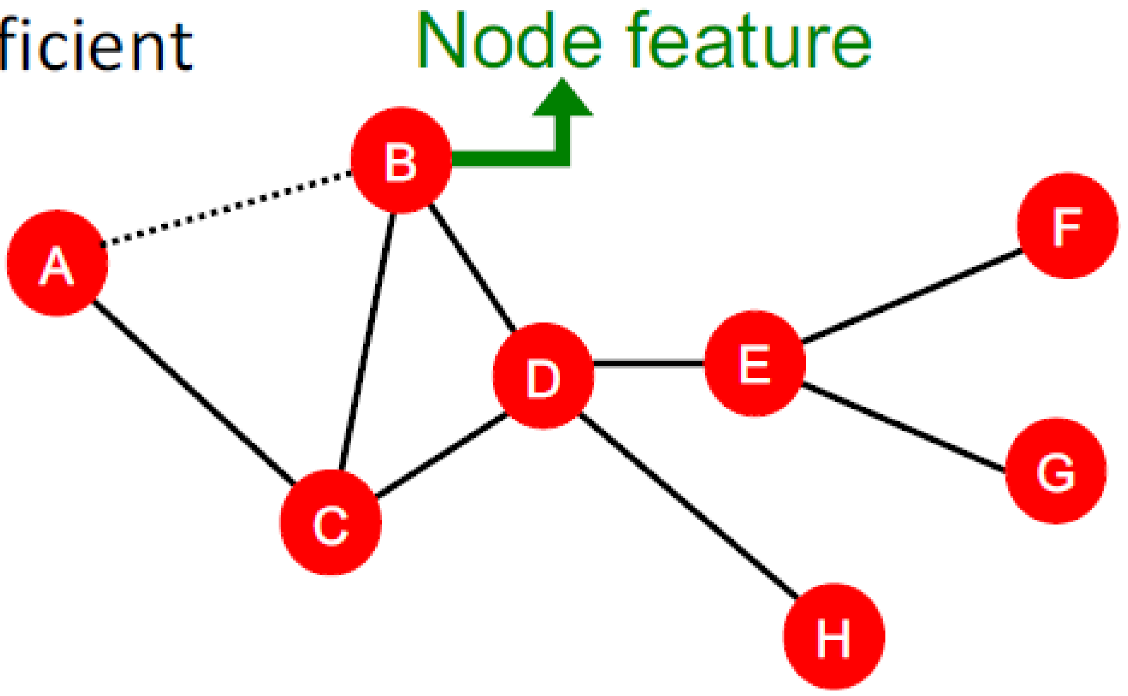
Node classification

ML needs features.

Node Level Features

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

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets

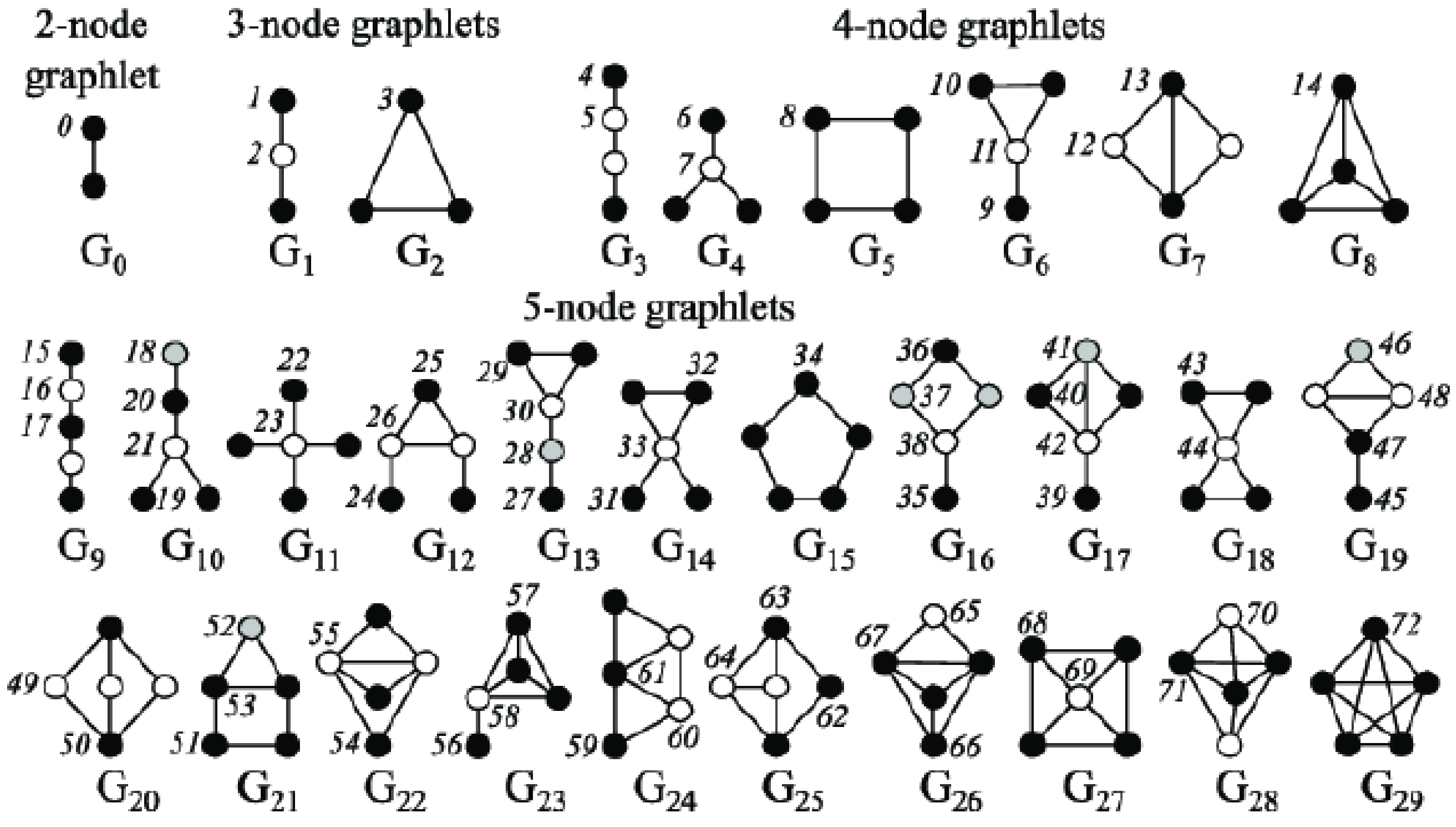


Node Level Features

- Node degrees : Simple but treats each of the neighbors equally
- Node Centrality : Measures node importance
  - Eigen Vector, PageRank, Betweenness, Closeness
- Clustering Coefficient : Measures cohesiveness
  - Counts # of triangles in the graph
  - We can generalize the counting to a pre-specified subgraph size (Graphlets)
- Graphlet counts

Node Level Features

- Graphlets
  - Rooted connected non-isomorphic subgraphs



Node features: Graphlets

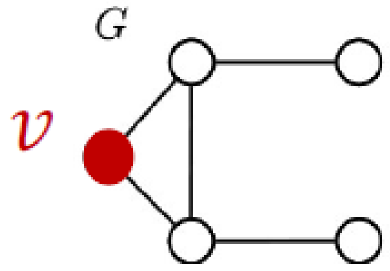
- **Graphlet Degree Vector (GDV)**: Graphlet-base features for nodes

- **Degree** counts **#(edges)** that a node touches
- **Clustering coefficient** counts **#(triangles)** that a node touches.

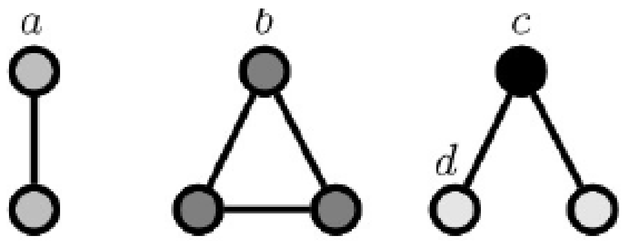
- **GDV** counts **#(graphlets)** that a node touches

Node Features: Graphlets

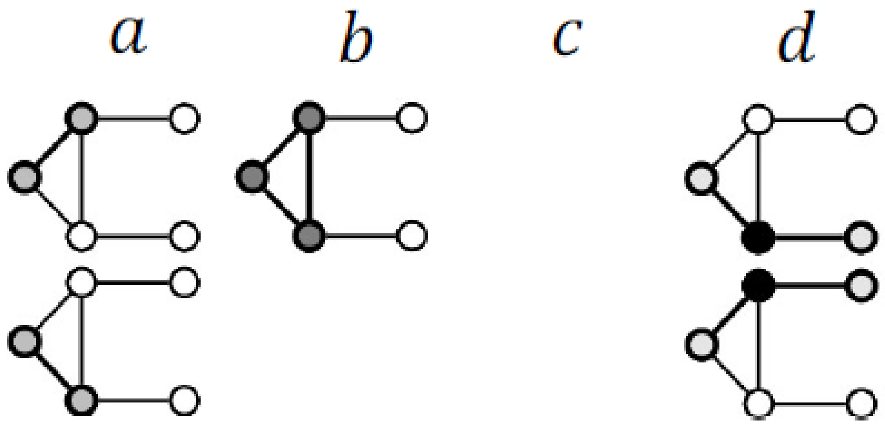
- **Graphlet Degree Vector (GDV):** A count vector of graphlets rooted at a given node.
- **Example:**



List of graphlets



Graphlet instances:



GDV of node v:

a, b, c, d

[2,1,0,2]

Node Features: 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.

Node Level Feature Summary

- Importance-based features:
  - Node degree
  - Different node centrality measures
- Structure-based features:
  - Node degree
  - Clustering coefficient
  - Graphlet count vector

Node Level Feature Summary

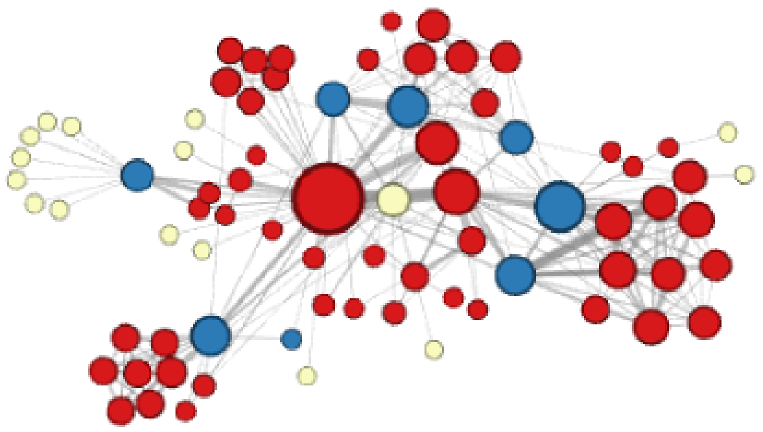
- **Importance-based features**: capture the importance of a node is in a graph
  - Node degree:
    - Simply counts the number of neighboring nodes
  - Node centrality:
    - Models importance of neighboring nodes in a graph
    - Different modeling choices: eigenvector centrality, betweenness centrality,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

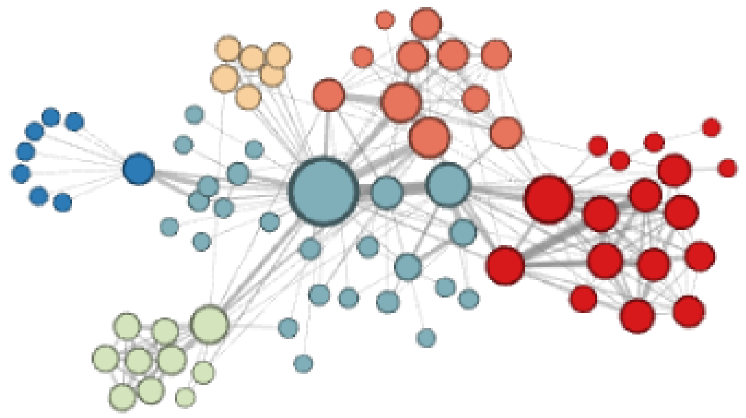
Node Level Feature Summary

- **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.

What it still misses?

Different ways to label nodes of the network:





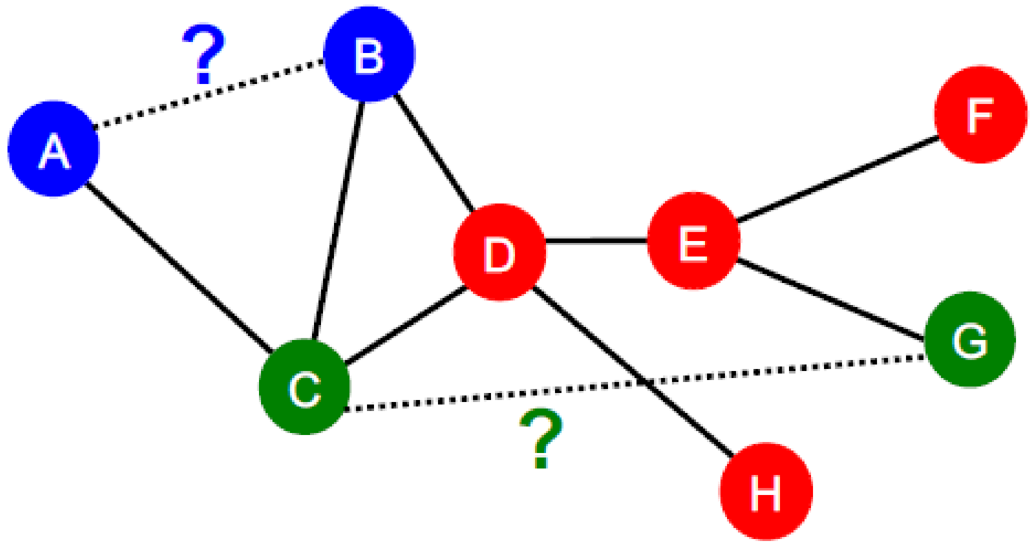
Node features defined so far would allow to distinguish nodes in the above example

However, the features defines so far would not allow for distinguishing the above node labelling

Link Level Prediction Tasks

Link Level Prediction

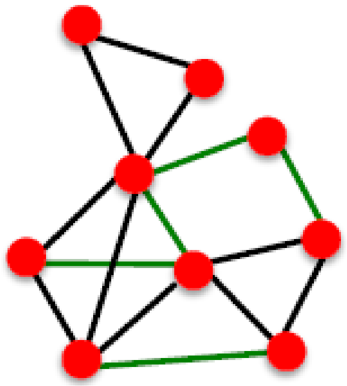
- The task is to predict **new links** based on existing links.
- At test time, all node pairs (no existing links) are ranked, and top K node pairs are predicted.
- The key is to design features for **a pair of nodes**.



Link Prediction as a task

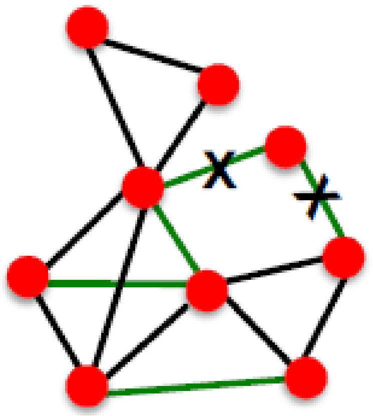
Two formulations of the link prediction task:

- **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 on edges up to time t'_0, **output a ranked list L** of links (not in $G[t_0, t'_0]$) that are predicted to appear in $G[t_1, t'_1]$
  - **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


$G[t_0, t'_0]$
$G[t_1, t'_1]$

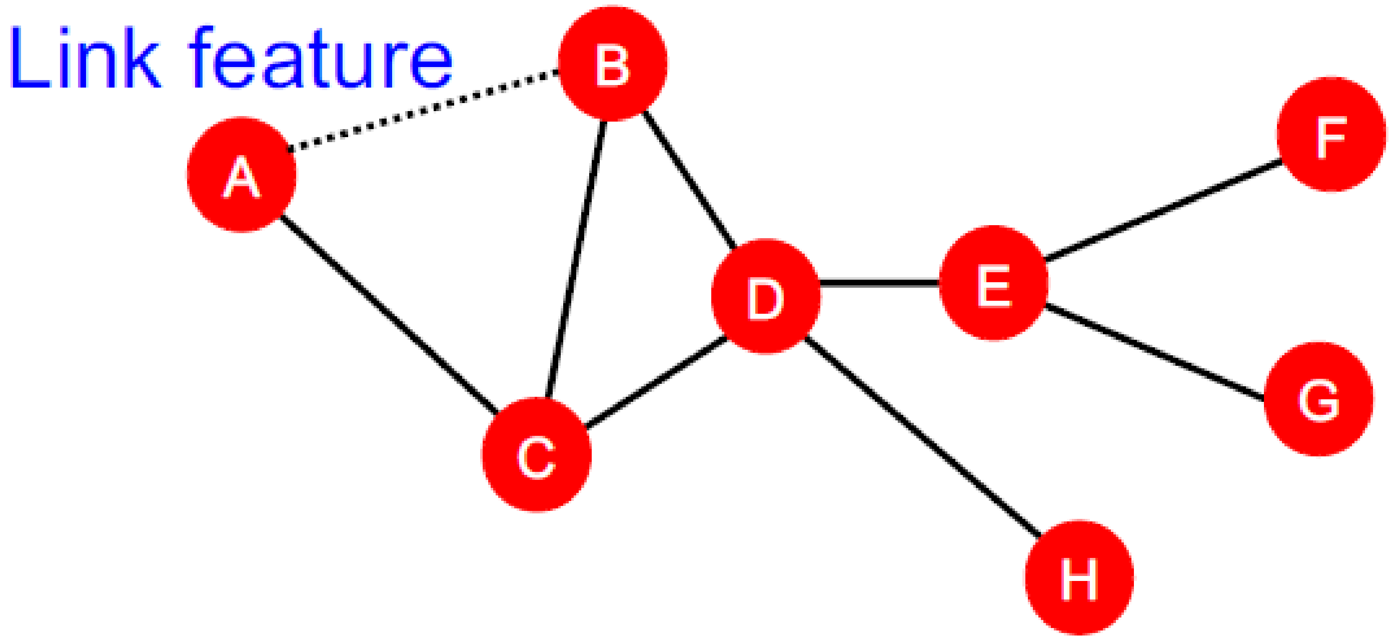
Link Prediction via Proximity

- **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

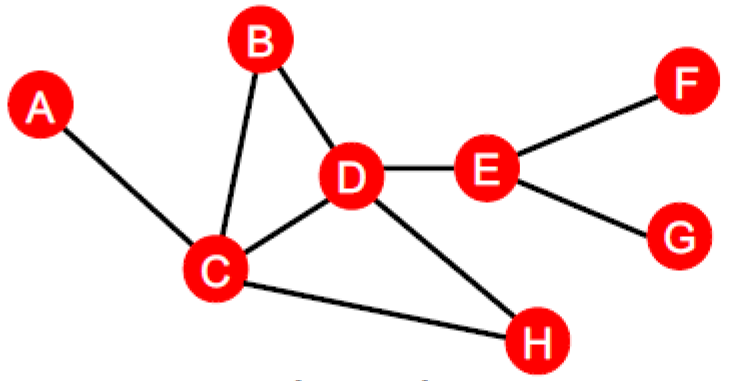
- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap



Distance based features

Shortest-path distance between two nodes

- Example:



$$S_{BH} = S_{BE} = S_{AB} = 2$$
$$S_{BG} = S_{BF} = 3$$

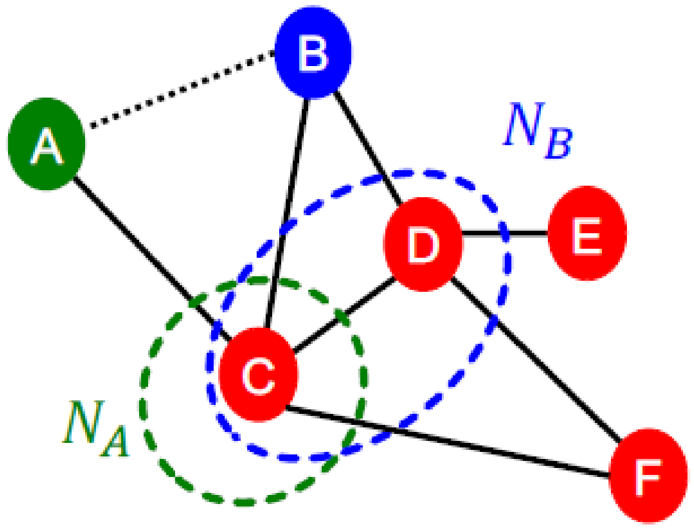
- However, this does not capture the degree of neighborhood overlap:
  - Node pair *(B, H)* has 2 shared neighboring nodes, while pairs *(B, E)* and *(A, B)* only have 1 such node.

Local Neighborhood Overlap

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

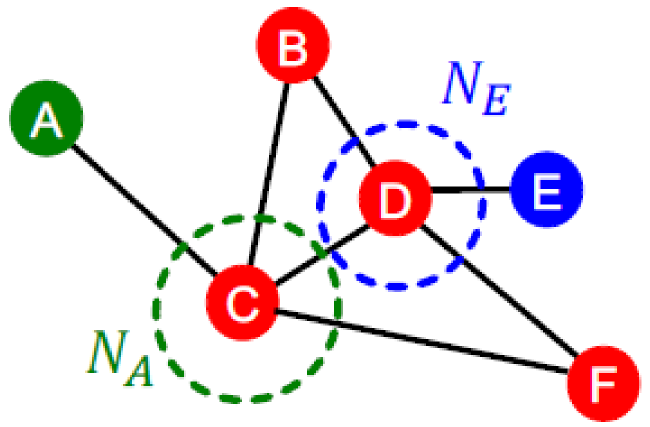
- **Common neighbors:** $|N(v_1) \cap N(v_2)|$
  - Example: $|N(A) \cap N(B)| = |\{C\}| = 1$
- **Jaccard's coefficient:** $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$
  - Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C,D\}|} = \frac{1}{2}$
- **Adamic-Adar index:**

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



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

  - 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

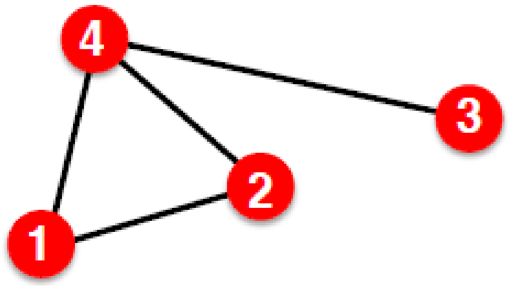
- Katz index: count the number of paths of all lengths between a given pair of nodes.

- Q: **How to compute #paths between two nodes?**
- Use **powers of the graph adjacency matrix**!

Power of Adjacency matrices

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

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



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

Power of Adjacency matrix

- **How to compute $P_{uv}^{(2)}$?**
  - **Step 1:** Compute **#paths** of length 1 **between each of u's neighbor and v**
  - **Step 2**: **Sum up** these #paths 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

#paths of length 1 between Node **1**'s neighbors and Node **2**

$P_{12}^{(2)} = A_{12}^2$

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

Power of adjacency

Global Neighborhood Overlap

- **Katz index:** count the number of paths of all lengths between a pair of nodes.

- How to compute #paths between two nodes?
- Use **adjacency matrix powers**!
  - $\boldsymbol{A}_{uv}$ specifies #paths of length 1 (direct neighborhood) between u and v.
  - $\boldsymbol{A}_{uv}^{\boldsymbol{2}}$ specifies #paths of **length 2** (neighbor of neighbor) between u and v.
  - And, $\boldsymbol{A}_{uv}^{\boldsymbol{l}}$ specifies #paths of **length $\boldsymbol{l}$**.

Global Neighborhood Overlap

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

Sum over *all path lengths*

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \boxed{\beta^l} \boxed{A^l_{v_1 v_2}}$$

#paths of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

$$S = \sum_{i=1}^{\infty} \beta^i A^i = \underbrace{(I - \beta A)^{-1}}_{= \sum_{i=0}^{\infty} \beta^i A^i} - I,$$

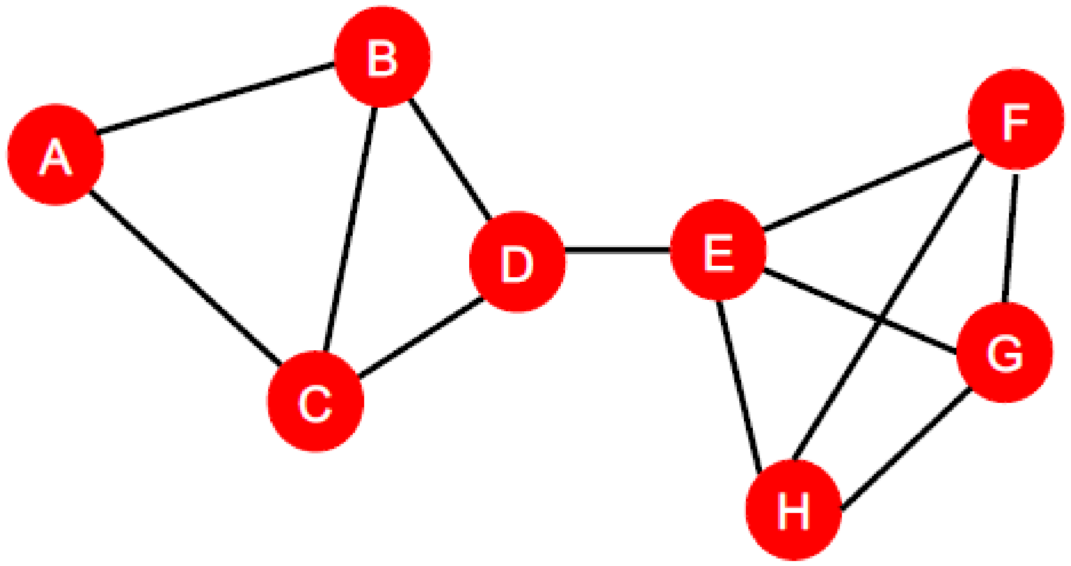
by geometric series of matrices

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 #paths of all lengths between two nodes.

Graph Level Features

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



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 $\boldsymbol{K} = \left(K(G, G')\right)_{G,G'}$ must always be positive semidefinite (i.e., has positive eigenvals)
  - There exists a feature representation $\phi(\cdot)$ such that $K(G, G') = \phi(\mathrm{G})^{\mathrm{T}}\phi(G')$
  - Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions.

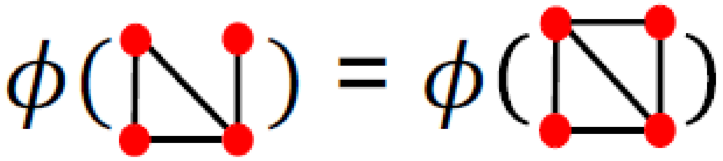
Graph Level Features

- **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

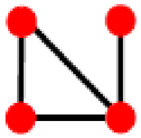
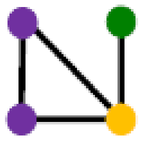
- **Goal**: Design graph feature vector $\phi(G)$
- **Key idea**: Bag-of-Words (BoW) for a graph
  - **Recall:** BoW simply uses the word counts as features for documents (no ordering considered).
  - Naïve extension to a graph: Regard nodes as words.
  - Since both graphs have 4 red nodes, we get the same feature vector for two different graphs…



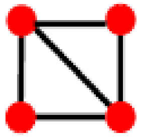
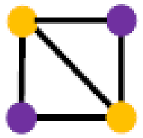
Graph Kernel

What if we use Bag of **node degrees**?

Deg1: ● Deg2: ● Deg3: ●

$\phi($  $)$ = count() = [1, 2, 1]

$\neq$ Obtains different features for different graphs!

$\phi($  $)$ = count() = [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

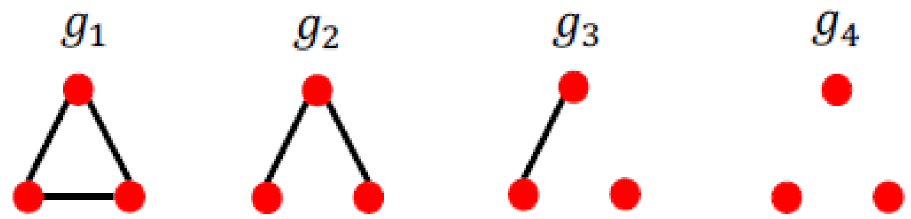
- Key idea: Count the number of different graphlets in a graph.

  - **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.

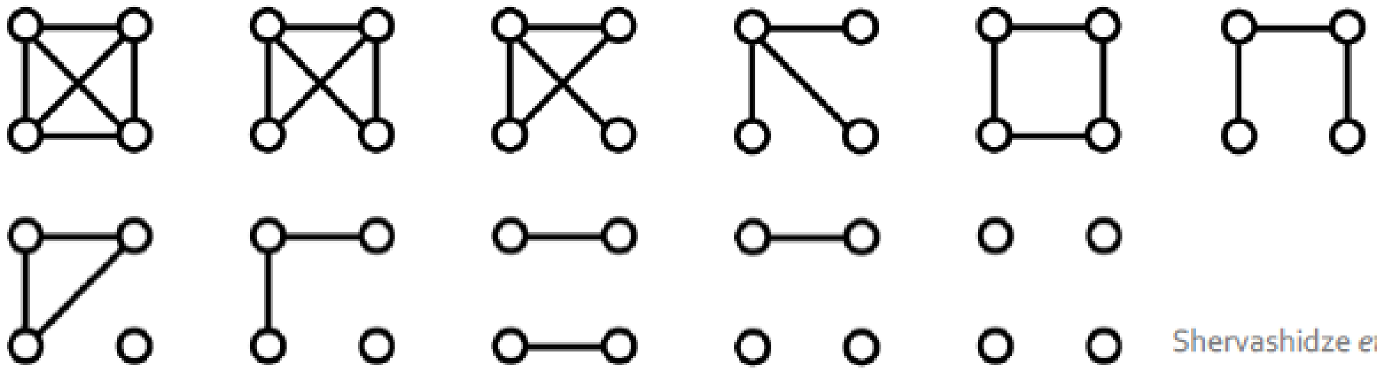
Graphlet features Example

Let $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$ 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

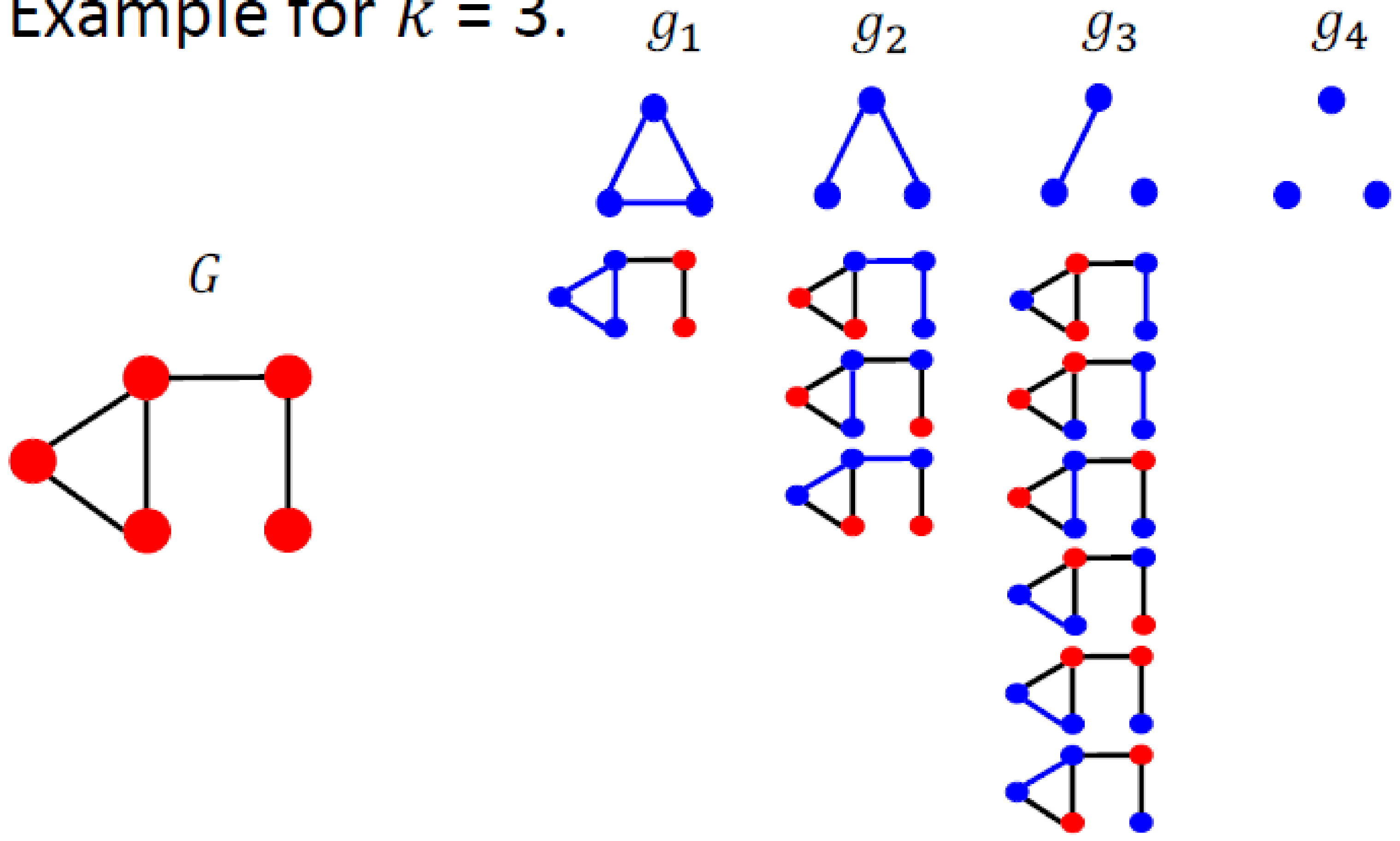
Graphlet features

- Given graph G, and a graphlet list $\mathcal{G}_k = (g_{1,} g_2, \ldots, g_{n_k})$, define the graphlet count vector $\boldsymbol{f}_G \in \mathbb{R}^{n_k}$ as

$$(\boldsymbol{f}_G)_i = \#(g_i \subseteq G) \text{ for } i = 1,2, \ldots, n_k.$$

Graphlet Features

- Example for k = 3.



$$f_G = (1, \quad 3, \quad 6, \quad 0)^{\mathrm{T}}$$

Graphlet Kernel

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

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

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

$$\boldsymbol{h}_G = \frac{\boldsymbol{f}_G}{\mathrm{Sum}(\boldsymbol{f}_G)} \qquad K(G, G') = \boldsymbol{h}_G{}^{\mathrm{T}} \boldsymbol{h}_{G'}$$

Graph 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

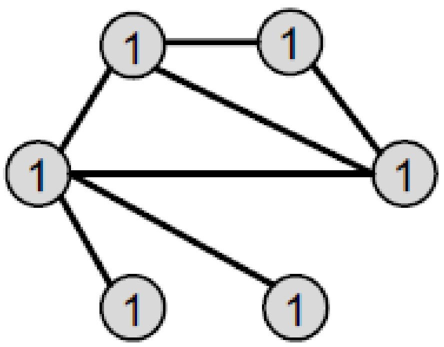
    $$c^{(k+1)}(v) = \mathrm{HASH}\left(\left\{c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right\}\right),$$

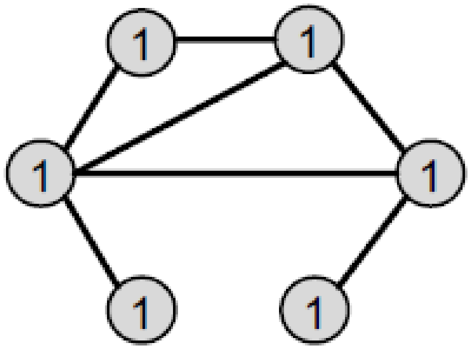
    where HASH maps different inputs to different colors.
  - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood

Color refinement

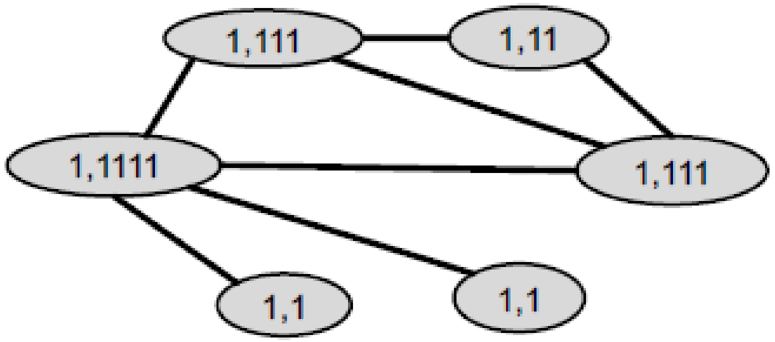
Example of color refinement given two graphs

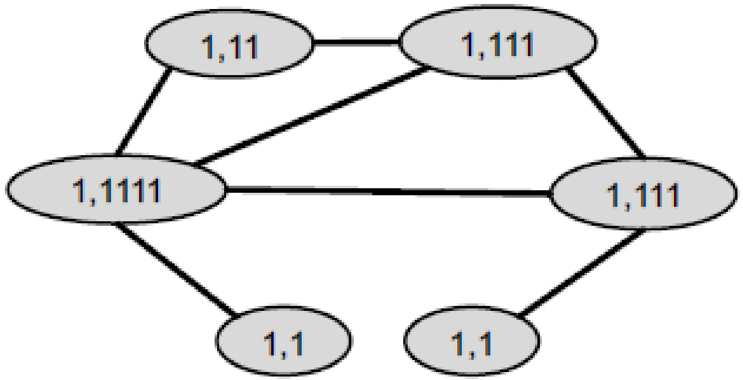
- Assign initial colors





- Aggregate neighboring colors

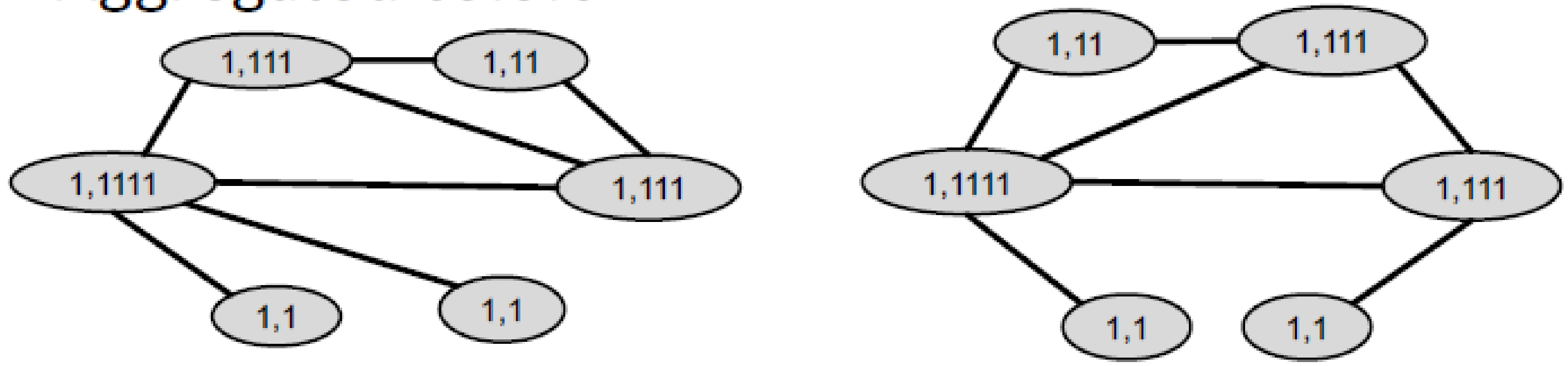




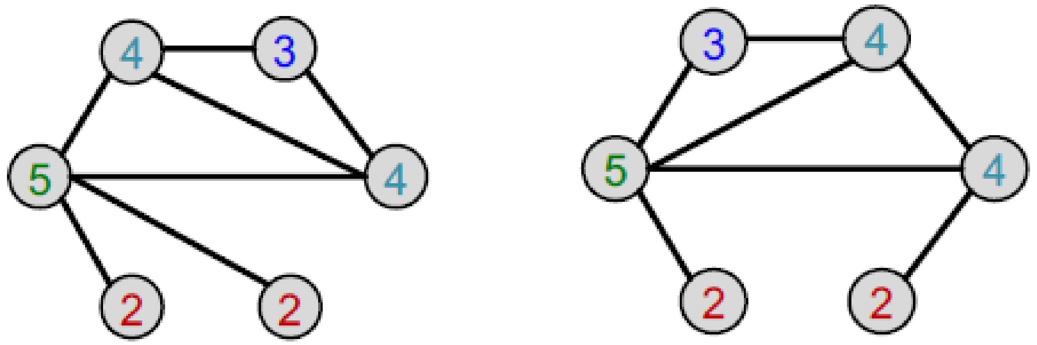
Color refinement

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors



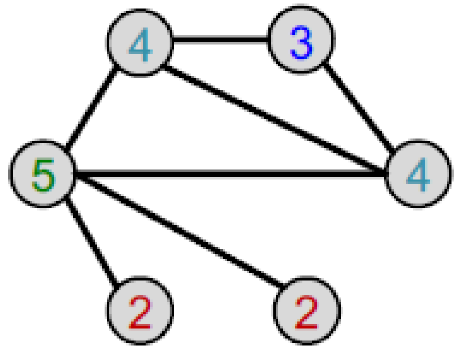
Hash table

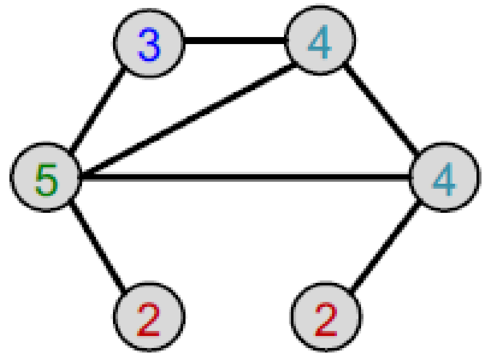
1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Color refinement

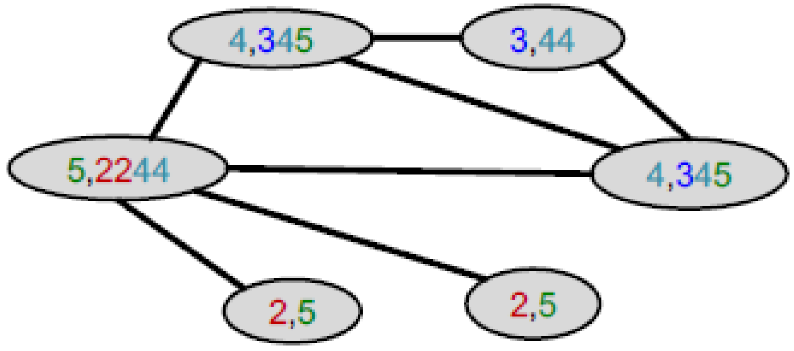
Example of color refinement given two graphs

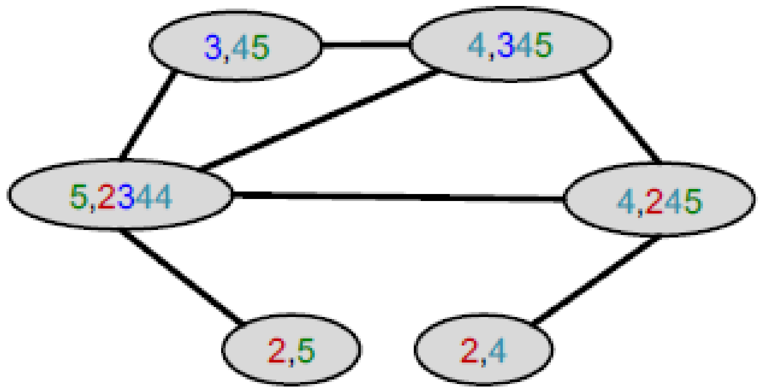
- Aggregated colors





- Hash aggregated colors

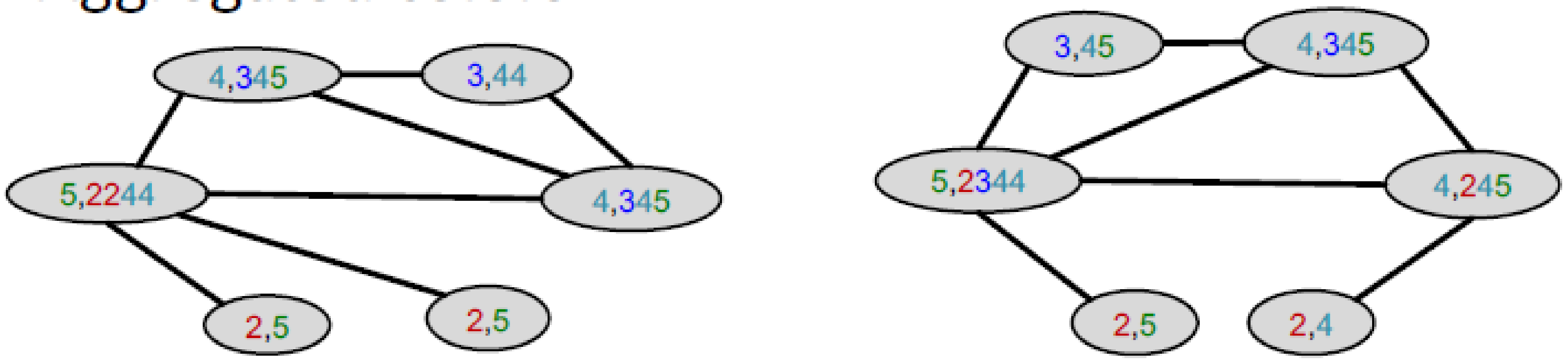




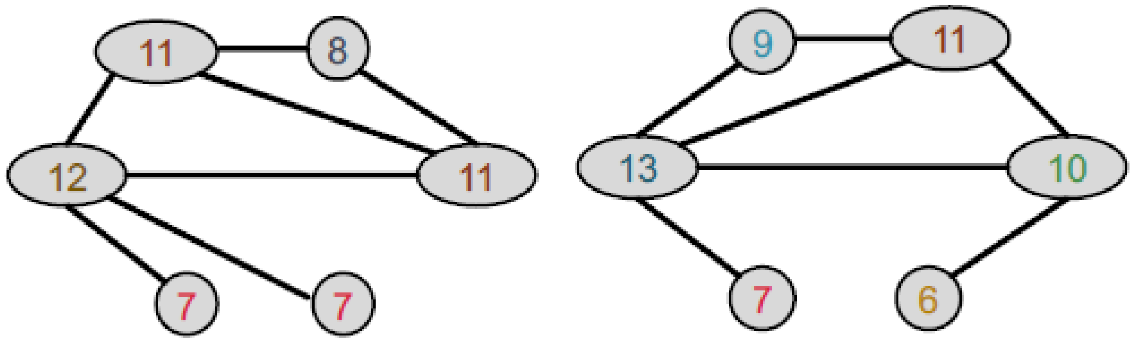
Color refinement

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors

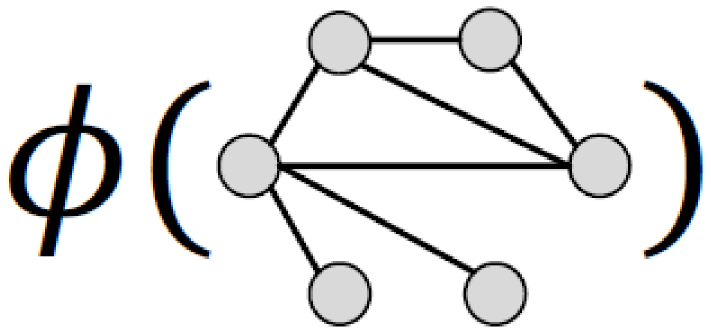


Hash table

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

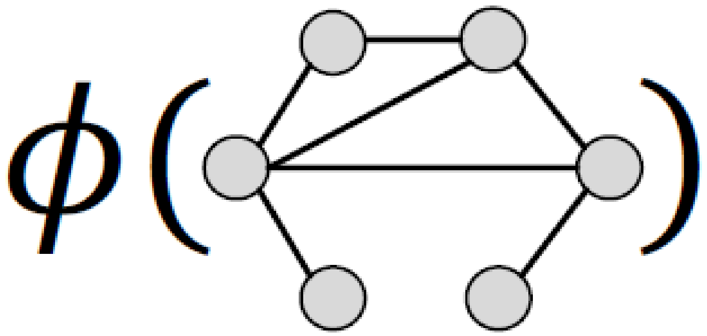
After color refinement, WL kernel counts number of nodes with a given color.



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]

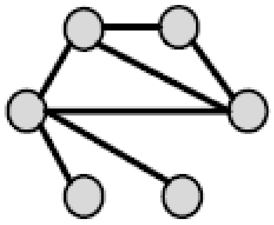
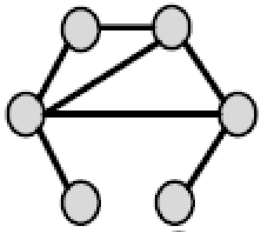
Counts

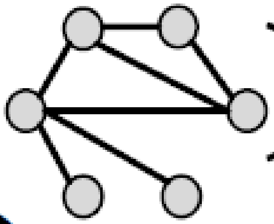
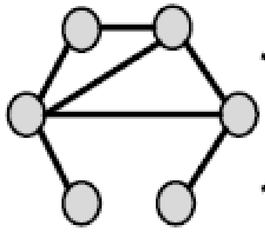


1,2,3,4,5,6,7,8,9,10,11,12,13
= [6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

Weisfeiler-Lehman Kernel

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

$K($  $,$  $)$

$= \phi($  $)^{T} \phi($  $)$

$= 49$

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!)