Traditional Methods for Machine Learning in Graphs

Graph Properties

- **Degree**:
  how many friends do I have?
- **Weights**:
  how strong are the ties?
- **Path**:
  how far am I from another vertex?
- **Connectivity**:
  can I reach all other vertices?
- **Diameter**:
  how dense are they?
- **Centrality**
  (e.g., betweenness, closeness): Am I in the center of everyone?

Graph Analysis Problem

- Existing Graph Analysis Problems
    1. Clique identification
    2. Shortest path
    3. K-core decomposition

  and more…

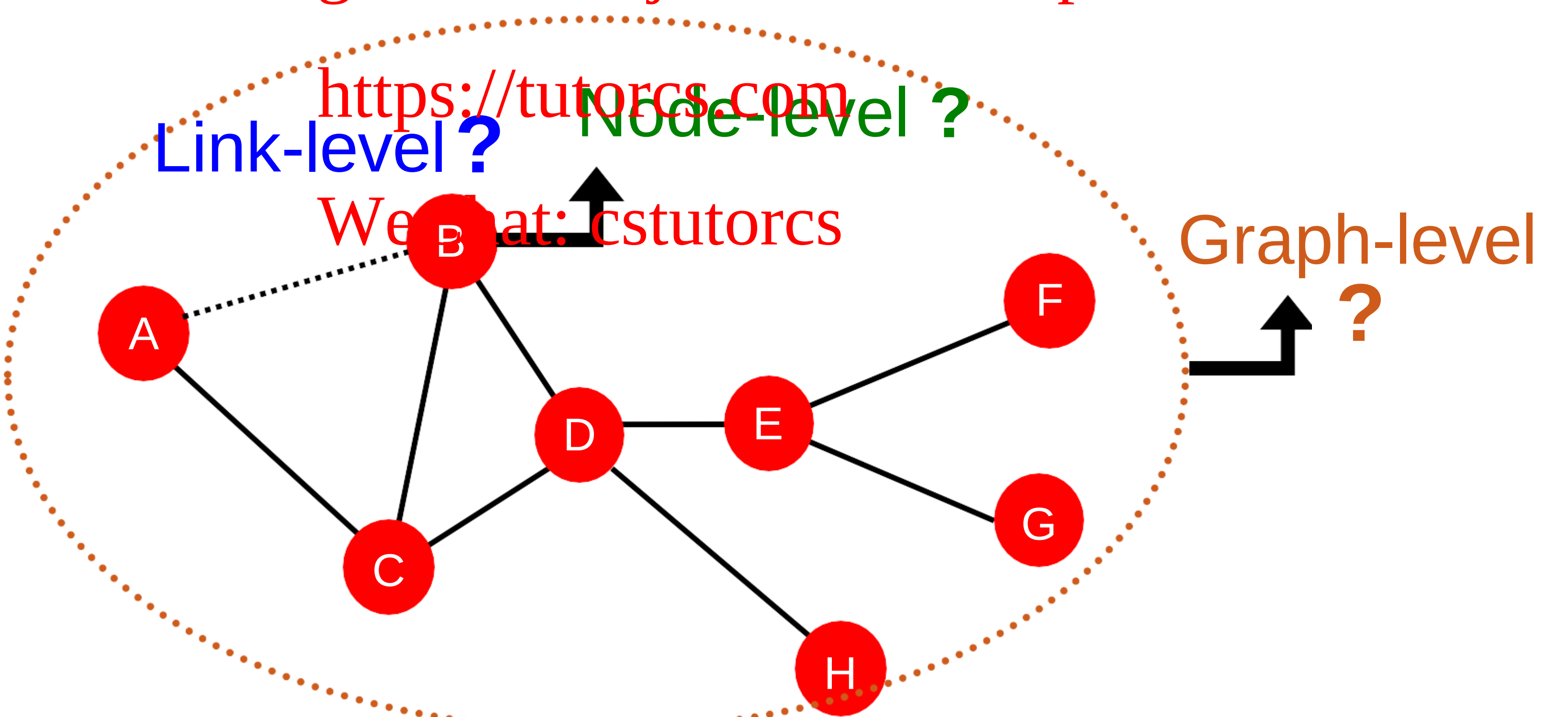
We need machine learning for graphs

- Many Applications
- Challenge:
  - finding a way to represent, or encode, graph structure so that it can be easily exploited by machine learning models.

- Traditionally, machine learning approaches relied on user-defined heuristics to extract features encoding structural information about a graph (e.g., degree statistics or kernel functions).

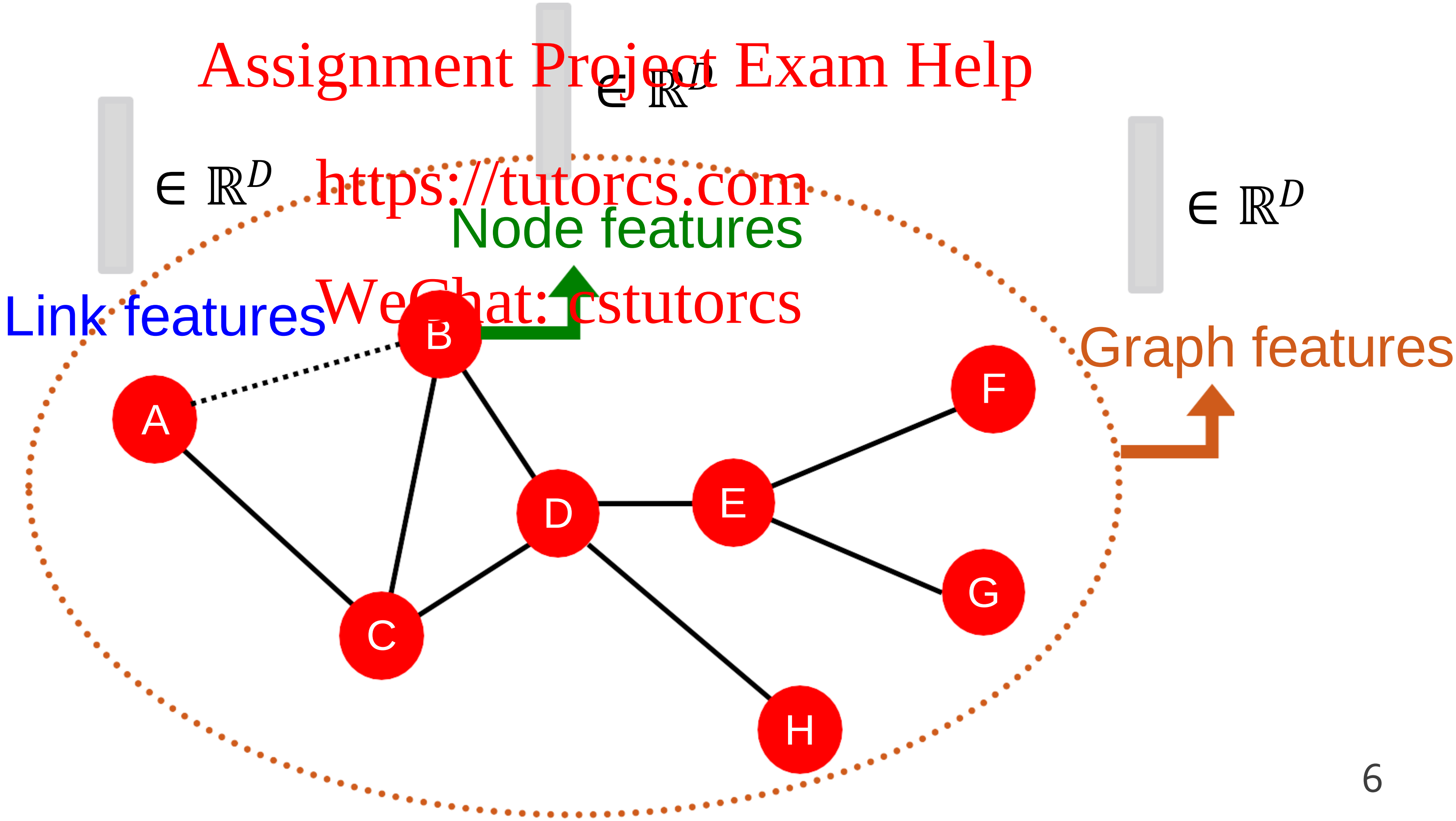
Machine Learning Tasks on Graphs

- Node-level prediction
- Link-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.

$$\mathbf{x}_1 \rightarrow y_1$$

$$\vdots$$

$$\mathbf{x}_N \rightarrow y_N$$

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

$$\mathbf{x} \rightarrow y$$

Feature Design

- **Using effective features over graphs is the key to achieving good model 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.

ML 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?

ML in Graphs

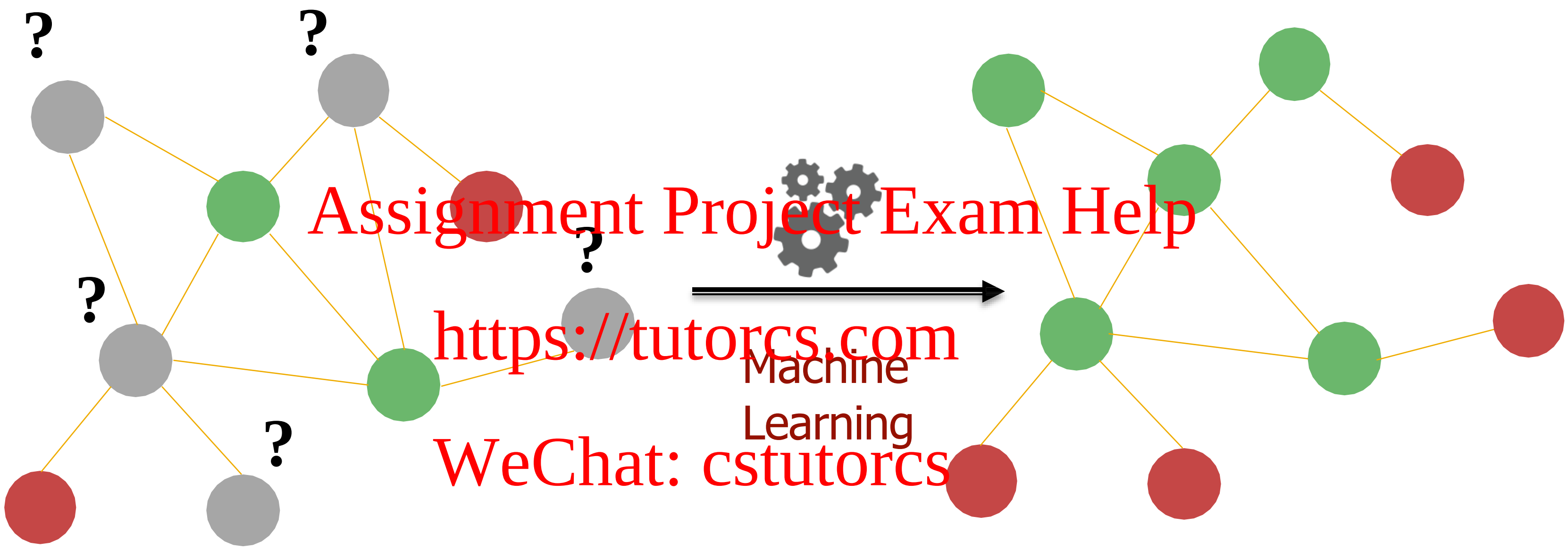
Example: Node-level prediction

- Given: $G = (V, E)$
- Learn a function: $f : V \to \mathbb{R}$

How do we learn the function?

Node-level Tasks and Features

Node-level Tasks

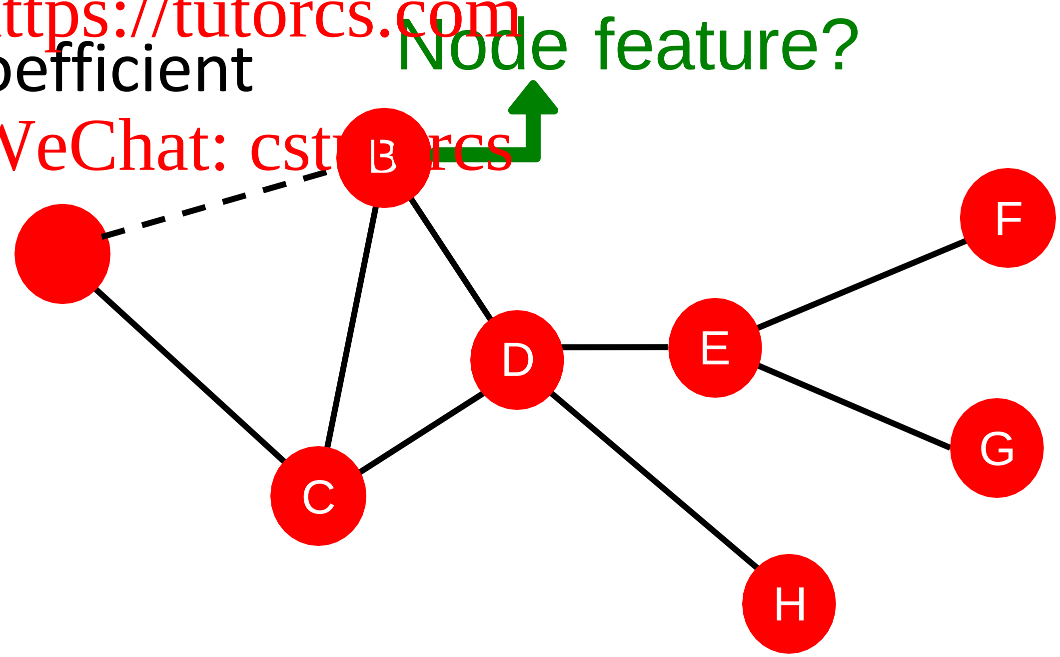


Node classification!

ML needs features.

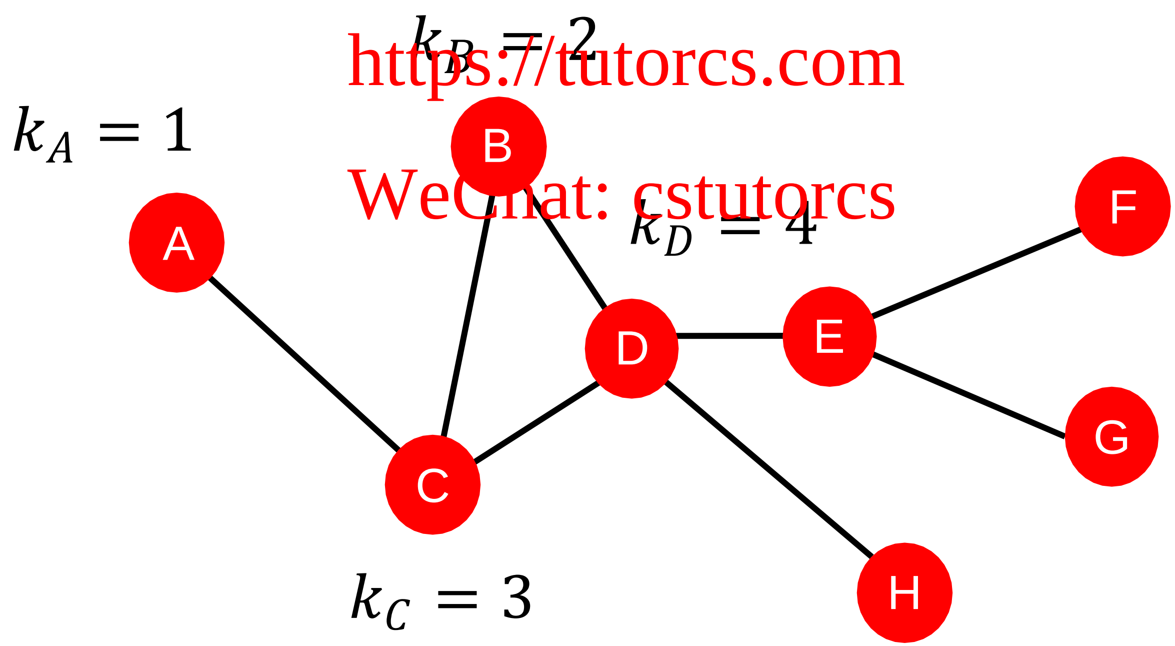
Node-level Features: Overview

- **Goal:** Characterize the structure and position of a node in the network:
  - Node degree
  - Node centrality
  - Clustering coefficient
  - Graphlets



Node-level Features: Node Degree

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



Node-level 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

  many others…

Node Centrality (1)

- **Eigenvector centrality:**
  - A node v is important if surrounded by important neighboring nodes $u \in N(v)$.

  - We model the centrality of node v as the sum of the centrality of neighboring nodes:

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

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

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

Node Centrality (1)

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

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

λ is normalization const (largest eigenvalue of A)

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

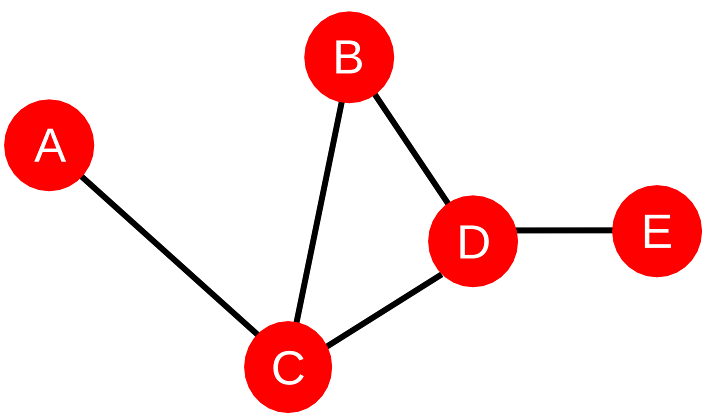
- We see that centrality c is the **eigenvector of $\boldsymbol{A}$**!
- The largest eigenvalue λ_{max} is always positive and unique (by Perron-Frobenius Theorem).
- The eigenvector $\boldsymbol{c}_{max}$ corresponding to λ_{max} is used for centrality.

Node Centrality (2)

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

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

  - **Example:**



$c_A = c_B = c_E = 0$

$c_C = 3$

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

$c_D = 3$

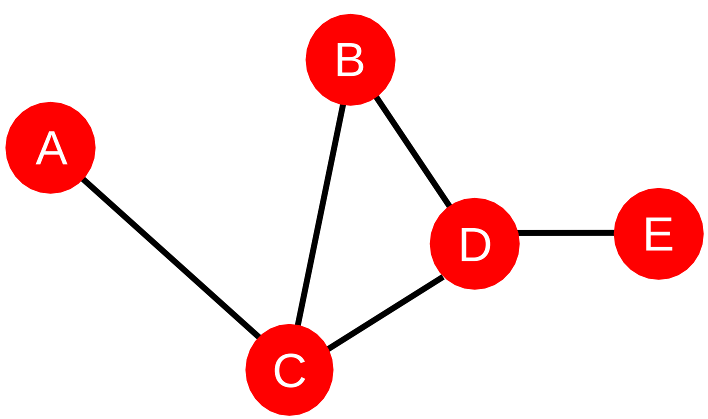
(A-C-D-E, B-D-E, C-D-E)

Node Centrality (3)

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

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

  - **Example:**



$c_A = 1/(2 + 1 + 2 + 3) = 1/8$
(A-C-B, A-C, A-C-D, A-C-D-E)

$c_D = 1/(2 + 1 + 1 + 1) = 1/5$
(D-C-A, D-B, D-C, D-E)

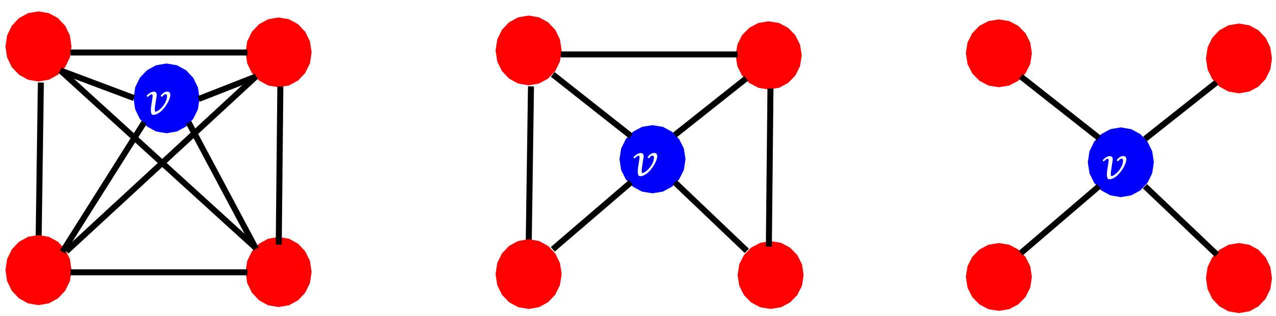
Node Features: Clustering Coefficient

- Measures how connected $v's$ neighboring nodes are:

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

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

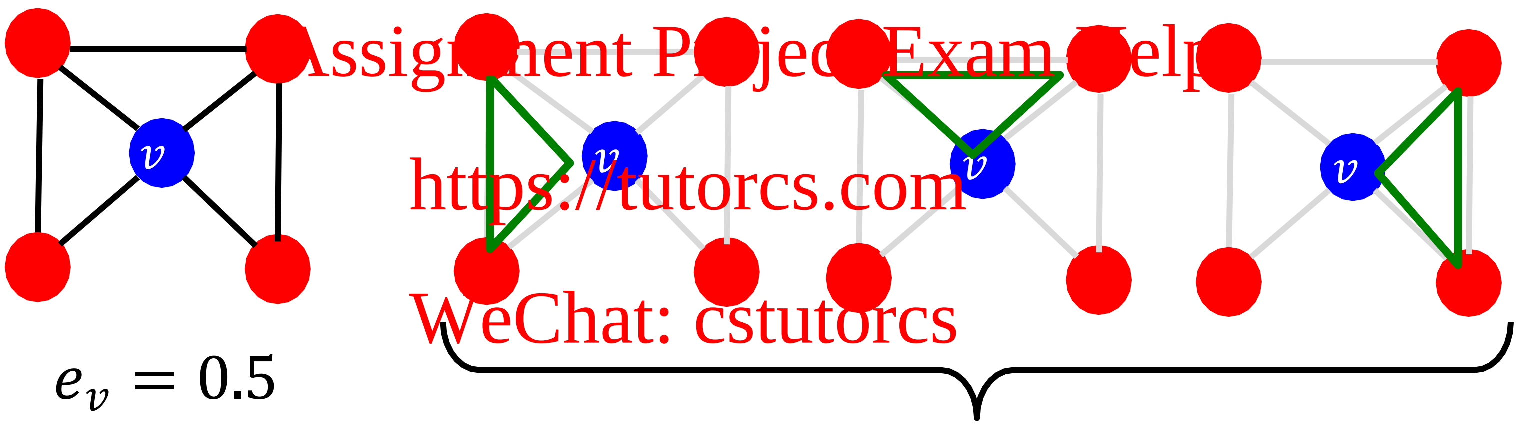
- **Examples:**



$e_v = 1$ $e_v = 0.5$ $e_v = 0$

Node Features: Graphlets

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



$e_v = 0.5$

3 triangles (out of 6 node triplets)

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

Node Features: Graphlets

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

Analogy:

- **Degree**

  counts **#(edges)** that a node touches

- **Clustering coefficient**

  counts **#(triangles)** that a node touches.

- **Graphlet Degree Vector (GDV)**:

  Graphlet-base features for nodes

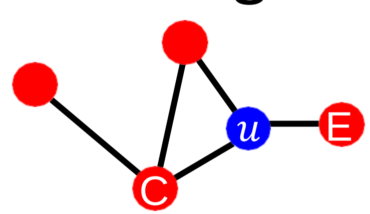
  - **GDV** counts **#(graphlets)** that a node touches

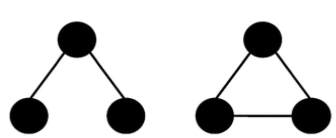
Node Features: Graphlets

- Considering graphlets of size 2-5 nodes we get:
  - **Vector of 73 coordinates** is a signature of a node that describes the topology of node's neighborhood

- Graphlet degree vector provides a measure of a **node's local network topology**:

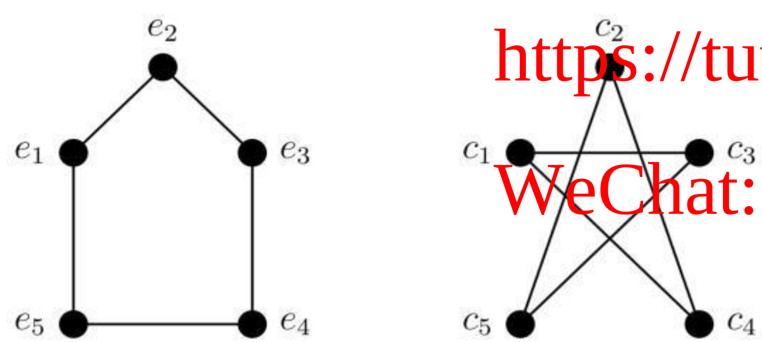
  - Comparing vectors of two nodes provides a more detailed measure of local topological similarity than node degrees or clustering coefficient.

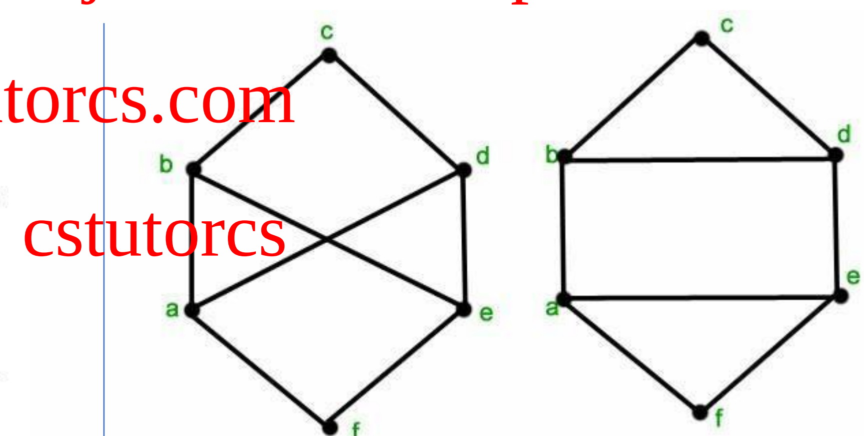




Isomorphism

- **Def: Graph Isomorphism**
  - Two graphs which contain the same number of nodes connected in the same way are said to be isomorphic.





Isomorphic

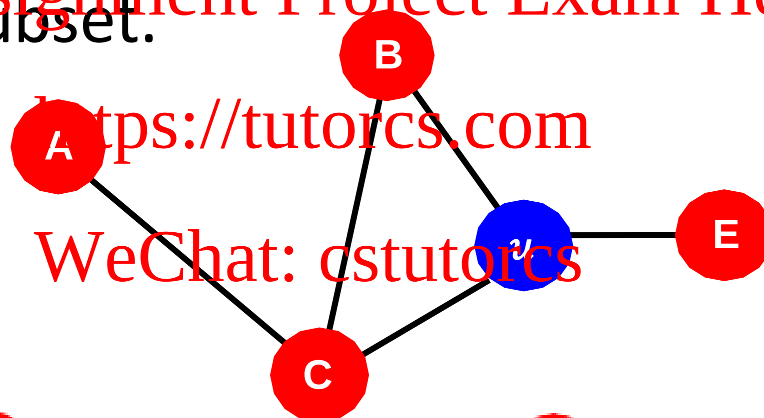
Node mapping: (e2,c2), (e1, c5), (e3,c4), (e5,c3), (e4,c1)

Non-Isomorphic

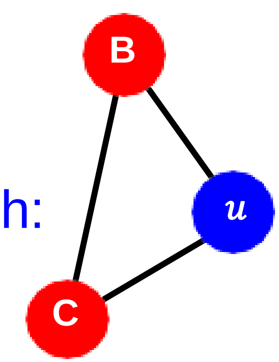
The right graph has cycles of length 3 but he left graph does not, so the graphs cannot be isomorphic.

Induced Subgraph

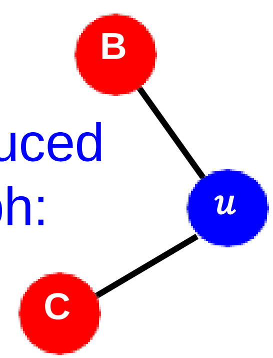
- **Def: Induced subgraph**
  - A graph, formed from a subset of vertices and *all* of the edges connecting the vertices in that subset.



Induced subgraph:



Non-induced Subgraph:

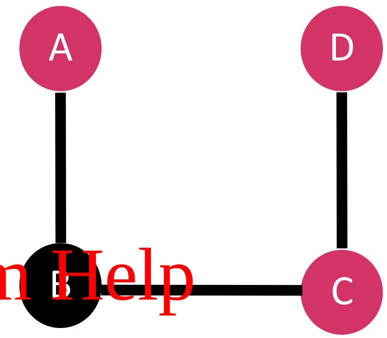


Subgraph Isomorphism

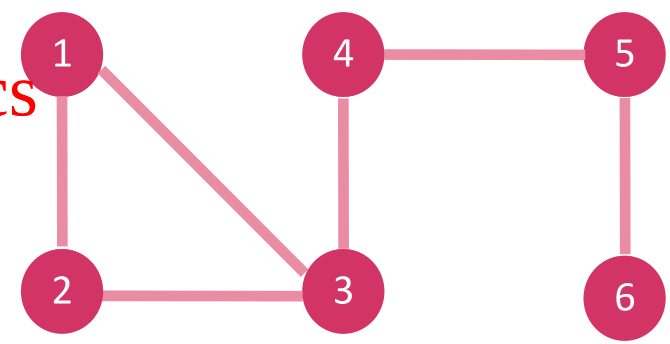
- Let $G = (V, E)$ be a data graph and $p = (V_p, E_p)$ be a pattern graph.
- A function $f\colon V_p \to V$ is called a homomorphism of p if for each edge $(u, u') \in E_p$, we have $(f(u), f(u')) \in E$.
- A homomorphism f of p is called a subgraph isomorphism of p if f is injective such that f never maps distinct nodes in V_p to the same node in V.

An Example

- Let $G = (V, E)$ be a data graph and $p = (V_p, E_p)$ be a pattern graph.



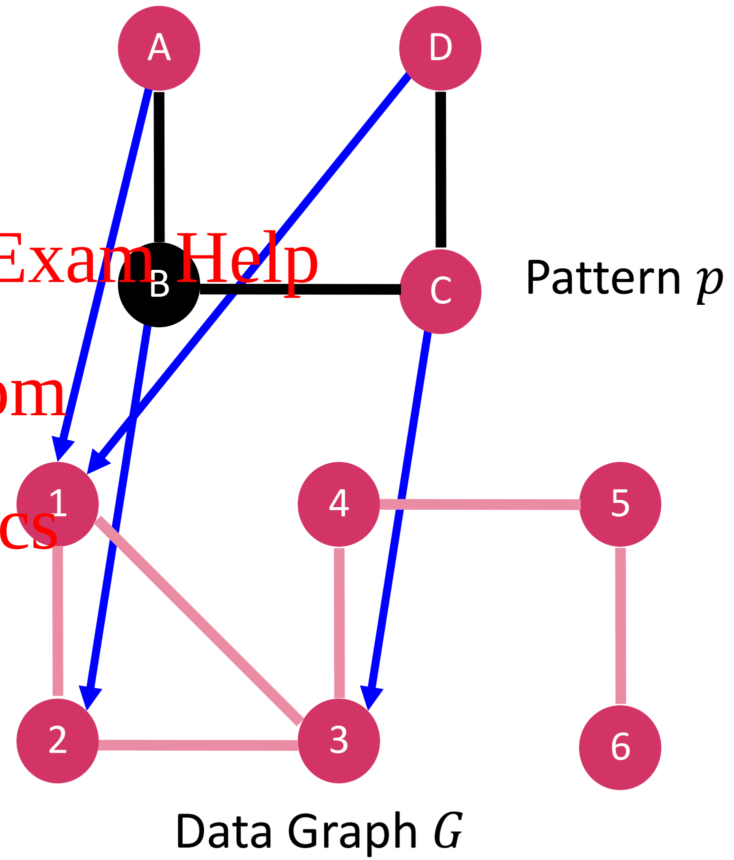
Pattern p



Data Graph G

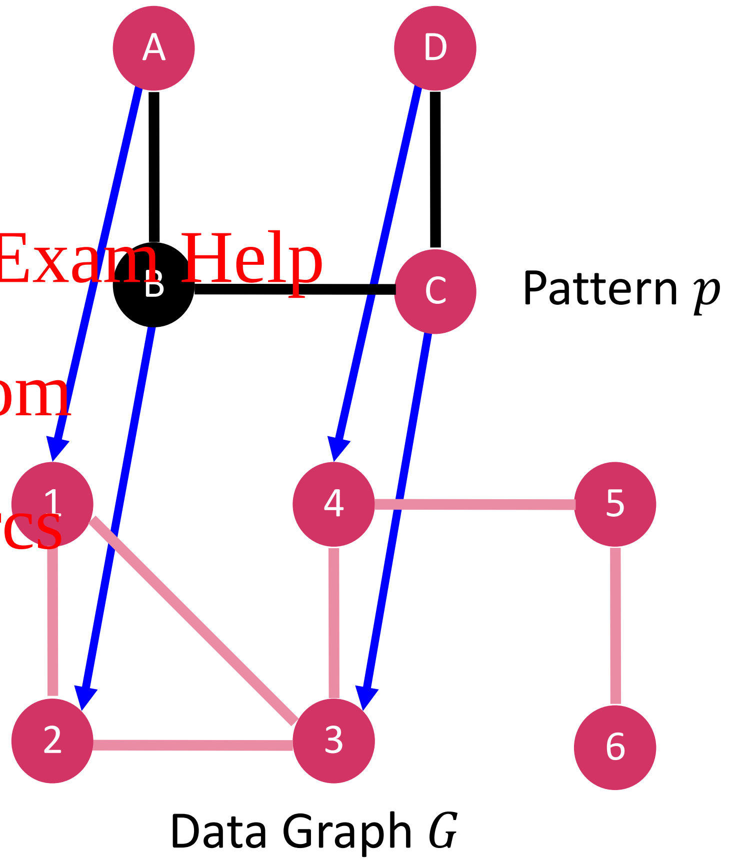
Homorphism

- A function $f: V_p \to V$ is called a homomorphism of p if for each edge $(u, u') \in E_p$, we have $(f(u), f(u')) \in E$.



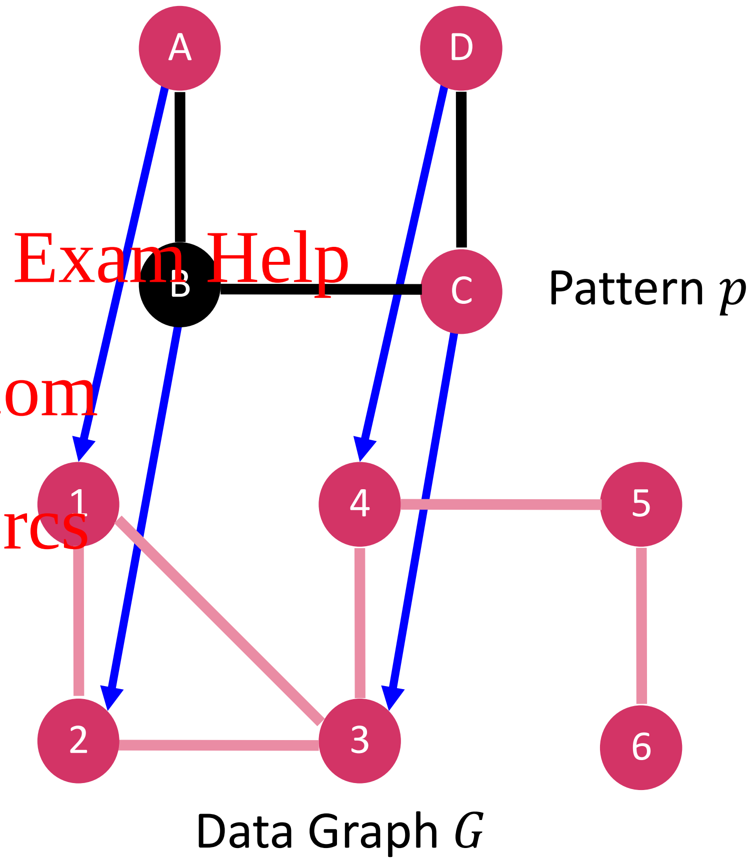
Subgraph Isomorphism

- A function $f\colon V_p \to V$ is called a homomorphism of p if for each edge $(u, u') \in E_p$, we have $(f(u), f(u')) \in E$.
- A homomorphism f of p is called a subgraph isomorphism of p if f is injective such that f never maps distinct nodes in V_p to the same node in V.



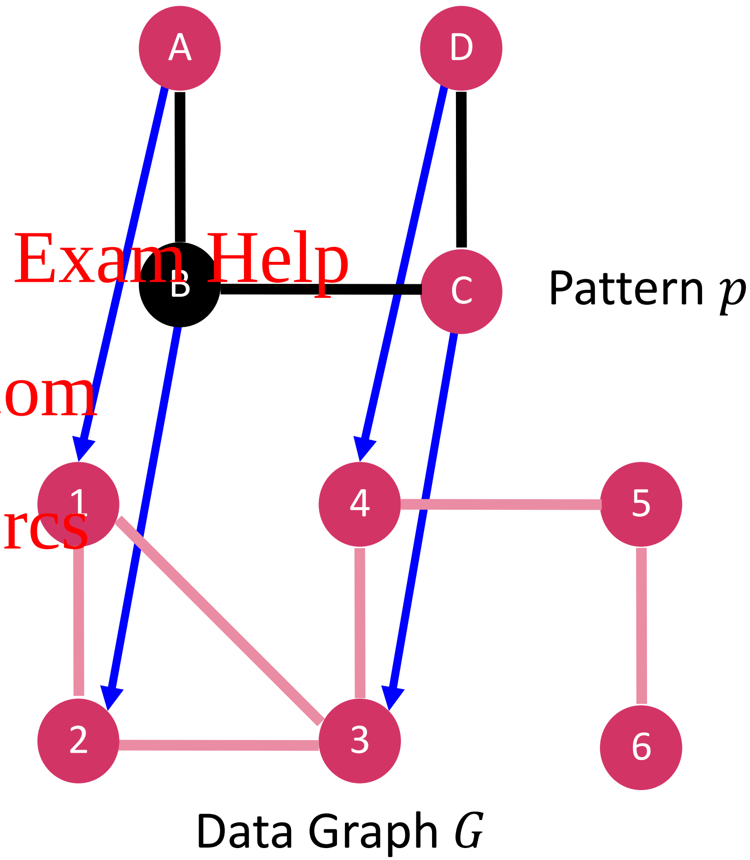
Subgraph Isomorphism (Non-Induced)

- A function $f\colon V_p \to V$ is called a homomorphism of p if for each edge $(u, u') \in E_p$, we have $(f(u), f(u')) \in E$.
- A homomorphism f of p is called a subgraph isomorphism of p if f is injective such that f never maps distinct nodes in V_p to the same node in V.



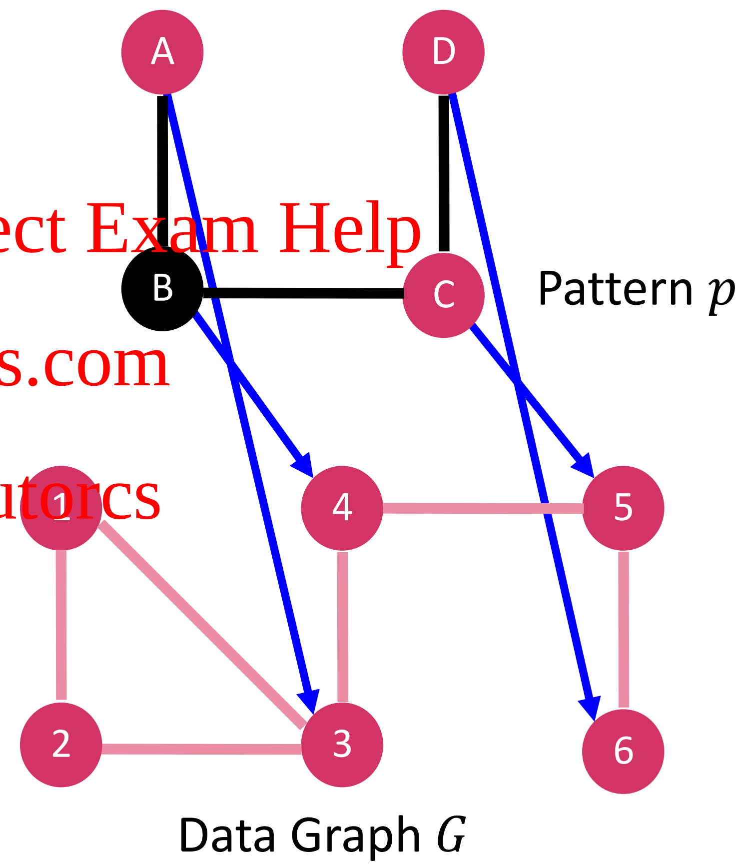
Subgraph Isomorphism (Non-Induced)

- A function $f\colon V_p \rightarrow V$ is called a homomorphism of p if for each edge $(u, u') \in E_p$, we have $(f(u), f(u')) \in E$.
- A homomorphism f of p is called a subgraph isomorphism of p if f is injective such that f never maps distinct nodes in V_p to the same node in V.

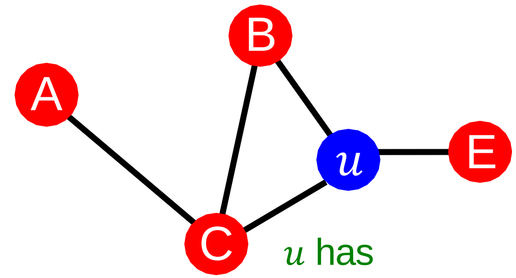


Subgraph Isomorphism (Induced)

- A function $f\colon V_p \to V$ is called a homomorphism of p if for each edge $(u, u') \in E_p$, we have $(f(u), f(u')) \in E$.
- A homomorphism f of p is called a subgraph isomorphism of p if f is injective such that f never maps distinct nodes in V_p to the same node in V.



Node Features: Graphlets

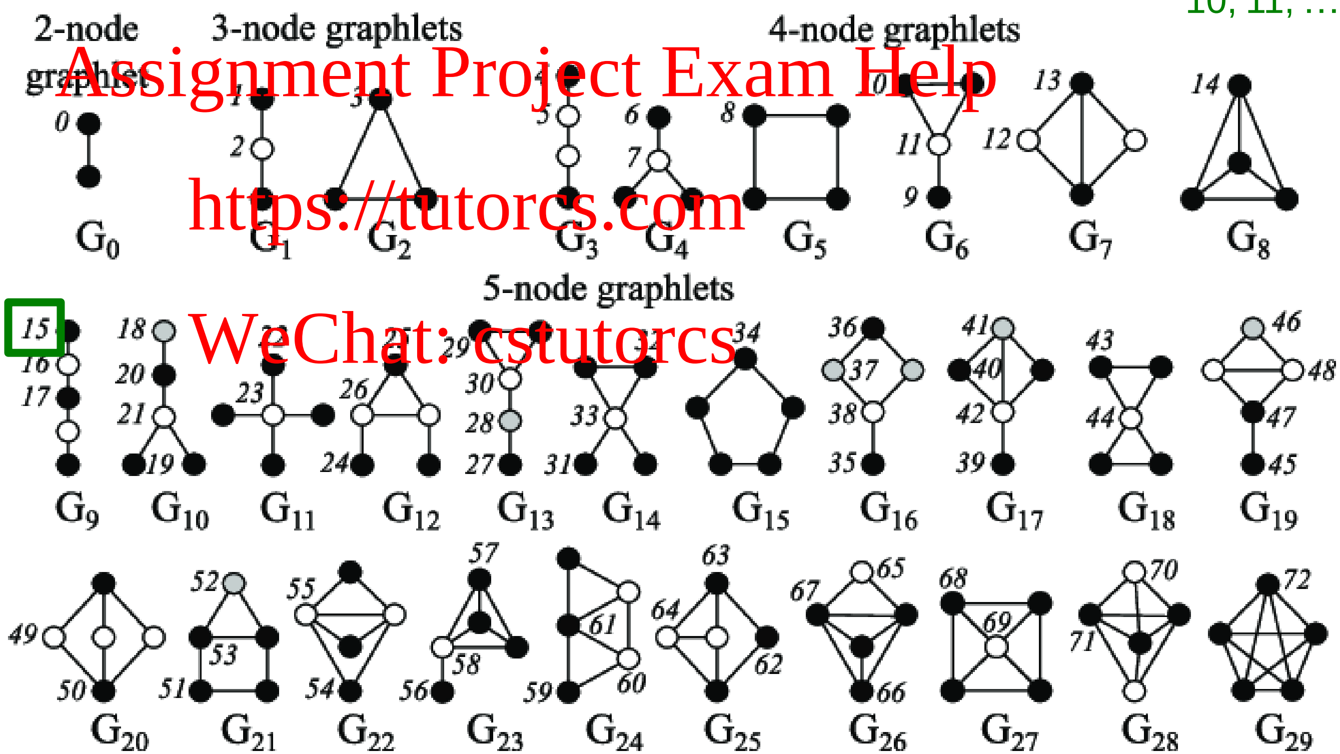


u has graphlets: 0, 1, 2, 3, 5, 10, 11, …

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

Take some nodes and all the edges between them.

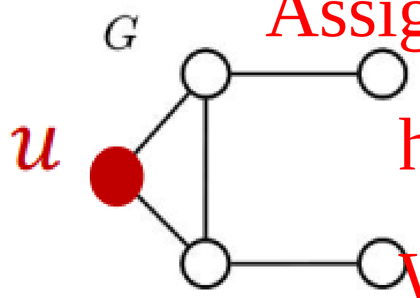
Graphlet id (Root of node u)



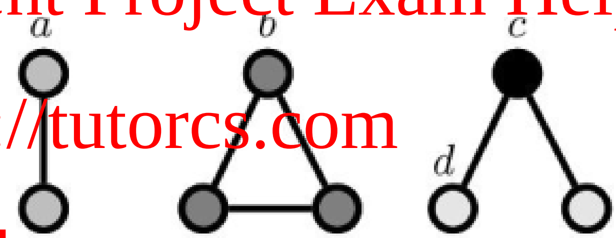
There are 73 different graphlet of up to 5 nodes

Node Features: Graphlets

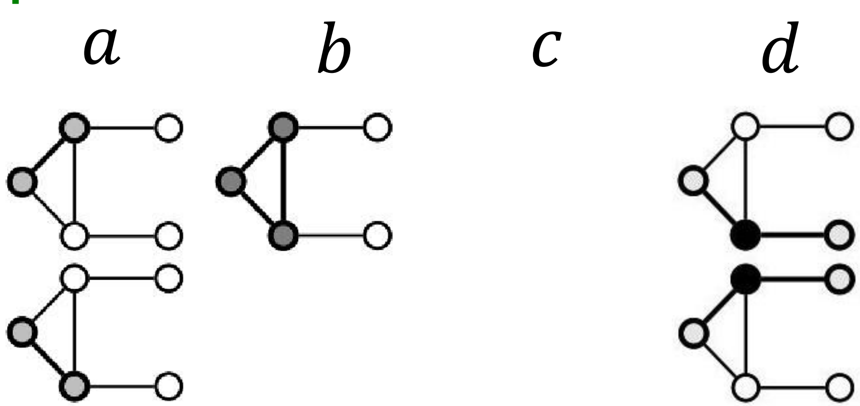
- **Graphlet Degree Vector (GDV):** A count vector of graphlets rooted at a given node.
- **Example:**



Possible graphlets up to size 3



Graphlet instances of node u:



GDV of node u:

a, b, c, d

[2,1,0,2]

Node-level Feature: Summary

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

Node-level Feature: Summary

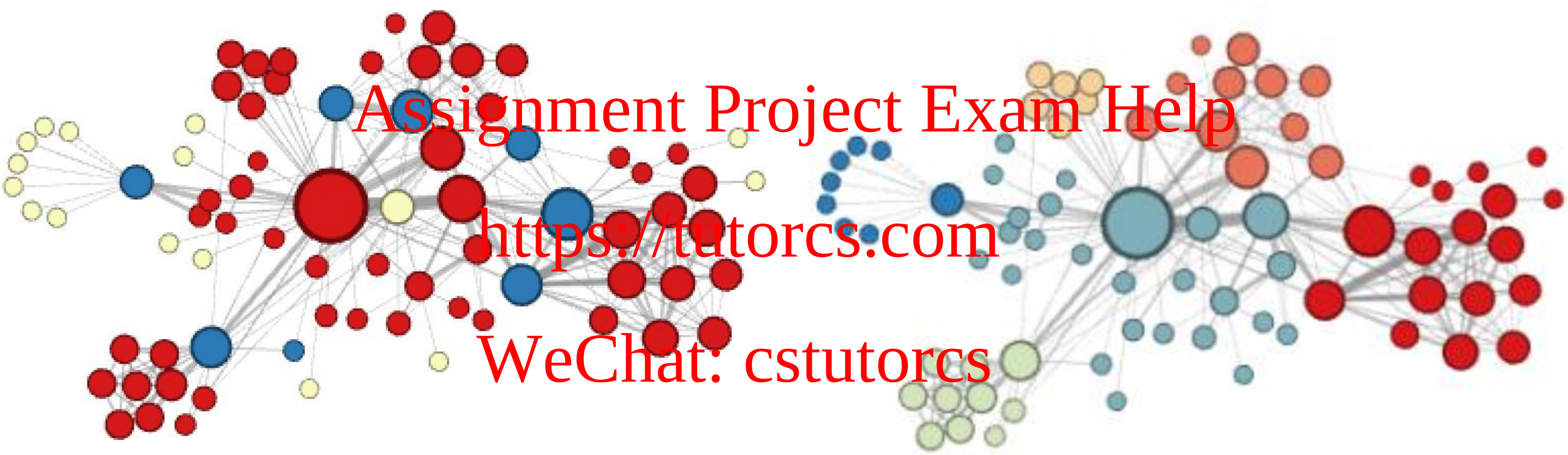
- **Importance-based features**: capture the importance of a node in a graph
  - Node degree:
    - Simply counts the number of neighboring nodes
  - Node centrality:
    - Models importance of neighboring nodes in a graph
    - Different modeling choices: eigenvector centrality, betweenness centrality,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

Node-level Feature: Summary

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

Discussion

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

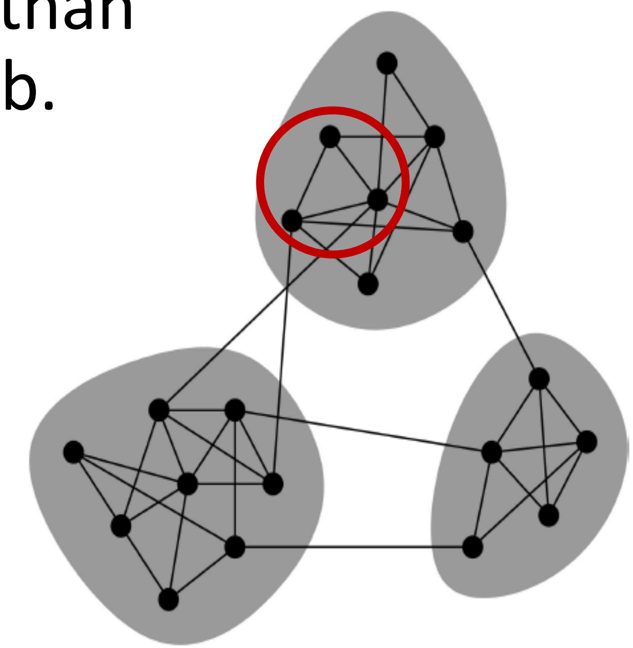
Link Prediction Task and Features

When can people be friends?

- In 1960s, *Mark Granovetter interviewed people who had recently changed employers to learn how they discovered their new jobs.*

- Surprising: Acquaintances rather than close friends help to find a new job.

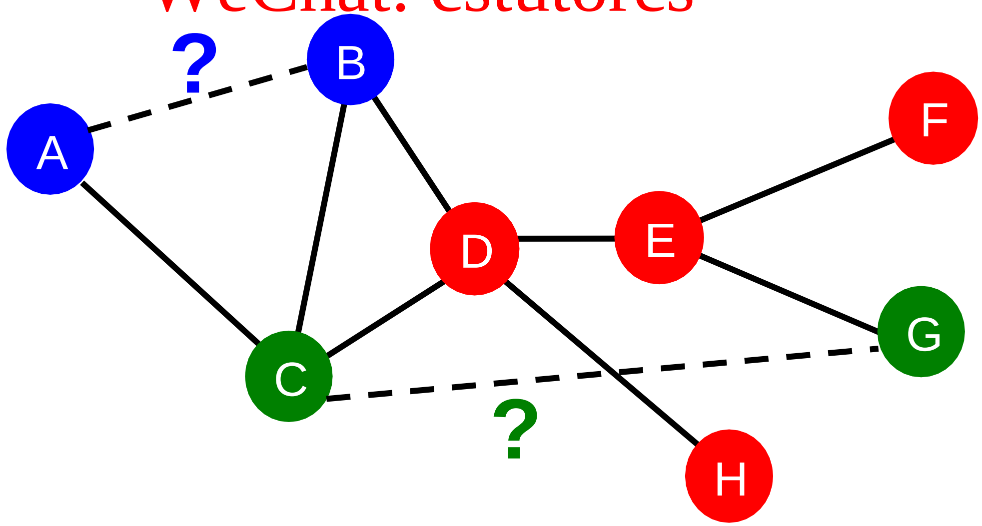
- *Triadic Closure*: *If two people in a social network have a friend in common, then there is an increased likelihood that they will become friends themselves at some point in future.*



Refer to Networks Crowds and Markets by D Easley and J. Kleinberg, and Jure Leskovec's lecture notes.

Link-level Prediction Task: Recap

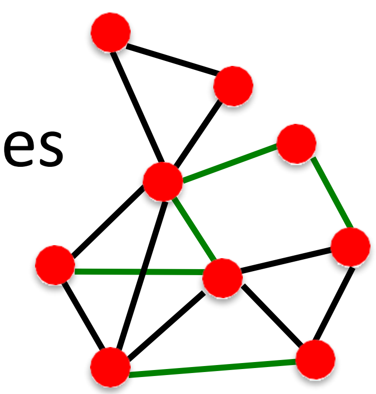
- The task is to predict **new links** based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- The key is to design features for **a pair of nodes**.



Link-level Prediction as a Task

Two formulations of the link prediction task:

- **1) Links missing at random:**
  - Remove a random set of links and then aim to predict them
- **2) Links over time:**
  - Given $G[t_0, t_0']$ a graph defined by edges up to time t_0', **output a ranked list L** of edges (not in $G[t_0, t_0']$) that are predicted to appear in time $G[t_1, t_1']$

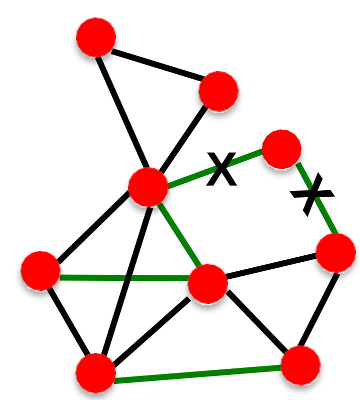


$G[t_0, t_0']$
$G[t_1, t_1']$

1

Link Prediction via Proximity

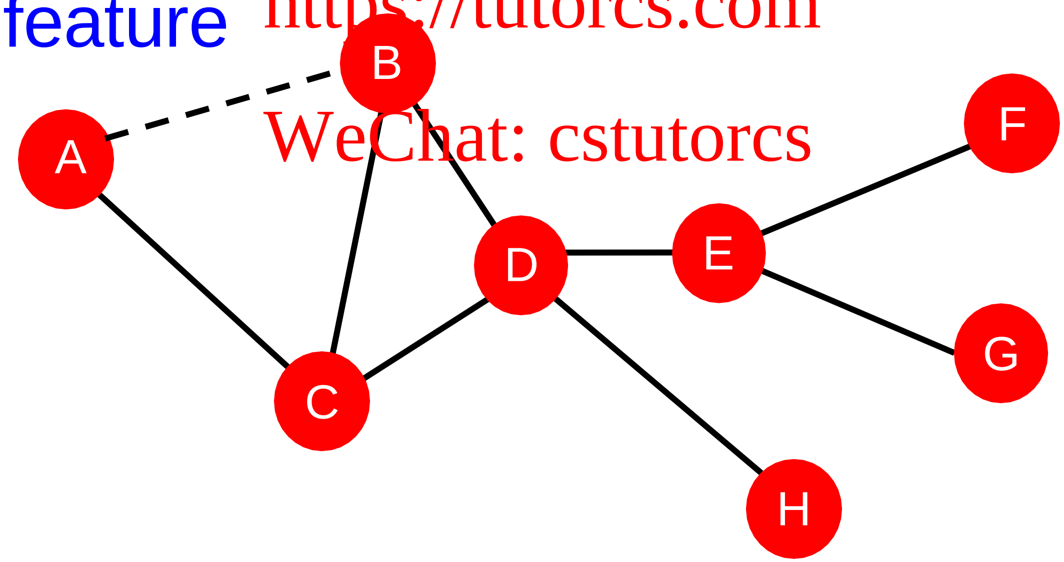
- **Methodology:**
  - For each pair of nodes *(x,y)* compute **score** *c(x,y)*
    - For example, *c(x,y)* could be the # of common neighbors of *x* and *y*
  - Sort pairs *(x,y)* by the decreasing score *c(x,y)*
  - **Predict top *n* pairs as new links**
  - **See which of these links actually appear in $G[t_1, t_1']$**



Link-level Features: Overview

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

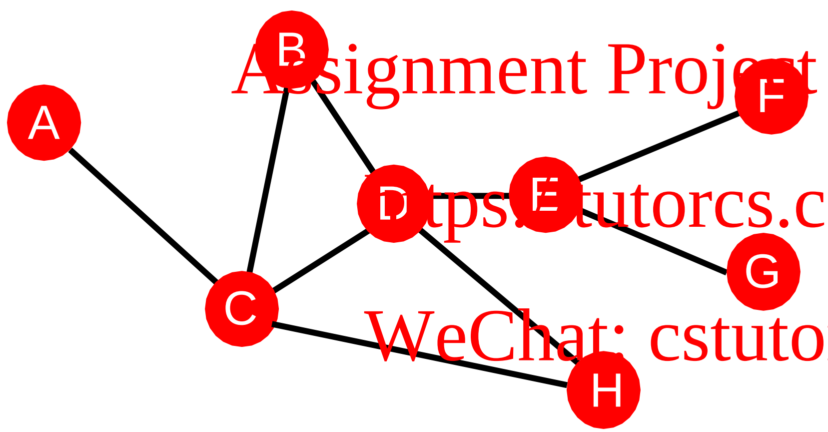
Link feature



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.

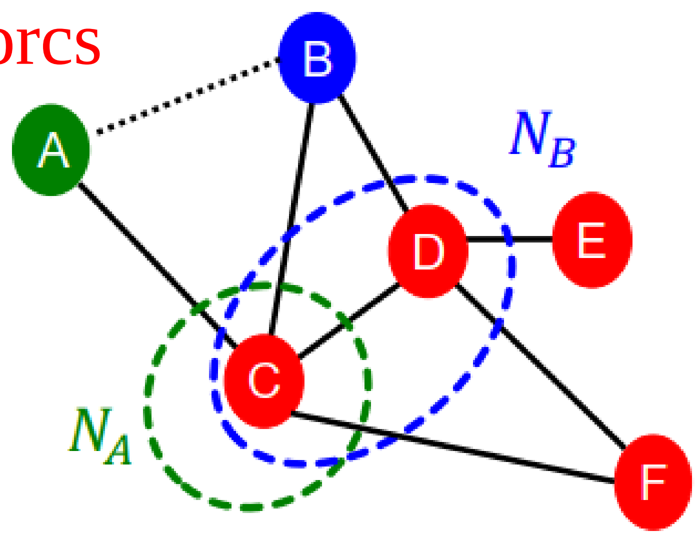
Local Neighborhood Overlap

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:

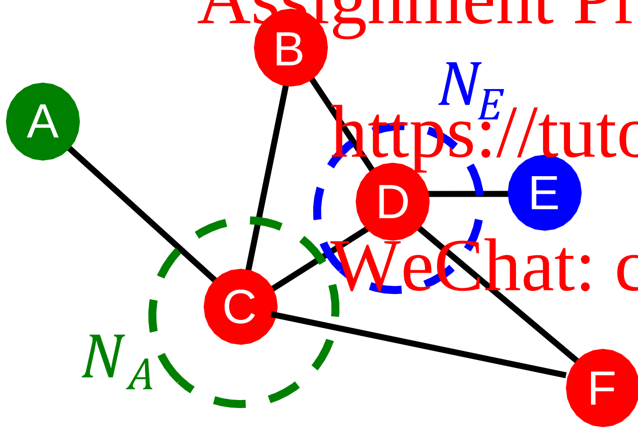
$$\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$$

  - 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 Overlap

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

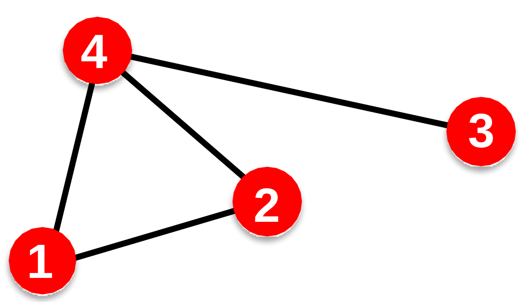
- **Q: How to compute #walks between two nodes?**

- Use **powers of the graph adjacency matrix**!

Intuition: Powers of Adj Matrices

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

$$\boldsymbol{P}_{\boldsymbol{12}}^{(1)} = \boldsymbol{A}_{\boldsymbol{12}}$$



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

Intuition: Powers of Adj Matrices

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

Node 1's neighbors

#walks of length 1 between Node 1's neighbors and Node 2

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

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

Power of adjacency

Global Neighborhood Overlap

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

- How to compute #walks between two nodes?
- Use **adjacency matrix powers**!

  - $\boldsymbol{A}_{uv}$ specifies #walks of length 1 (direct neighborhood) between u and v.

  - $\boldsymbol{A}_{uv}^{\mathbf{2}}$ specifies #walks of **length 2** (neighbor of neighbor) between u and v.
  - And, $\boldsymbol{A}_{uv}^{\boldsymbol{l}}$ specifies #walks of **length $\boldsymbol{l}$**.

Global Neighborhood Overlap

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

Sum over *all* *walk lengths*

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

#walks of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

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

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 #walks of all lengths between two nodes.

Learning Outcomes

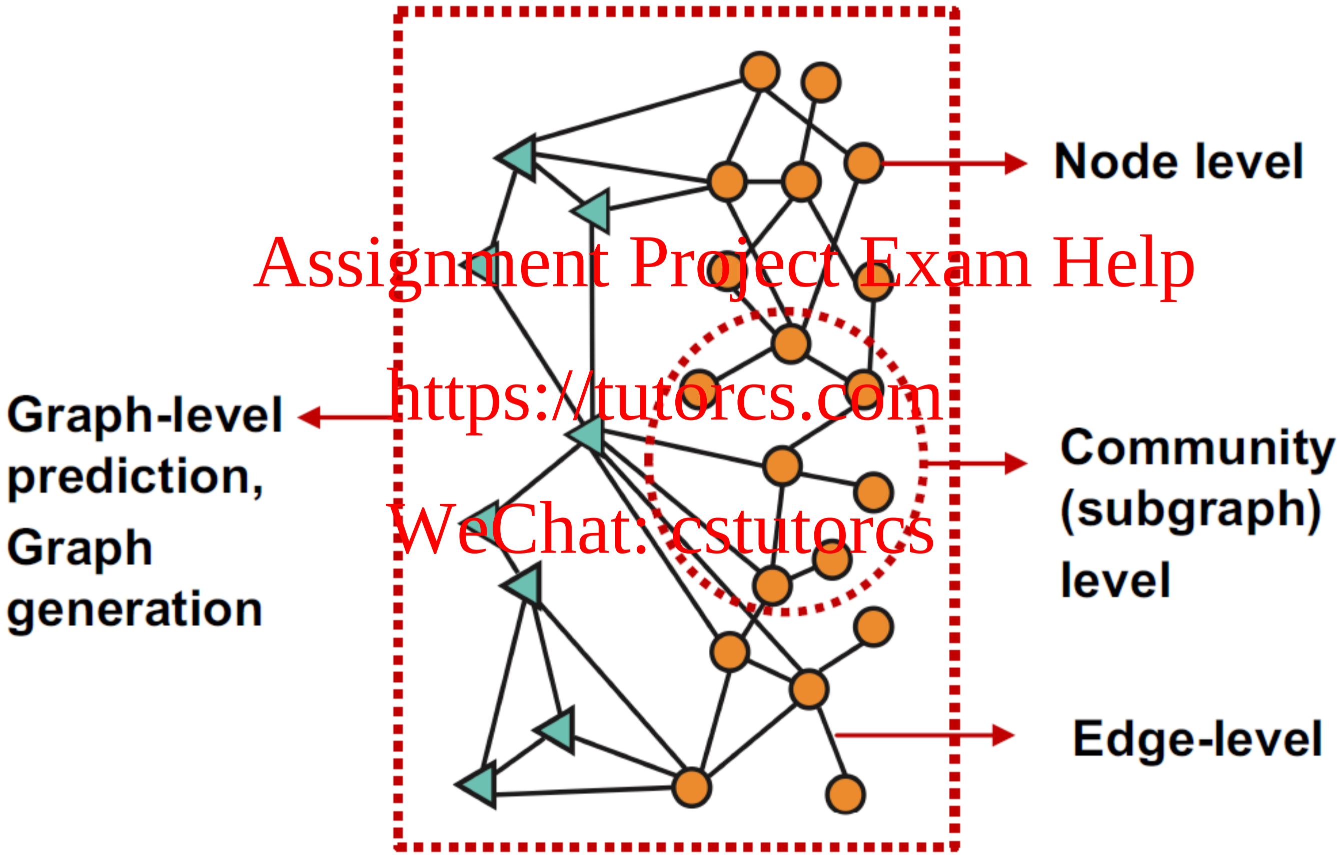
- **Traditional ML Pipeline**
  - Hand-crafted feature + ML model
- **Hand-crafted features for graph data**
  - **Node-level:**
    - Node degree, centrality, clustering coefficient, graphlets
  - **Link-level:**
    - Distance-based feature
    - local/global neighborhood overlap

Acknowledgement: Jure Leskovec, Stanford University

Classic Graph ML Tasks

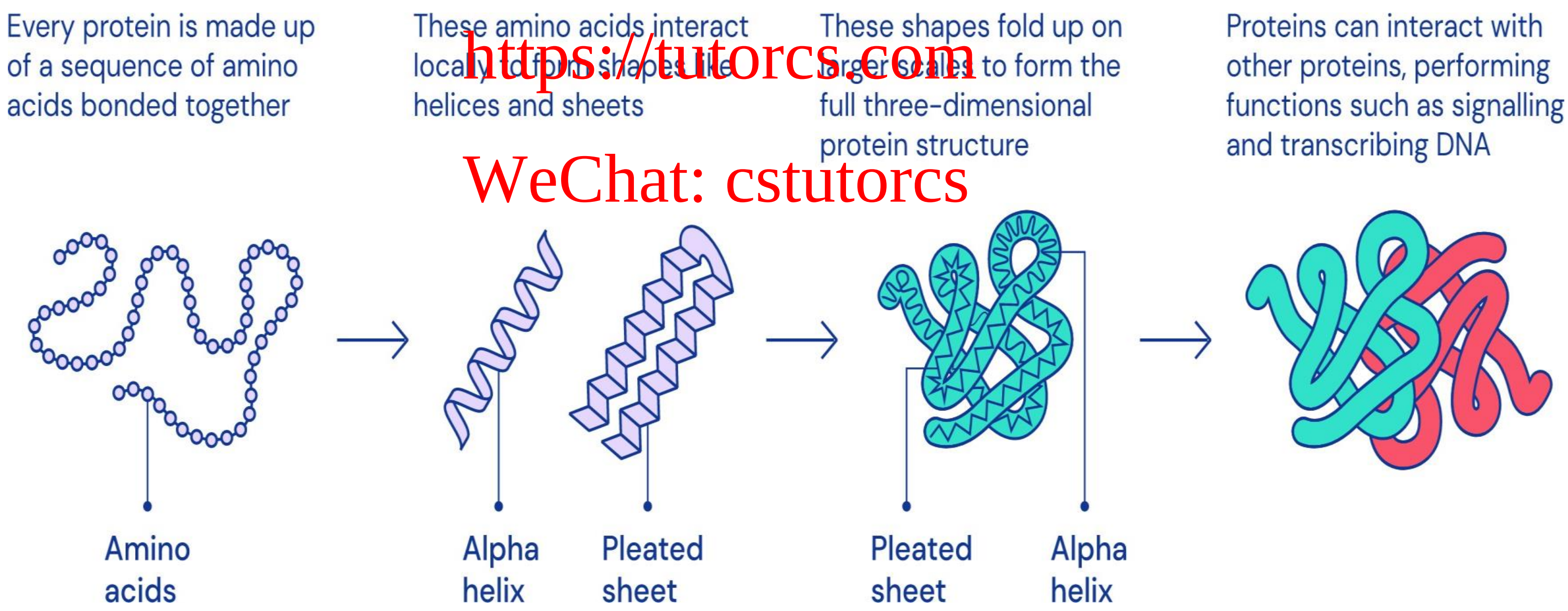
- **Node classification**:
  - Predict a property of a node
  - **Example:** Categorize online users / items
- **Link prediction**:
  - Predict whether there are missing links between two nodes
  - **Example:** Knowledge graph completion
- **Graph classification**:
  - Categorize different graphs
  - **Example:** Molecule property prediction

Different Types of Tasks



Protein Folding Problem

- Computationally **predict** a protein's 3D structure based solely on its **amino acid sequence**



AlphaFold

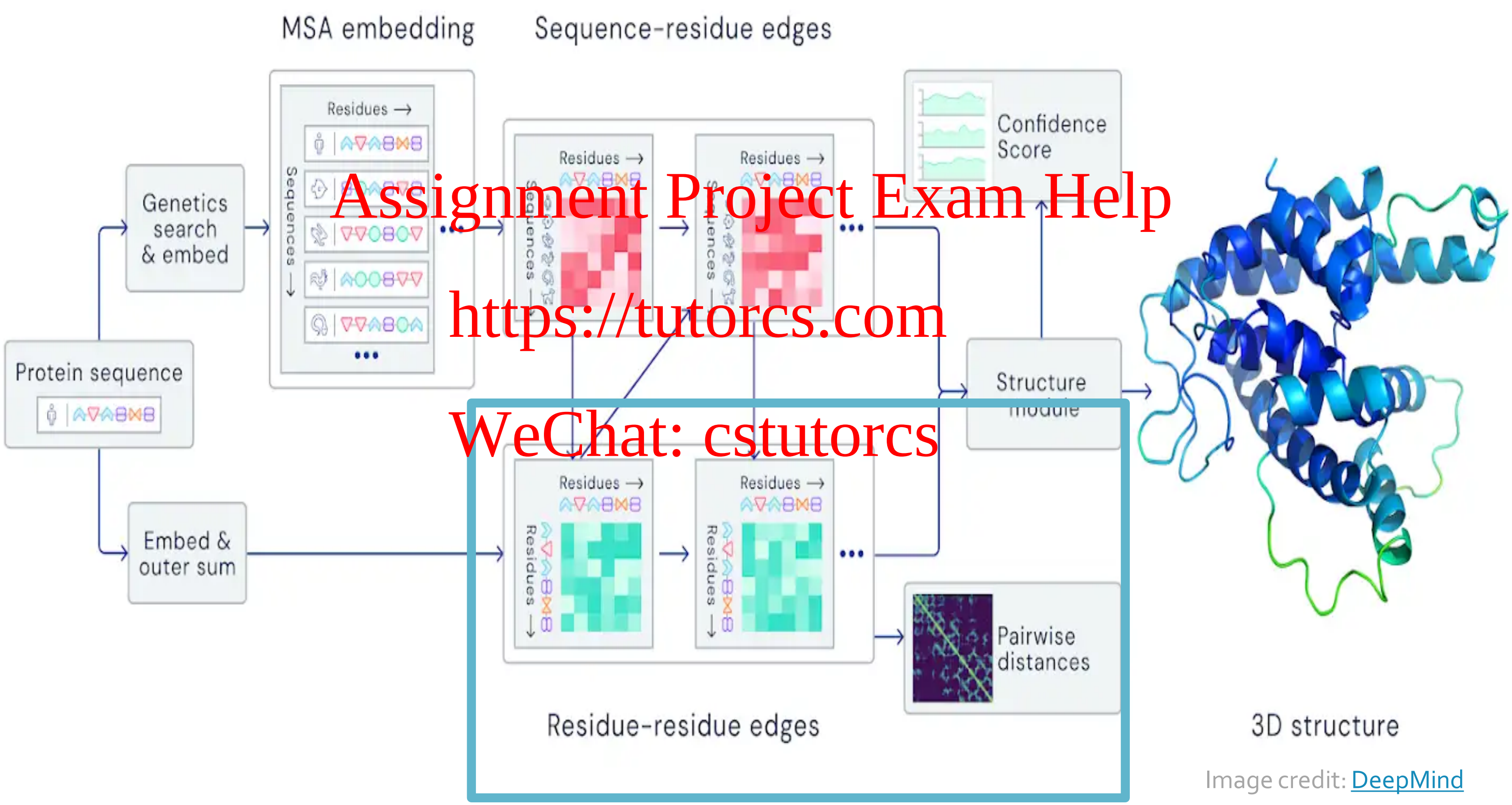


Image credit: DeepMind

Spatial graph

AlphaFold: Impact

Image credit: SingularityHub

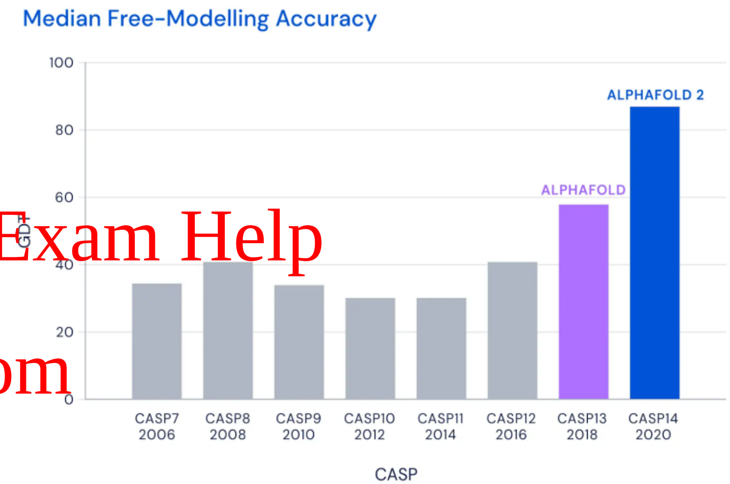


Image credit: DeepMind

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

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

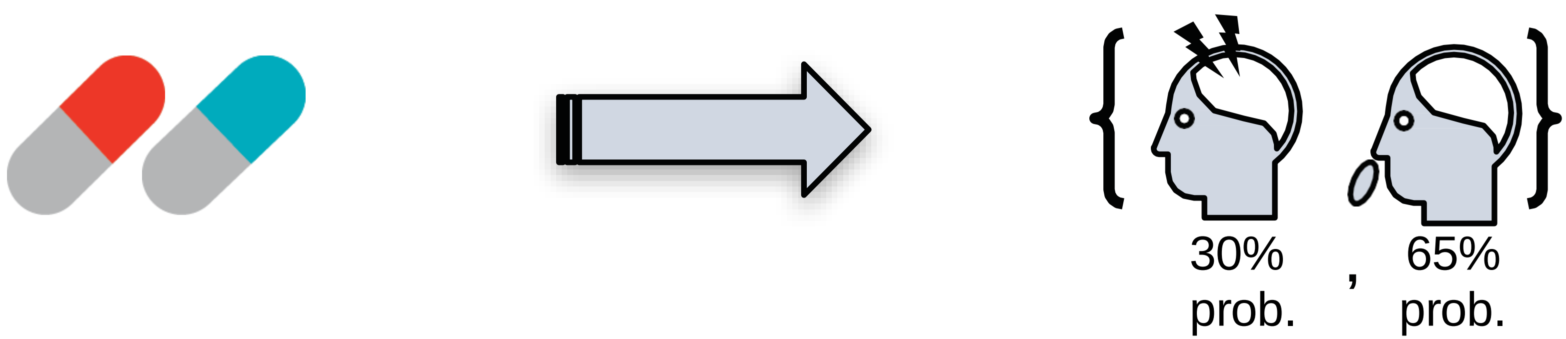
12-14-20

DeepMind's latest AI breakthrough could turbocharge drug discovery

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

Example 2: Drug Side Effects

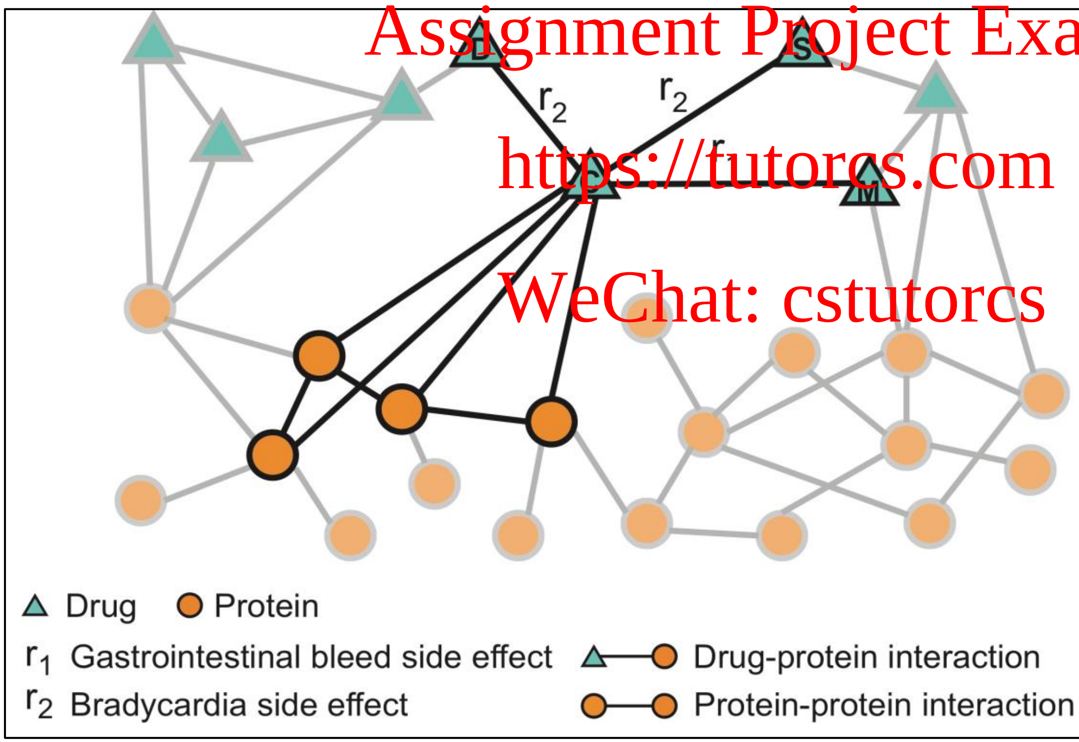
- Polypharmacy is the use of drug combinations and is commonly used for treating complex and terminal diseases.
  - 46% of people ages 70-79 take **more than 5** drugs
  - Many patients take more than 20 drugs to treat heart disease, depression, insomnia, etc.
- Despite its effectiveness in many cases, it poses high risks of adverse side effects.

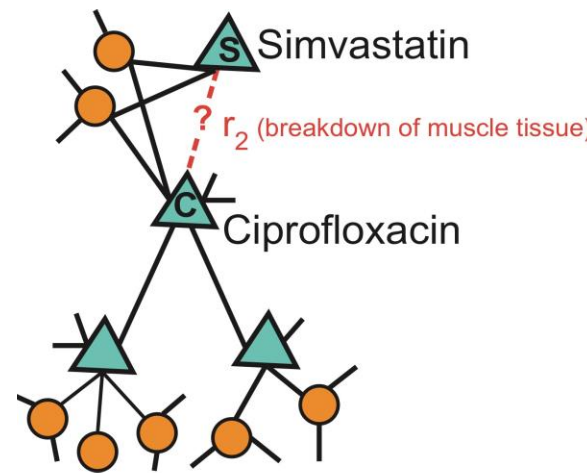


Link Prediction: Biomedical Graph

- **Nodes**: Drugs & Proteins
- **Edges**: Interactions

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





Link Prediction: Biomedical Graph

Rank	Drug c	Drug d	Side effect r	Evidence found
1	Pyrimethamine	Aliskiren	Sarcoma	Stage *et al.* 2015
2	Tigecycline	Bimatoprost	Autonomic neuropathy	
3	Omeprazole	Dacarbazine	Telangiectases	
4	Tolcapone	Pyrimethamine	Breast disorder	Bicker *et al.* 2017
5	Minoxidil	Paricalcitol	Cluster headache	
6	Omeprazole	Amoxicillin	Renal tubular acidosis	Russo *et al.* 2016
7	Anagrelide	Azelaic acid	Cerebral thrombosis	
8	Atorvastatin	Amlodipine	Muscle inflammation	Banakh *et al.* 2017
9	Aliskiren	Tioconazole	Breast inflammation	Parving *et al.* 2012
10	Estradiol	Nadolol	Endometriosis	

Case Report

Severe Rhabdomyolysis due to Presumed Drug Interactions between Atorvastatin with Amlodipine and Ticagrelor