

# Online Social Networks and Media

Graph ML

# Graph Machine Learning

## Outline

Part I: Introduction, Traditional ML

Part II: Graph Embeddings

Part III: GNNs

Part IV (if time permits): Knowledge Graphs

Slides used based on:

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>

# Part I:

Types of ML Tasks

Traditional ML

Feature Engineering

# Tools



## PyG (PyTorch Geometric):

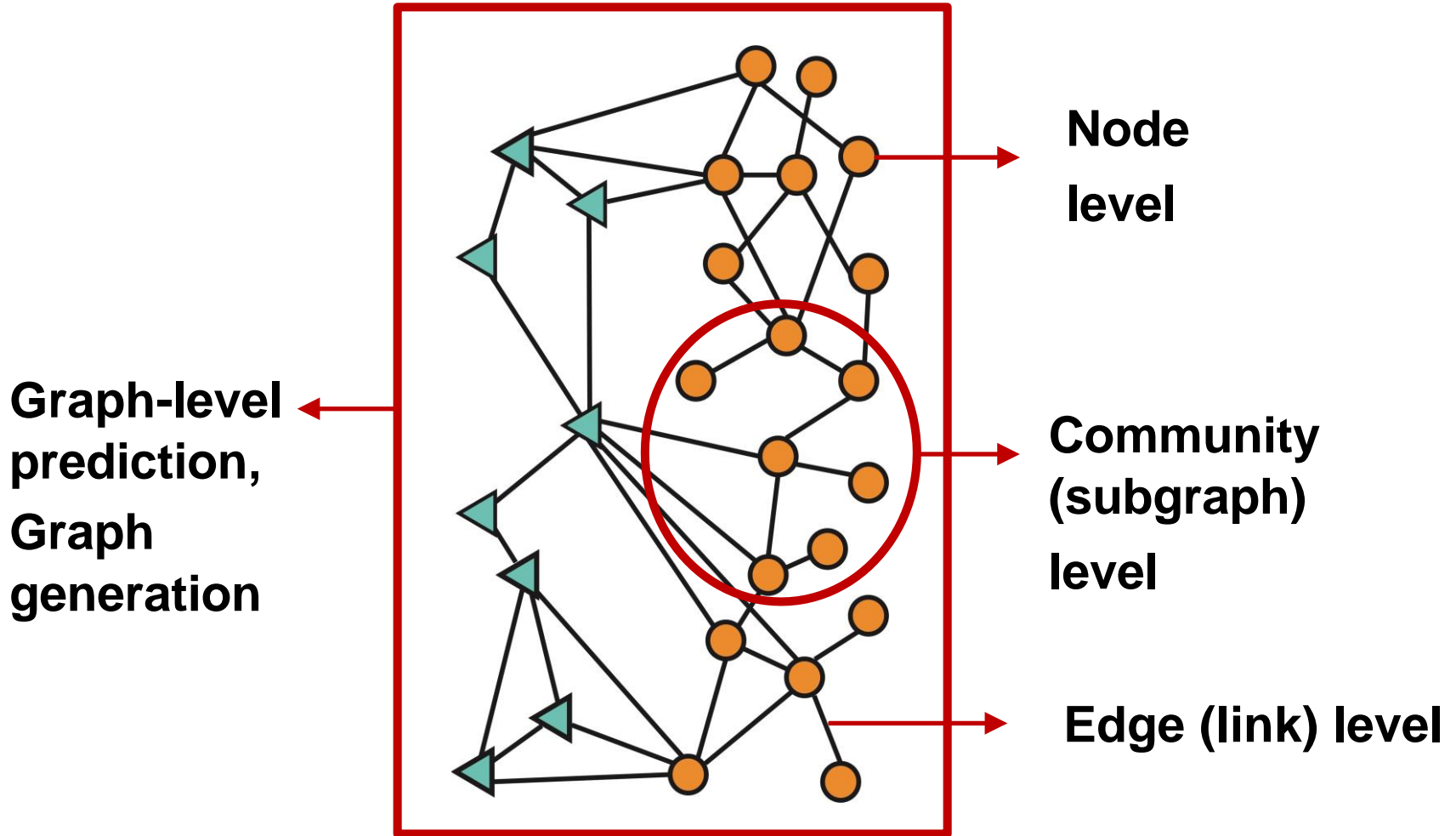
- The ultimate library for Graph Neural Networks

## **We further recommend:**

- GraphGym: Platform for designing Graph Neural Networks.
  - Modularized GNN implementation, simple hyperparameter tuning, flexible user customization
- **Other network analytics tools:** SNAP.PY, NetworkX

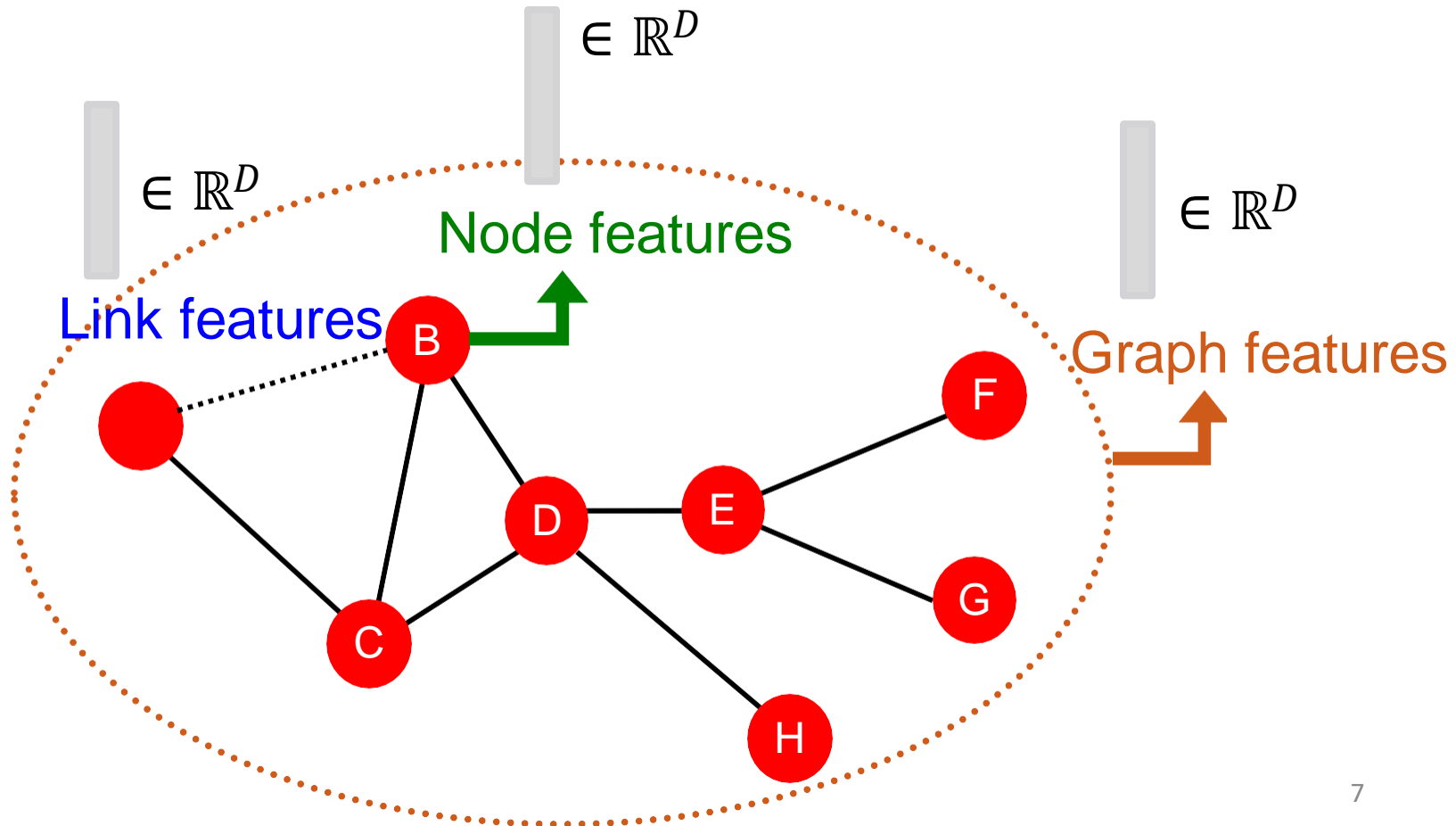
# Types of ML tasks in graphs

# Types of ML tasks in graphs



# Traditional ML Pipeline

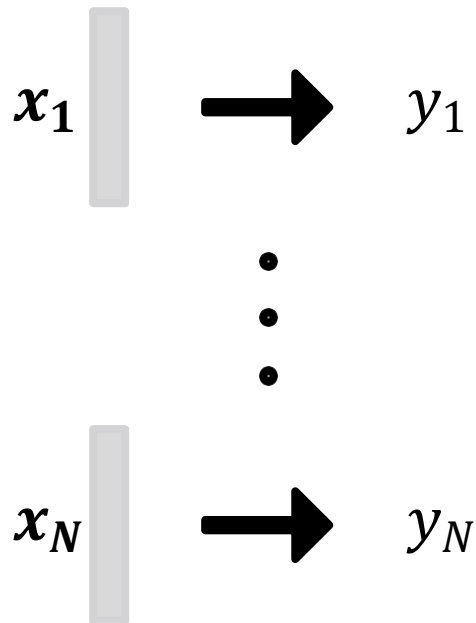
- Design features for nodes/links/graphs
- Obtain features for all training data



# Traditional ML Pipeline

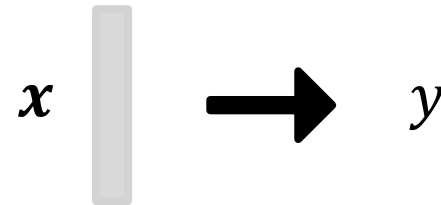
- **Train an ML model:**

- Random forest
- SVM
- Neural network, etc.



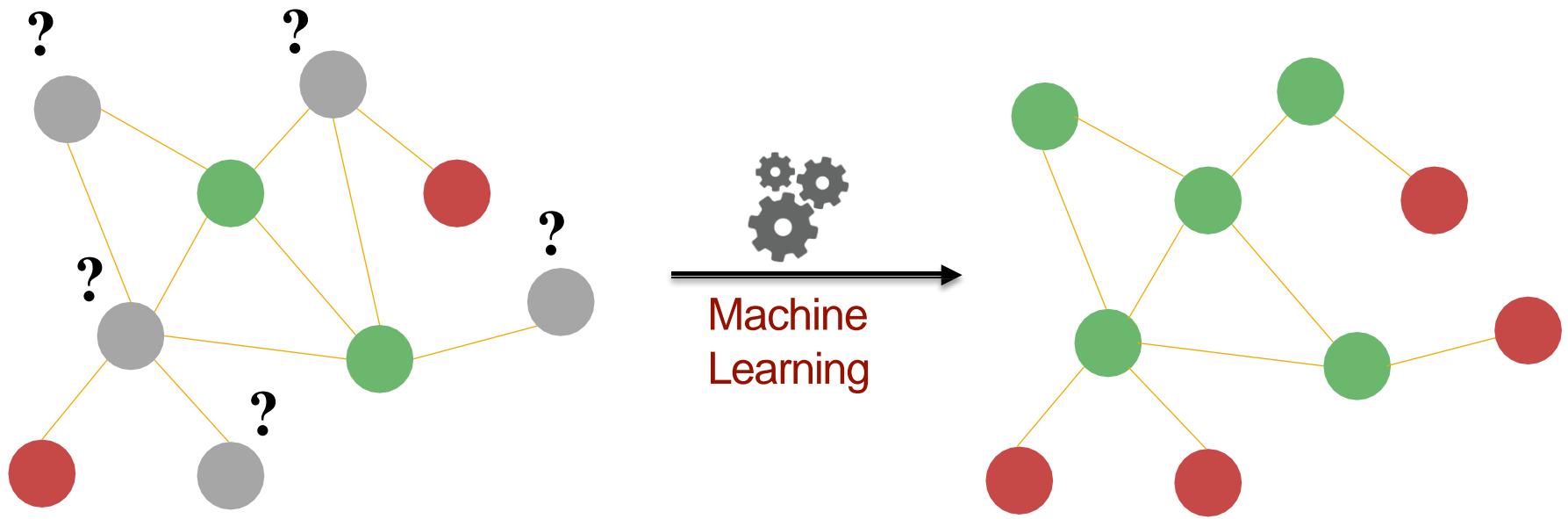
- **Apply the model:**

- Given a new node/link/graph, obtain its features and make a prediction





# Node Level Tasks (example)



Node classification

# Feature Design

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

**Goal:** Make predictions for a set of objects

**Design choices:**

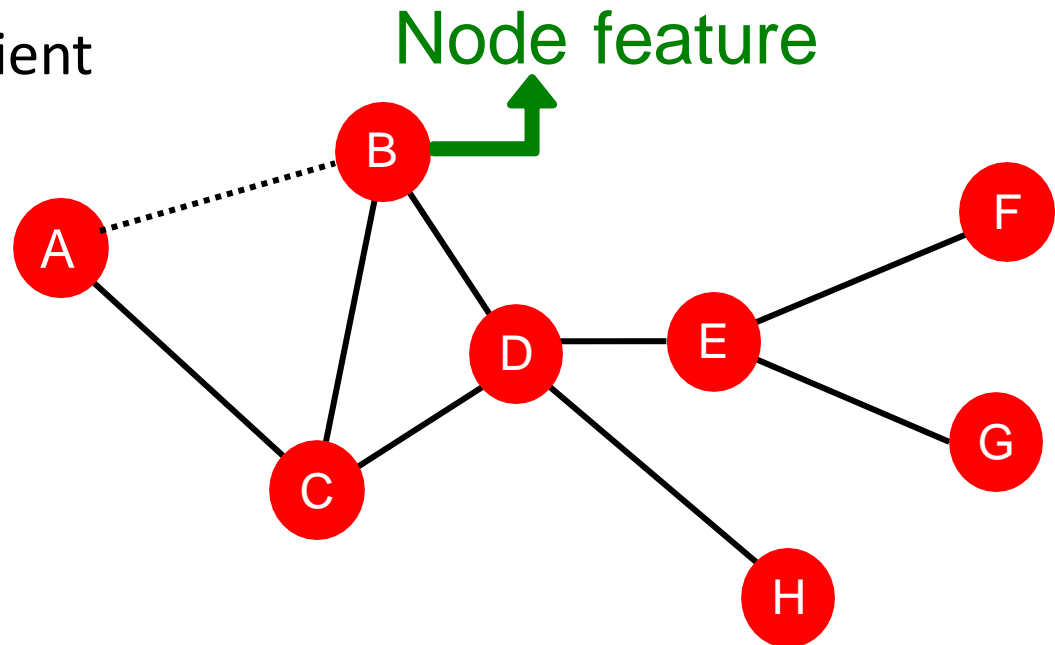
- **Features:**  $d$ -dimensional vectors
- **Objects:** Nodes, edges, sets of nodes, entire graphs
- **Objective function:**
  - What task are we aiming to solve?

# **NODE LEVEL FEATURES AND TASKS**

# Node Level Features

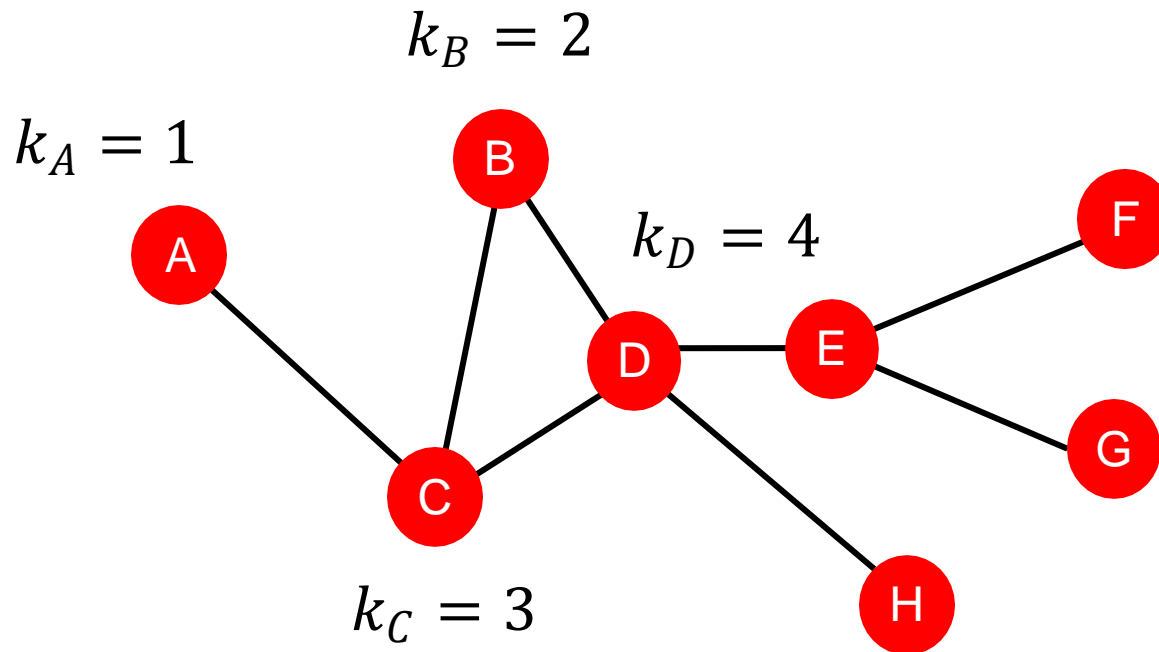
Goal: Characterize the structure and position of a node in the network:

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



# 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 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 (Pagerank) centrality
  - Betweenness centrality
  - Closeness centrality
  - and many others...

# Pagerank 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**:

$$p(v) = \sum_{u \rightarrow v} \frac{p(u)}{\text{OutDegree}(u)}$$

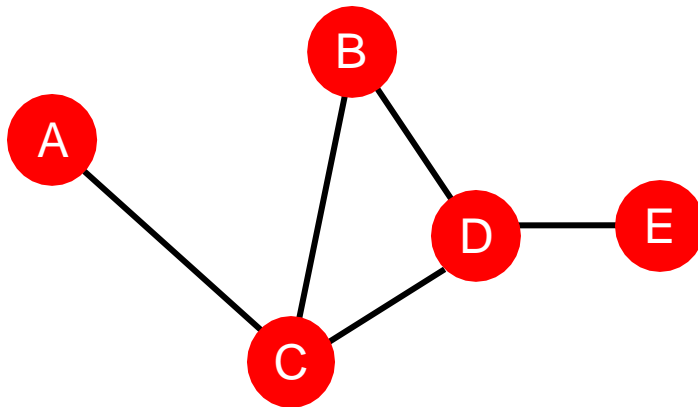


# Betweenness centrality

- 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:



$$c_A = c_B = c_E = 0$$

$$c_C = 3$$

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

$$c_D = 3$$

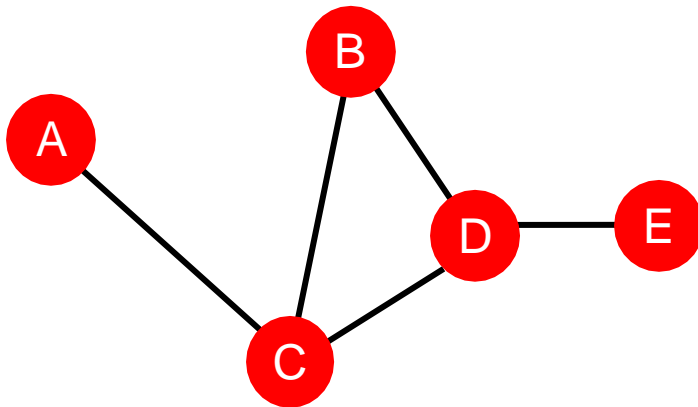
(A-C-D-E, B-D-E, C-D-E)

# 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)

# Clustering coefficient

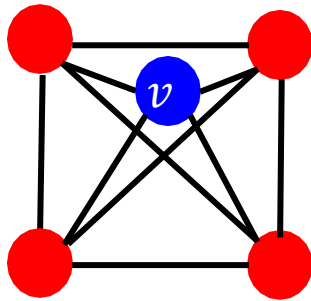
- Measures how connected the neighboring nodes of  $v$  are:

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

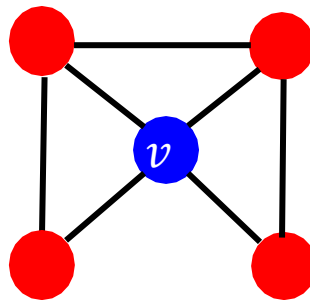
#(node pairs among  $k_v$  neighboring nodes)

In our examples below the denominator is 6 (4 choose 2).

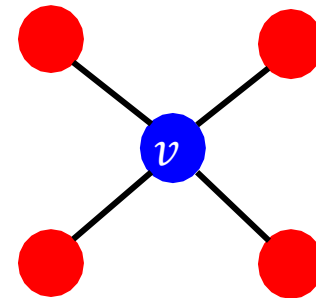
Examples:



$$e_v = 1$$



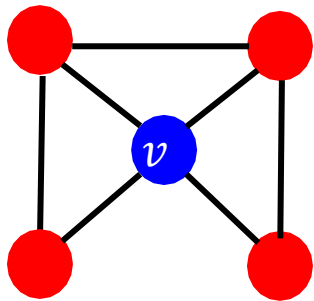
$$e_v = 0.5$$



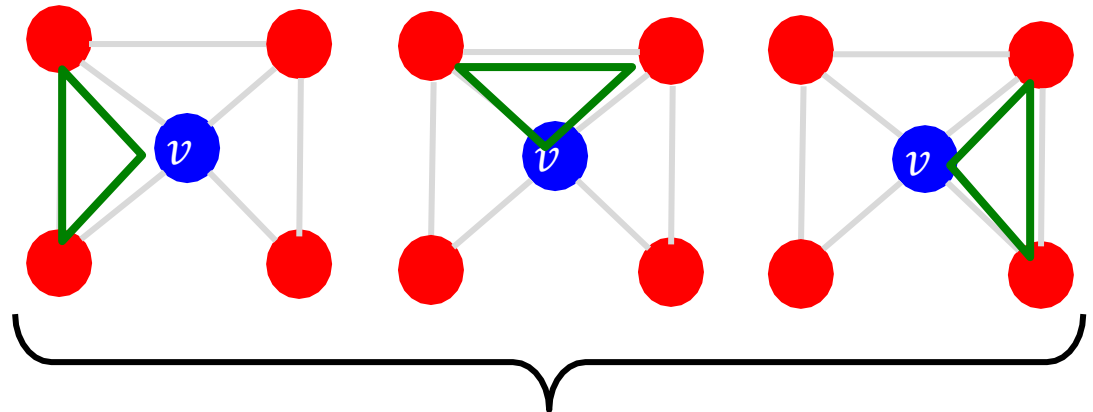
$$e_v = 0$$

# Graphlets

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



$$e_v = 0.5$$

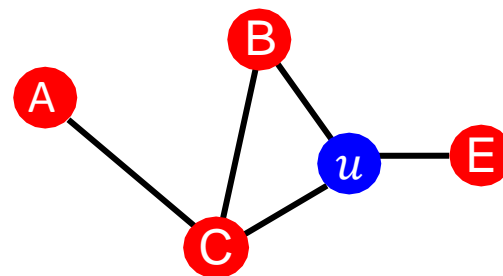


3 triangles (out of 6 node triplets)

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

# Graphlets

- **Goal:** Describe network structure around node  $u$ 
  - **Graphlets** are small subgraphs that describe the structure of node  $u$ 's network neighborhood



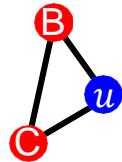
Analogy:

- **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

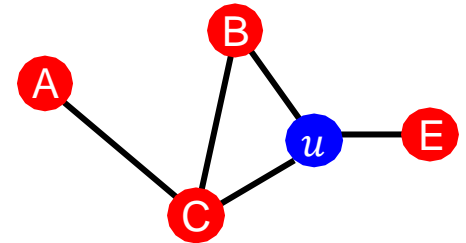
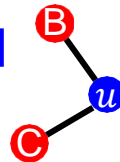
# Graphlets

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

Induced subgraph:



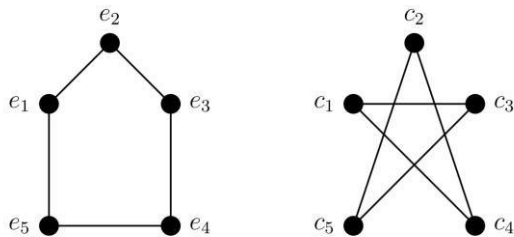
Not induced subgraph:



# Graphlets

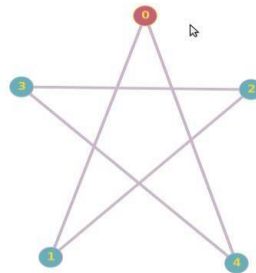
## ■ Def: Graph Isomorphism

- Two graphs which contain the same number of nodes connected in the same way are said to be isomorphic.  
(one-to-one mapping of their nodes)

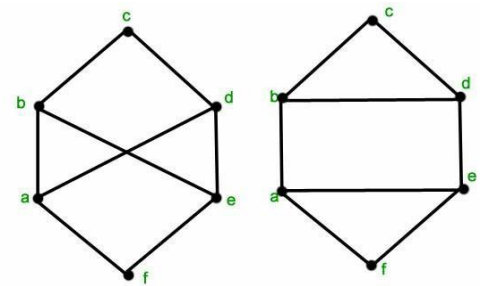


Isomorphic

Node mapping:  $(e_2, c_2)$ ,  $(e_1, c_5)$ ,  
 $(e_3, c_4)$ ,  $(e_5, c_3)$ ,  $(e_4, c_1)$



Source: Mathoverflow



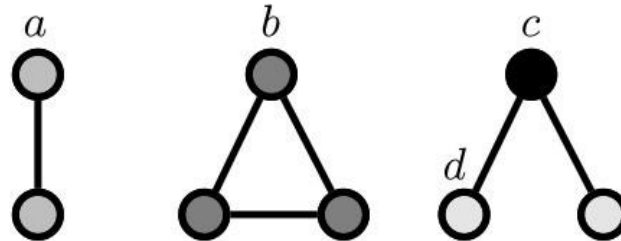
Non-Isomorphic

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

# Graphlets

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

All possible graphlets on up to 3 nodes



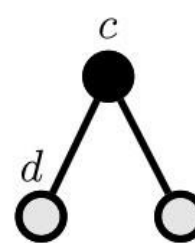
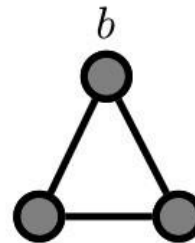
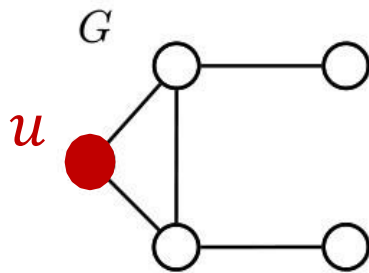
**Graphlet Degree Vector (GDV):** A count vector of graphlets rooted at a given node.



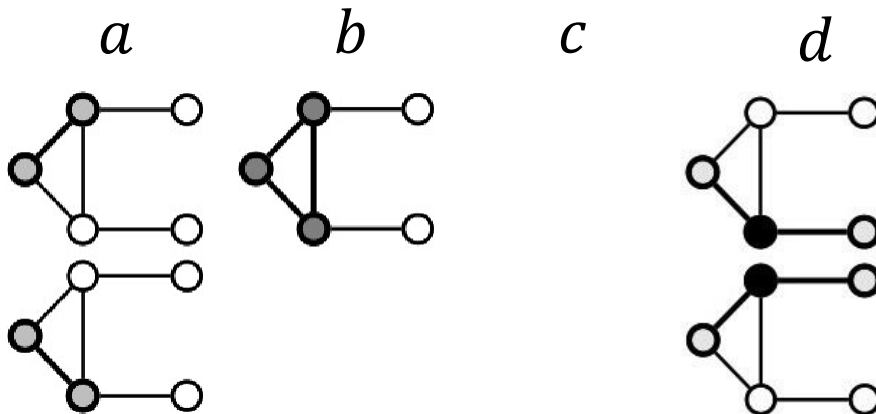
# Graphlets

Example:

All possible graphlets on up to 3 nodes

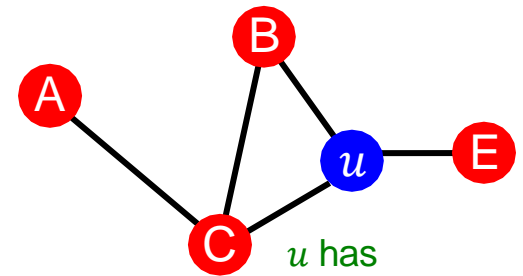


Graphlet instances of node  $u$ :



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

# Graphlets



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

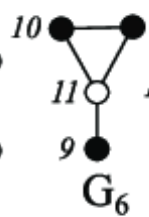
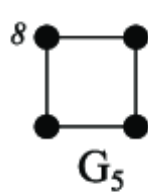
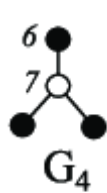
2-node graphlet



3-node graphlets



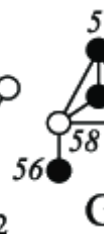
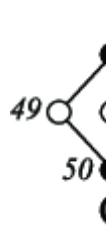
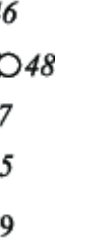
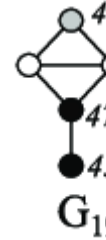
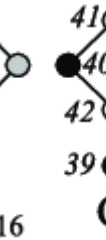
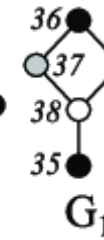
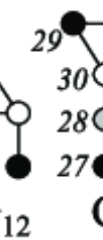
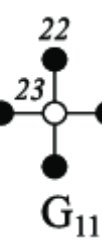
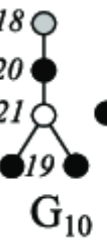
4-node graphlets



Graphlet id (Root/ "position" of node *u*)

15

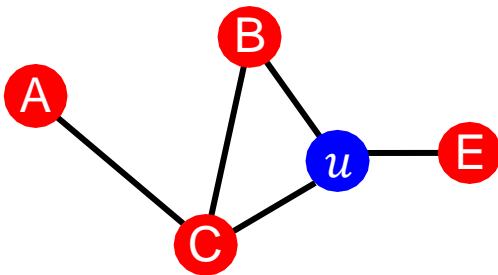
5-node graphlets



There are 73 different graphlets on up to 5 nodes

# 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



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

# Node Level Features

- **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

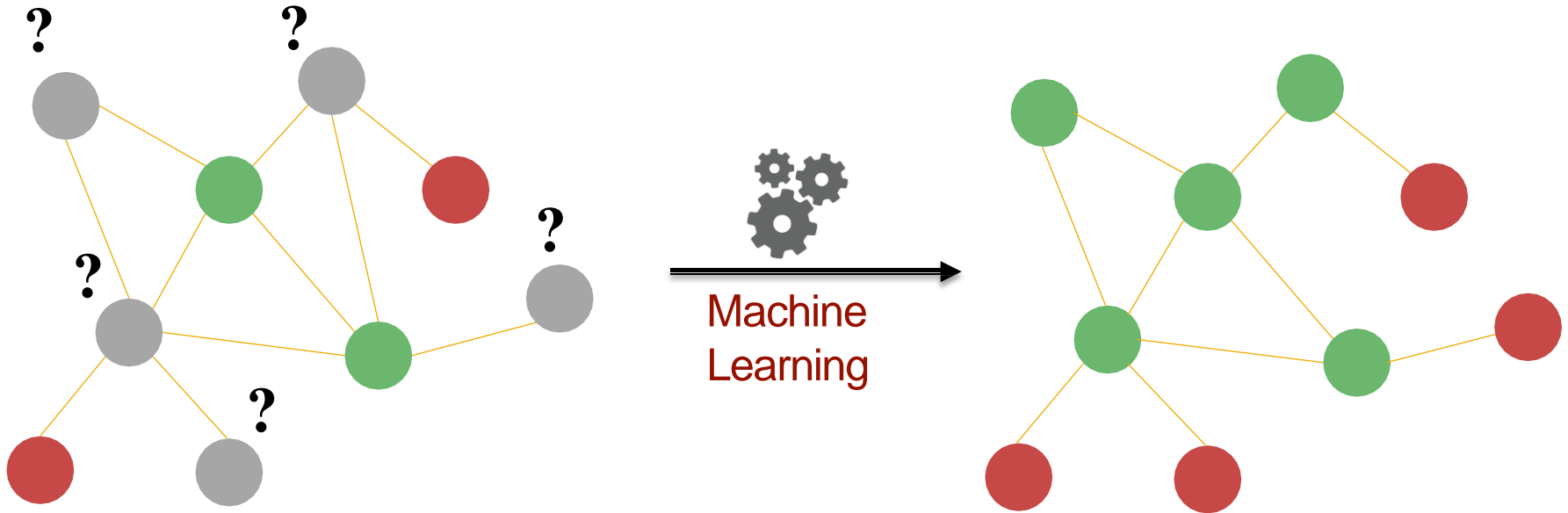
# Node Level Features

- **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 Features

- **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.

# Node Level Tasks

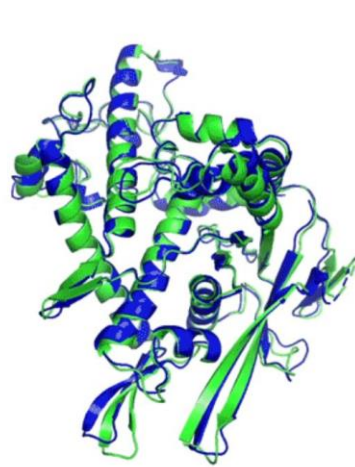


Node classification

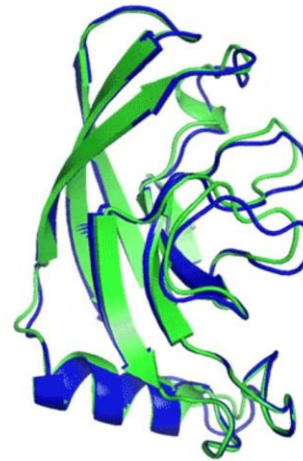
# Protein Folding

Computationally predict the 3D structure of a protein based solely on its amino acid sequence:

For each node predict its 3D coordinates



T1037 / 6vr4  
90.7 GDT  
(RNA polymerase domain)



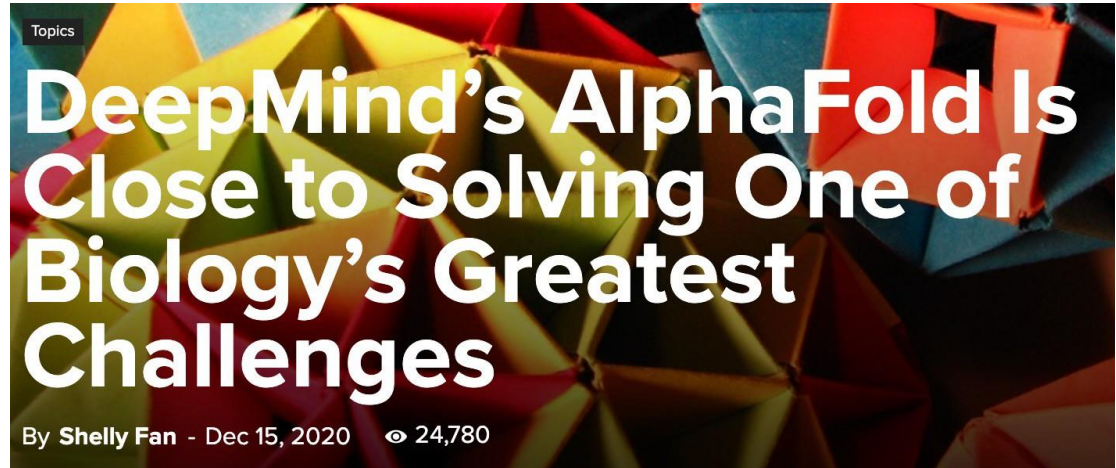
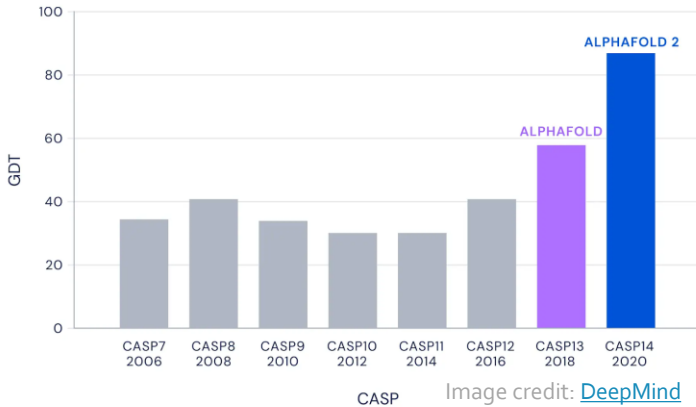
T1049 / 6y4f  
93.3 GDT  
(adhesin tip)

- Experimental result
- Computational prediction



# AlphaFold: Impact

Median Free-Modelling Accuracy



## AlphaFold's AI could change the world of biological science as we know it

**DeepMind's latest AI breakthrough can accurately predict the way proteins fold**

**Has Artificial Intelligence 'Solved' Biology's Protein-Folding Problem?**

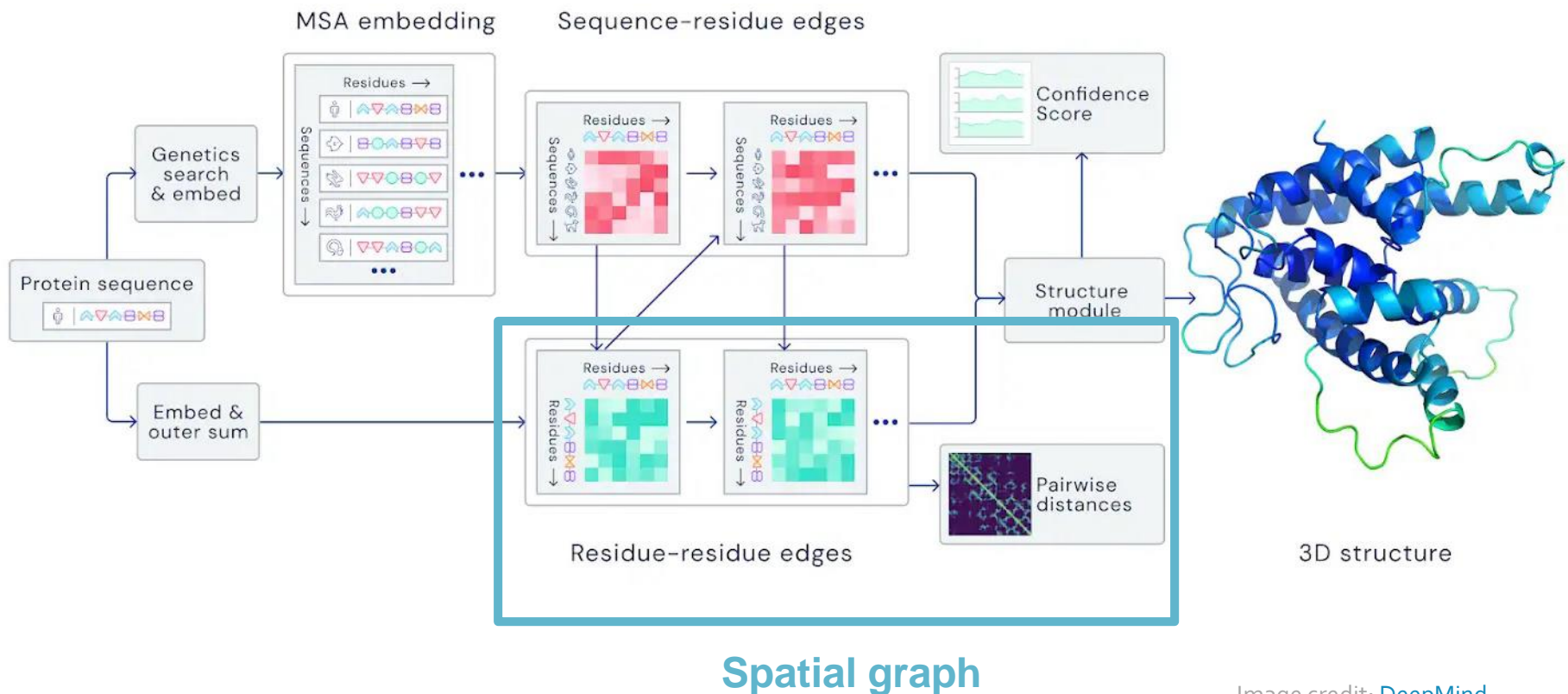
12-14-20

**DeepMind's latest AI breakthrough could turbocharge drug discovery**

# AlphaFold: Solving Protein Folding

## Key idea: “Spatial graph”

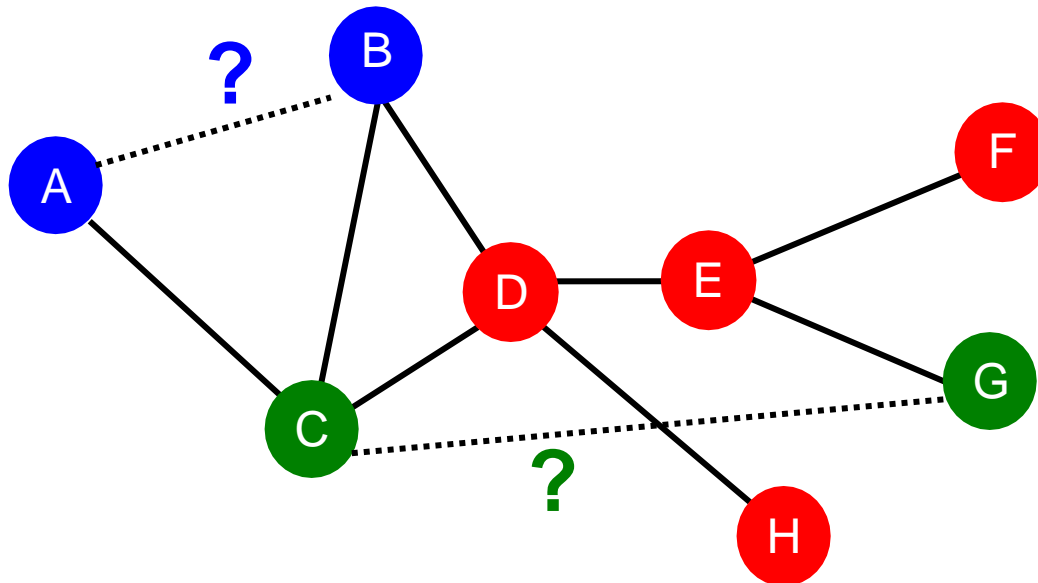
- **Nodes:** Amino acids in a protein sequence
- **Edges:** Proximity between amino acids (residues)



# LINK PREDICTION

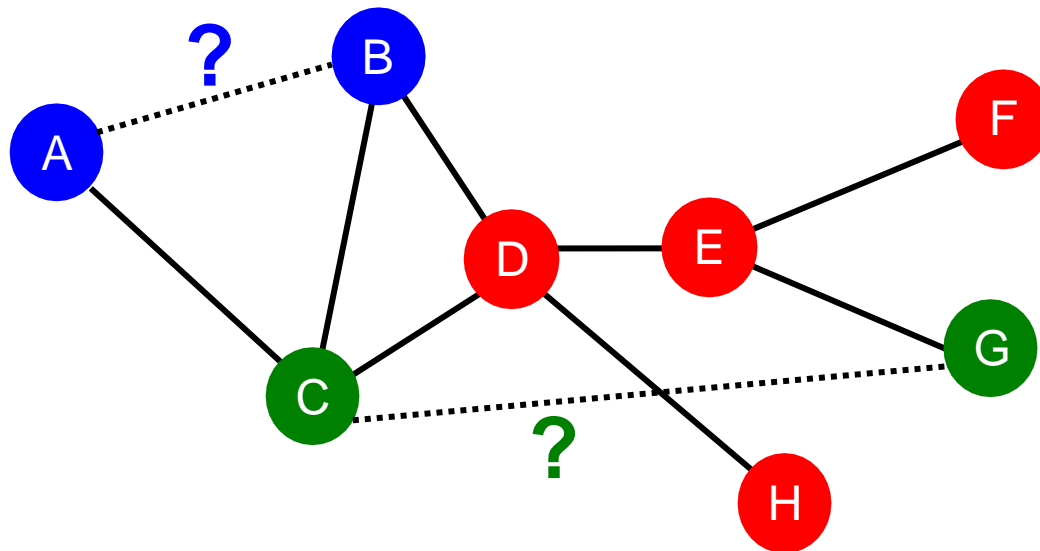
# Link Prediction

- The task is to predict **new links** based on the existing links.
- Two ways: (a) define a score for each pair of nodes, rank pairs, return top K ones, (b) build a classifier with input pair of nodes, output probability of existence



# Link Prediction

- The key is to design features for a **pair of nodes**.  
(for computing the score, as input to the classifier)
- First, score



# Link Prediction

## **(1) Links missing at random:**

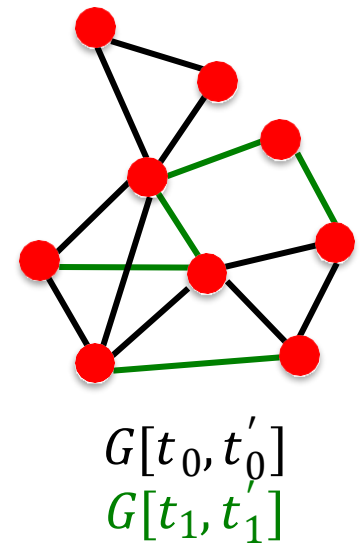
Missing/unknown, incomplete information

- Remove a random set of links and then aim to predict them

# Link Prediction

## (2) Temporal Links Prediction

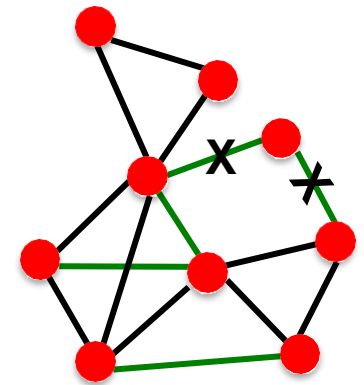
- Given  $G[t_0, t'_0]$  a graph defined by edges up to time  $t'_0$ , **output a ranked list  $L$**  of edges (not in  $G[t_0, t'_0]$ ) that are predicted to appear in time  $G[t_1, t'_1]$



# Score-based Link Prediction

## Methodology:

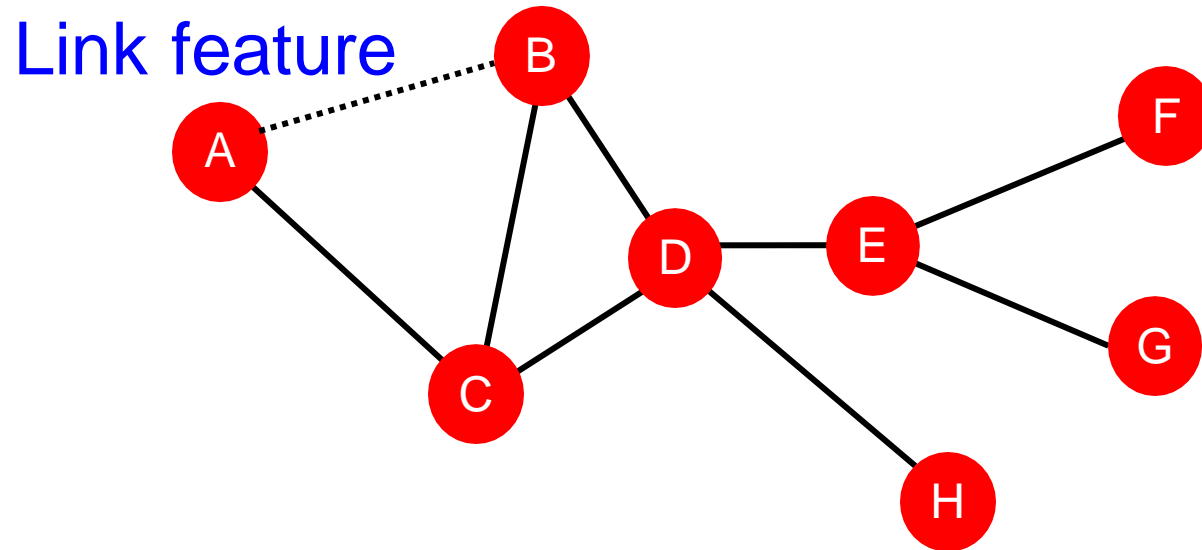
- For each pair of nodes  $(x,y)$  compute score  $c(x,y)$ 
    - For example,  $c(x,y)$  could be the # of common neighbors of  $x$  and  $y$
  - **Sort** pairs  $(x,y)$  by the decreasing score  $c(x,y)$
  - **Predict top  $n$  pairs as new links**
- Evaluation:
- $n = |E_{new}|$ : # new edges that appear during the test period  $[t_1, t']$
  - Take top  $n$  elements of  $L$  and count correct edges





# Link Level Features

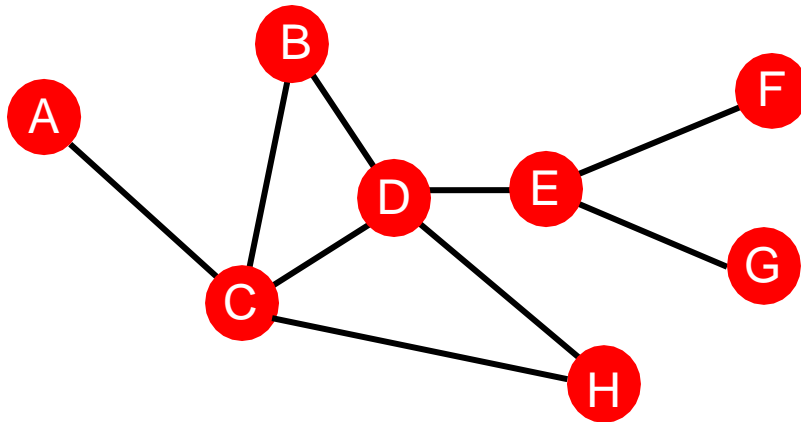
- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap



# 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 Features

**Local Neighborhood Features:** Captures # neighboring nodes shared between two nodes

**Common neighbors:**

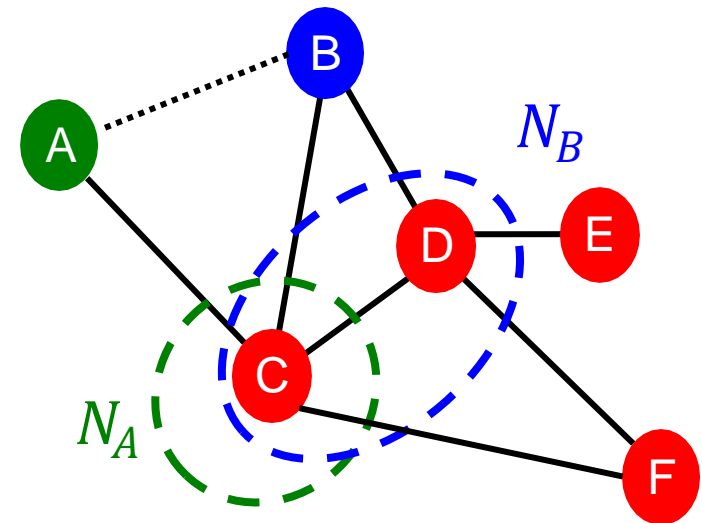
$$|N(v_1 \cap v_2)|$$

**Jaccard coefficient:**

$$\frac{|N(v_1 \cap v_2)|}{|N(v_1 \cup v_2)|}$$

**Example:**

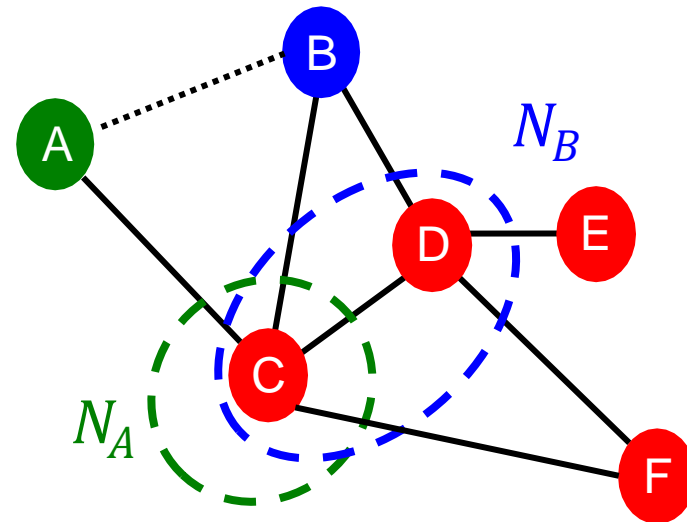
$$c(A, B)$$



# Local Neighborhood Overlap Features

Adamic-Adar index:

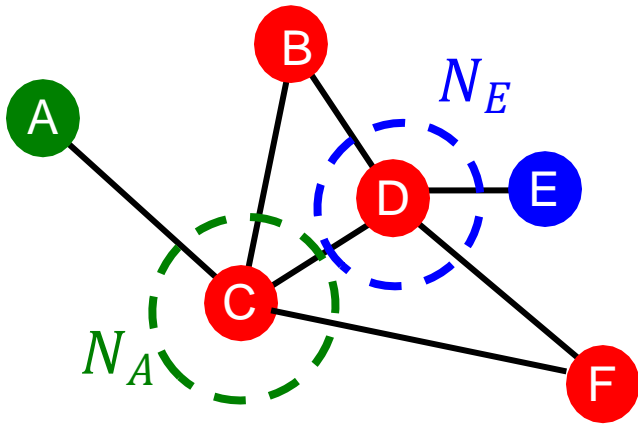
$$\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$$



# Global Neighborhood Overlap Features

## 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 connect in the future.

**Global neighborhood overlap** metrics resolve the limitation by considering the entire graph.

# Global Neighborhood Overlap Features

**Katz index:** counts the number of walks of all lengths between a given pair of nodes.

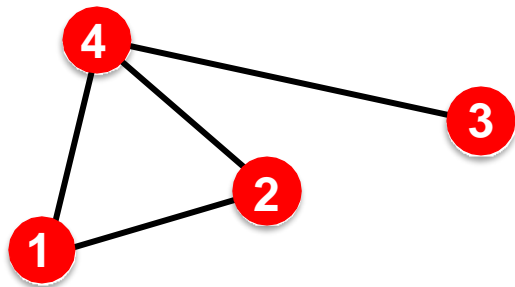
*How to compute #walks between two nodes?*

- Use powers of the adjacency matrix!

# Global Neighborhood Overlap Features

## Computing #walks between two nodes

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



$$P_{12}^{(1)} = A_{12}$$
$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

# Global Neighborhood Overlap Features

- 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

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

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



# Global Neighborhood Overlap Features

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 Features

**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:

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

by geometric series of matrices

# Link Level Features

## 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.

# Classification for Link Prediction

Predict link  $e = (v, u)$

## **Input**

Features describing  $v$  and  $u$

## **Output**

Prediction

positive class: link

negative class: no-link

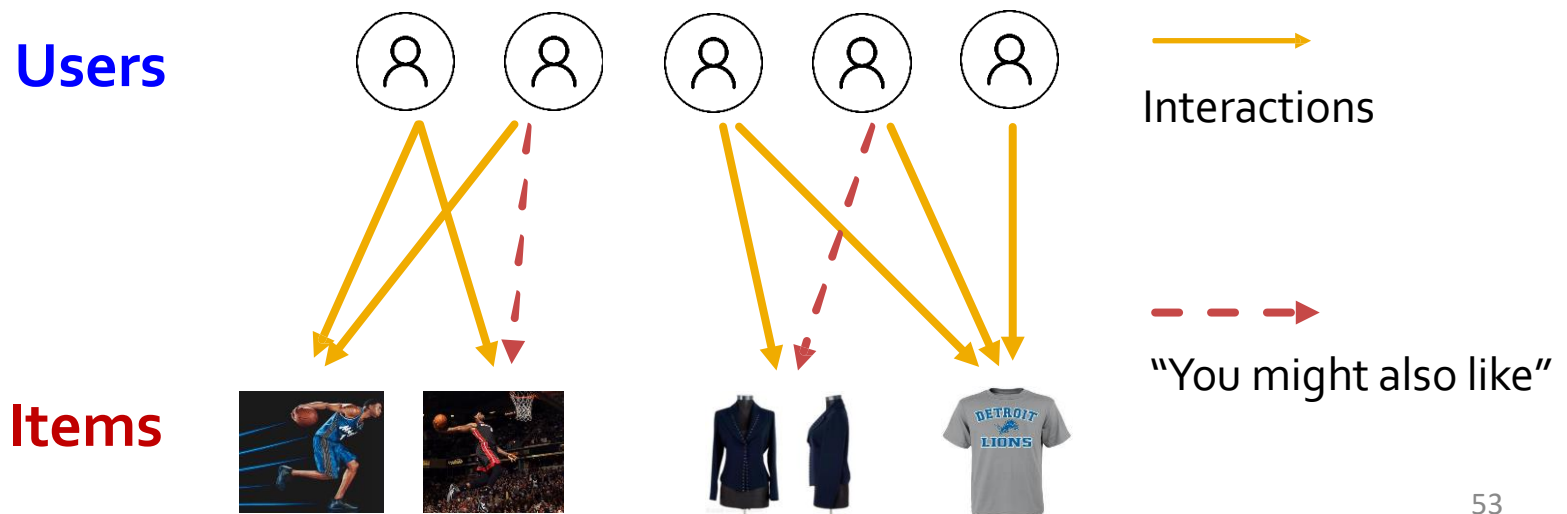
# Example: Recommender Systems

Users interact with items

Watch movies, buy merchandise, listen to music

- **Nodes:** Users and items
- **Edges:** User-item interactions

Goal: Recommend items users might like

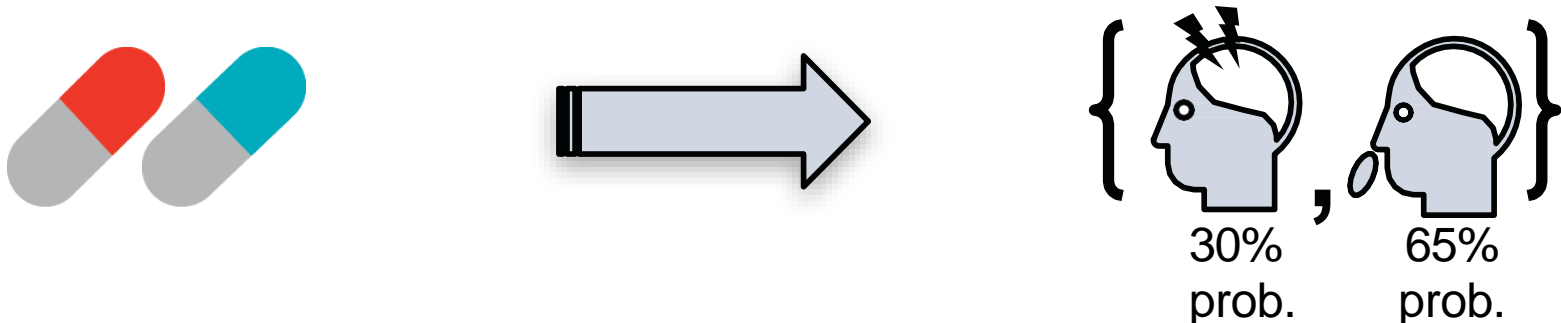


# Example: Drug Side Effects

Many patients **take multiple drugs** to treat **complex or co-existing diseases**:

- 46% of people ages 70-79 take more than 5 drugs
- Many patients take more than 20 drugs to treat heart disease, depression, insomnia, etc.

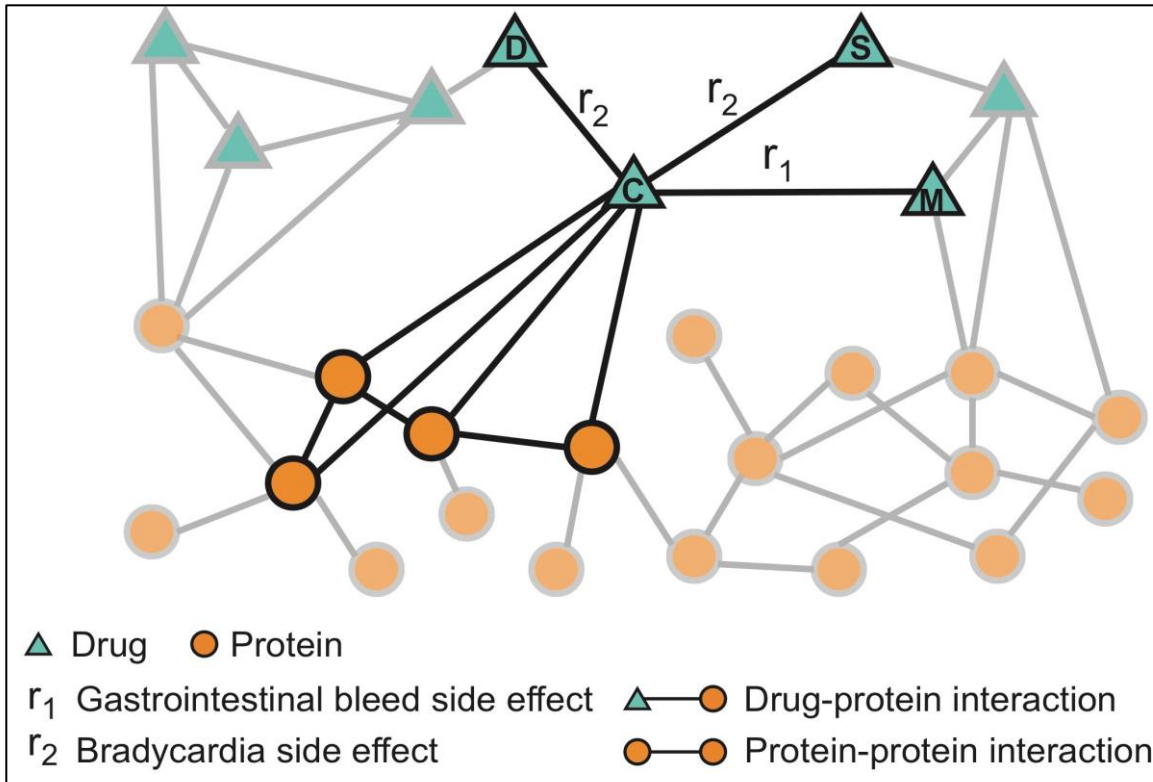
Task: Given a pair of drugs predict adverse side effects



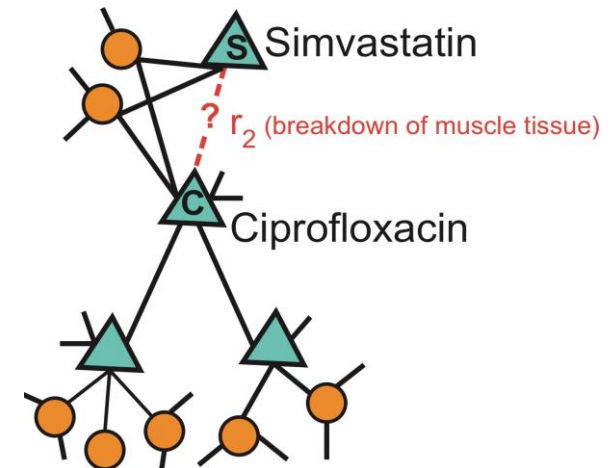
# Example: Drug Side Effects

**Nodes:** Drugs & Proteins

**Edges:** Interactions



**Query:** How likely will Simvastatin and Ciprofloxacin, when taken together, break down muscle tissue?



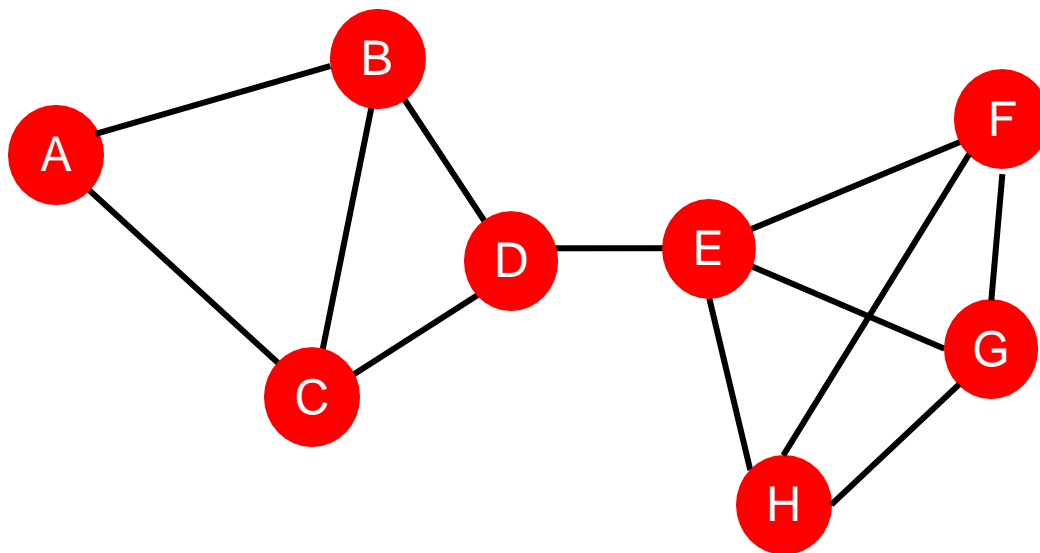
# GRAPH LEVEL FEATURES AND TASKS



# Graph Level Features

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

For example:



# Graph Kernels

**Graph Kernels:** Measure *similarity* between two graphs

- Kernel  $K(G, G') \in \mathbb{R}$  measures similarity
- Kernel matrix  $\mathbf{K} = \left( K(G, G') \right)_{G, G'}$  must always be positive semidefinite (i.e., has positive eigenvalues)
- There exists a feature representation  $\phi(\cdot)$  such that  $K(G, G') = \phi(G)^T \phi(G')$
- Once the kernel is defined, off-the-shelf ML model, such as **kernel SVM**, can be used to make predictions.

# Graph Kernels

**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.

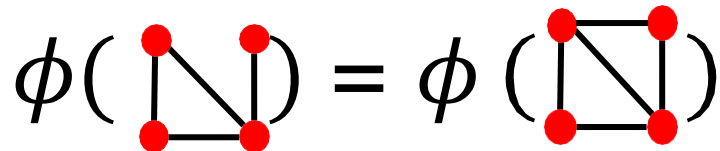
2 Shervashidze, Nino, et al. "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research* 12.9 (2011).

# Graph Kernels

**Goal:** Design graph feature vector  $\phi(G)$

**Key idea:** Bag-of-Words (BoW) for a graph

- 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

$$\phi\left(\begin{array}{c} \bullet \\ | \\ \bullet \end{array} \begin{array}{c} \bullet \\ | \\ \bullet \end{array}\right) = \phi\left(\begin{array}{c} \bullet \\ | \\ \bullet \end{array} \begin{array}{c} \bullet \\ | \\ \bullet \end{array}\right)$$


# Graph Kernels

What if we use Bag of node degrees?

Deg1: ● Deg2: ● Deg3: ●

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



Obtains different features for different graphs!

$$\phi(\text{Graph 3}) = \text{count}(\text{Graph 4}) = [0, 2, 2]$$

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

# Graphlet Features

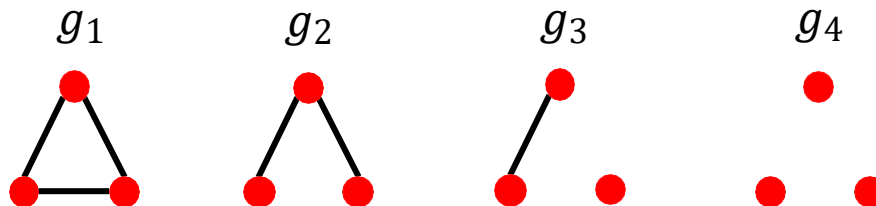
**Key idea:** Count the number of *different graphlets* in a graph.

- **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**.

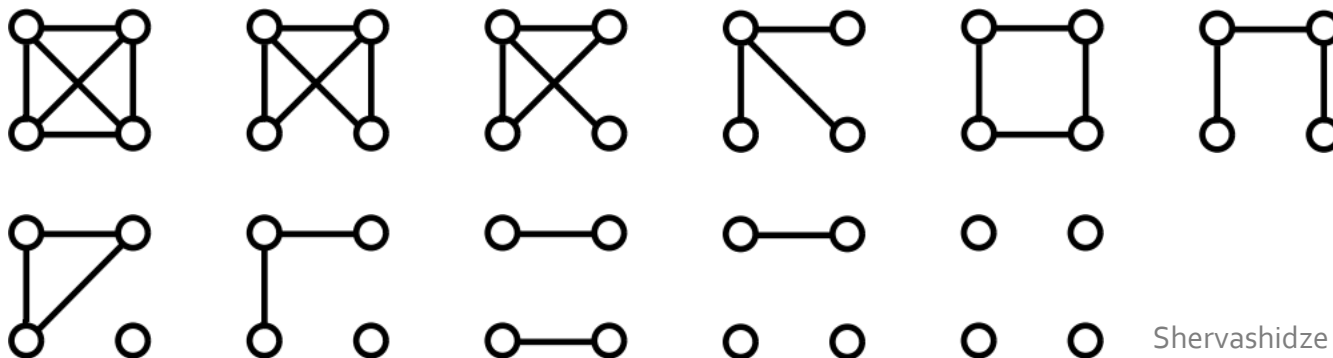
# Graphlet Features

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.



# Graphlet Features

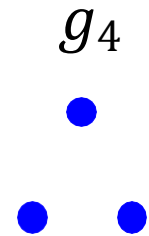
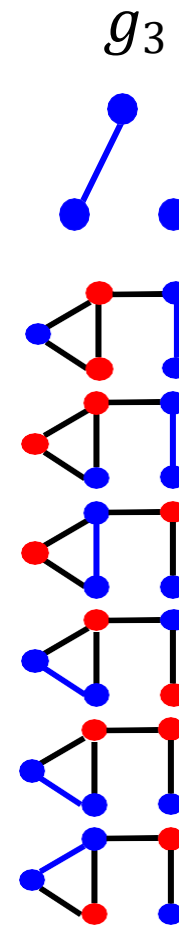
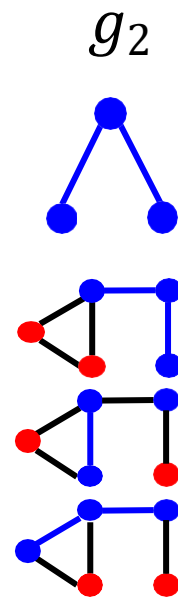
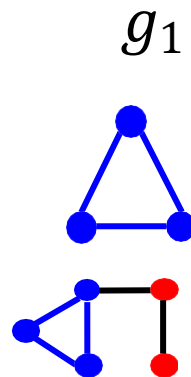
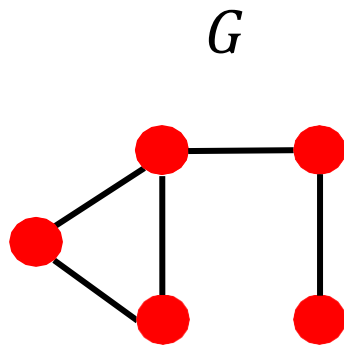
Given graph  $G$ , and a graphlet list  $\mathcal{G}_k = (g_1, g_2, \dots, g_{n_k})$ , define the graphlet count vector  $f_G \in \mathbb{R}^{n_k}$  as

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



# Graphlet Features

Example for  $k = 3$ .



$$f_G = (1, 3, 6, 0)^T$$

# 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

Limitation: 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 the node degree of a graph 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

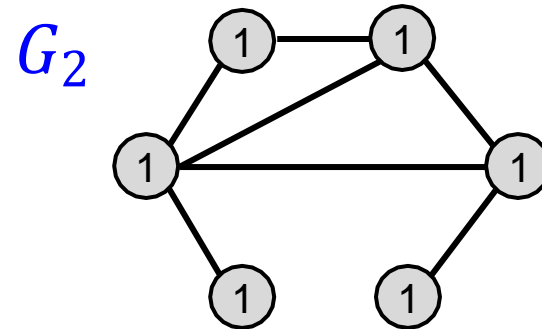
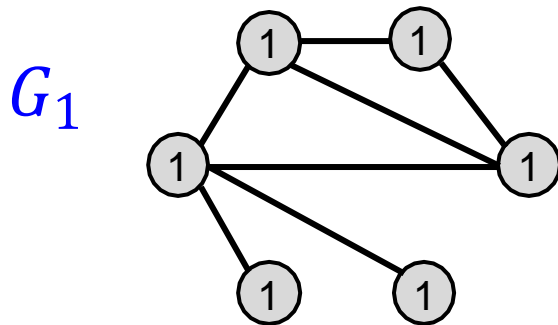
$$c^{(k+1)}(v) = \text{HASH} \left( \left\{ c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)} \right\} \right),$$

where **HASH** maps different inputs to different colors.

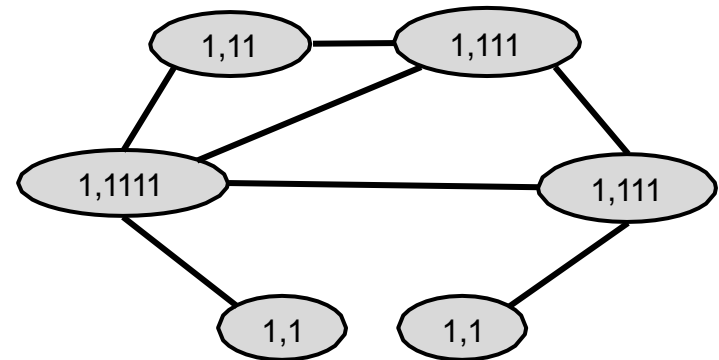
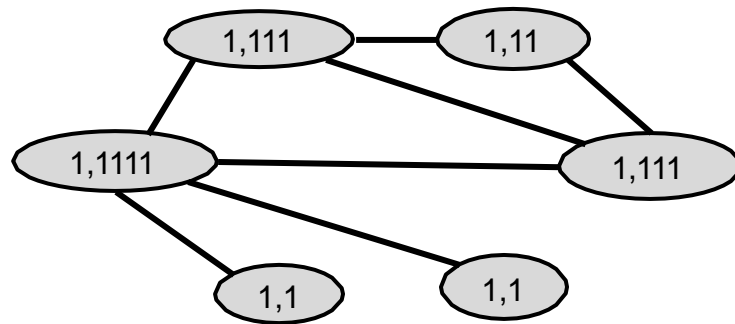
- After  $K$  steps of color refinement,  $c^{(K)}(v)$  summarizes the structure of  $K$ -hop neighborhood

# Color Refinement

- Assign initial colors

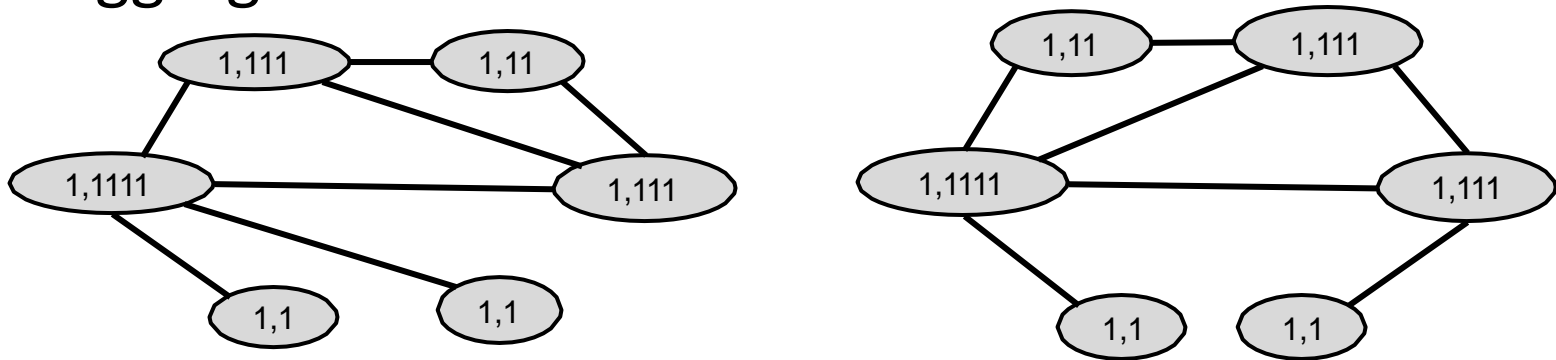


- Aggregate neighboring colors

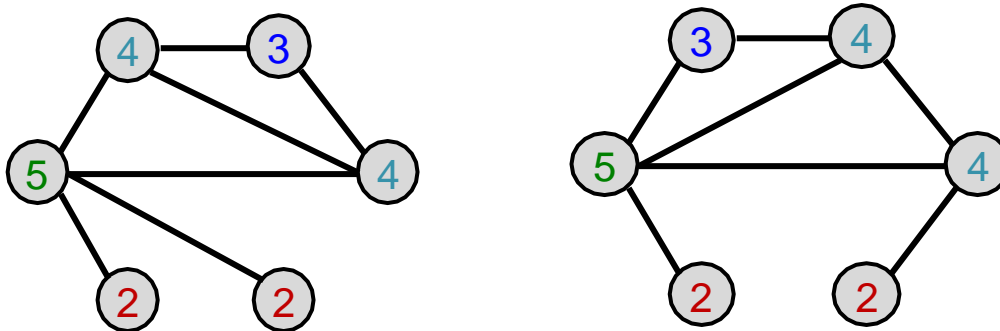


# Color Refinement

- Aggregated colors



- Hash aggregated colors

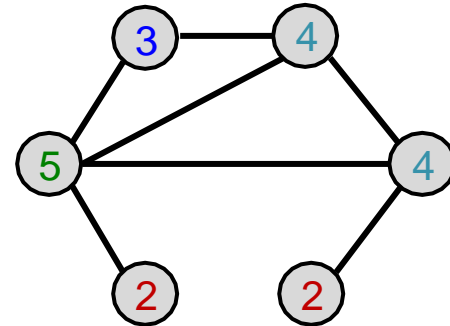
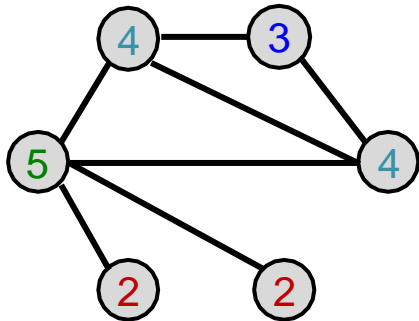


## Hash table

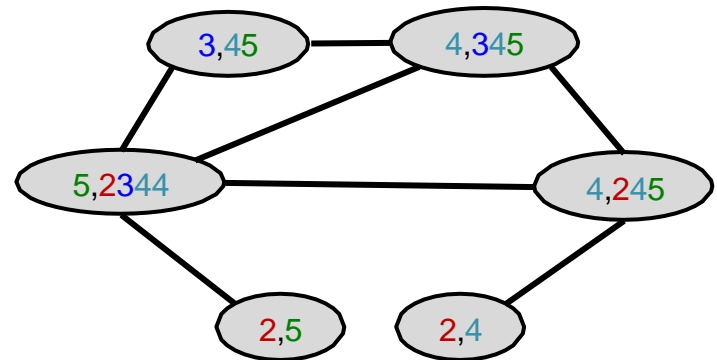
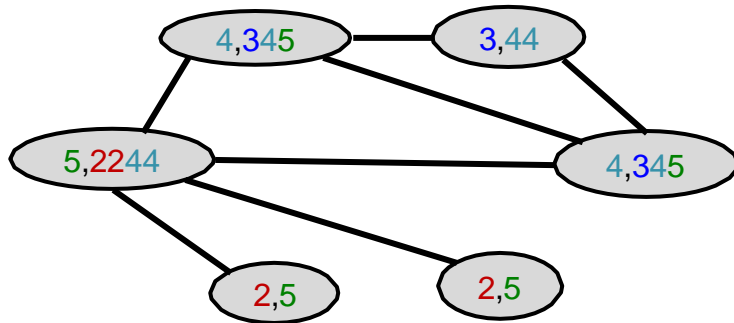
1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

# Color Refinement

- Aggregated colors



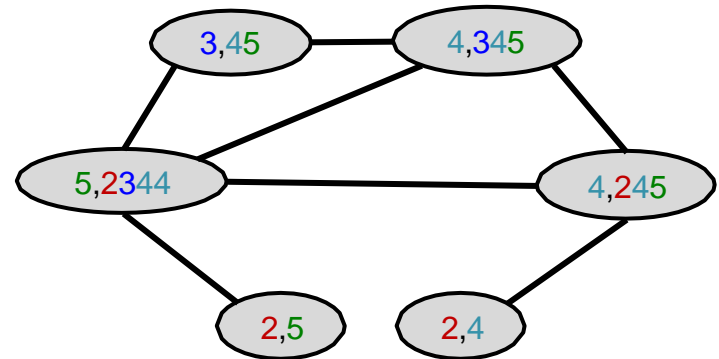
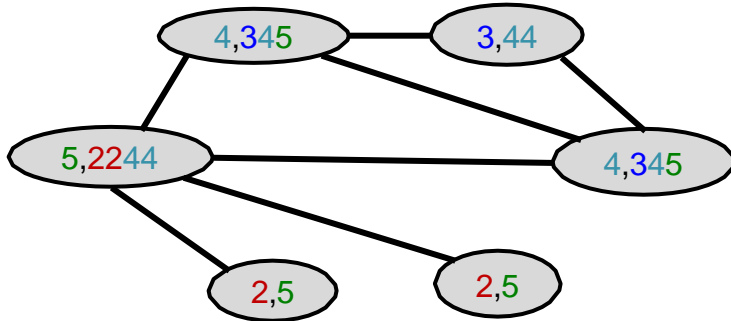
- Hash aggregated colors



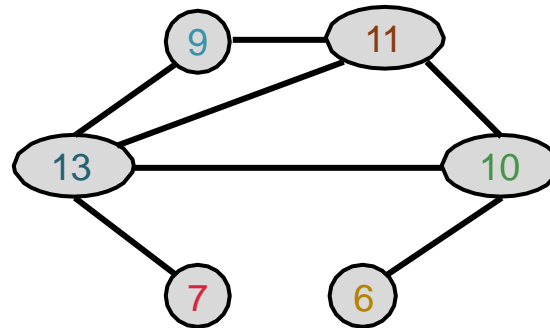
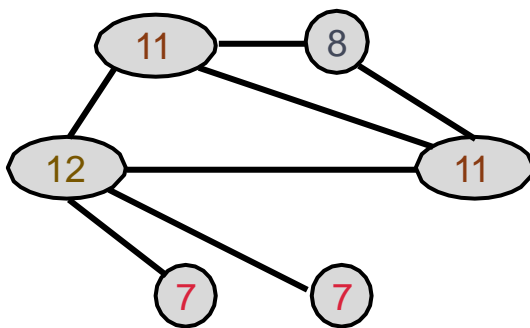


# Color Refinement

- Aggregated colors



- Hash aggregated colors



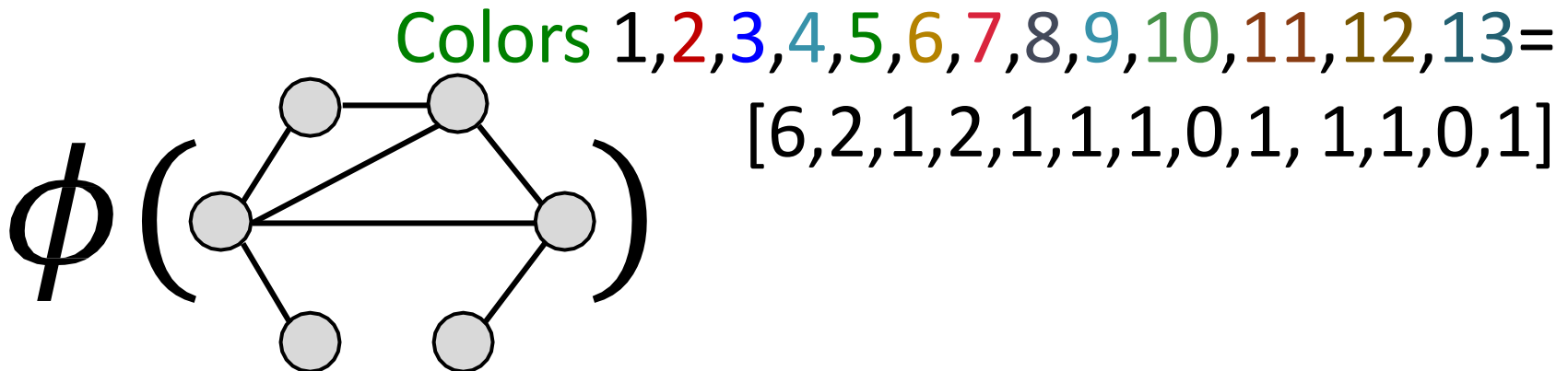
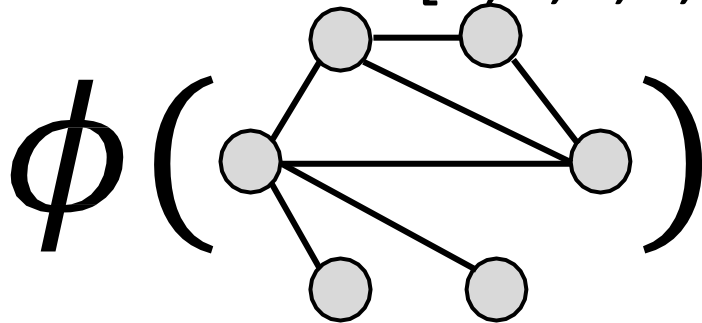
Hash table

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 Kernel

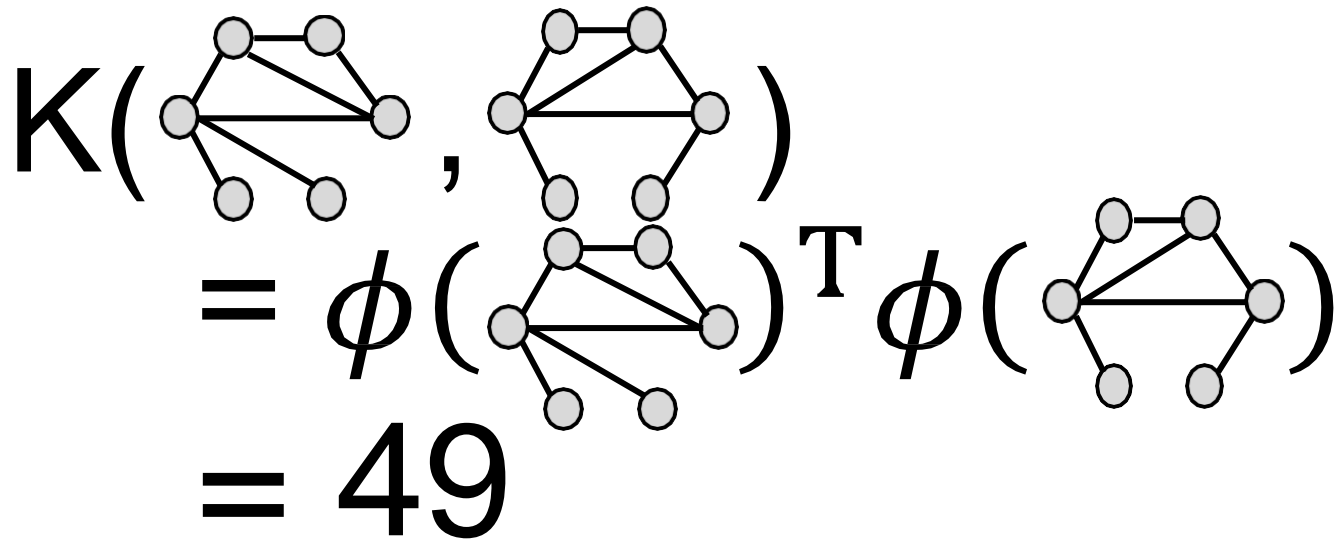
After color refinement, WL kernel counts number of nodes with a given color.

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]



# 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  $\#(\text{edges})$ , since it involves aggregating neighboring colors.
- When computing a kernel value, only colors appeared in the two graphs need to be tracked.
  - Thus,  $\#(\text{colors})$  is at most the total number of nodes.
- Counting colors takes linear-time w.r.t.  $\#(\text{nodes})$ .
- In total, time complexity is **linear in  $\#(\text{edges})$** .

# Graph Kernels

## ■ 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!)

# Example 1: Traffic Prediction

The screenshot displays a Google Maps interface with a route from Stanford University to the University of California, Berkeley. The map shows a blue route starting from Stanford University in the south and heading north through the San Francisco Bay Area. Three route options are listed on the left side of the map:

- via I-880 N**: 51 min, 38.9 miles. Description: Fastest route now, avoids road closure on University Ave.
- via I-280 N**: 52 min, 46.2 miles.
- via CA-84 E and I-880 N**: 52 min, 41.1 miles.

The map also shows various landmarks and cities, including San Francisco, Oakland, Alameda, San Leandro, Hayward, Union City, Fremont, Newark, Palo Alto, Redwood City, and San Mateo. A sidebar on the left provides navigation options and a search bar. The bottom of the map shows the Google logo and map data information.

# Traffic Prediction

Road networks as graphs

**Nodes:** Road segments

**Edges:** Connectivity between road segments

**Prediction:** Time of Arrival (ETA)

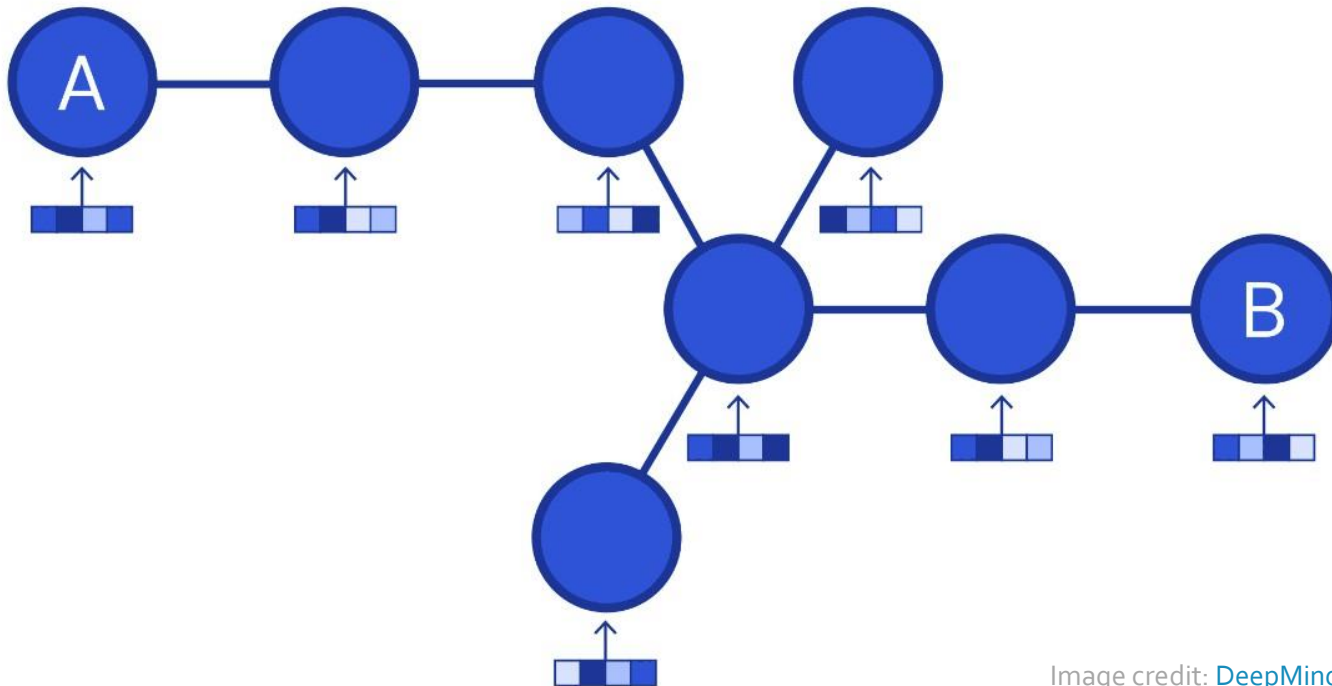
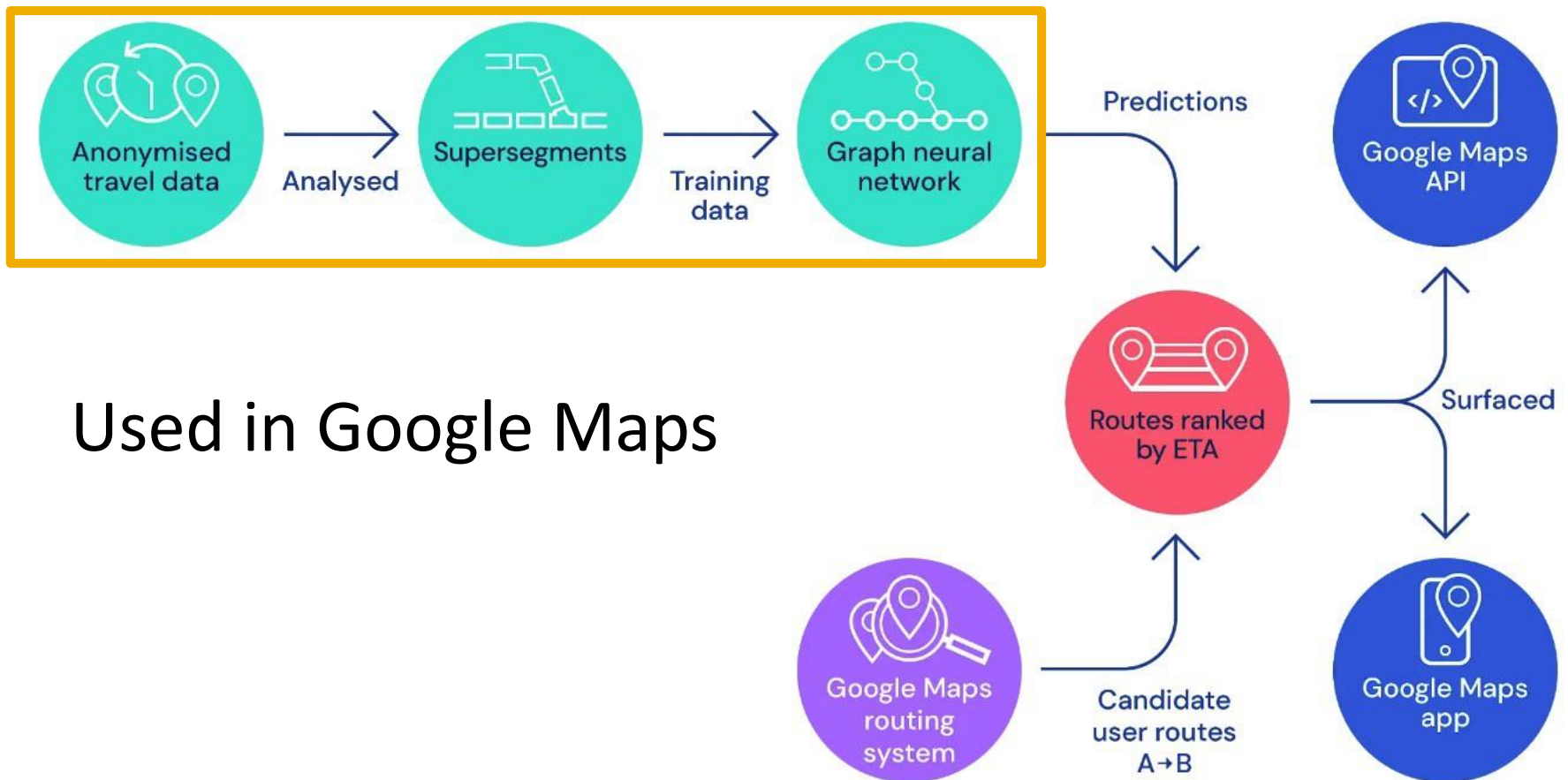


Image credit: [DeepMind](#)

# Traffic Prediction with GNNs

## Predicting Time of Arrival with GNNs



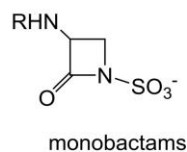
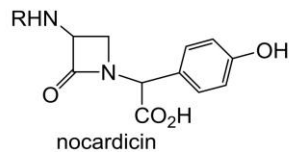
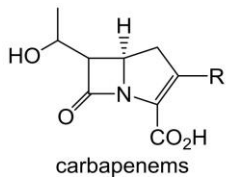
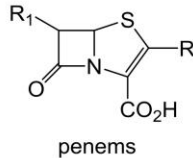
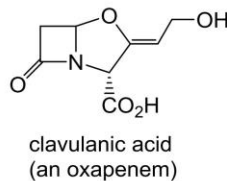
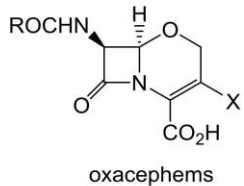
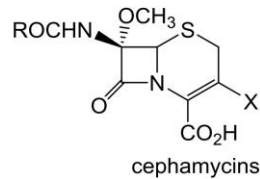
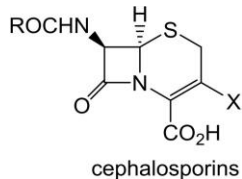
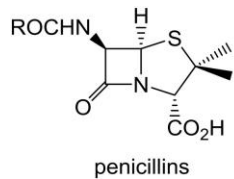


# Example 2: Drug Prediction

Antibiotics are small molecular graphs

**Nodes:** Atoms

**Edges:** Chemical bonds

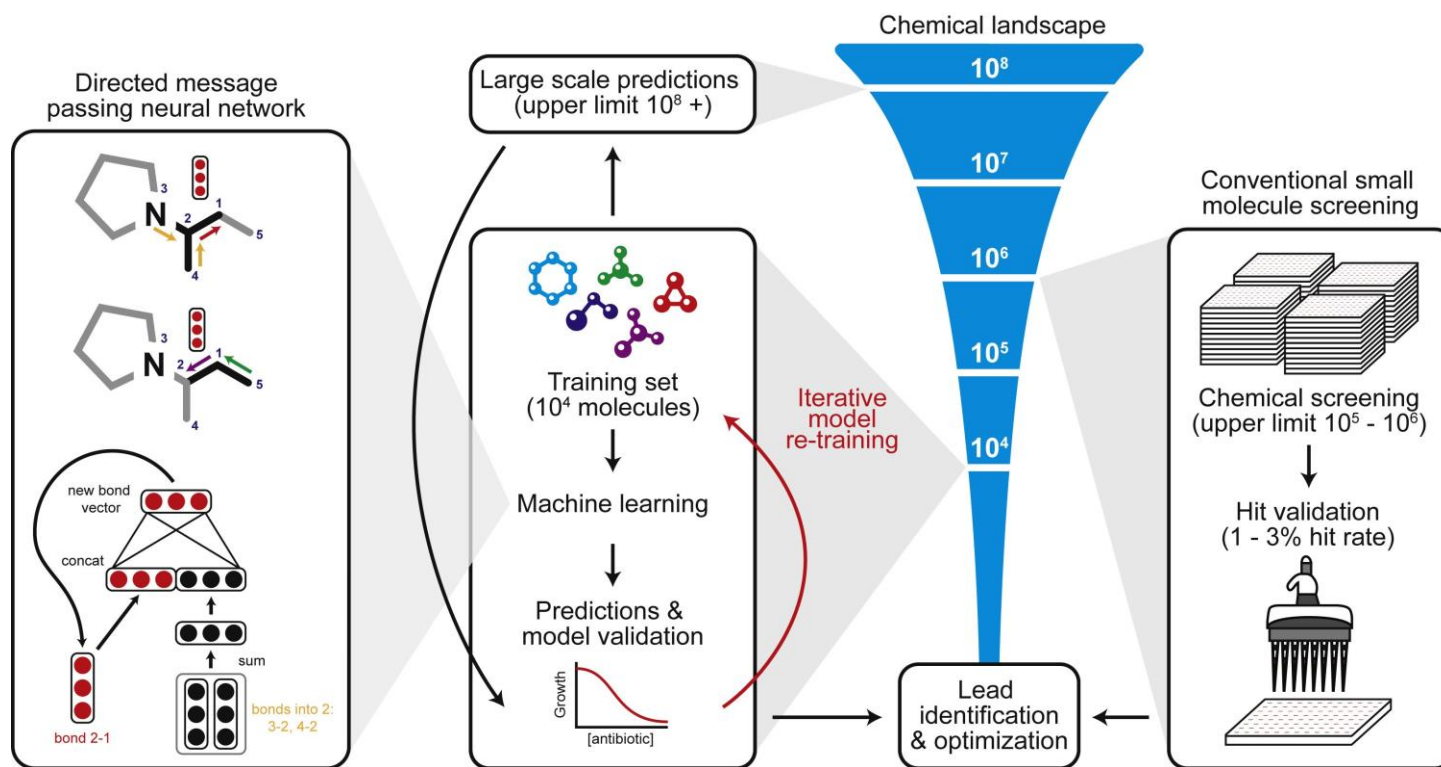


Konaklieva, Monika I. "Molecular targets of  $\beta$ -lactam-based antimicrobials: beyond the usual suspects." *Antibiotics* 3.2 (2014): 128-142.

Image credit: [CNN](#)

# Drug Prediction

- A Graph Neural Network **graph classification model**
- Predict promising molecules from a pool of candidates



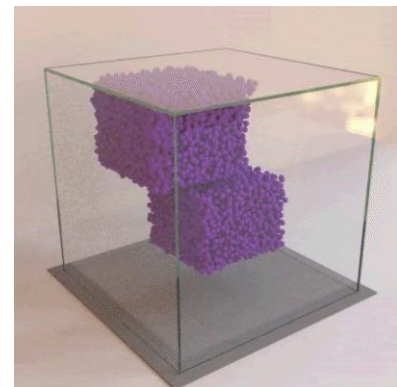
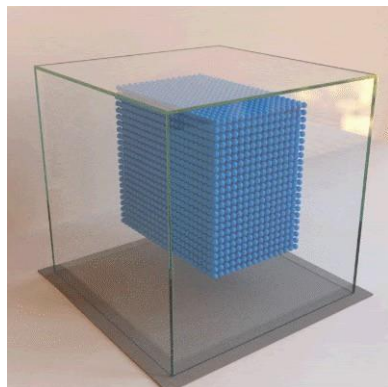
Stokes, Jonathan M., et al. "A deep learning approach to antibiotic discovery." Cell 180.4 (2020): 688-702.

# Example 3: Physical Simulation

Physical simulation as a graph:

**Nodes:** Particles

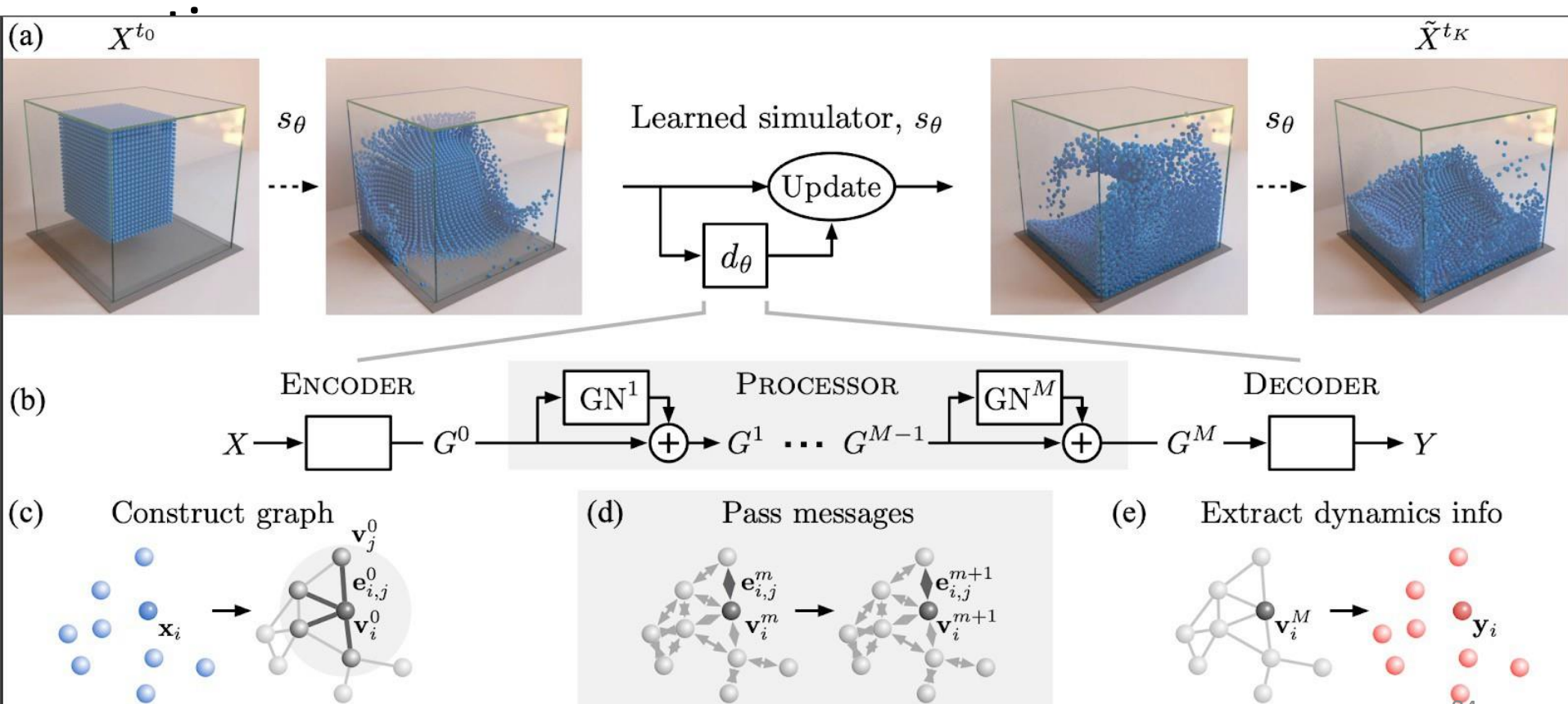
**Edges:** Interaction between particles



# Physical Simulation

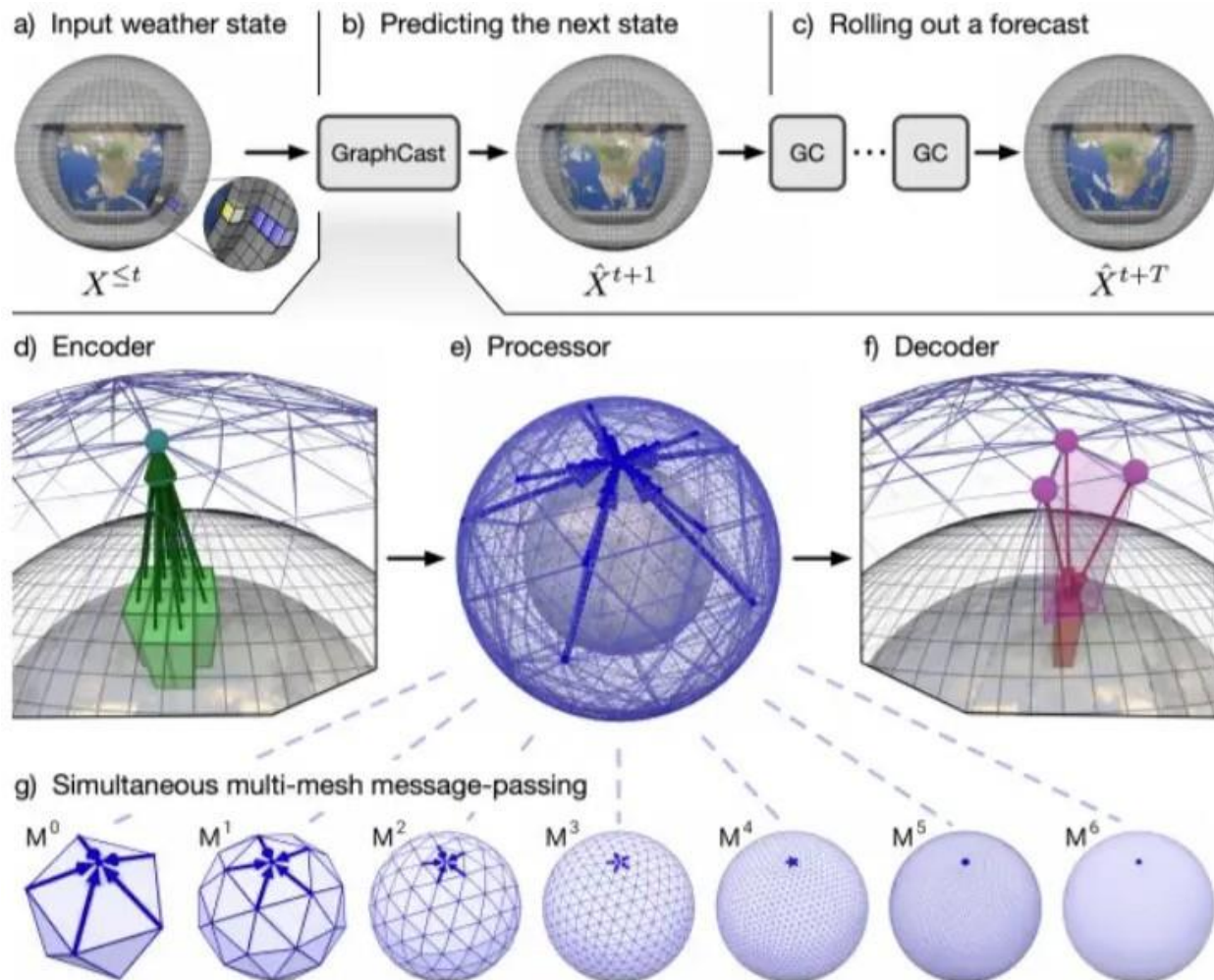
A graph evolution task:

- **Goal:** Predict how a graph will evolve over





# Application: Weather Forecasting



# 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

# Acknowledgement

Most slides from

CS224W: Machine Learning with Graphs, Jure Leskovec, Stanford University, <http://cs224w.stanford.edu>