Yale

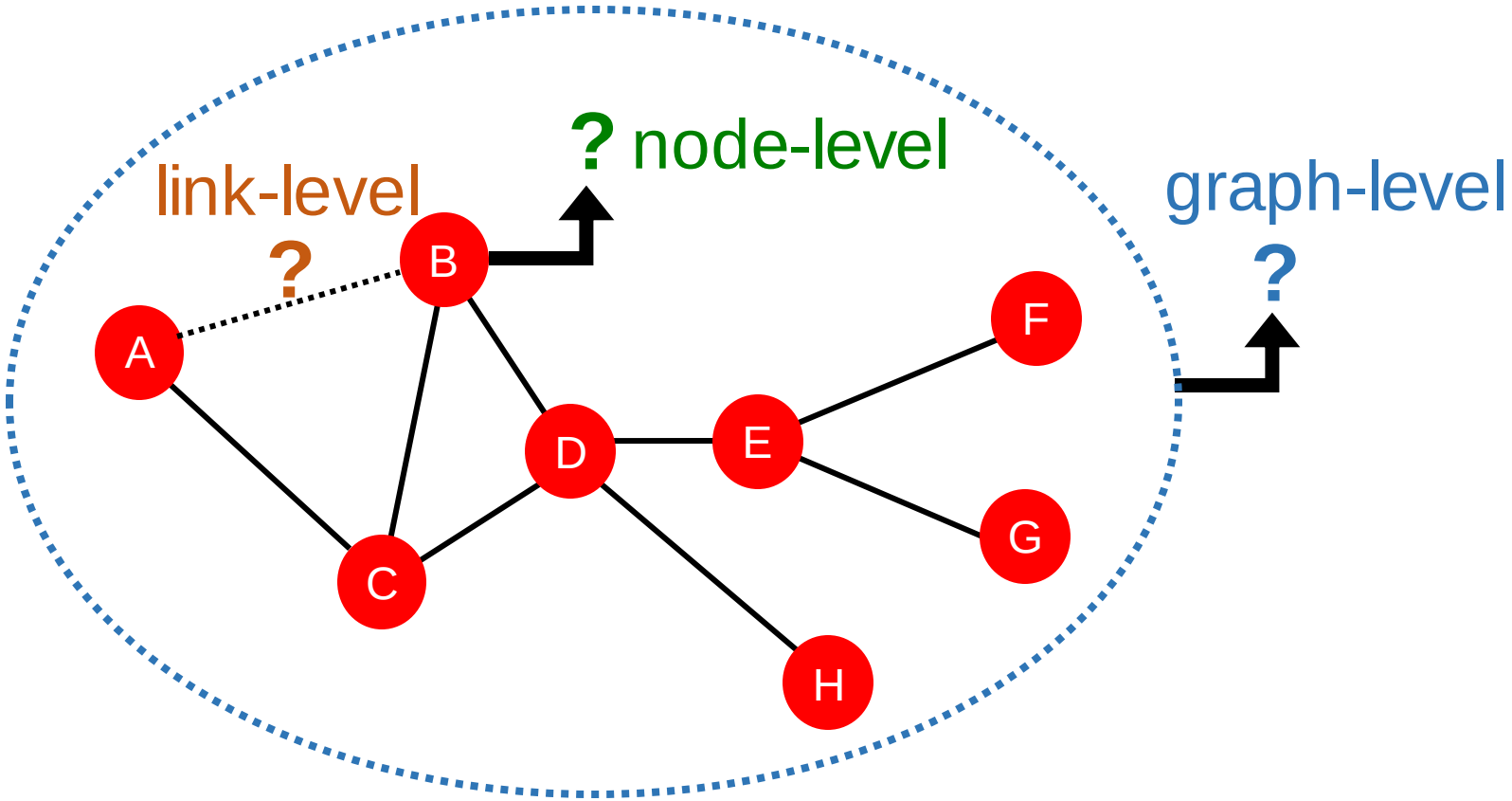
Machine Learning Tasks for Graph-Structured Data

CPSC483: Deep Learning on Graph-Structured Data

Rex Ying

Graph Machine Learning Tasks: Overview

- Node-level prediction
- Link-level prediction
- Graph-level prediction
- Graph generation
  - Generative model



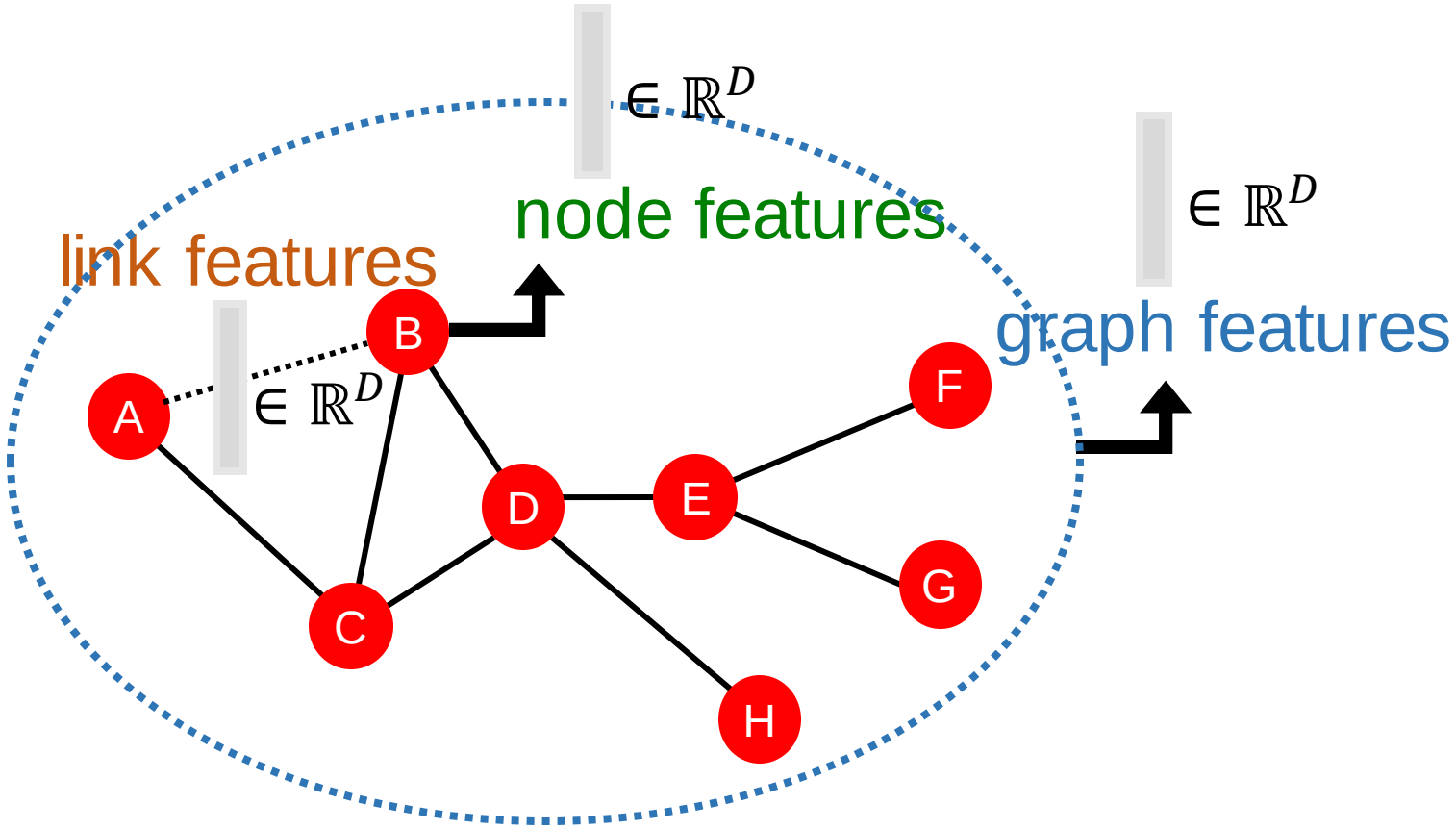
Traditional Graph ML Pipeline: Overview

1. Handcraft features
   - node features
   - link features
   - graph features
2. Train an ML model
   - Decision Tree
   - Logistic Regression
   - Support Vector Machine (SVM)
   - …
3. Apply the model
   - Given a new node/link/graph, obtain its features and make a prediction

Traditional Graph ML Pipeline (1)

1. Handcraft features
   - node features
   - link features
   - graph features

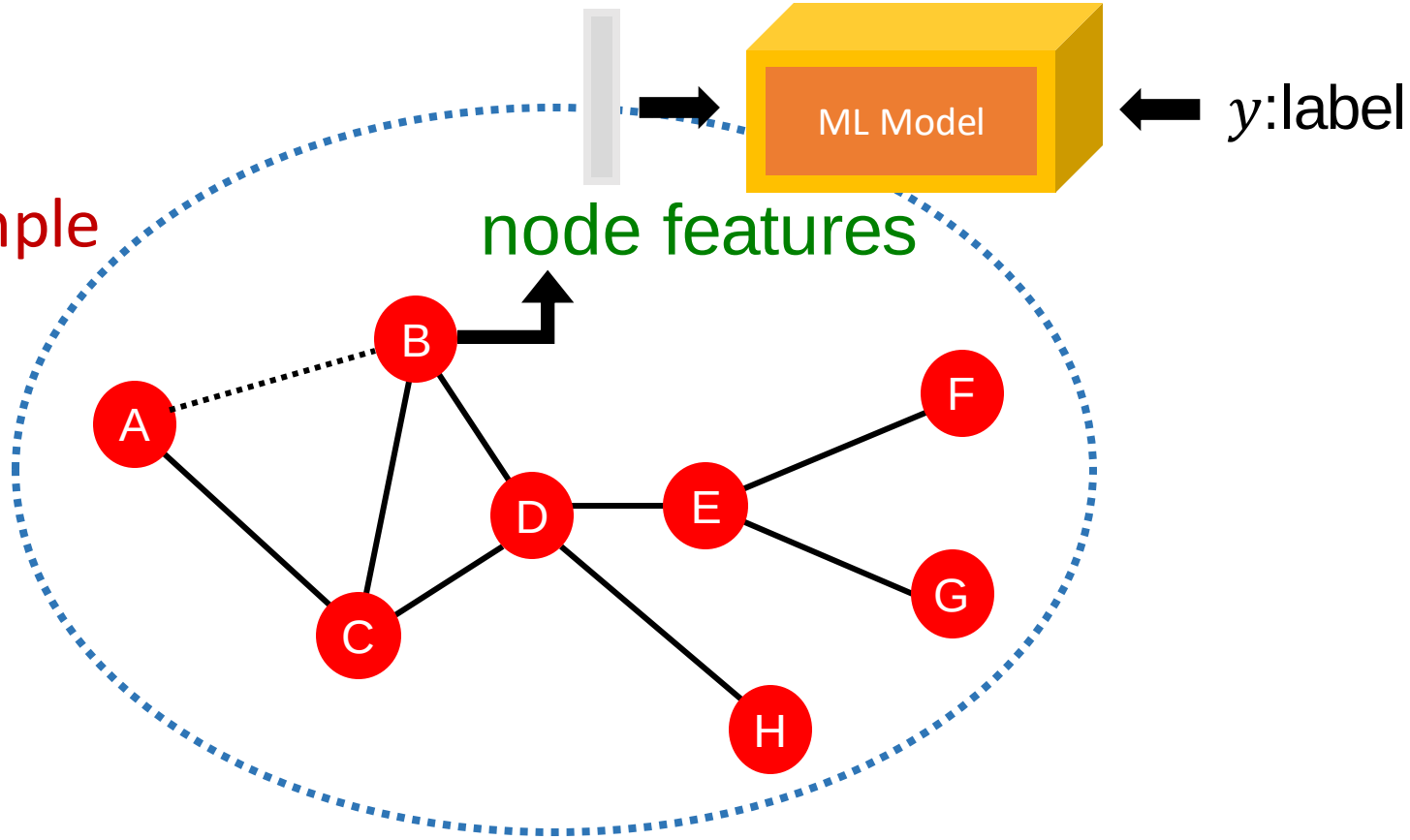
Let's take node features for example.



Traditional Graph ML Pipeline (2)

Taking node features for example

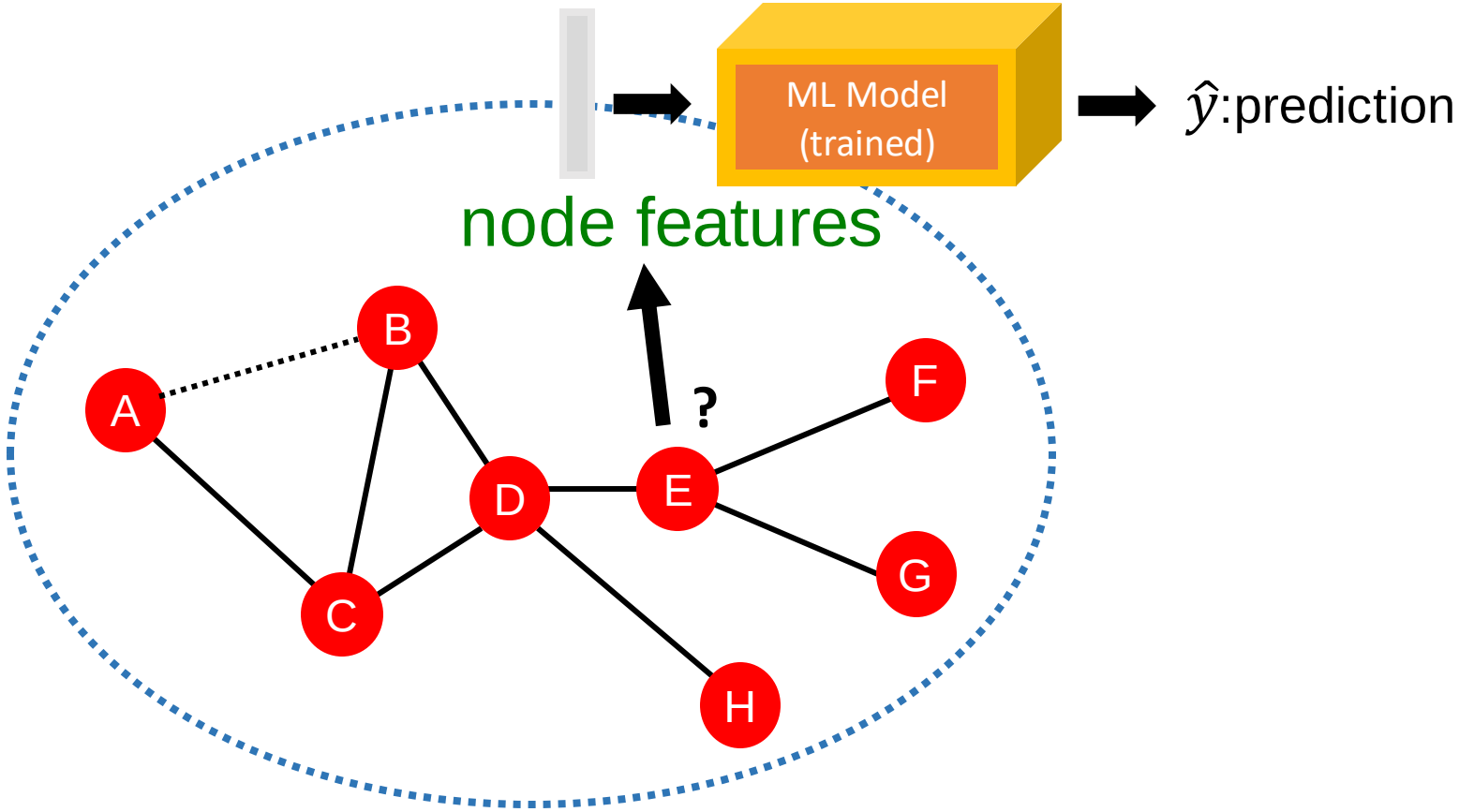
2. Train an ML model
   - Decision Tree
   - Logistic Regression
   - Support Vector Machine
   - …



Traditional Graph ML Pipeline (3)

Taking node features for example

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



This Lecture: Feature Design

- **Using effective features over graphs is the key to achieving good test performance.**
- Traditional ML pipeline uses hand-designed features.
- **In this lecture, we overview the traditional features for**
  - Node-level prediction
  - Link-level prediction
  - Graph-level prediction
- For simplicity, we focus on undirected graphs.

Feature Design: Inherent Features

- **Sometimes elements (nodes, edges, etc.) of a graph have inherent features**. For example,
  - In **social networks**, a user's
    - Age
    - Gender
    - Occupation
    - Education
  - In **molecules**, a bond's
    - Type
    - Order
    - Aromaticity
    - Conjugation

Machine Learning in Graphs

Design choices:

- **Features:** *d*-dimensional vectors
  - Data-specific node / edge / graph features
  - Extract features from graph structure
- **We want to predict:** nodes, edges, sets of nodes, entire graphs
- **Objective function:**
  - Classification: categorical
  - Regression: numerical

Machine Learning Tasks for Graph-structured Data

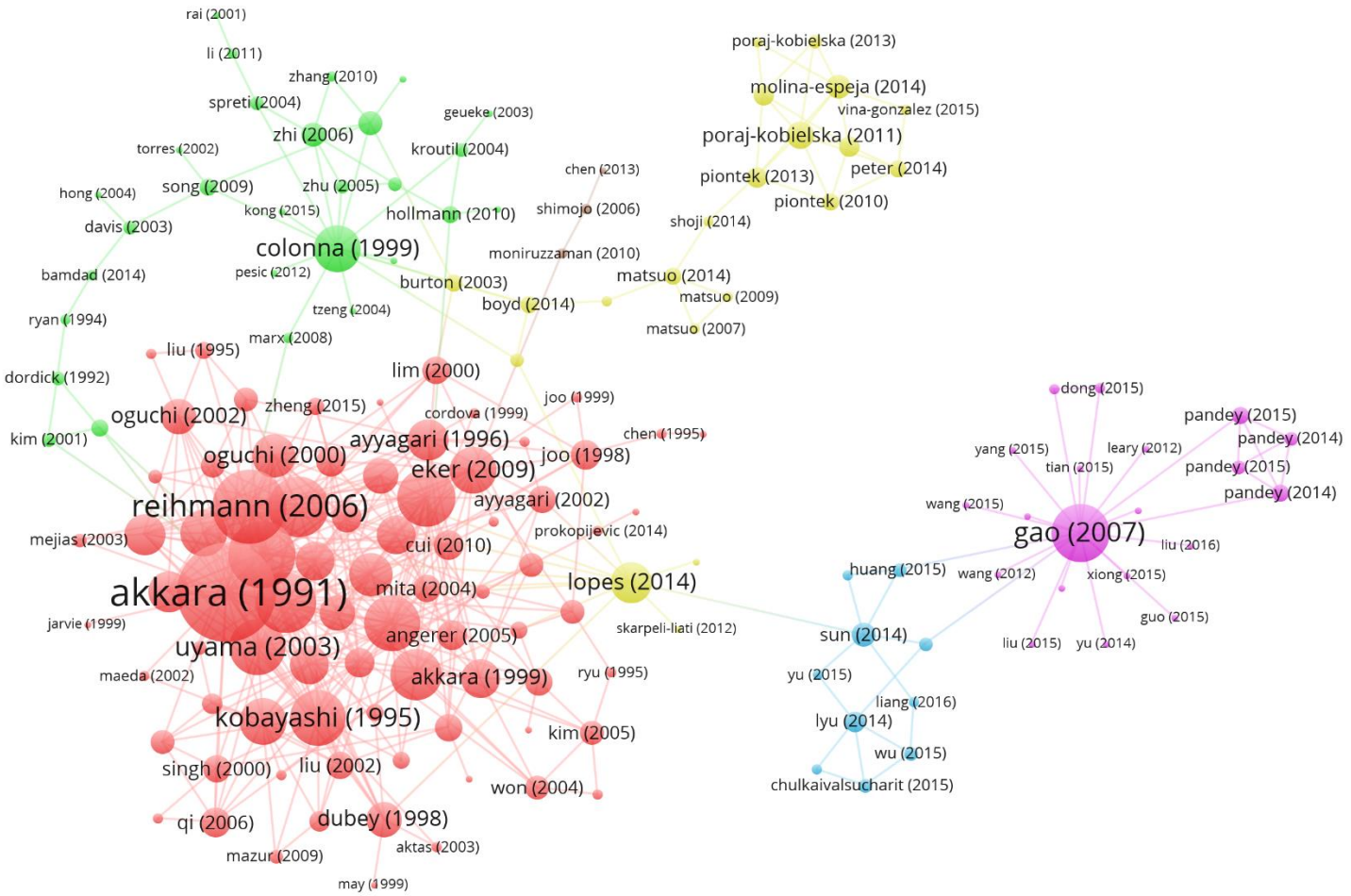
- **Node-level Tasks and Features**
- **Link / edge Prediction Tasks and Features**
- **Graph-Level Tasks, Features and Graph Kernels**

Machine Learning Tasks for Graph-structured Data

- **Node-level Tasks and Features**
- **Link / edge Prediction Tasks and Features**
- **Graph-Level Tasks, Features and Graph Kernels**

Node-Level Task (1)

- Citation Networks
  - **Nodes**: papers
  - **Edges**: citations
  - **Predictions**: papers' research fields.



Node-Level Task (2)

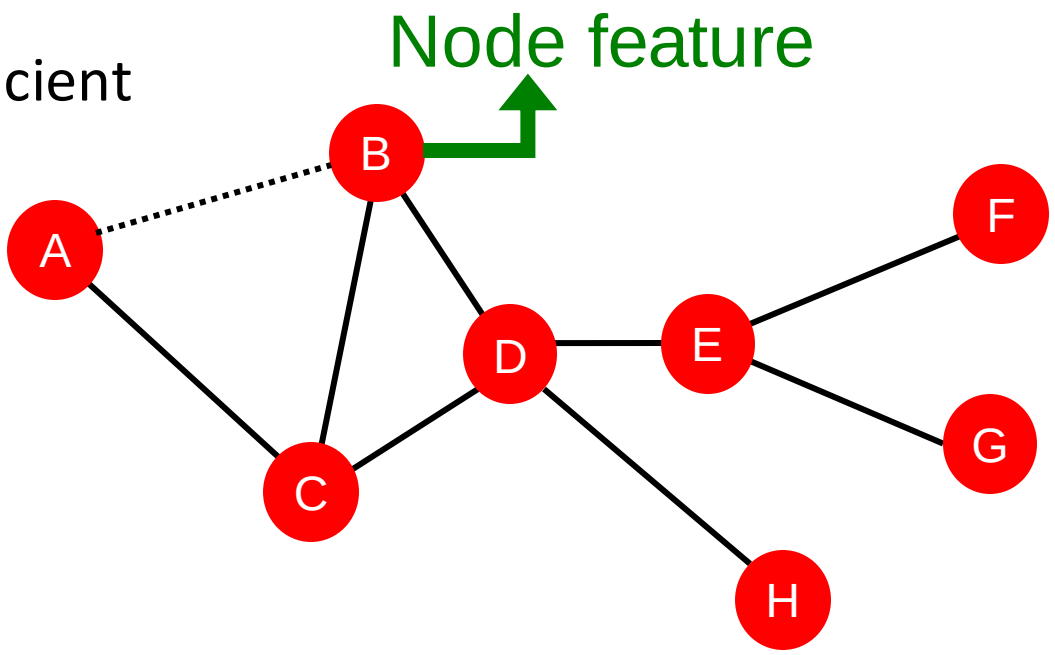
- Molecule's Conformation Generation
  - **Nodes**: atoms in the molecule
  - **Edges**: chemical bonds
  - **Predictions**: 3D position of each atom

Node-Level Features: Overview

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

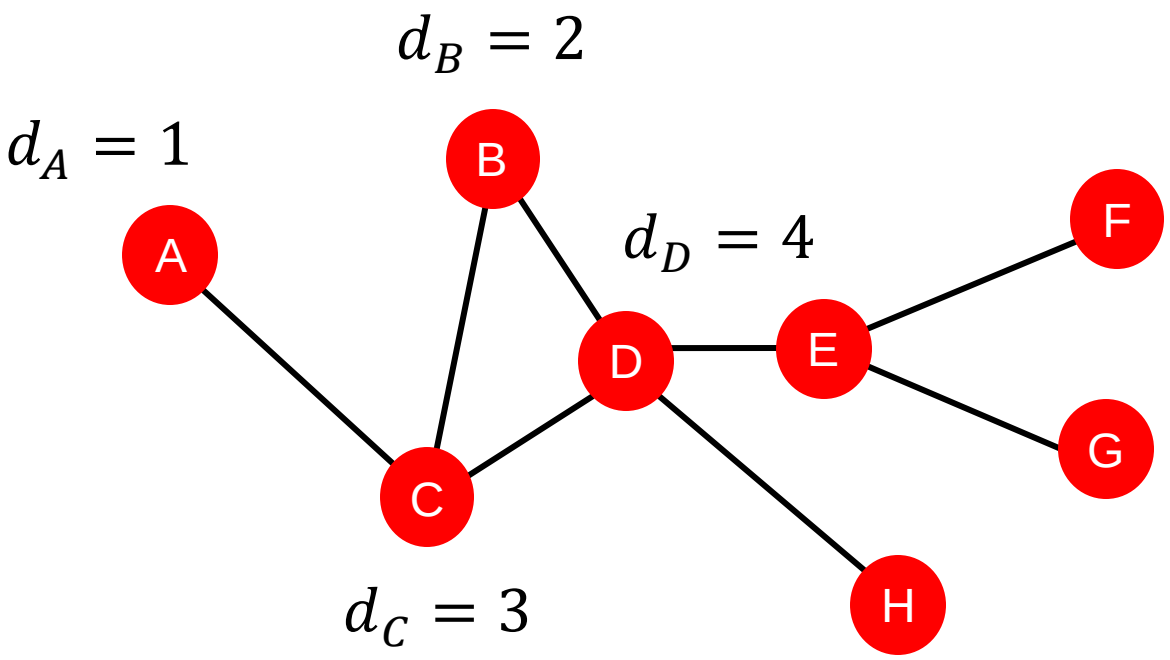
To achieve this goal, we can consider the following types of node features

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



Node Features: Node Degree

- The degree d_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:

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

where λ is some positive constant.

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

Node Centrality (1) Cont.

- **Eigenvector centrality:**
  - Rewrite the recursive equation in the matrix form.

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

where $\boldsymbol{A}$: Adjacency matrix, $\boldsymbol{A}_{uv} = 1$ if $u \in N(v)$;

$\boldsymbol{c}$:Centrality vector.

  - We see that centrality is the **eigenvector**!
  - The largest eigenvalue λ_{max} is always positive and unique (by Perron-Frobenius Theorem).
  - The leading eigenvector $\boldsymbol{c}_{max}$ is used for centrality.

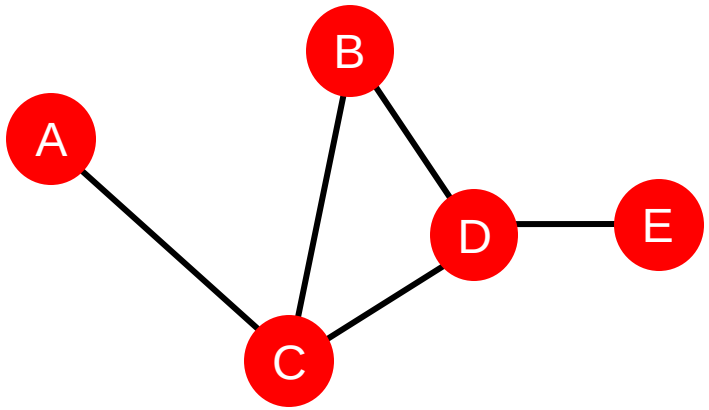
Node Centrality (2)

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

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

Where s is the source node and t is the target node.

  - Example:



$c_A = c_B = c_E = 0$

$c_C = 3$

(A-<u>C</u>-B, A-<u>C</u>-D, A-<u>C</u>-D-E)

$c_D = 3$

(A-C-<u>D</u>-E, B-<u>D</u>-E, C-<u>D</u>-E)

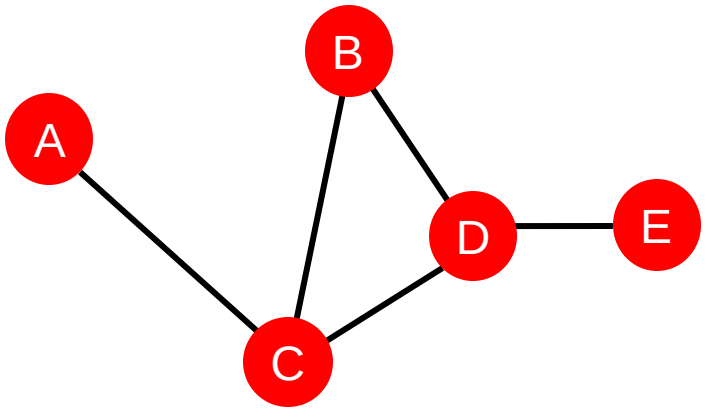
underscore denotes the node v

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)

underscore denotes the node v

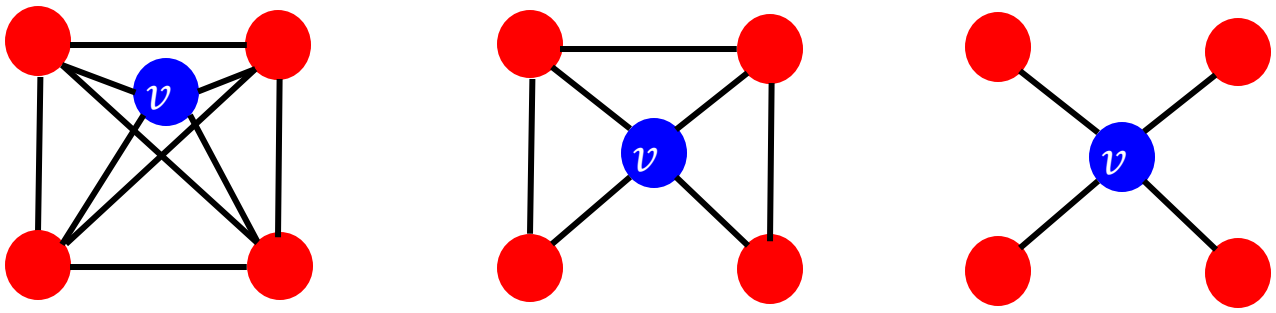
Node Features: Clustering Coefficient

- Measures how connected v's neighboring nodes are:

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

#(node pairs among d_v neighboring nodes)

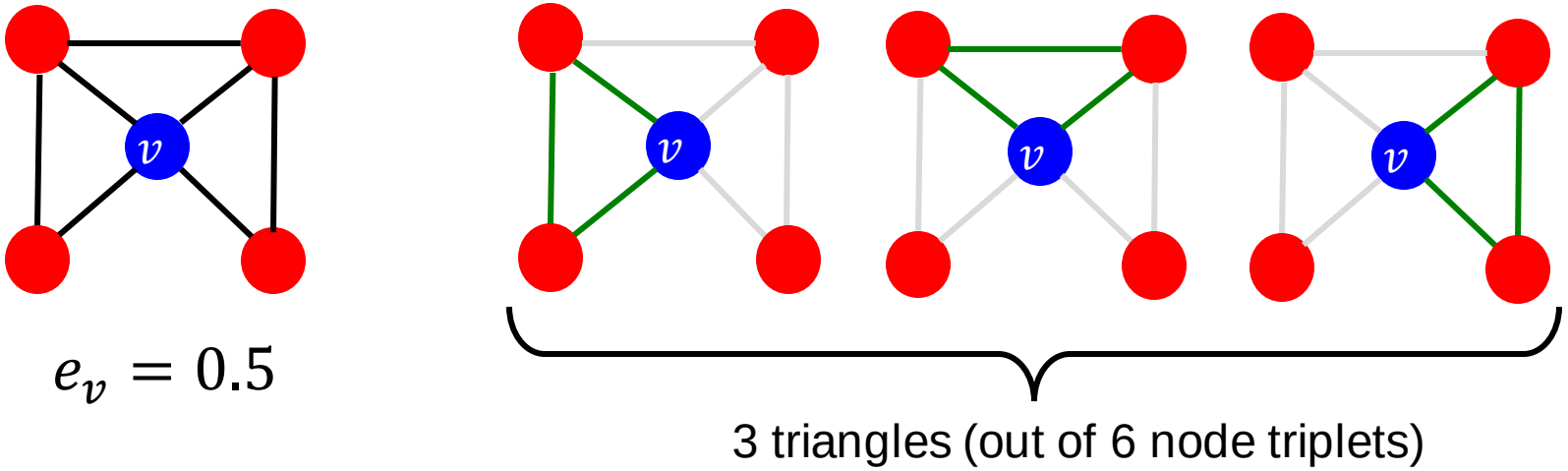
- Examples



$e_v = 1$ $\quad$ $e_v = 0.5$ $\quad$ $e_v = 0$

Node Features: Graphlets

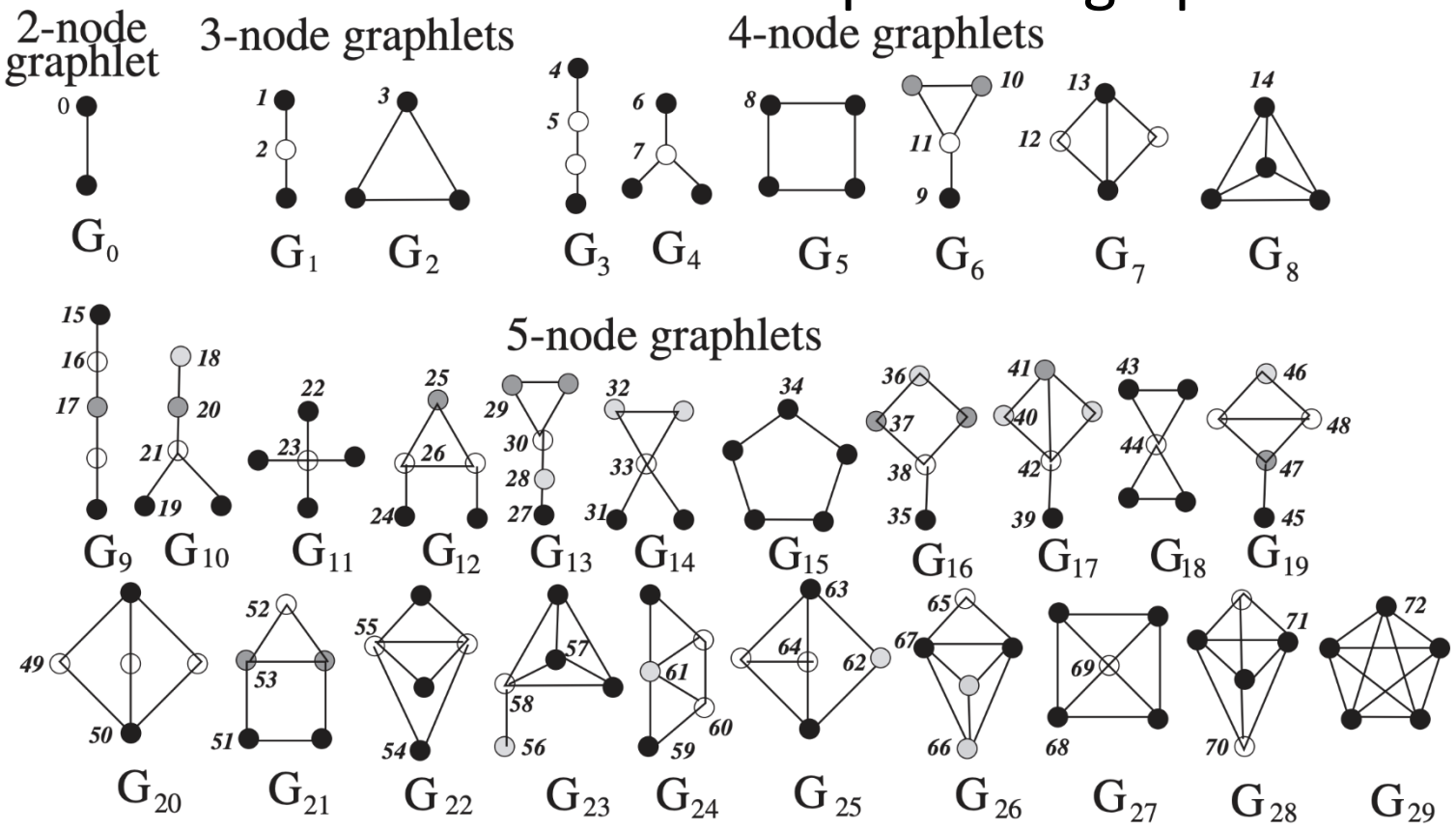
- **Observation:** Clustering coefficient counts the #(triangles) in the **ego-network** centered around v.



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

Node Features: Graphlets (1)

- **Graphlets**: Rooted connected non-isomorphic subgraphs:



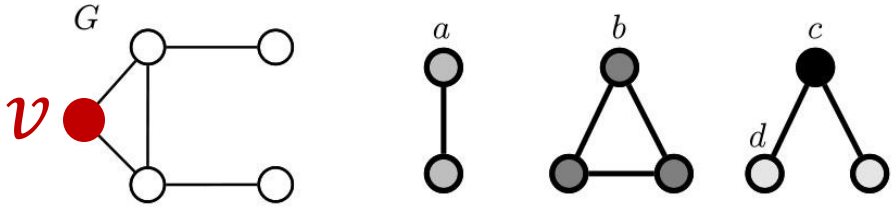
(Przulj et al., 2004)

equivalent nodes are of the same shade

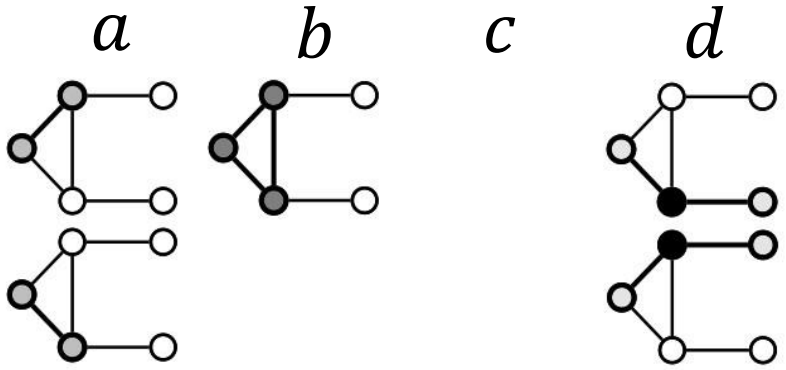
Node Features: Graphlets (2)

- **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 (3)

- 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 Features: Summary (1)

- **We have introduced different ways to obtain node features (except inherent 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
- These features can be **concatenated** to form a feature vector.

Node-Level Features: Summary (2)

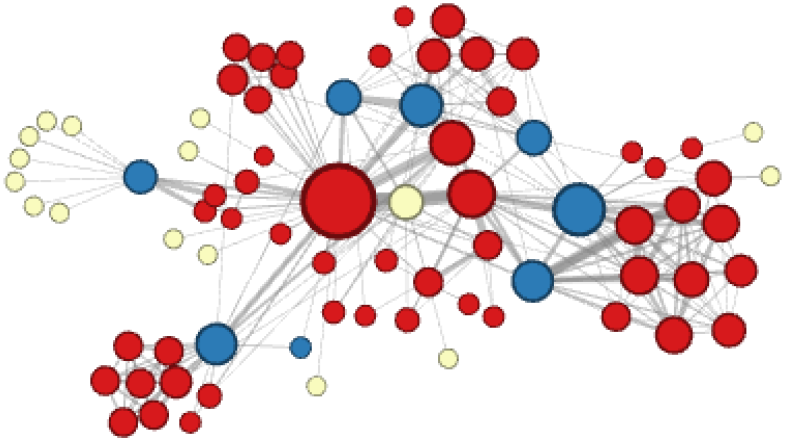
- **Importance-based features**: capture the importance of a node is in a graph
  - Node degree:
    - Simply counts the number of neighboring nodes
  - Node centrality:
    - Models importance of neighboring nodes in a graph
    - Different modeling choices: eigenvector centrality, betweenness centrality,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

Node-Level Features: Summary (3)

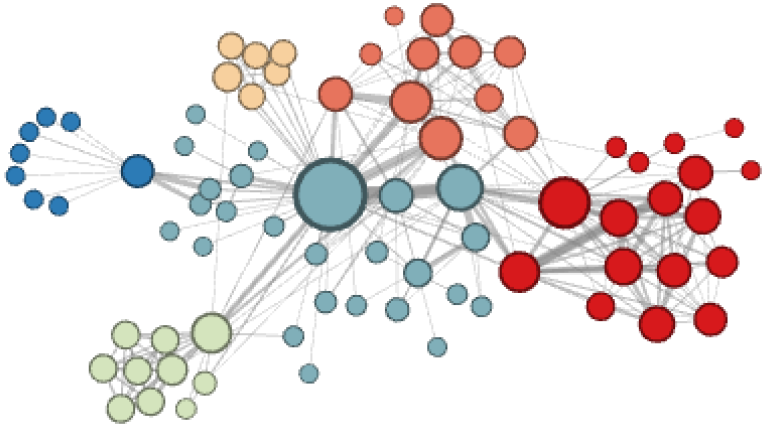
- **Structure-based features**: Capture topological properties of local neighborhood around a node.
  - **Node degree:**
    - Counts the number of neighboring nodes
  - **Clustering coefficient:**
    - Measures how connected neighboring nodes are
  - **Graphlet degree vector:**
    - Counts the occurrences 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.

Discussion

- **Different ways to label nodes of the network:**



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



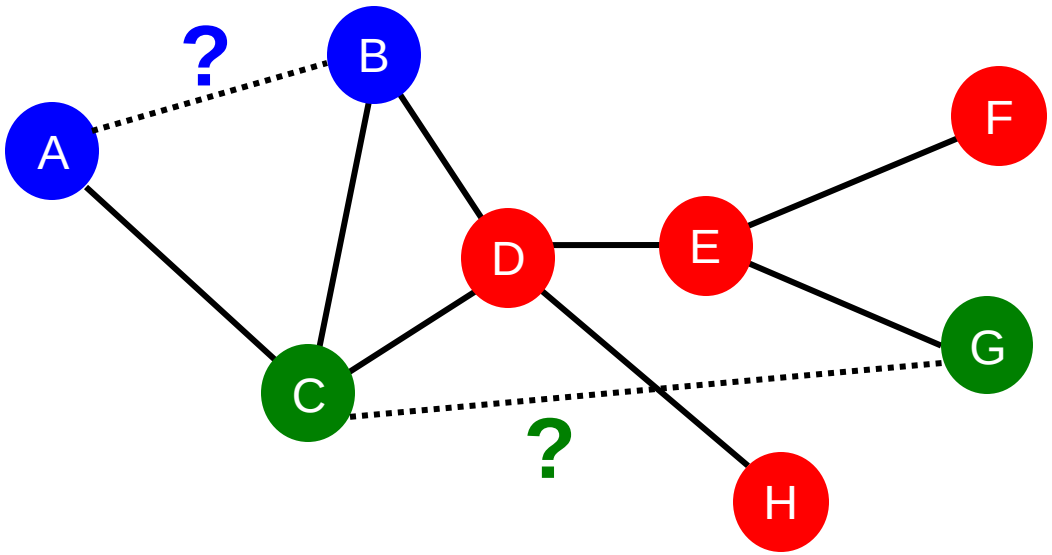
However, the features defines so far would not allow for distinguishing the above node labelling

Machine Learning Tasks for Graph-structured Data

- Node-level Tasks and Features
- **Link / edge Prediction Tasks and Features**
- Graph-Level Tasks, Features and Graph Kernels

Link-Level Prediction Task

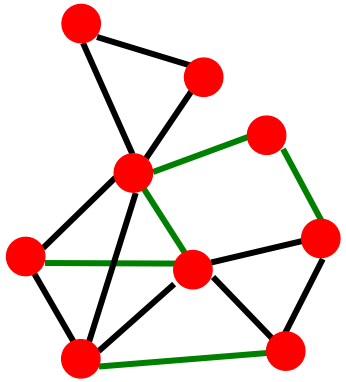
- The task is to predict **new / missing links** based on existing links.
- At test time, all node pairs (no existing links) are ranked, and top K node pairs are predicted.
- The key is to design features for **a pair of nodes**.



Link Prediction as a Task

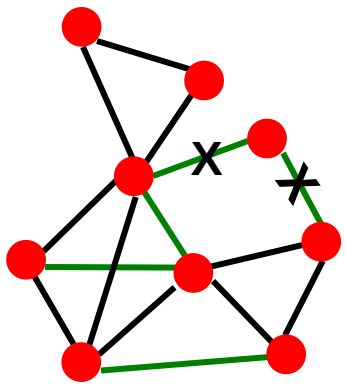
Two formulations of the link prediction task:

- **1) Links missing:**
  - In some networks, links may be not fully discovered, e.g. knowledge graph, protein-protein interaction network.
- **2) Links over time:**
  - We will talk more about **temporal graph** tasks in future lectures



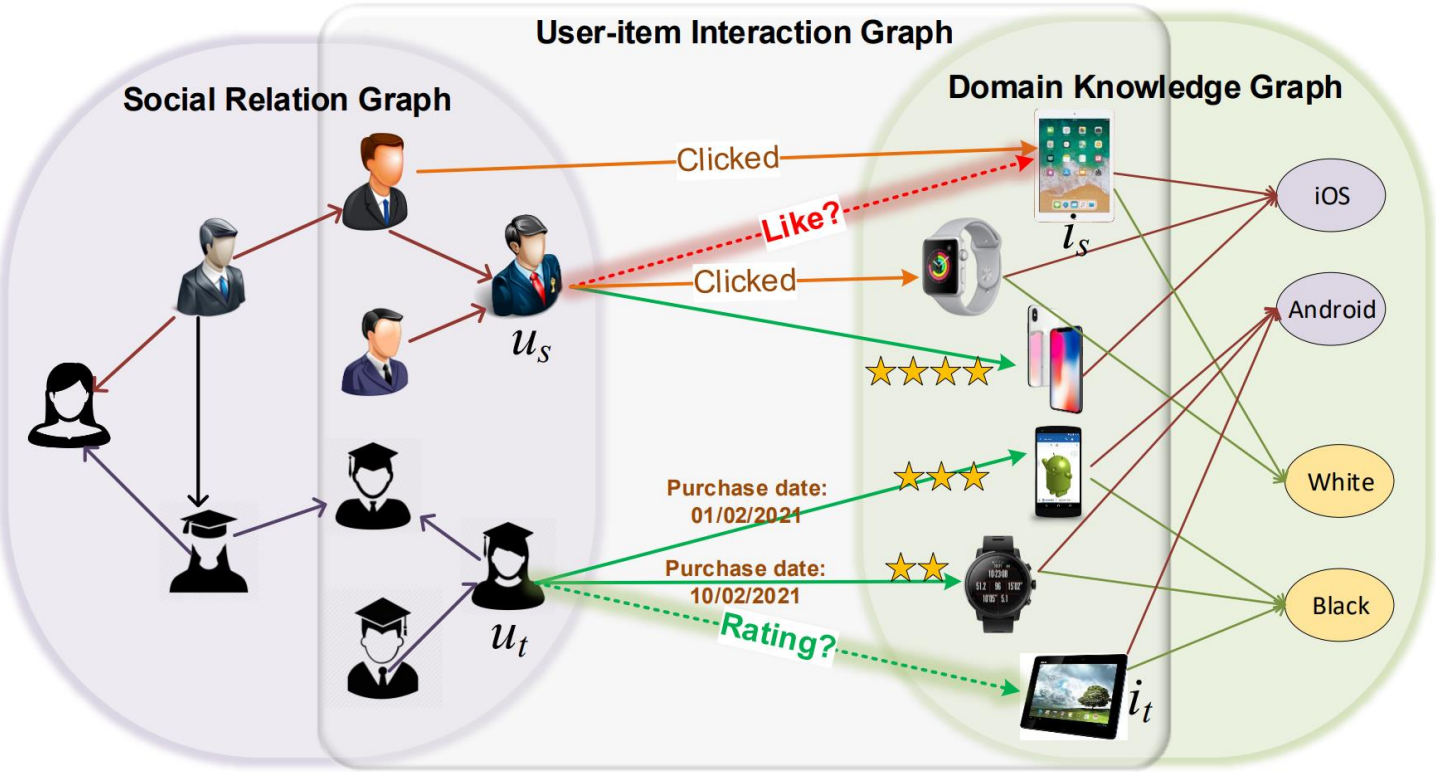
Link Prediction via Proximity

- **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 the ground truths.**



Link-Level Task (1)

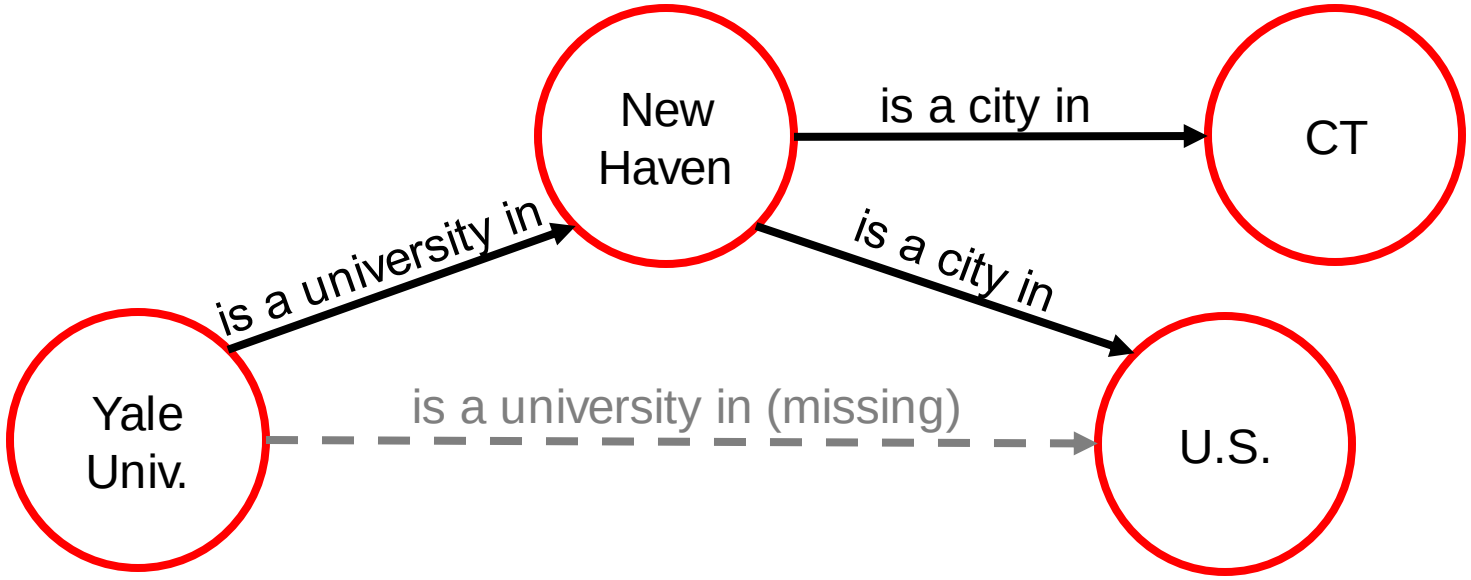
- **User-item interactions**
  - **Nodes**: users, items
  - **Edges**: interactions (e.g. clicks, purchases)
  - **Predictions**: potential interactions



(Wang et al., 2020)

Link-Level Task (2)

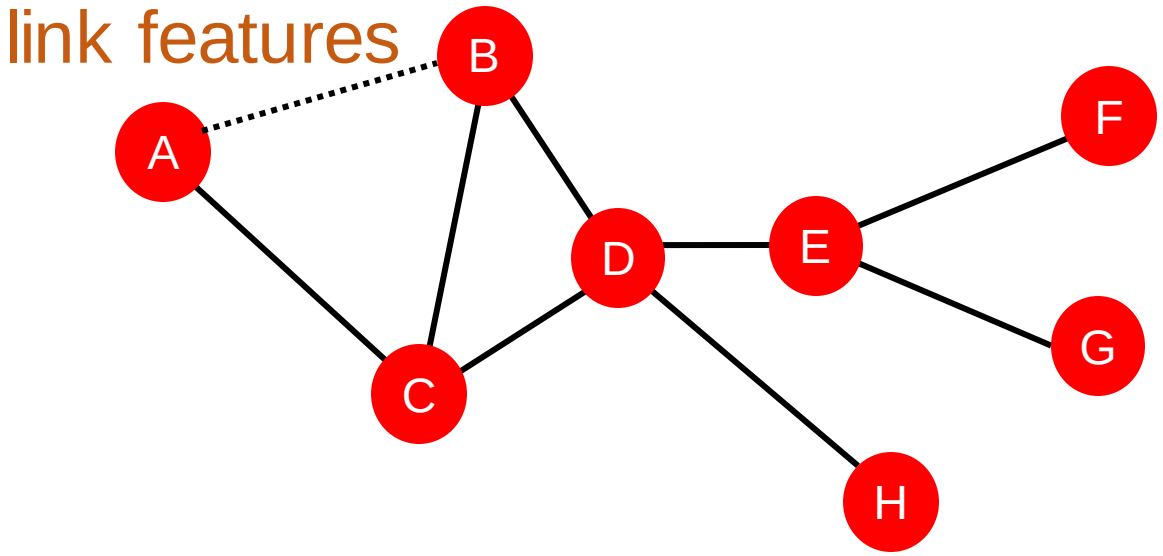
- **Knowledge Graph Completion**
  - **Nodes**: entities
  - **Edges**: relations
  - **Predictions**: missing relations



Some famous knowledge graphs:
- FreeBase
- WordNet
- DDB
- NELL

Link-Level Features: Overview

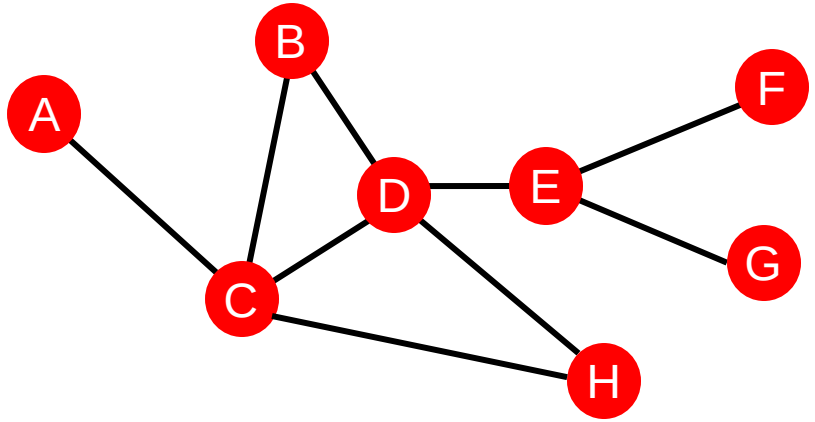
- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap



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.

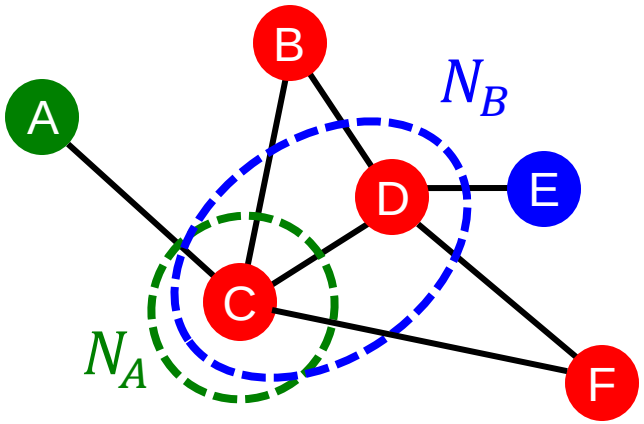
Local Neighborhood Overlap

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

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

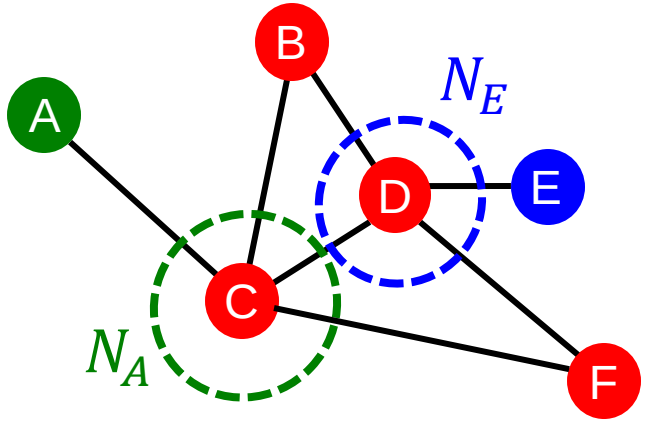
  - Example: $\frac{1}{\log(d_C)} = \frac{1}{\log 4}$



Make sure we define in the first lecture!

Global Neighborhood Overlap (1)

- **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 Overlap (2)

- **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**!

Powers of Adjacency Matrices (1)

- **Theorem**: Let G be a graph with adjacency matrix $\boldsymbol{A}$ w.r.t. the ordering v_1, v_2, …, v_n of the vertices of the graph. Let $P_{ij}^{(r)}$ be the number of different **paths** of **length $\boldsymbol{r}$** from v_i to v_j, where r is a positive integer. We have

$$P_{ij}^{(r)} = \boldsymbol{A}_{ij}^{r}$$

  where $\boldsymbol{A}_{ij}^{r}$ denotes the (i, j)th entry of $\boldsymbol{A}^{r}$.
- **Proof**: using **mathematical induction**.

Powers of Adjacency Matrices (2)

- **For example, to compute** $P_{ij}^{(2)}$
  - **Step 1:** Compute **#(paths)** of length 1 **between each of v_i's neighbor and v_j**
  - **Step 2**: **Sum up** these **#(paths)** across v_i's neighbors v_k
  - $P_{ij}^{(2)} = \sum_k \boldsymbol{A}_{ik} * P_{kj}^{(1)} = \sum_k \boldsymbol{A}_{ik} * \boldsymbol{A}_{kj} = \boldsymbol{A}_{ij}^2$

Node **1**'s neighbors

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

$$\boldsymbol{P}_{\mathbf{12}}^{(\mathbf{2})} = \boldsymbol{A}_{12}^2$$

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

Power of adjacency matrix

Global Neighborhood Overlap (3)

- **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**!
  - $\boldsymbol{A}_{ij}$ specifies #(paths) of length 1 (direct neighborhood) between v_i and v_j.
  - $\boldsymbol{A}_{ij}^{\mathbf{2}}$ specifies #(paths) of **length 2** (neighbor of neighbor) between v_i and v_j.
  - And, $\boldsymbol{A}_{ij}^{\boldsymbol{l}}$ specifies #paths of **length $\boldsymbol{l}$**.

Global Neighborhood Overlap (4)

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

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \boxed{\beta^l} \boxed{\boldsymbol{A}_{12}^l}$$

#(paths) of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

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

by geometric series of matrices

Link-Level Features: Summary

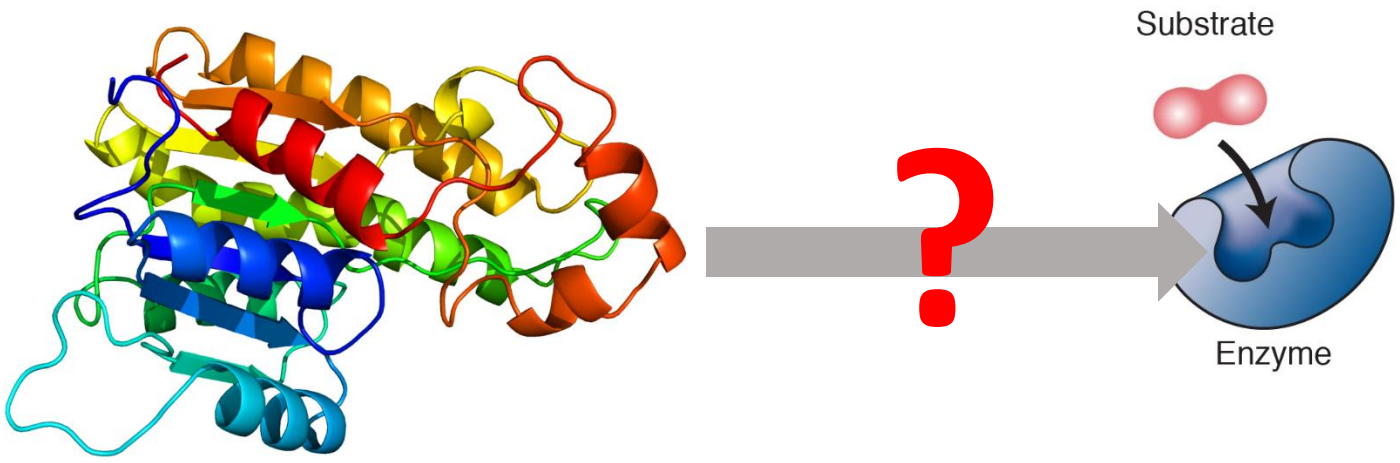
- **Distance-based features:**
  - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.
- **Local neighborhood overlap:**
  - Captures how many neighboring nodes are shared by two nodes.
  - Becomes zero when no neighbor nodes are shared.
- **Global neighborhood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

Machine Learning Tasks for Graph-structured Data

- Node-level Tasks and Features
- Link / edge Prediction Tasks and Features
- **Graph-Level Tasks, Features and Graph Kernels**

Graph-Level Task

- Proteins
  - **Nodes**: amino acids
  - **Edges**: bonds, distances (connect two nodes if the distance between which is less than a threshold)
  - **Predictions**: enzymes or non-enzymes

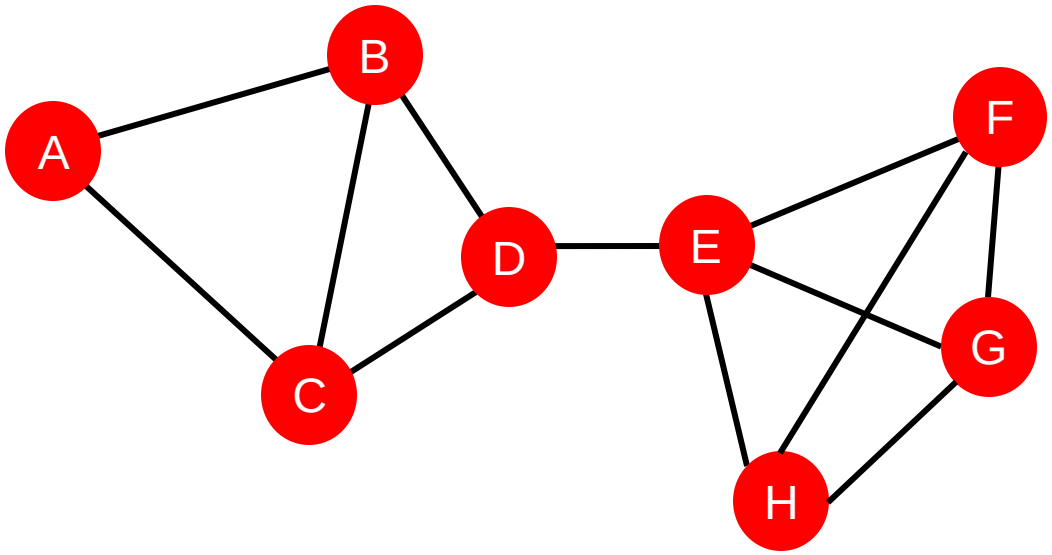


Protein Molecule

Graph-Level Features

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

- **For example:**



Background: Kernel Methods

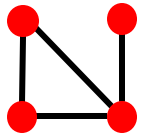
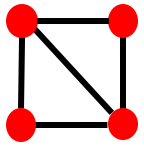
- **Kernel methods** are widely-used for traditional ML for graph-level prediction.
- **Idea: Design kernels instead of feature vectors.**
- **A quick introduction to Kernels:**
  - Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
  - Kernel matrix $\boldsymbol{K} = \left(K(G, G')\right)_{G,G'}$,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

Graph-Level Features: Overview

- **Graph Kernels**: Measure similarity between two graphs (for those interested):
  - Graphlet Kernel
  - Weisfeiler-Lehman Kernel
  - Other kernels are also proposed in the literature
    - Random-walk kernel
    - Shortest-path graph kernel
    - And many more…

Graph Kernel: Key idea (1)

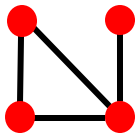
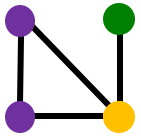
- **Goal**: Design graph feature vector $\phi(G)$
- **Key idea**: Bag-of-Words (BoW) for a graph
  - **Recall:** BoW simply uses the word counts as features for documents (no ordering considered).
  - Naïve extension to a graph: Regard nodes as words.
  - Since both graphs have 4 red nodes, we get the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

$\phi($  $) = \phi($  $)$

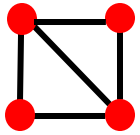
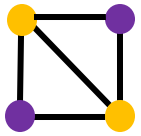
Graph Kernel: Key idea (2)

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

Degree1: ● Degree2: ● Degree3: ●

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

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!

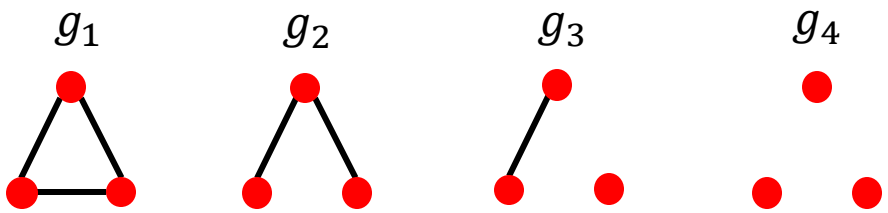
Graphlet Features (1)

- **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.

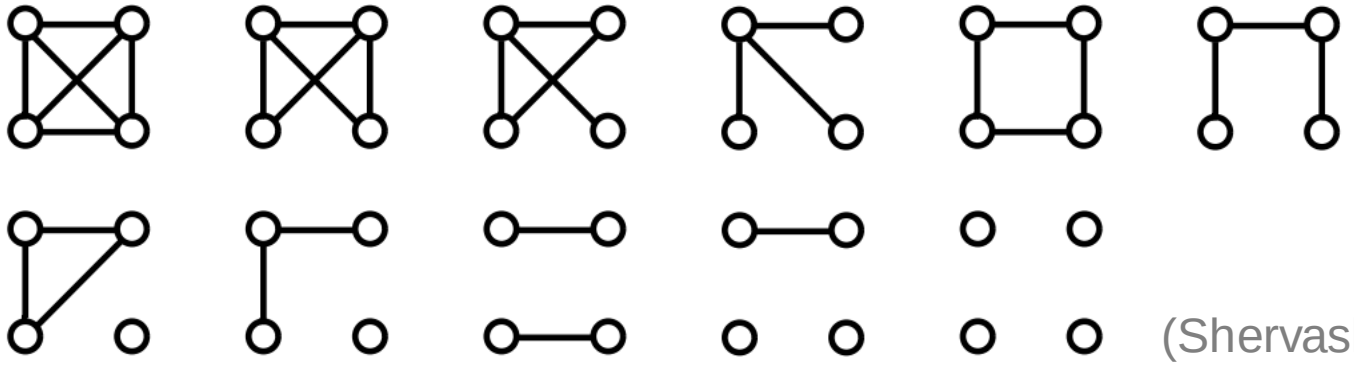
Graphlet Features (2)

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.*, 2011)

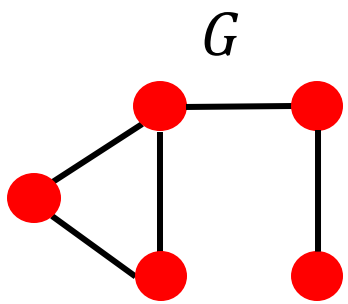
Graphlet Features (3)

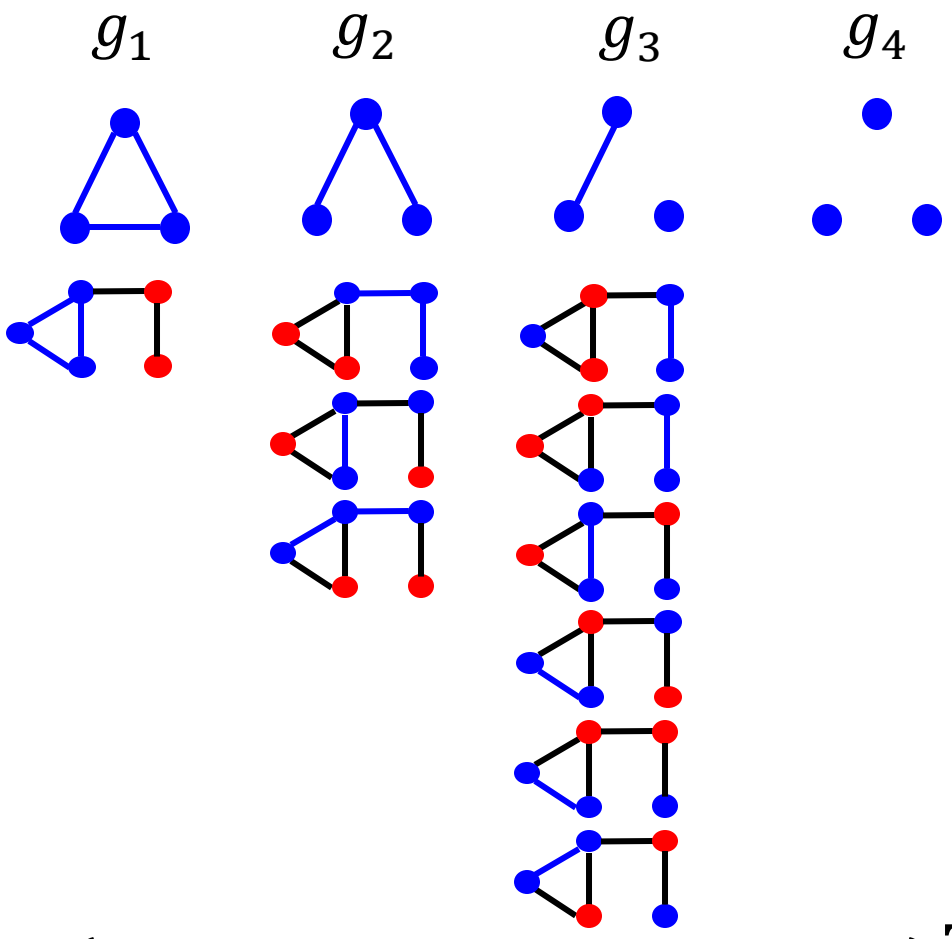
- Given graph G, and a graphlet list $\mathcal{G}_k = (g_1, g_2, \dots, g_{n_k})$, define the graphlet count vector $\boldsymbol{f}_G \in \mathbb{R}^{n_k}$ as

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

Graphlet Features (4)

- Example for k = 3.





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

Graph Kernel (1)

- 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 (2)

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**:

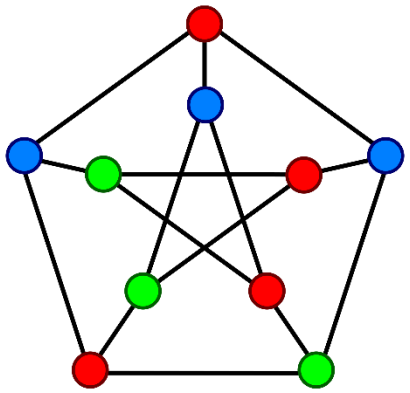
Color refinement

Color Refinement: Overview

- Coloring is an assignment of labels (called "**colors**") to elements (e.g. nodes) of a graph.
- **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 (1)

Example of color refinement given two graphs

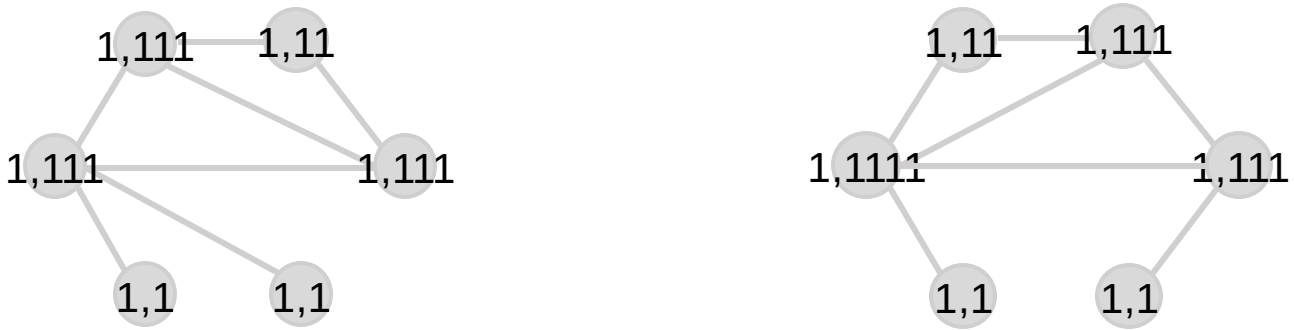
- Assign initial colors

- 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

- Hash aggregated colors

- Aggregated neighboring colors

Color Refinement (4)

Example of color refinement given two graphs

- Aggregated neighboring colors

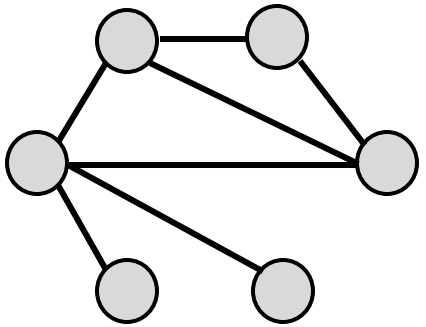
- Hash aggregated colors

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.

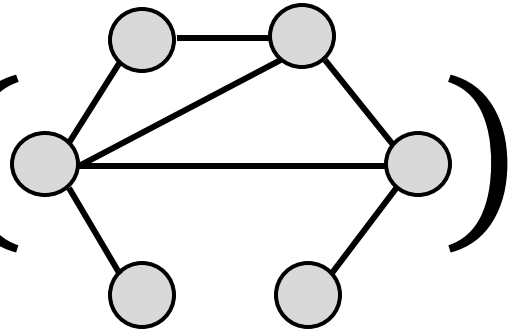
$\phi($  $)$

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

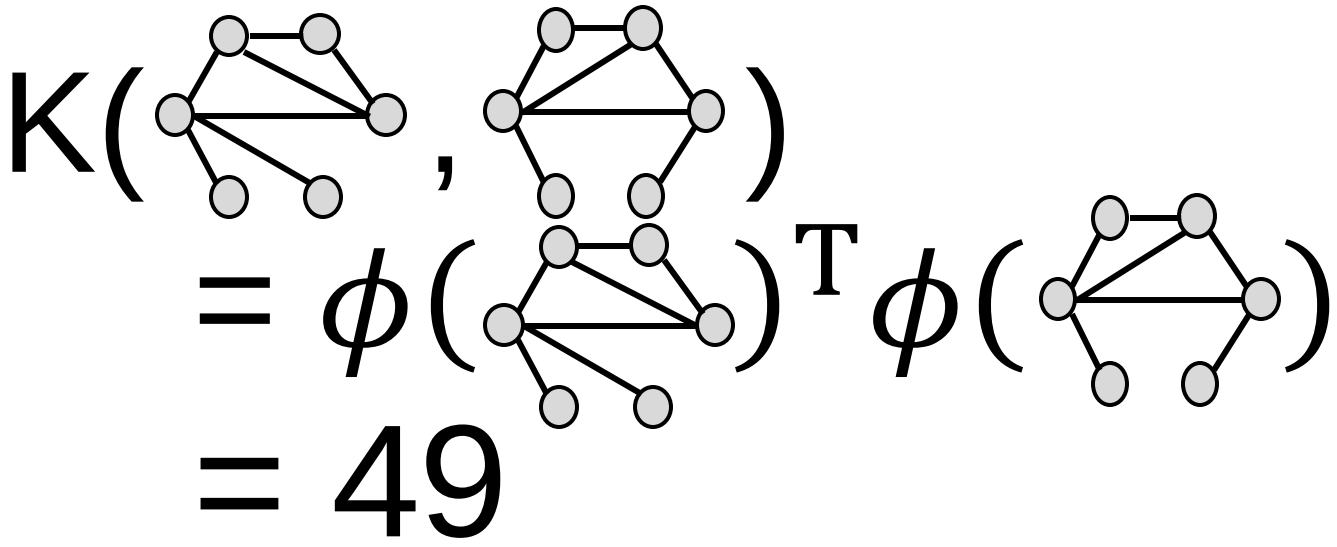
$\phi($  $)$

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 (1)

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



Weisfeiler-Lehman Kernel (2)

- 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

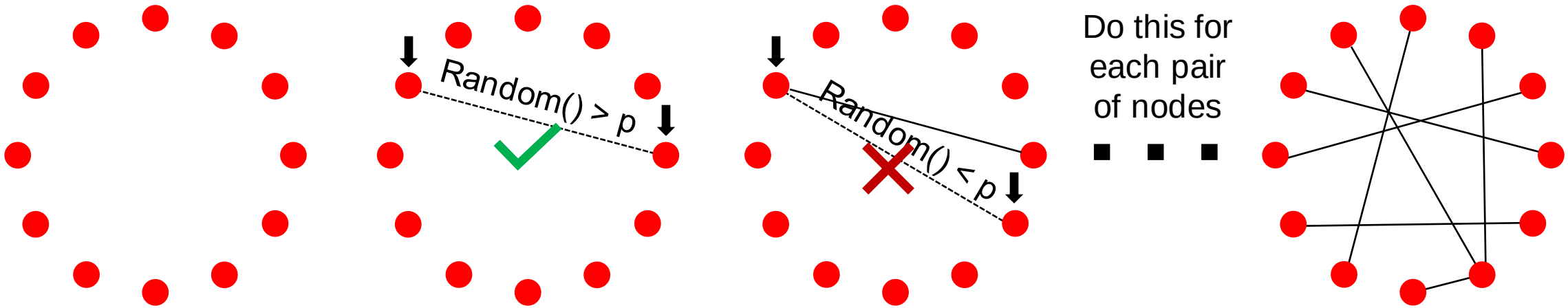
- **Graphlet Kernel**
  - Graph is represented as **Bag-of-graphlets**
  - **Computationally expensive**
- **Weisfeiler-Lehman Kernel**
  - Apply K-step color refinement algorithm to enrich node colors
    - Different colors capture different K-hop neighborhood structures
  - Graph is represented as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

Task: Graph Generation

- **Goal**: generate graphs that fit some properties of the real graphs.
- **Traditional methods**: generate by randomness based on some assumptions on the graph's formulation process.
  - Erdos-Renyi model
  - Watts-Strogatz model (small world model)
  - Barabási-Albert model
  - Kronecker model
  - …
- **Deep methods**: …

Erdos-Renyi Model

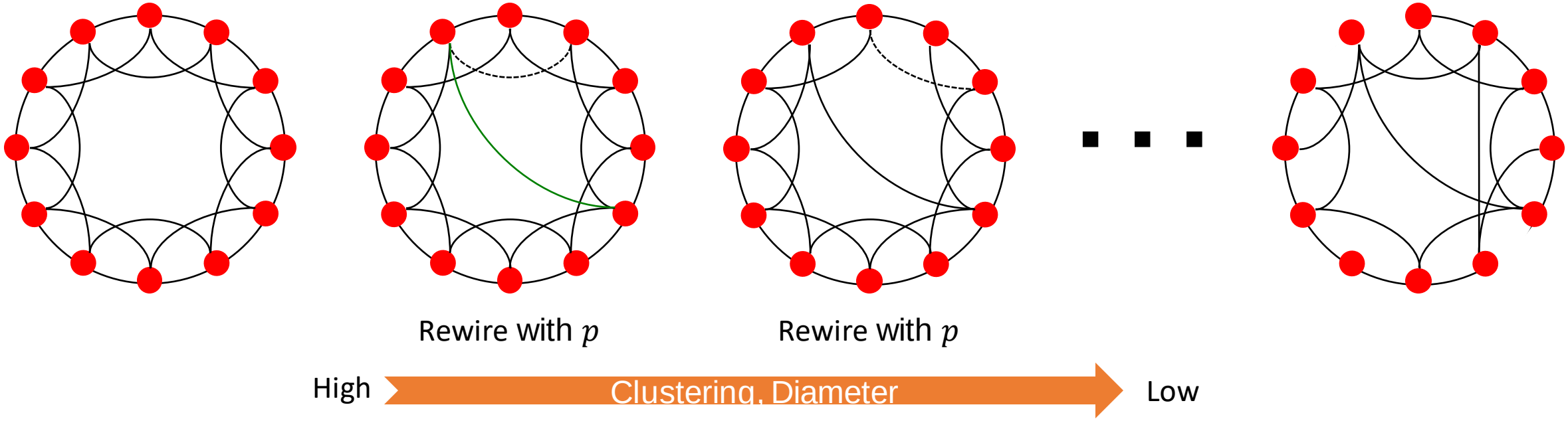
- Two variants
  - $G(n, p)$: a graph of n nodes, and each pair of nodes is connected with probability p.
  - $G(n, m)$: a graph of n nodes, and m edges are randomly placed.
- Example $G(n, p)$:



- Does it fit real-world graph?

Watts-Strogatz Model (Small World Model)

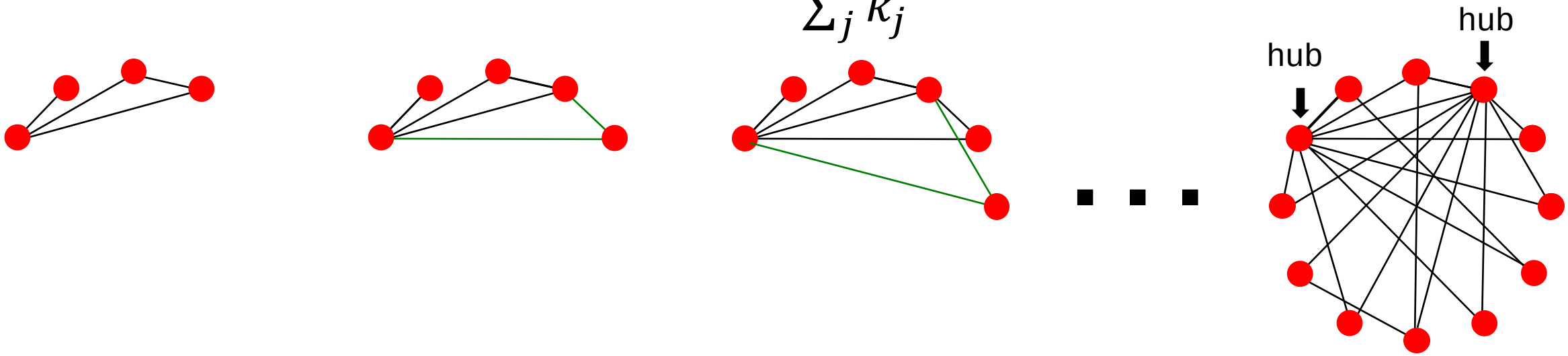
- Small world phenomenon
- The average clustering coefficient of real networks is much higher than expected for a random network of similar n and m.



Barabási-Albert Model

- **Growth**: the number of nodes are increased, not fixed at the beginning.
- **Preferential attachment**: a new node prefers to link the high-degree node.
- Probability of connecting to the node i:

$$\pi(i) = \frac{k_i}{\sum_j k_j}$$

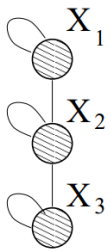


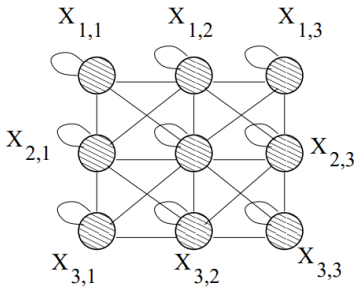
Kronecker Graph Model

- Recursively construct a graph by Kronecker product ($\otimes$).

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \cdots & a_{1m}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n1}\mathbf{B} & \cdots & a_{nm}\mathbf{B} \end{pmatrix}$$

Graph





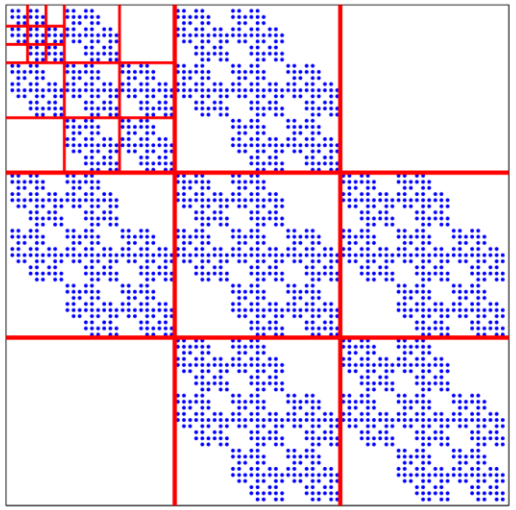
Adjacency Matrix

1	1	0
1	1	1
0	1	1

K_1

K_1	K_1	0
K_1	K_1	K_1
0	K_1	K_1

$K_2 = K_1 \otimes K_1$



$K_n = K_1 \otimes K_1 \otimes \cdots \otimes K_1$

(Leskovec et al., 2010)

Today's Summary

- **Traditional ML Pipeline**
  - Hand-crafted feature + ML model
- **Hand-crafted features for graph data**
  - **Node-level:**
    - Node degree, centrality, clustering coefficient, graphlets
  - **Link-level:**
    - Distance-based feature
    - local/global neighborhood overlap
  - **Graph-level:**
    - Graphlet kernel, WL kernel

  In traditional ML, node-level features can only be used for node-level tasks (we will alleviate in the future lectures)
- **Graph Generation**