IT5429E-1-24 (24.1A01)(Fall 2024): Graph Analytics for Big Data

Week 1: Introduction, Traditional Methods

Instructor: Thanh H. Nguyen

Many slides are adapted from https://web.stanford.edu/class/cs224w/

Course Information

- Course forum: https://join.slack.com/t/bk-soict-mlwithgraphs/shared_invite/zt-2nvsln7mc-JRcLScSfenC0mEmO6okH9A

- Instructor: Thanh H. Nguyen (nguyenhongthanh87@gmail.com)

- Coursework:
  - One programming project

Course Outline

- Machine Learning and Representation Learning for graph data:
  - Traditional ML methods for graphs
  - Methods for node embeddings
  - Graph neural networks
  - Graph Transformers
  - Knowledge graphs
  - Generative models for graphs
  - Scaling up to large graphs
  - Applications

Course Outline (Tentative)

Week	Topics
1	Introduction, Traditional Methods
2	Node Embedding, Link Analysis
3	Graph Neural Nets: Part 1
4	Graph Neural Nets: Part 2
5	Label Propagation, Heterogeneous Graphs
6	Knowledge Graphs
7	Subgraph Matching, GNN for Recommendations
8	Deep Generative Models, Advanced Topics
9	Graph Transformer, Scaling Up GNNs
10	Selective Topics
11	Class Projects
12	Class Projects

Prerequisites

- Background
  - Machine learning
  - Algorithms
  - Probability and statistics

- Programming
  - Write non-trivial Python programs
  - Familiar with Pytorch

Graph Machine Learning Tools

- PyG
  - Link: https://www.pyg.org
  - A library for Graph Neural Networks

- GraphGym
  - Link: https://github.com/snap-stanford/GraphGym
  - Platform for designing Graph Neural Networks.
  - This platform is now supported in PyG

- Other network analytics tools: SNAP.PY, NetworkX

Course Logistics

- Class time:
  - Weeks 1-2: Wednesdays (1:30pm – 5pm)
  - Weeks 3-End: Tuesdays (8am – 11:30am)
- All lecture slides and class discussions will be posted and held on Slack accordingly.
- Lecture structures
  - Two parts per class: approximately 90 minutes each part
  - 15 minutes for coffee break
  - 15 minutes for Q&A and discussions.

Course Logistics

- Readings
  - Book: Graph Representation Learning https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf
  - Research papers

Grading

- Class participation (10%)
  - Attendance: each student is allowed to be absent from class <= two times
  - Engagement (actively join discussions, Q&A)

- Class project (90%)

Class Projects: Real-world Applications of GNNs

- Determine a specific use case (e.g., fraud detection)
- Demonstrate how GNNs + PyG can be used to solve the problem
- Groups of two students
  - Message me on Slack

Class Projects: Tasks

- Identify an appropriate public dataset
- Formulate the use case as a clear graph ML problem
- Demonstrate how GNNs can be used to solve this problem.

- Important note:
  - It is not enough to simply apply an existing GNN model to a new dataset
  - Students are expected to provide some form of novelty: for example,
    - Develop a new GNN method for a specific problem, improving on existing methods in a non-trivial way,
    - Or comprehensive analyses (ablation studies, comparison between multiple model architectures) for the project.

Class Projects: Examples

- Example graphs and datasets
  - Open Graph Benchmark: https://ogb.stanford.edu
  - Datasets available in PyG

- Application examples:
  - Recommender systems
  - Fraud detection in transaction graphs
  - Friend recommendation
  - Paper citation graphs
  - Author collaboration networks
  - Heterogeneous academic graphs
  - Knowledge graphs
  - Drug-drug interaction networks

Class Projects: Components

- Project proposal (10%)
- Project report (60%)
- Project presentation (20%)

Class Projects: Project Proposal (10%)

- Application domain
  - Which dataset are you planning to use?
  - Describe the dataset/task/metric.
  - Why did you choose the dataset?
- Graph ML techniques that you want to apply
  - Which graph ML model are you planning to use?
  - Describe the model(s) (try using figures and equations).
  - Why is/are the model(s) appropriate for the dataset you have chosen?
- Submission:
  - Deadline: By the end of Week 4 (proposal file + student names)
  - 2-3 pages in pdf.
  - Format: NeurIPS2024 LaTeX style
  - Message me the file on the class Slack channel.

Class Projects: Project report (60%)

- Writing (40 points): 10-15 pages
  - Motivation & explanation of data/task (9 points)
  - Appropriateness & explanation of model(s) (9 points)
  - Insights + results (9 points)
  - Figures (9 points)
  - Code snippets (4 points)
- Submission:
  - Deadline: by the end of Week 12
  - Format: NeurIPS2024 LaTeX style
  - Message me the file on the class Slack channel.
- Colab (20 points)
  - Code: correctness, design (10 points)
  - Documentation: class/function descriptions, comments in code (10 points)

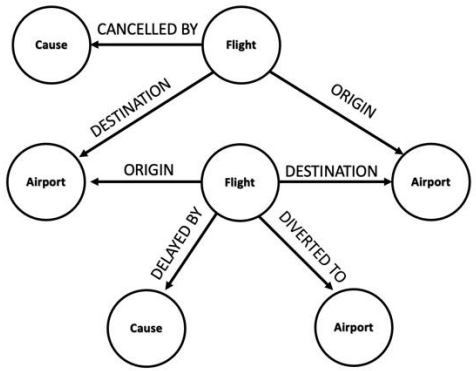
Class Projects: Project presentation (20%)

- Present at class with Q&A
- Time: 30 minutes

Machine Learning with Graphs: Why Graphs?

Graphs are a general language for describing *and* analyzing entities with relations/interactions

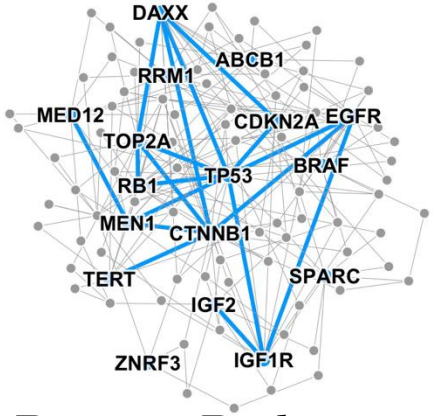
Many Types of Data are Graphs



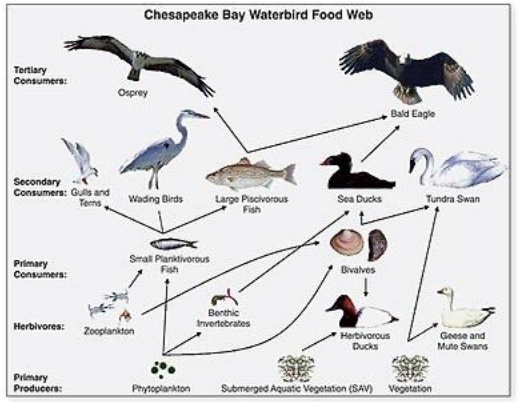
Event Graphs

Computer Networks

:

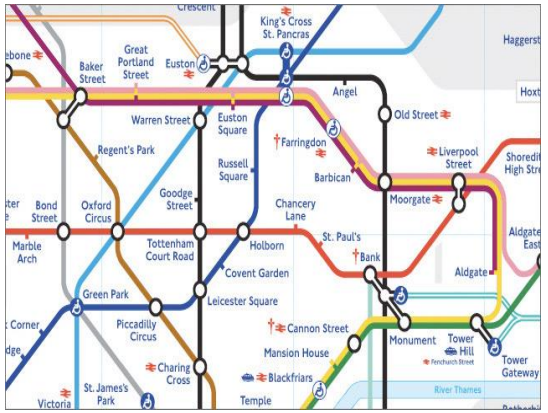


Disease Pathways



Food Webs

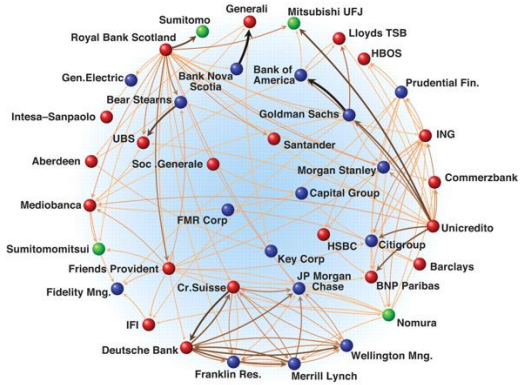
Particle Networks



Underground Networks

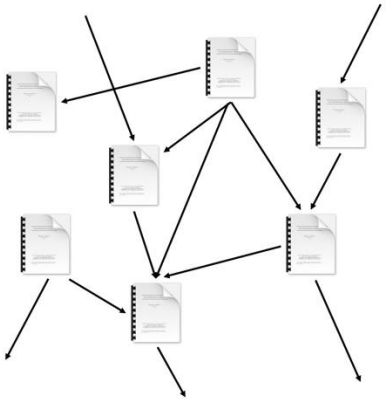
Many Types of Data are Graphs

Social Networks



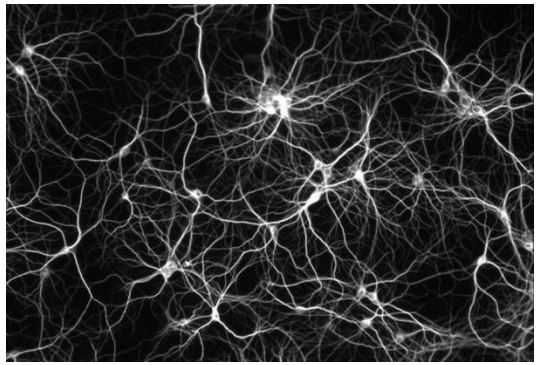
Economic Networks

Communication Networks



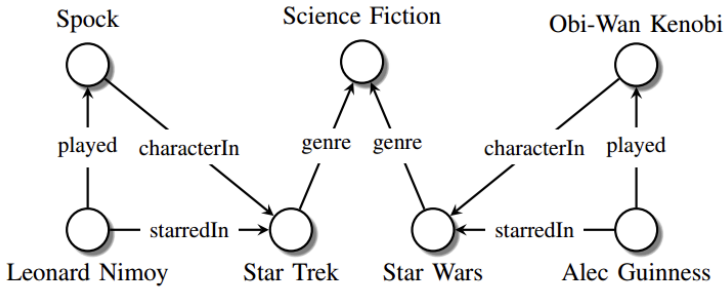
Citation Networks

Internet

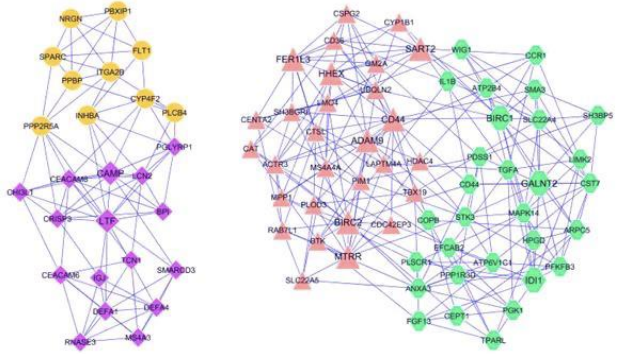


Networks of Neurons

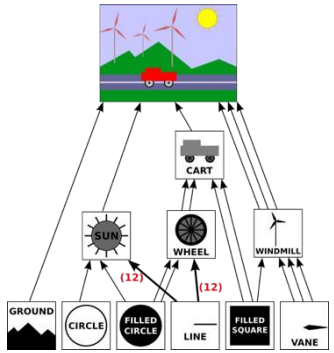
Many Types of Data are Graphs



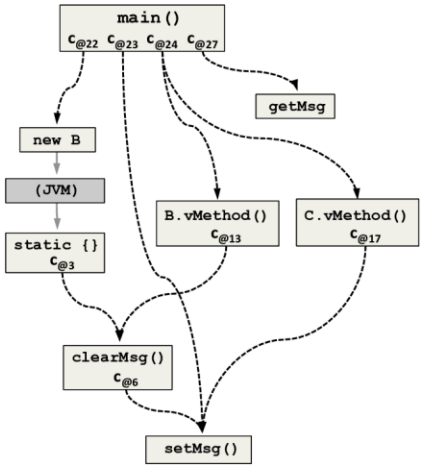
Knowledge Graphs



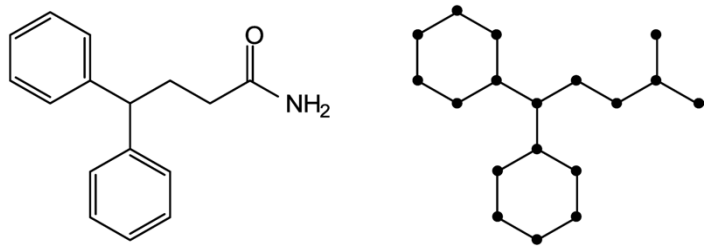
Regulatory Networks



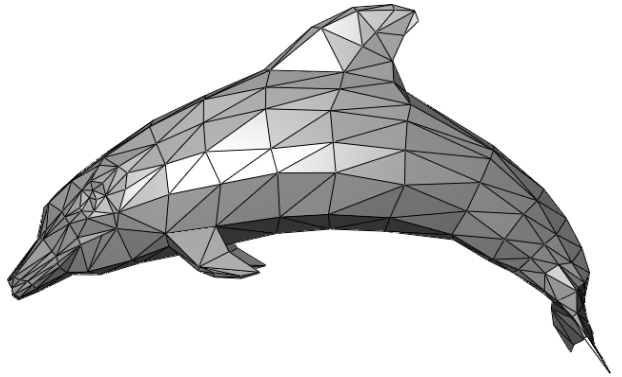
Scene Graphs



Code Graphs



Molecules



3D Shapes

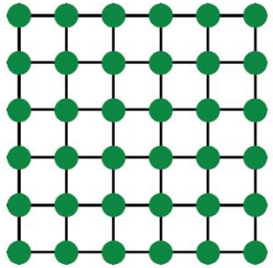
Graphs: Machine Learning

- Complex domains have a rich relational structure, which can be represented as a relational graph

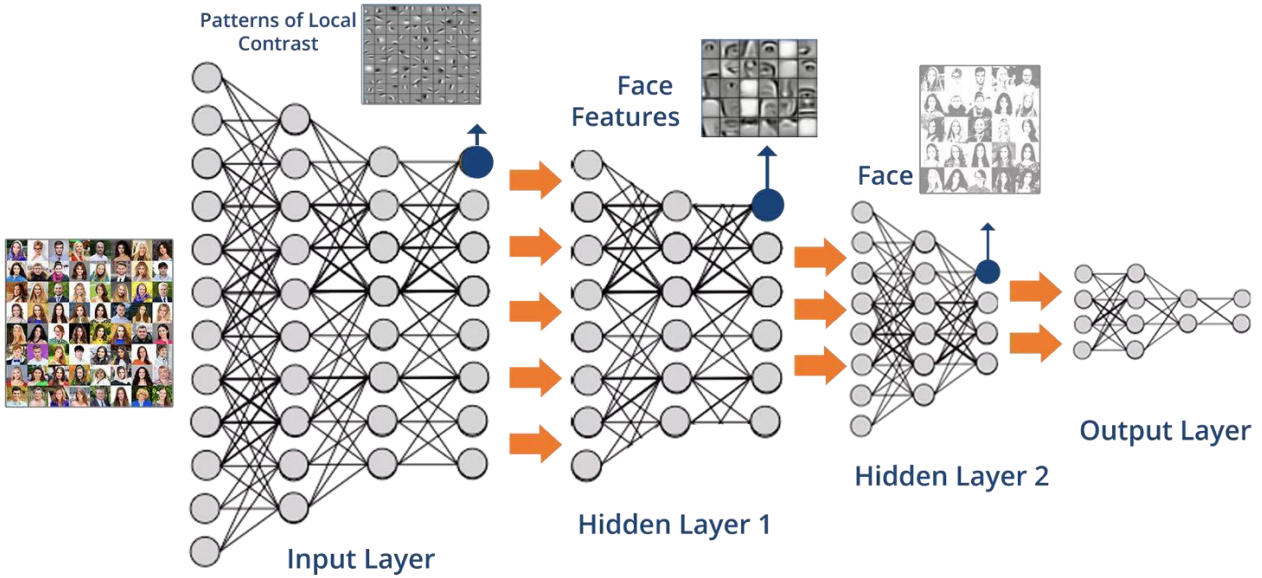
- By explicitly modeling relationships we achieve better performance!

- Main question: How do we take advantage of relational structure for better prediction?

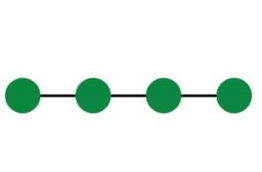
Today: Modern ML Toolbox

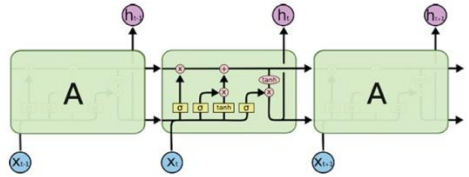


Images



Text/Speech





Modern deep learning toolbox is designed for simple sequences and grids

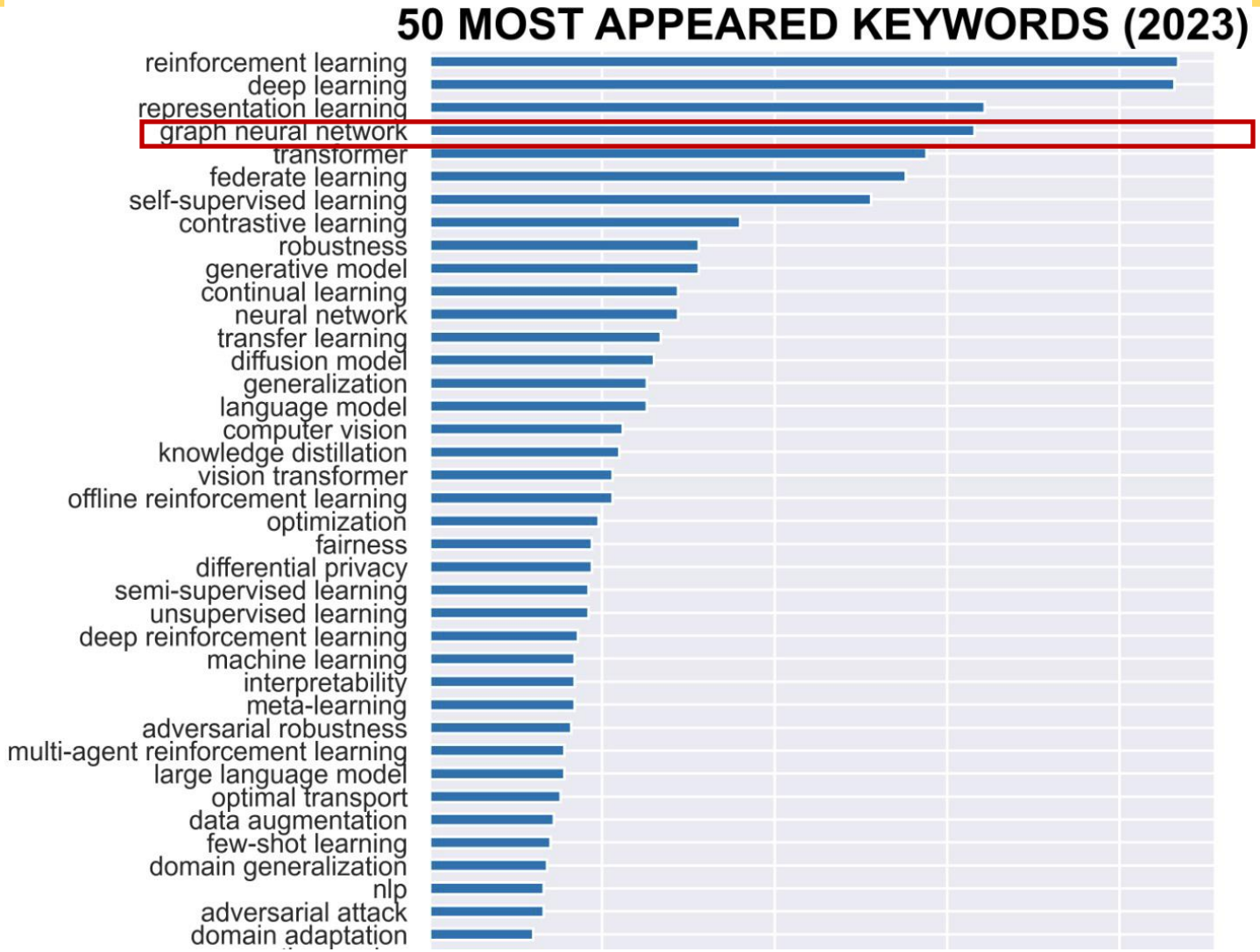
This Course

How can we develop neural networks that are much more broadly applicable?

Graphs are the new frontier of deep learning

Hot Subfield in Machine Learning

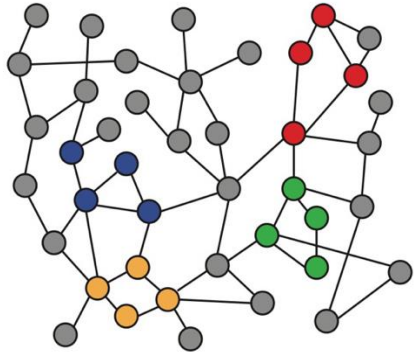
ICLR 2023 keywords



Why is Graph Deep Learning Hard?

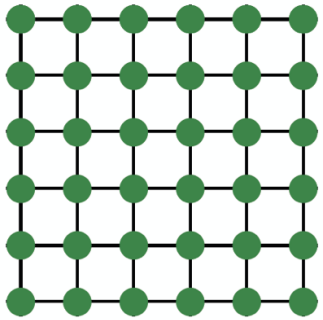
Networks are complex

- Arbitrary size and complex topological structure (i.e., no spatial locality like grids)



Networks

vs



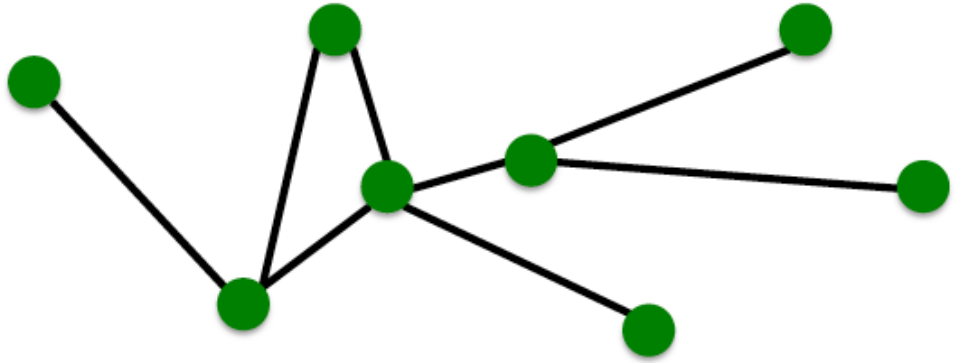
Images

Texts

- No fixed node ordering or reference point
- Often dynamic and have multimodal features

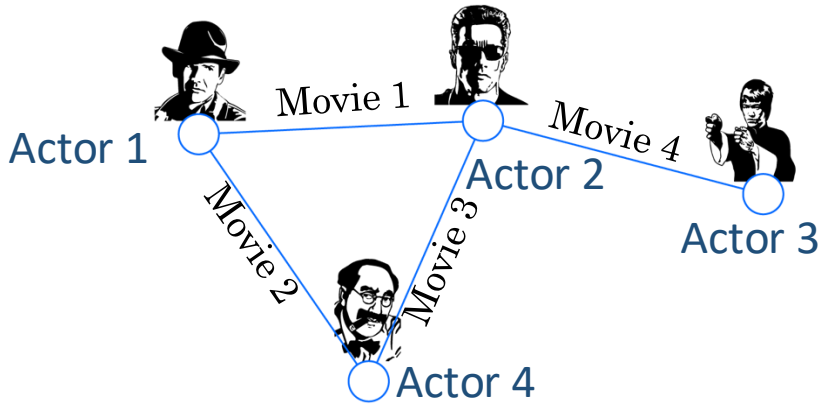
Choices of Graph Representation

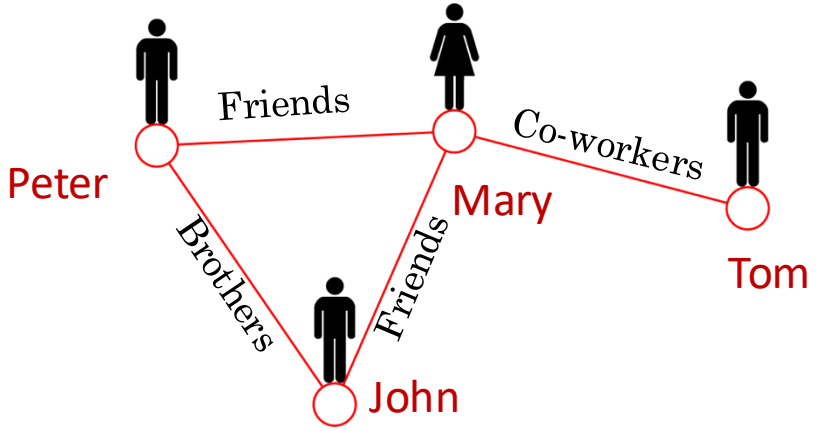
Components of A Network

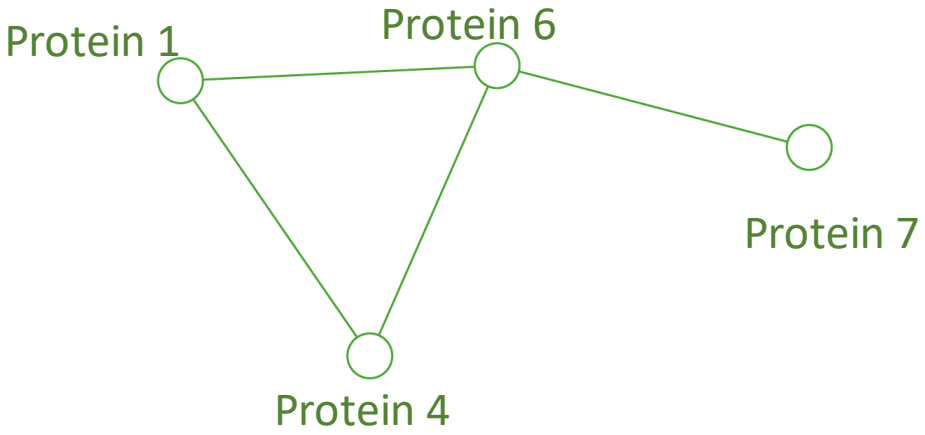


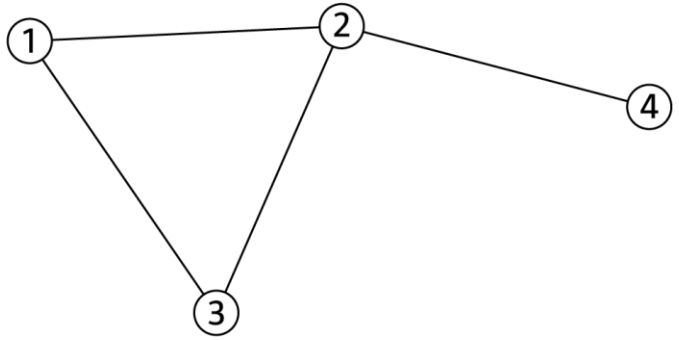
- Objects: nodes, vertices
- Interactions: edges, links
- Systems: graphs, networks

Graphs: A Common Language





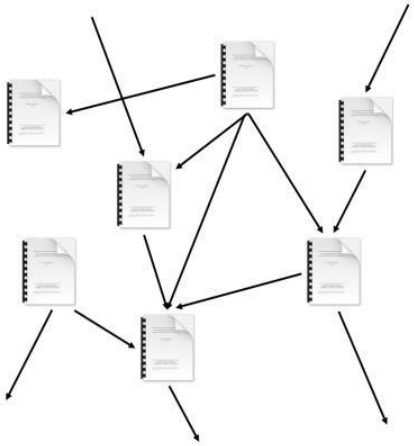




Choosing a Proper Representation

- If you connect individuals who work with each other, you will explore a professional network

- If you connect scientific papers that cite each other, you will be studying the citation network



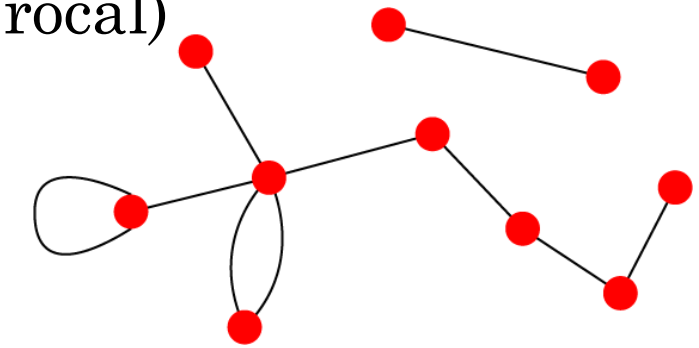
How to Define a Graph

- How to build a graph
  - What are nodes?
  - What are edges?

- Choice of proper network representation of a given domain/problem determines ability to use networks successfully
  - In some cases, representation is unique and unambiguous
  - In other cases, representation is by no mean unique
  - The way you assign links will determine the nature of the questions you study.

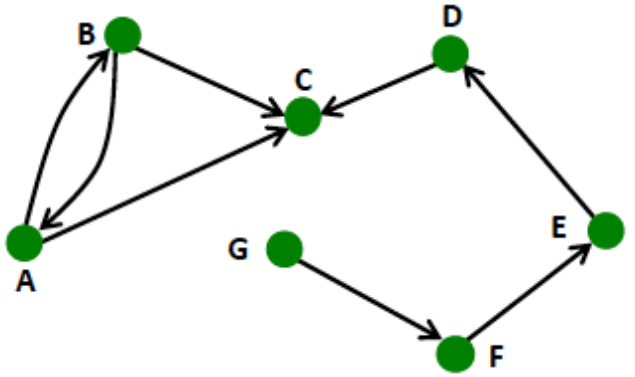
Directed and Undirected Graphs

- Undirected
  - Links: undirected (symmetrical, reciprocal)



  - Examples:
    - Collaborations
    - Friendships on Facebook

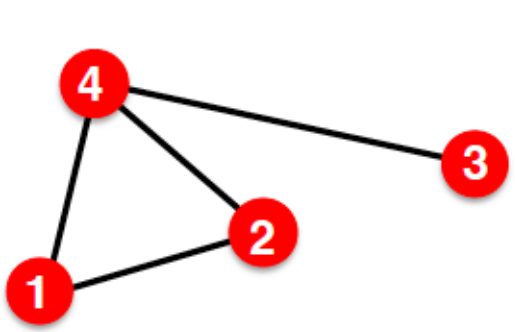
- Directed
  - Links: directed

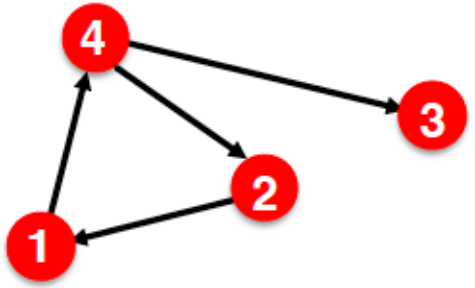


  - Examples:
    - Phone calls
    - Following on Twitter (X)

- Other considerations
  - Weights
  - Properties
  - Types
  - Attributes

Representing Graphs: Adjacency Matrix



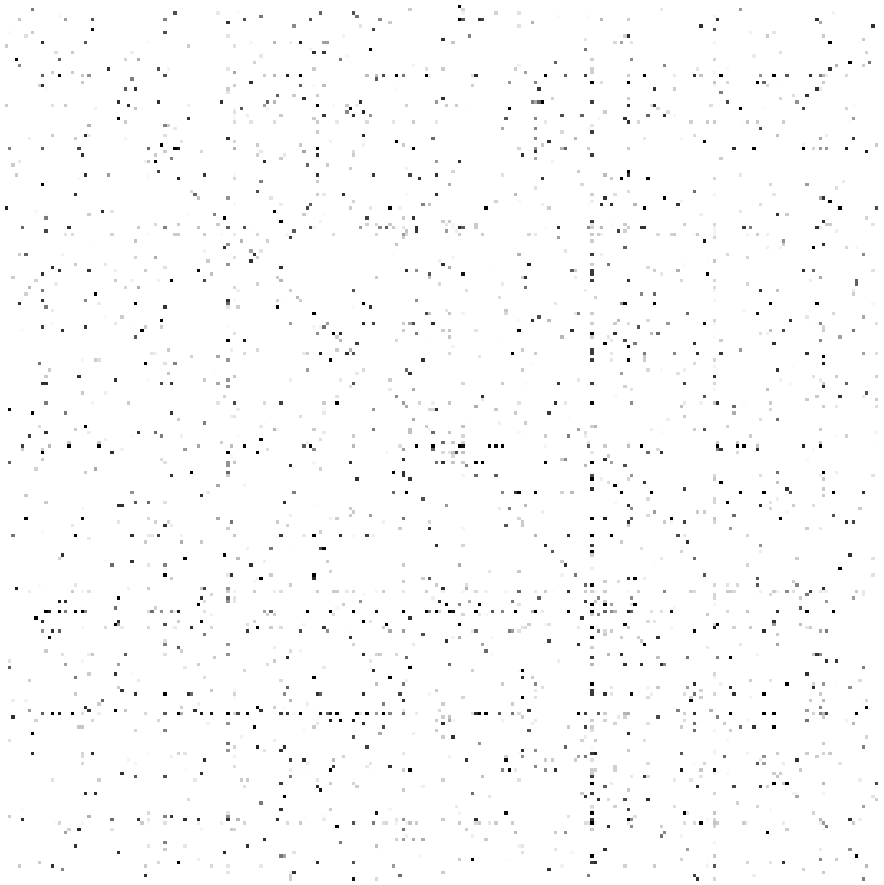


$A_{ij} = 1$ if there is a link from node i to node j

$A_{ij} = 0$ otherwise

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

Adjacency Matrices are Sparse



Networks are Sparse Graphs

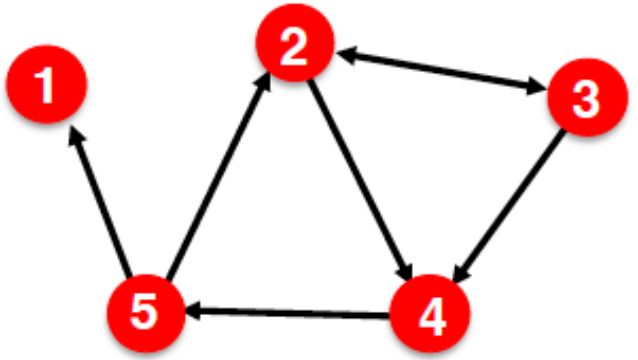
Most real-world networks are sparse

NETWORK	NODES	LINKS	DIRECTED/ UNDIRECTED	N	L	<k>
Internet	Routers	Internet connections	Undirected	192,244	609,066	6.33
WWW	Webpages	Links	Directed	325,729	1,497,134	4.60
Power Grid	Power plants, transformers	Cables	Undirected	4,941	6,594	2.67
Phone Calls	Subscribers	Calls	Directed	36,595	91,826	2.51
Email	Email Addresses	Emails	Directed	57,194	103,731	1.81
Science Collaboration	Scientists	Co-authorship	Undirected	23,133	93,439	8.08
Actor Network	Actors	Co-acting	Undirected	702,388	29,397,908	83.71
Citation Network	Paper	Citations	Directed	449,673	4,689,479	10.43
E. Coli Metabolism	Metabolites	Chemical reactions	Directed	1,039	5,802	5.58
Protein Interactions	Proteins	Binding interactions	Undirected	2,018	2,930	2.90

Consequences: Adjacency matrix is filled with zeros

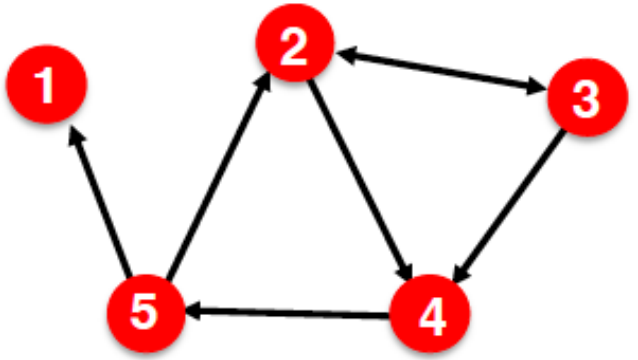
Representing Graphs: Edge List

- Representing graph as a list of edges
  - (2, 3)
  - (2, 4)
  - (3, 2)
  - (3, 4)
  - (4, 5)
  - (5, 2)
  - (5, 1)



Representing Graphs: Adjacency List

- Adjacency list
  - Easier to work with if network is
    - Large
    - Sparse
  - Allow us to quickly retrieve all neighbors of a given node
    - 1:
    - 2: 3, 4
    - 3: 2, 4
    - 4: 5
    - 5: 1, 2

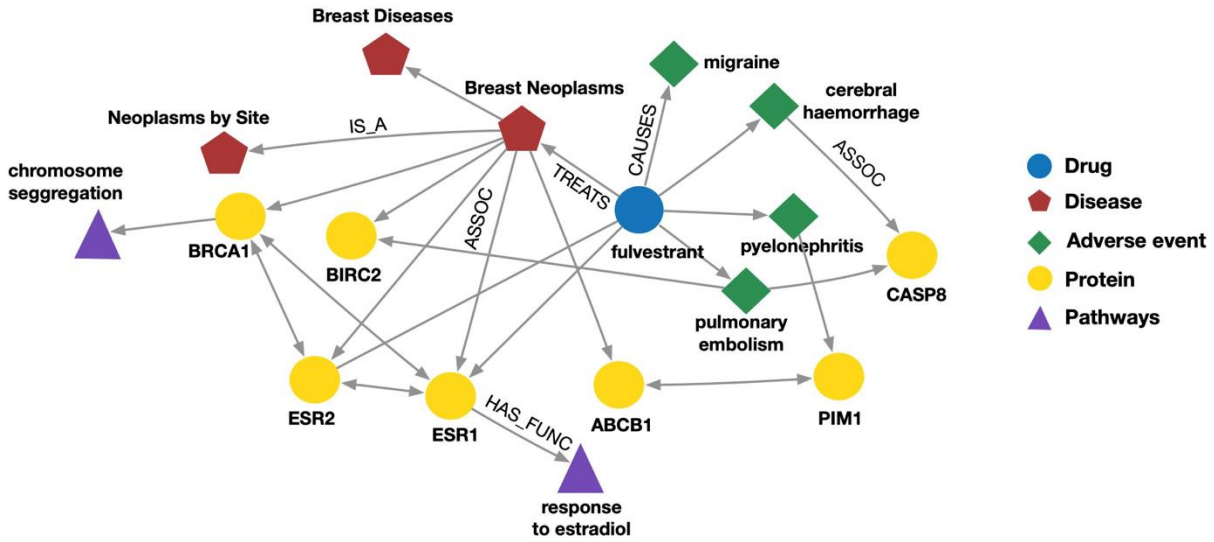


Heterogeneous Graphs

- A heterogeneous graph is defined as: $G = (V, E, R, T)$

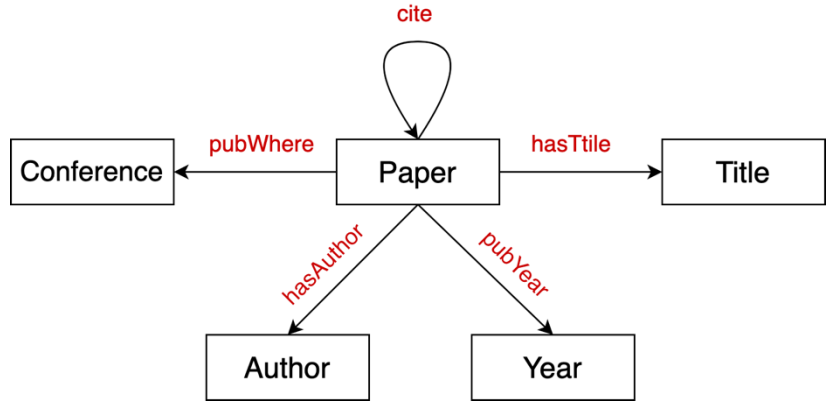
- Nodes with node types: $v_i \in V$
- Edges with relation types: $(v_i, r, v_j) \in E$
- Node type: $T(v_i)$
- Relation type: $r \in R$
- Nodes and edges have attributes/features

Many Graphs are Heterogeneous



Biomedical Knowledge Graphs

- Example node: Migraine
- Example edge: (fulvestrant, Treats, Breast Neoplasms)
- Example node type: Protein
- Example edge type (relation): Causes

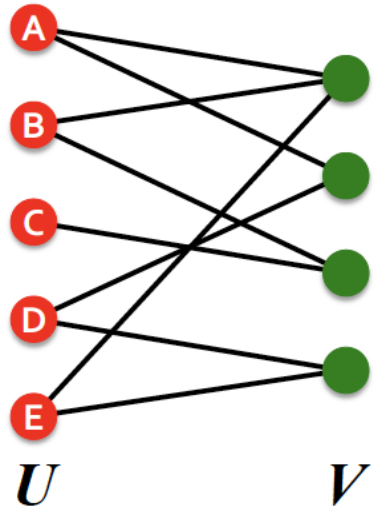


Academic Graphs

- Example node: ICML
- Example edge: (GraphSAGE, NeurIPS)
- Example node type: Author
- Example edge type (relation): pubYear

Bipartite Graph

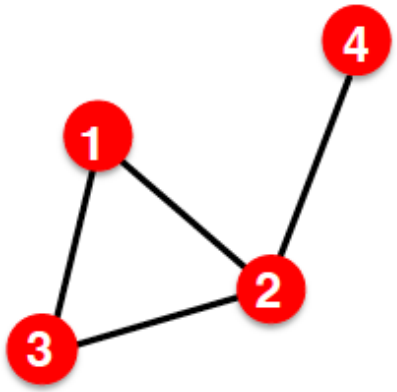
- Nodes can be divided into two disjoint sets U and V
  - Every link connects a node in U to one in V
  - U and V are independent sets



- Examples:
  - Authors-to-Papers (they authored)
  - Actors-to-Movies (they appeared in)
  - Users-to-Movies (they rated)
  - Recipes-to-Ingredients (they contain)

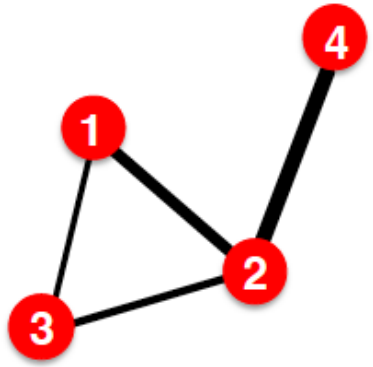
More Types of Graphs

- Unweighted



$$A_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

- Weighted

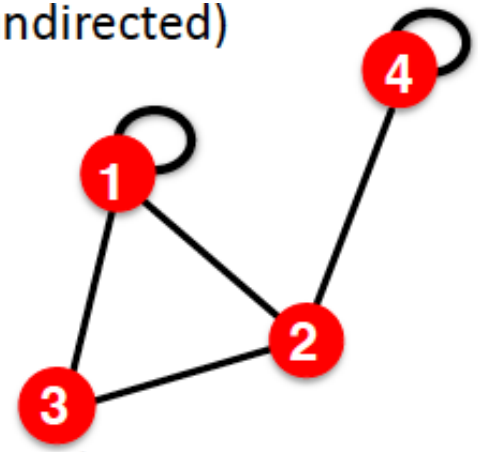


$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

More Types of Graphs

- Self-edges (self-loop)

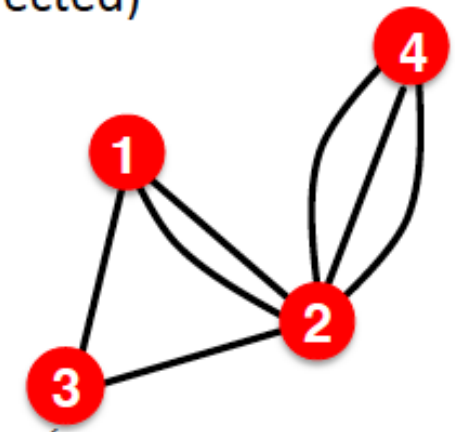
(undirected)



$$A_{ij} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

- Multi-graph

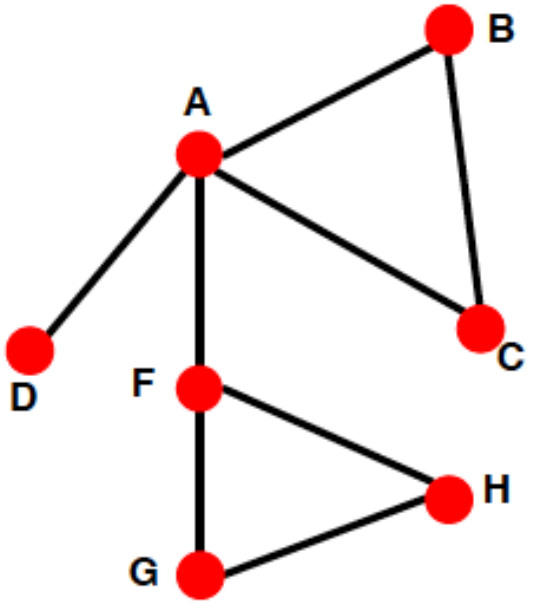
(undirected)

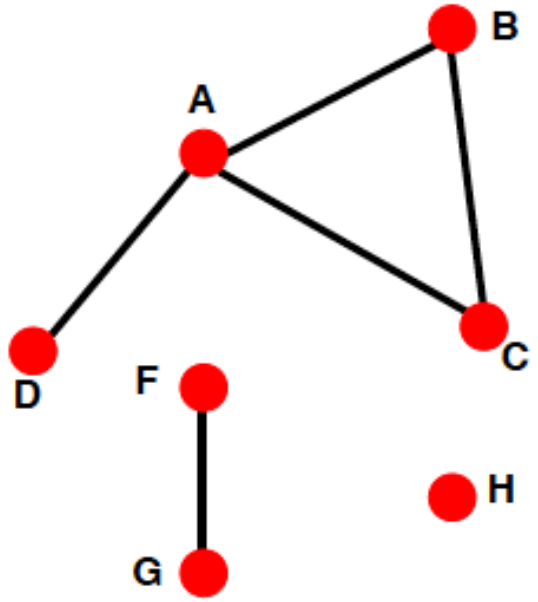


$$A_{ij} = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 3 \\ 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{pmatrix}$$

Connectivity of Undirected Graphs

- Connected undirected graph
  - Any two vertices can be joined by a path
- A disconnected graph consists of two or more connected components

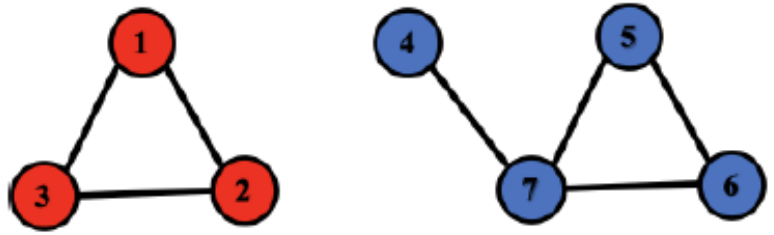




Connectivity Example

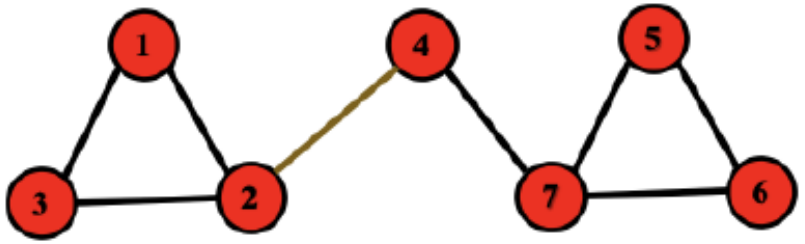
- The adjacency matrix of a network with several components can be written in a block-diagonal form, so that nonzero elements are confined to squares, with all other elements being zero:

Disconnected



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

Connected

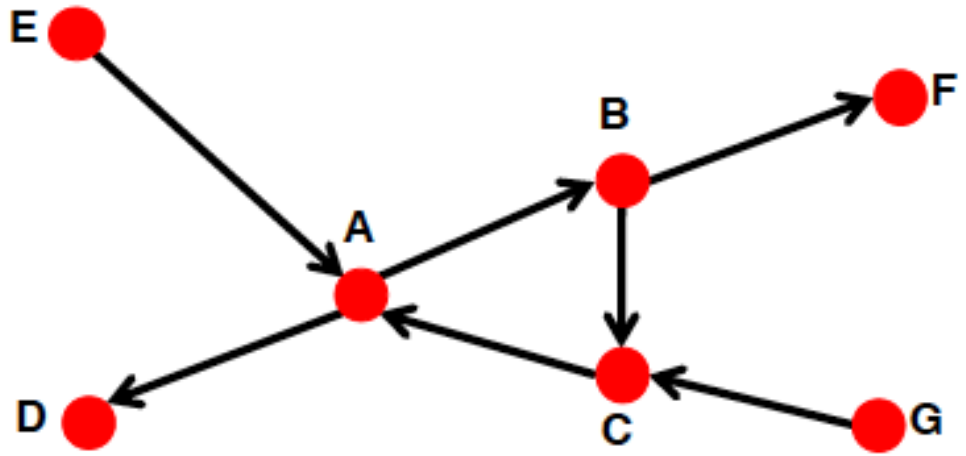


$$\begin{pmatrix} 0&1&1&0&0&0&0\\ 1&0&1&1&0&0&0\\ 1&1&0&0&0&0&0\\ 0&1&0&0&0&0&1\\ 0&0&0&0&0&1&1\\ 0&0&0&0&1&0&1\\ 0&0&0&1&1&1&0 \end{pmatrix}$$

Connectivity of Directed Graphs

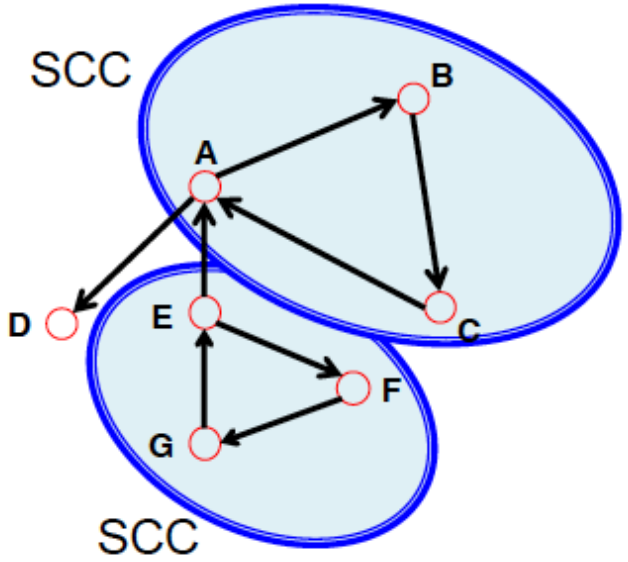
- Strongly connected directed graphs
  - Have paths from each node to every other nodes

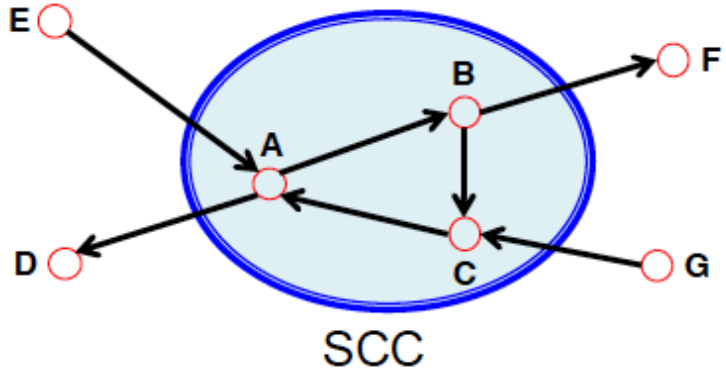
- Weakly connected directed graphs
  - Is connected if we disregard edge directions
  - Example:



Connectivity of Directed Graphs

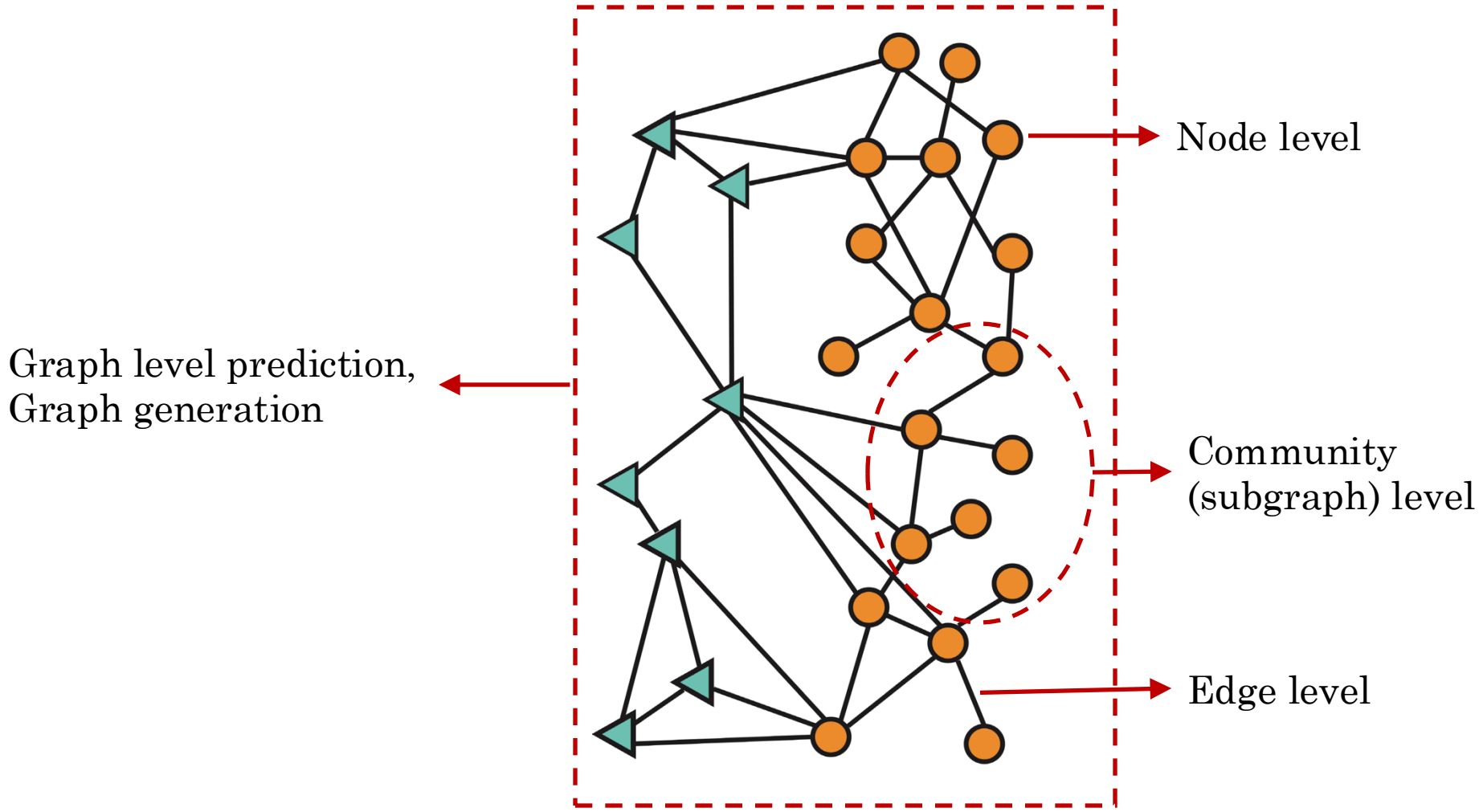
- Strongly connected components (SCCs): not every node is part of a non-trivial SCC



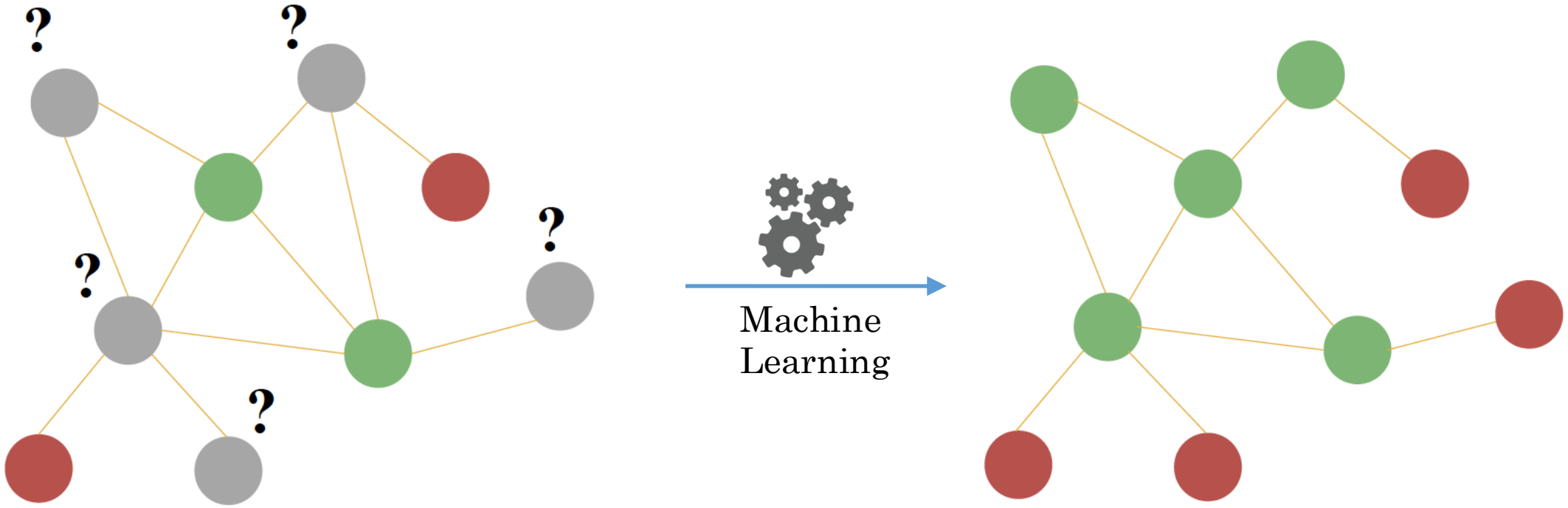


Applications of Graph ML

Different Types of Tasks



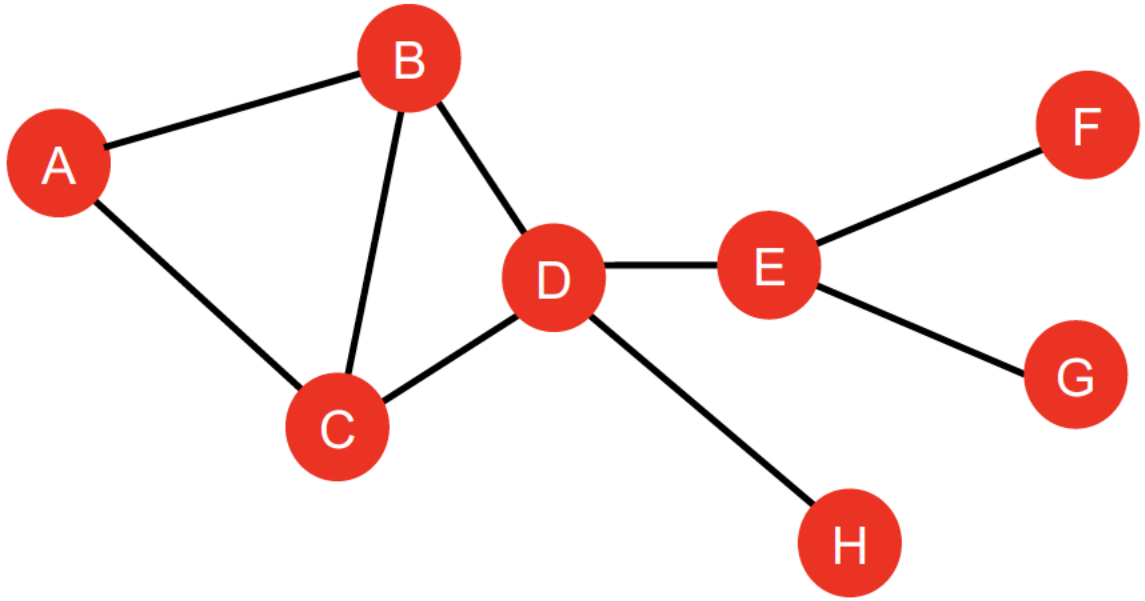
Node-Level Tasks



Node Classification

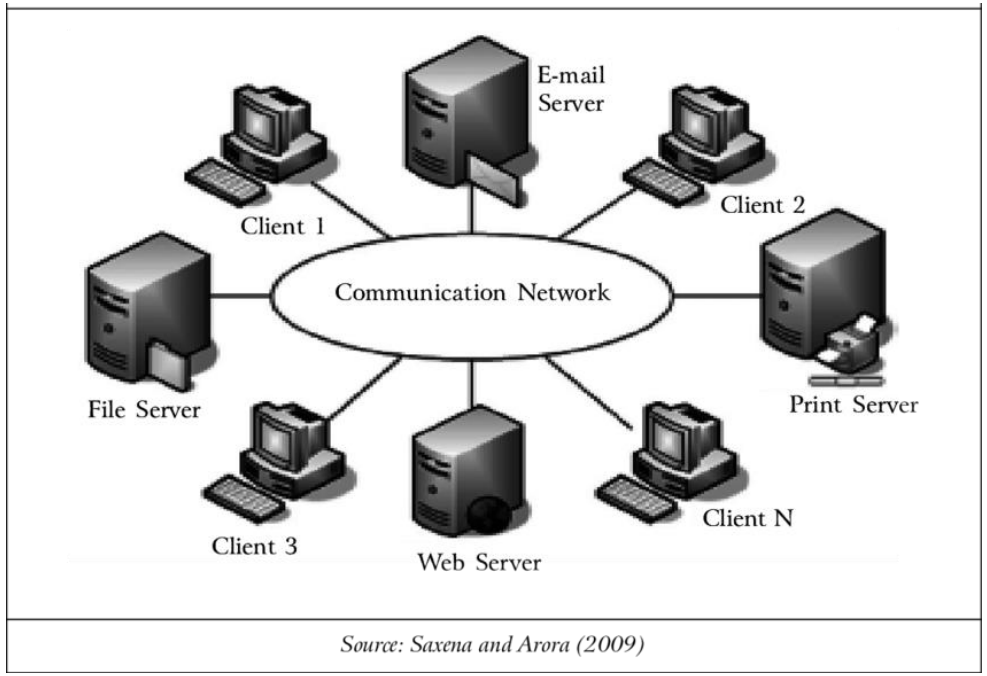
Node-level Network Structure

- Goal: Characterize the structure and position of a node in the network
  - Node degree
  - Node importance and position
    - E.g., Number of shortest paths passing through a node
    - E.g., Avg. shortest path length to other nodes
  - Substructures around the node



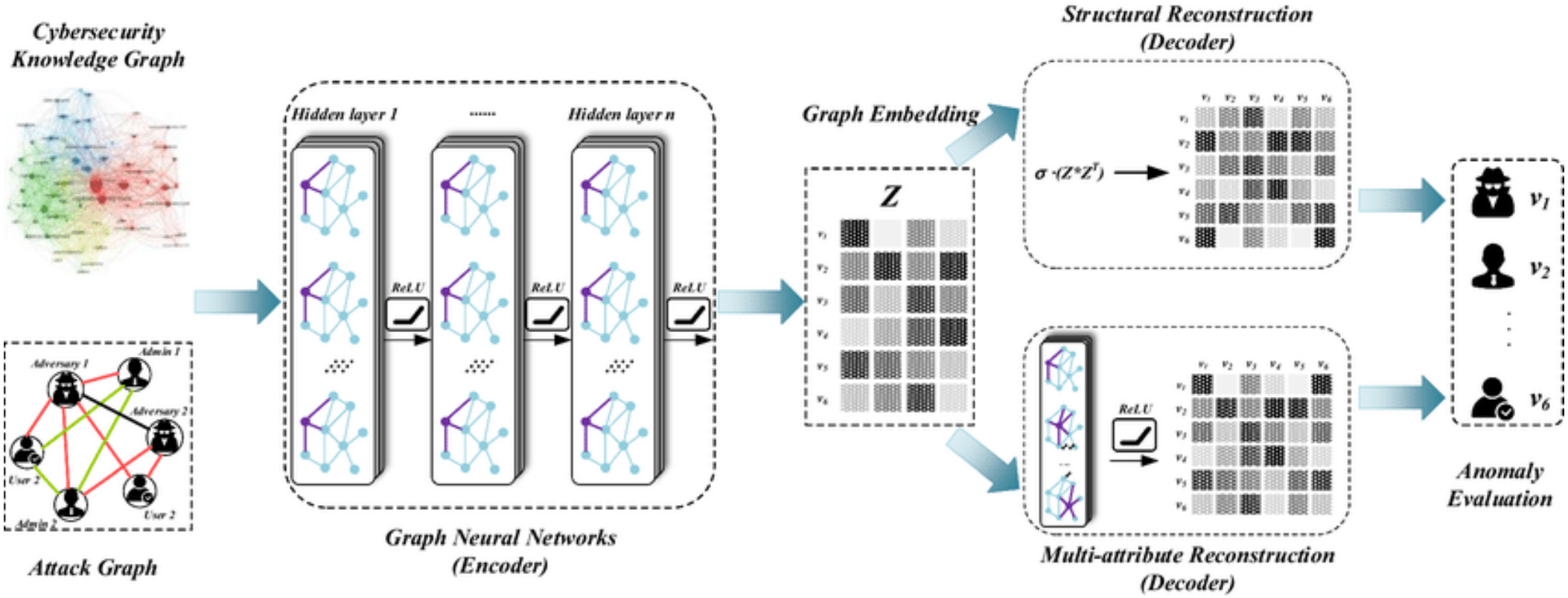
Example (1): Anomaly Detection

- Computer network
  - Nodes: computers/machines
  - Edges: connection/communication between computers
  - Task: detect compromised computers



Source: Saxena and Arora (2009)

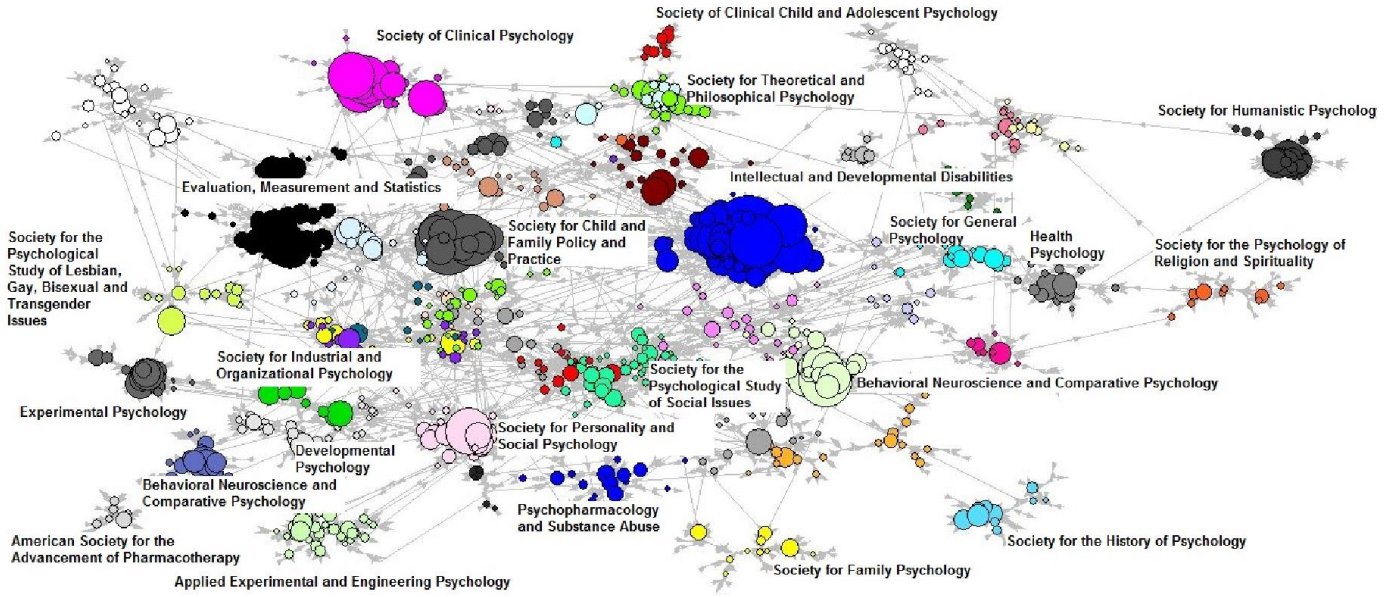
Example (1): Anomaly Detection



Zhuo, Ming, Leyuan Liu, Shijie Zhou, and Zhiwen Tian. "Survey on security issues of routing and anomaly detection for space information networks." *Scientific Reports* 11, no. 1 (2021): 22261.

Example (2): Research Paper Topics

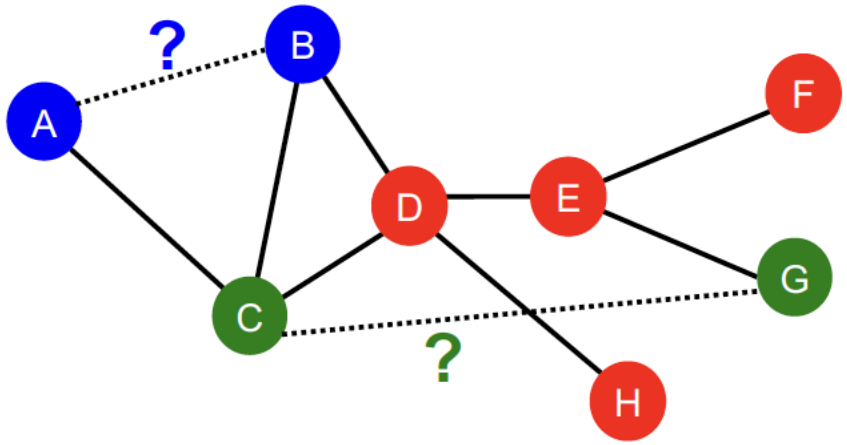
- Citation networks
  - Node: research papers
  - Edges: citations
  - Task: predict topics of papers



Valmarska, Anita, and Janez Demšar. "Analysis of citation networks." PhD diss., Diploma Thesis, Faculty of Computer and Information Science, University of Ljubljana, 2014.

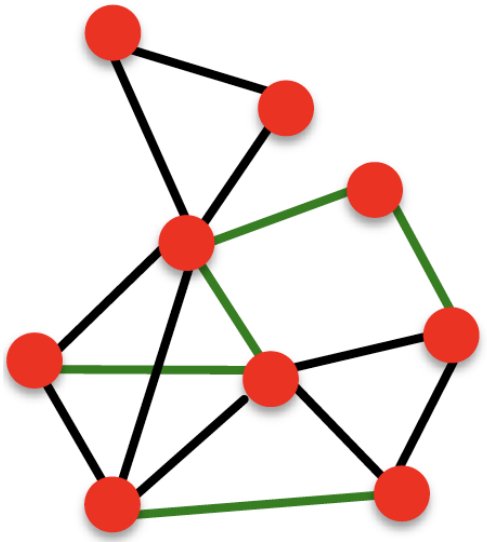
Link-level Prediction Task

- The task is to predict new/missing/unknown links based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- Task: Make a prediction for a pair of nodes.



Link Prediction as a Task

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

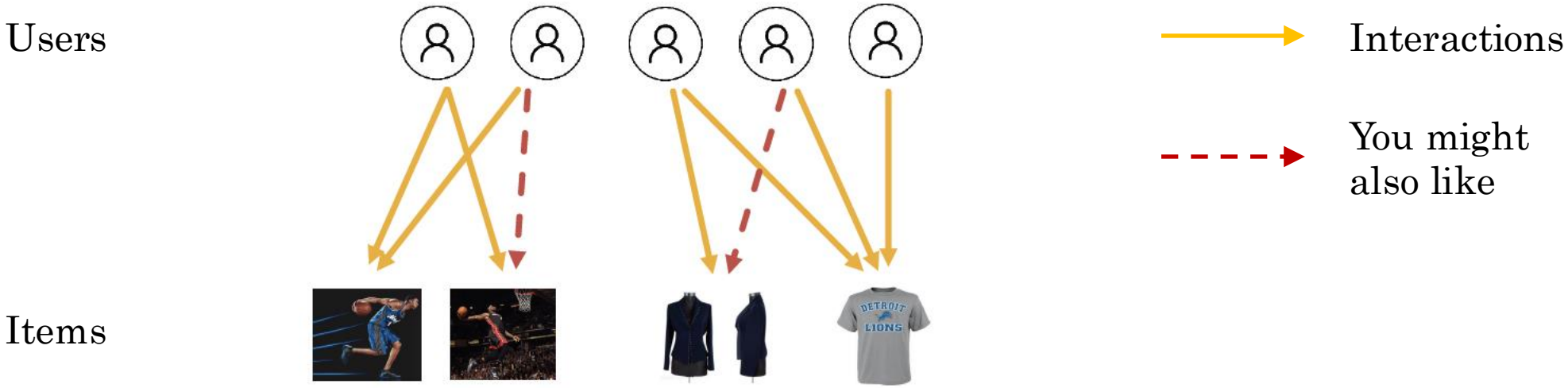


$G[t_0, t_0']$

$G[t_1, t_1']$

Example (1): Recommender Systems

- Users interacts with items
  - Watch movies, buy merchandise, listen to music
  - Nodes: Users and items
  - Edges: User-item interactions
- Goal: recommend items that users might like

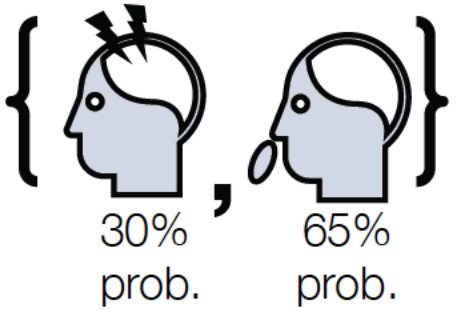


Example (2): Drug Side Effects

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

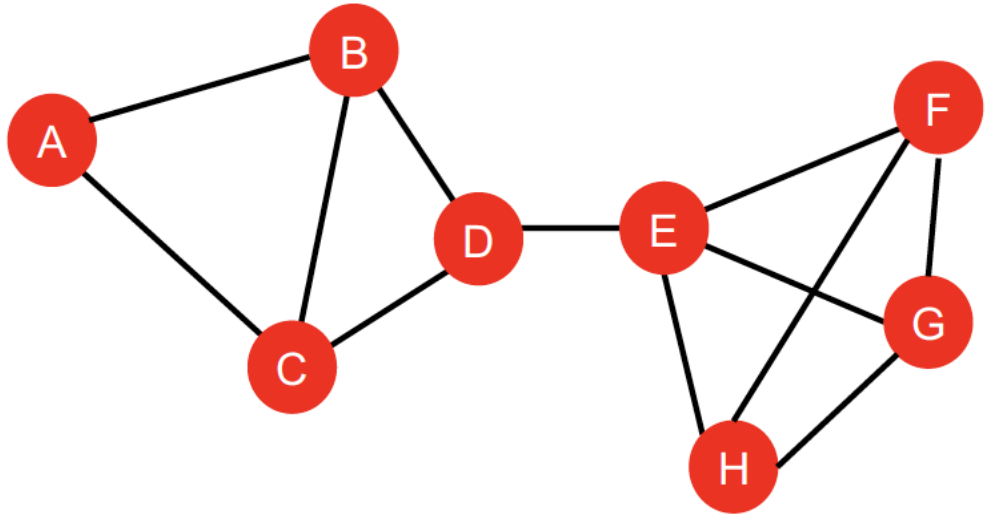


Graph-level Tasks

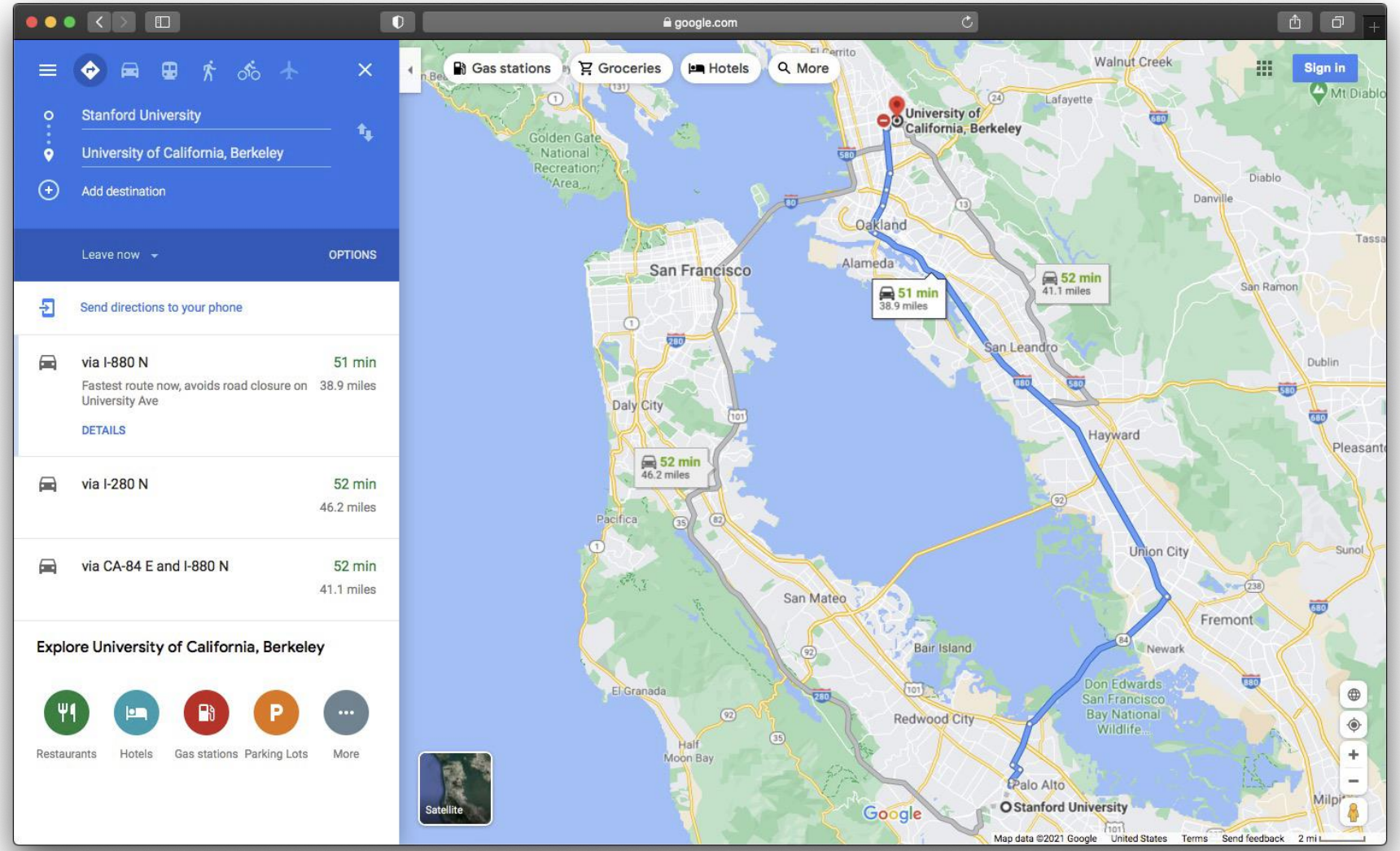
Graph-level Features

- Goal: We want make a prediction for an entire graph or a subgraph of the graph.

- Example



Example (1): Traffic Prediction



Road Network as a Graph

- Nodes: Road segments
- Edges: Connectivity between road segments
- Prediction: Time of Arrival (ETA)

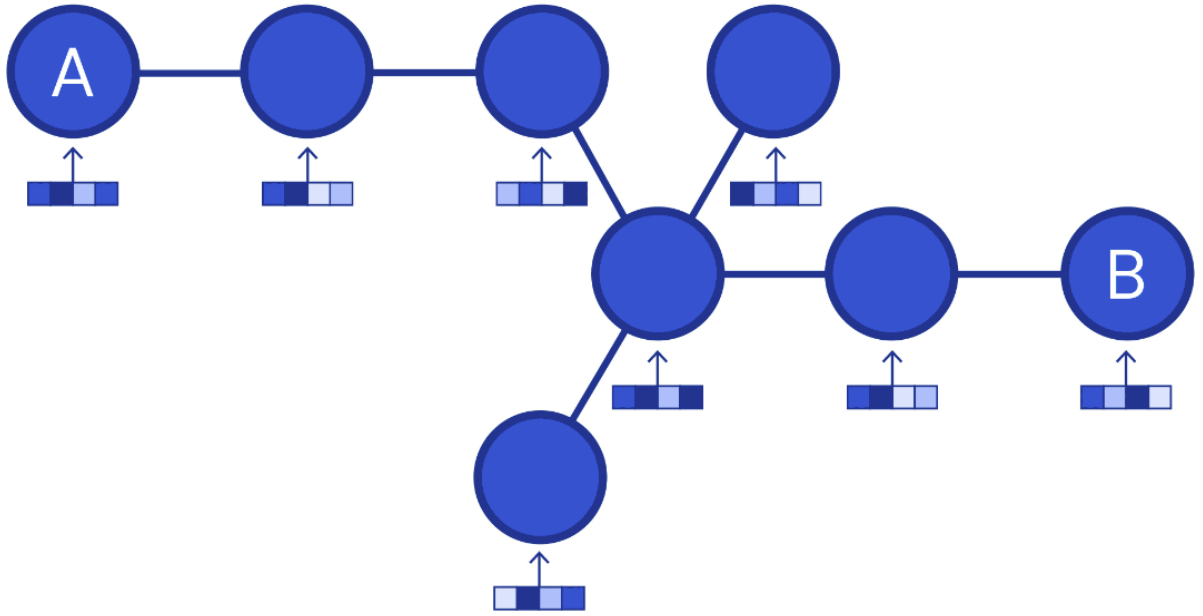
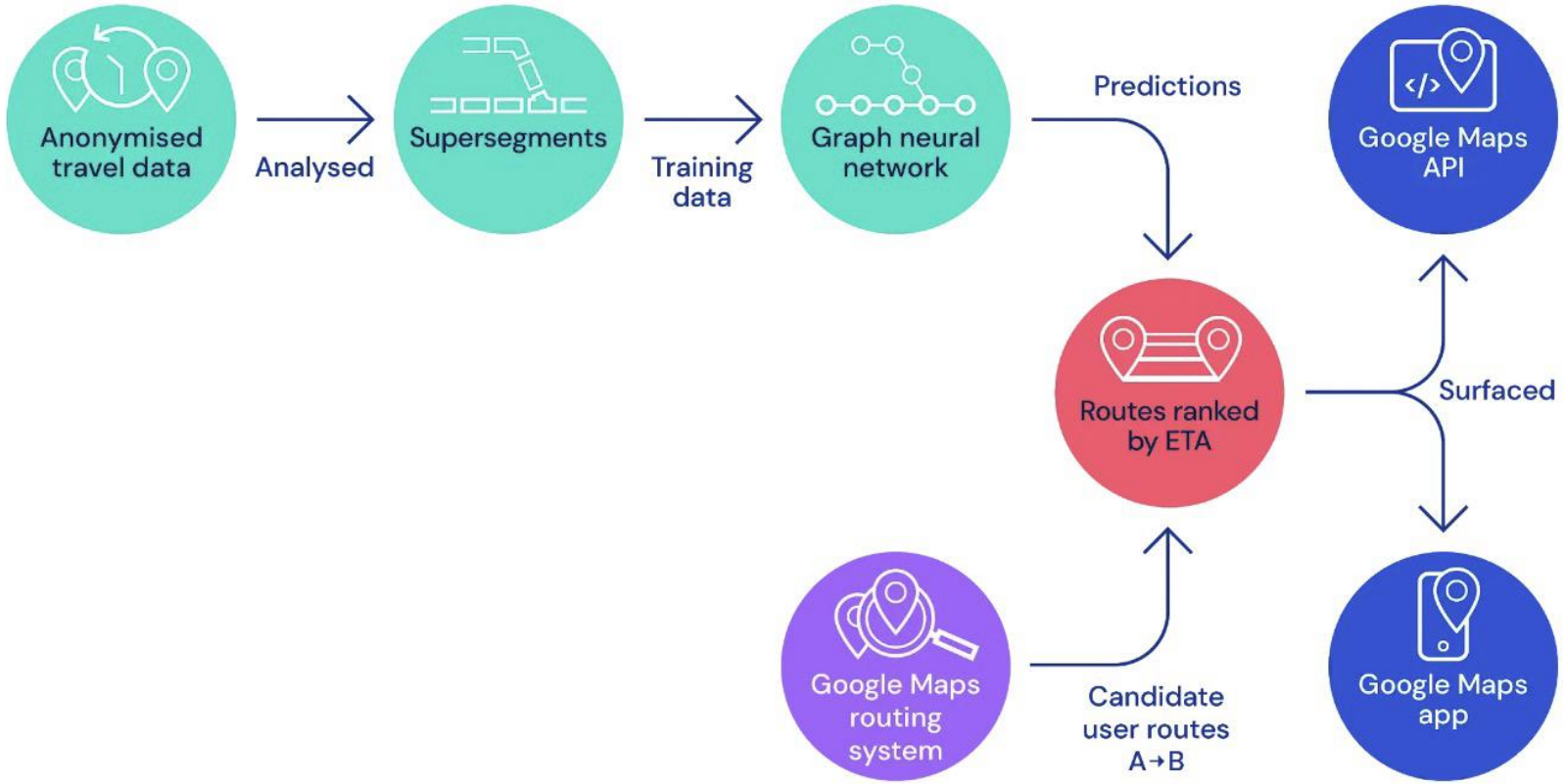


Image credit: DeepMind

Traffic Prediction via GNN

- Predict Time of Arrival with Graph Neural Nets

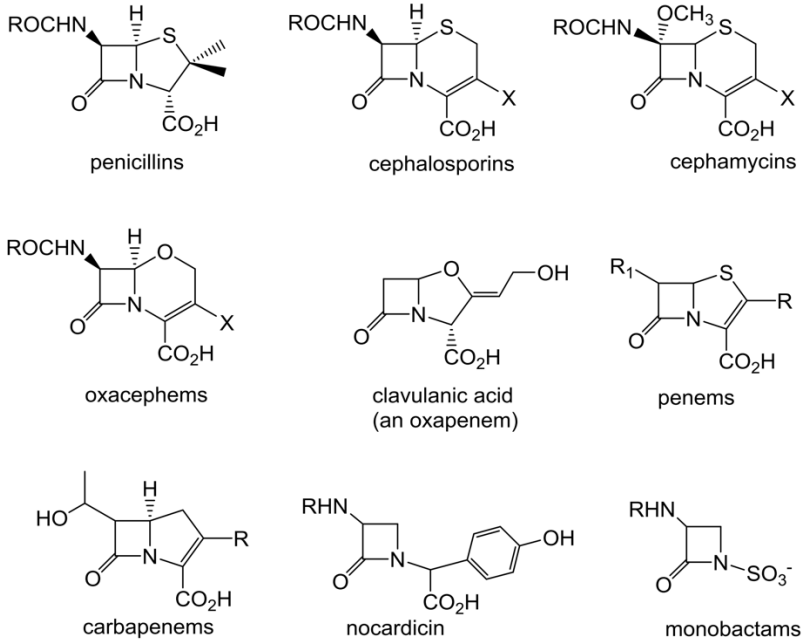


THE MODEL ARCHITECTURE FOR DETERMINING OPTIMAL ROUTES AND THEIR TRAVEL TIME.

Image credit: DeepMind

Example (2): Drug Discovery

- Antibiotics are small molecular graphs
  - Nodes: Atoms
  - Edges: Chemical bonds



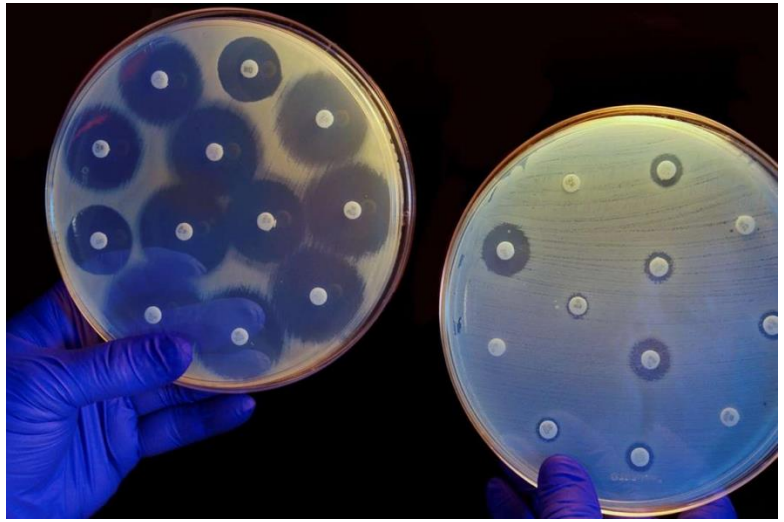
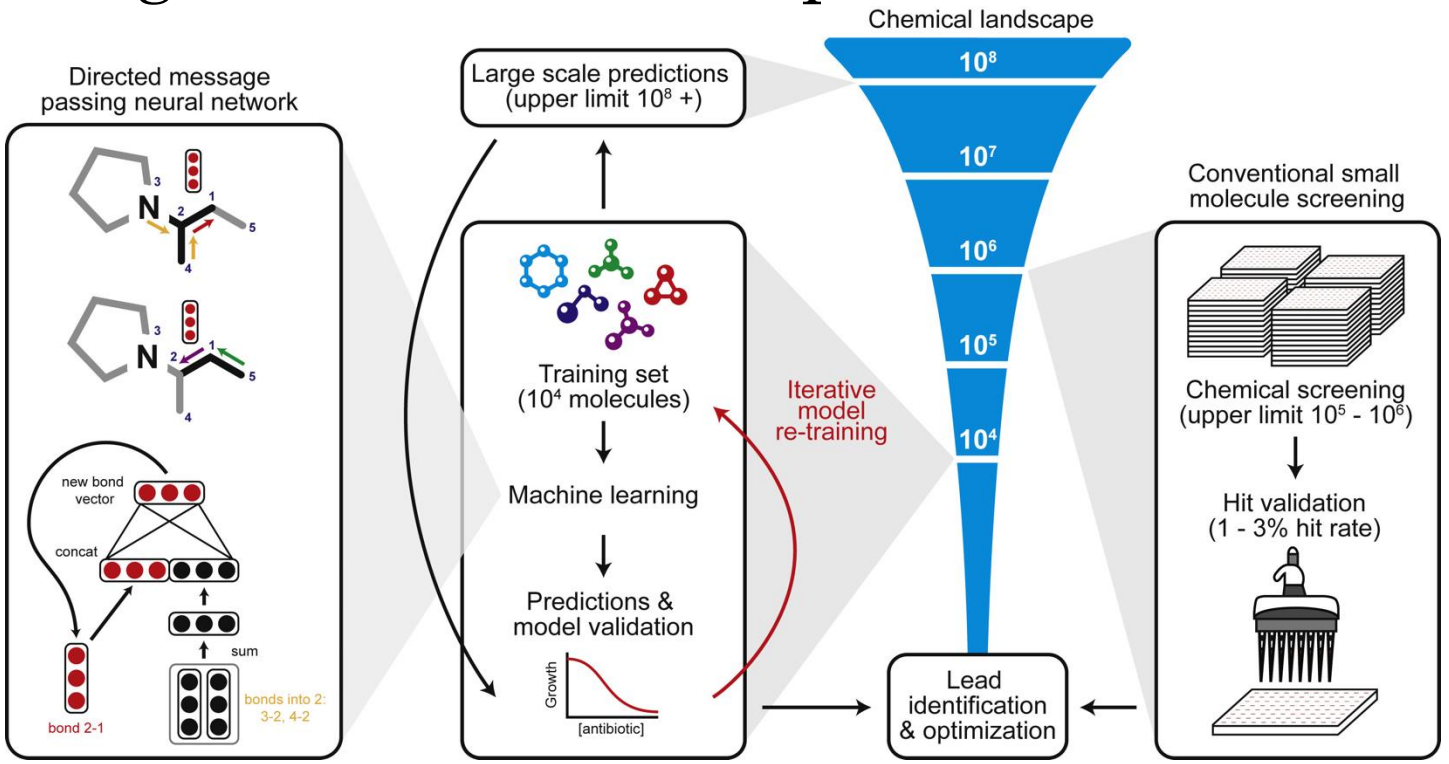


Image credit: CNN

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

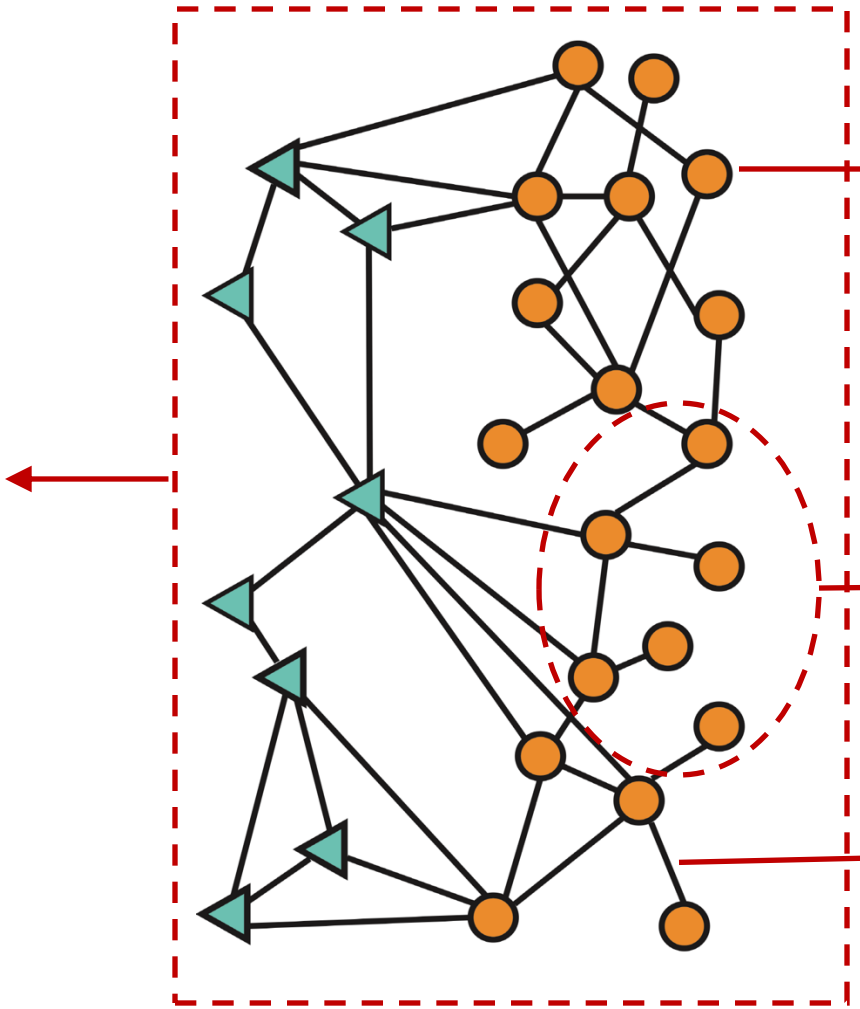
Deep Learning for Antibiotic Discovery

- 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.

Summary



Graph level prediction,
Graph generation

Node level

Community
(subgraph) level

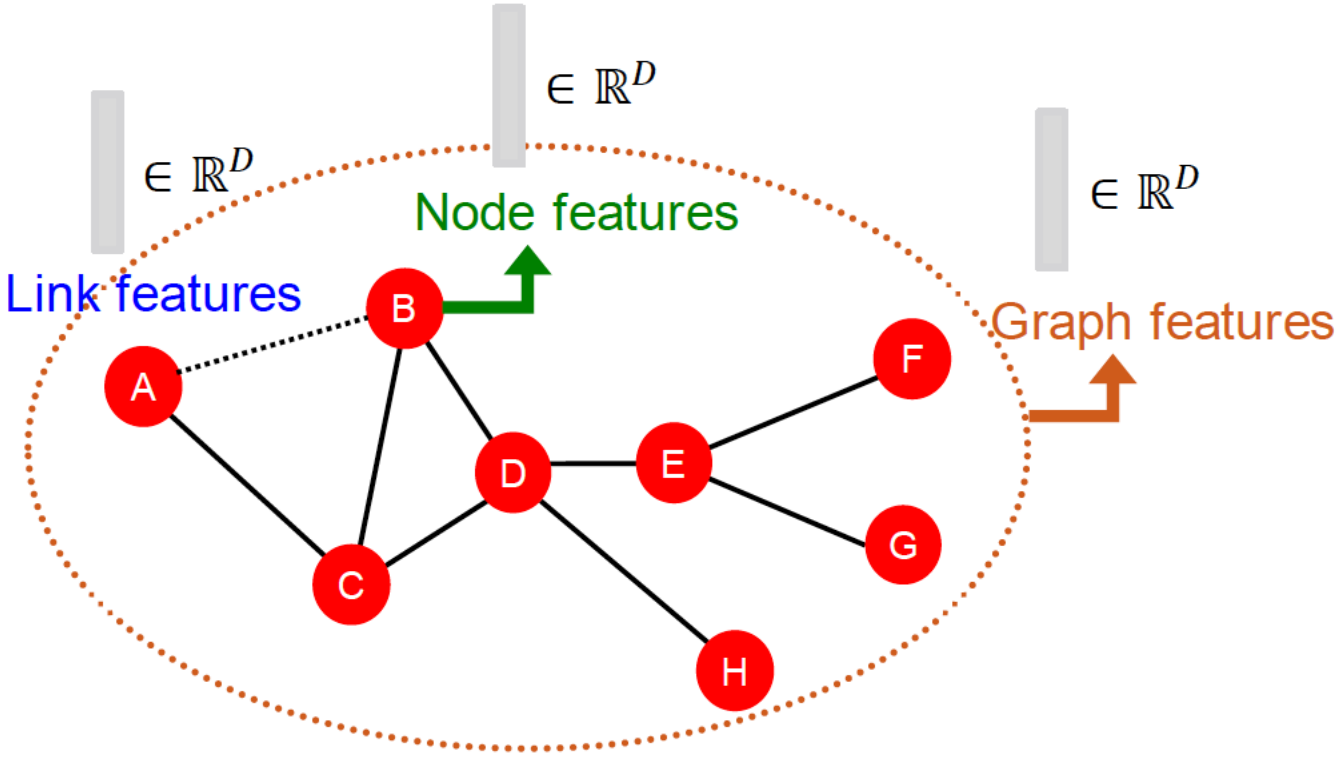
Edge level

Coffee Break!!!

Traditional ML Methods for Graphs

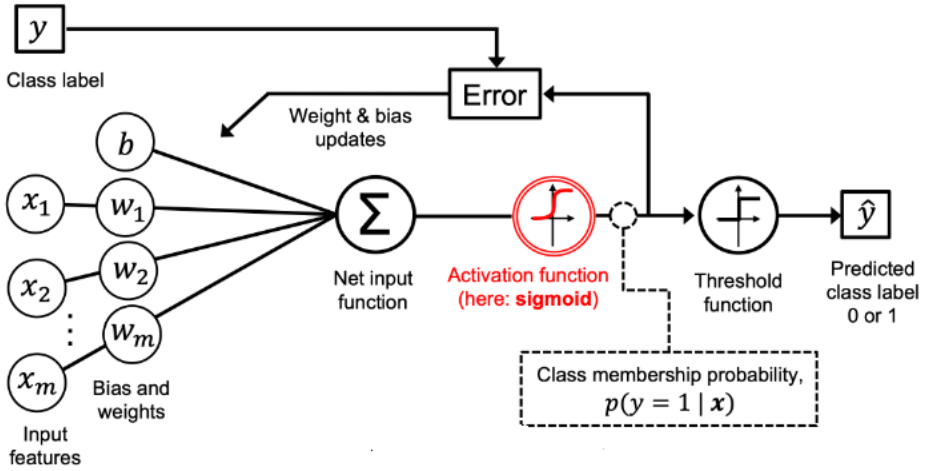
Traditional ML Pipeline

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



Traditional ML Pipeline

- Train an ML model
  - Logistic regression
  - Random forest
  - Neural network, etc



$\boldsymbol{x}_N \rightarrow y_N$

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

$\boldsymbol{x} \rightarrow y$

This Lecture: Feature Design

- Use effective features x over graphs
- Traditional ML pipeline uses hand-designed features

- In this lecture, we will overview the traditional features for:
  - Node-level prediction
  - Link-level prediction
  - Graph-level prediction
- For simplicity, we focus on undirected graphs

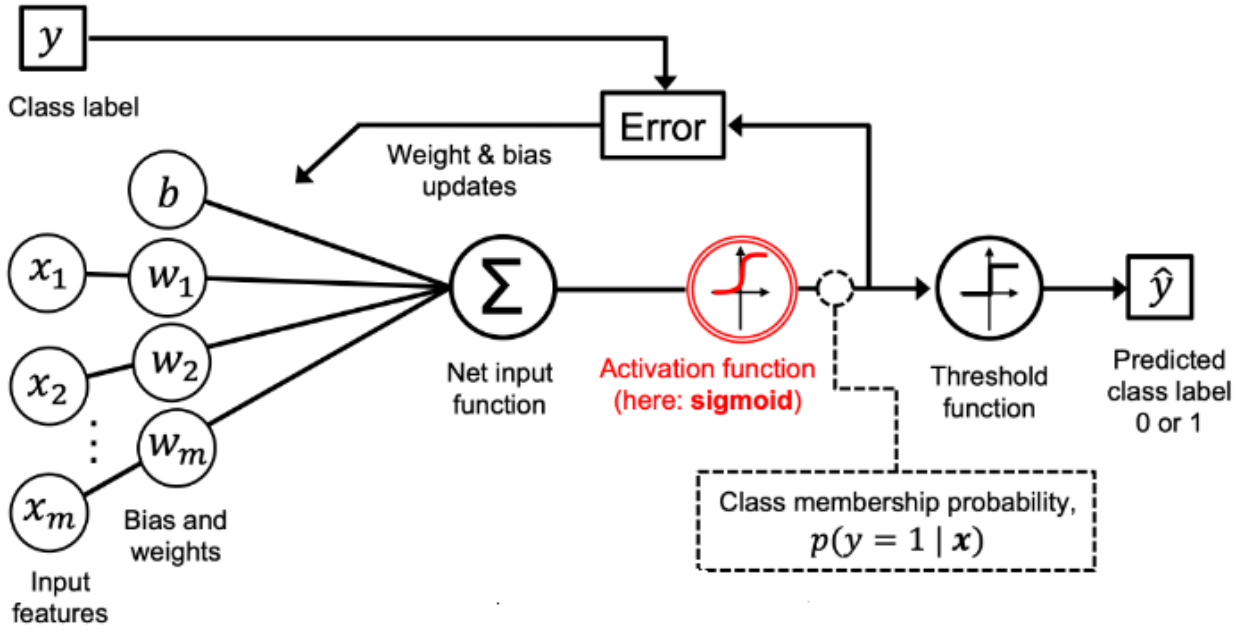
Machine Learning in Graphs

- Goal: Make predictions for a set of objects

- Design choices
  - Features: d-dimensional vectors x
  - Objects: Nodes, edges, sets of nodes, entire graphs
  - Objective functions: What tasks are we aiming to solve?

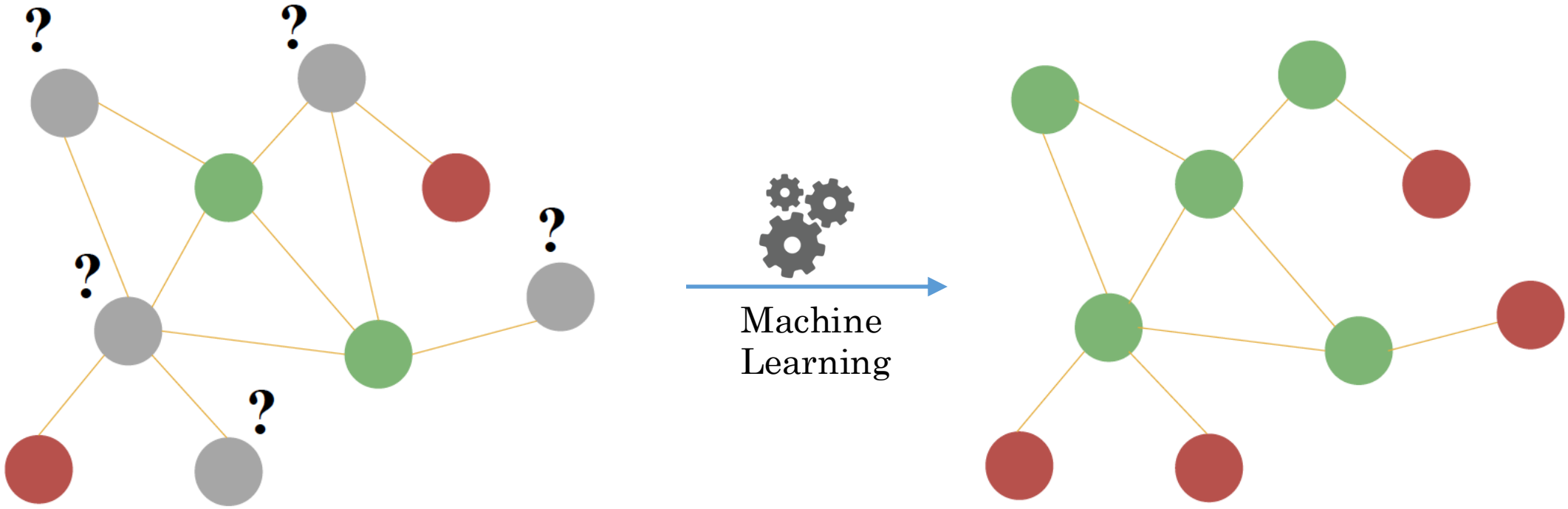
Machine Learning in Graphs

- Example: Node-level prediction
- Given: $G = (V, E)$
- Learn a function: $f: V \rightarrow \mathbb{R}$



Node-level Tasks and Features

Node-Level Tasks

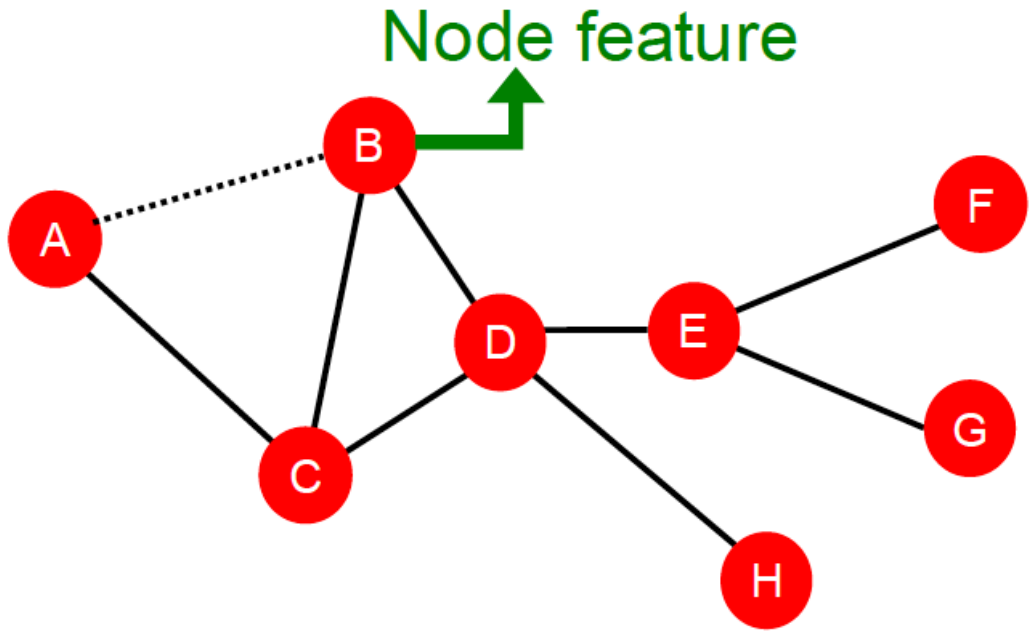


Node Classification

ML needs features

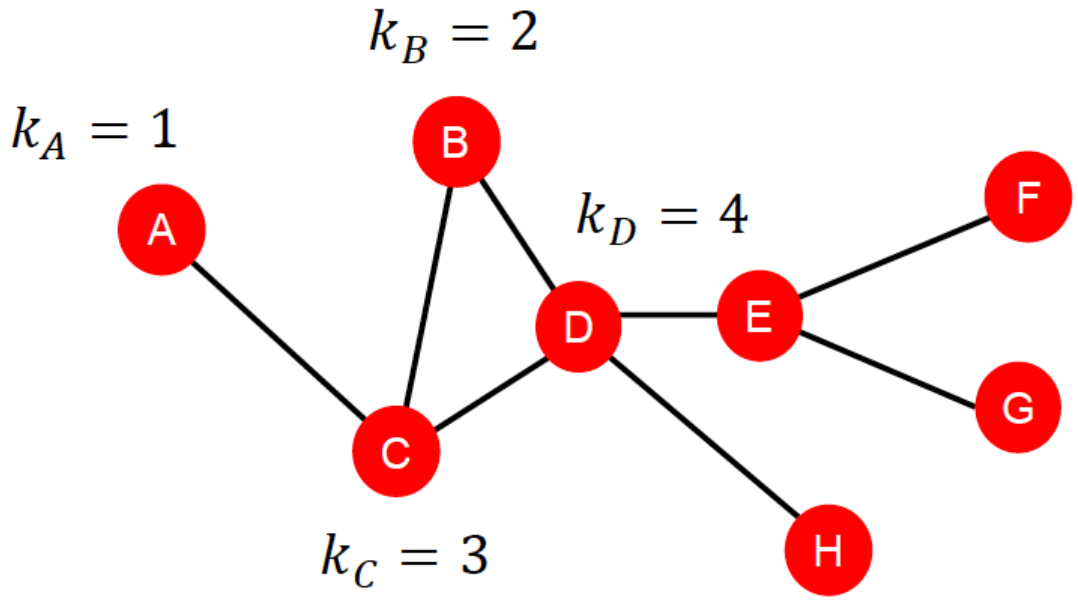
Node-level Features: Overview

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



Node Features: Node Degree

- The degree k_v of node v is the number of edges (neighboring nodes) the node has
- Treat 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
  - Closest centrality
  - And many others

Node Centrality: 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 normalizing constant (it will turn out to be the largest eigenvalue of the adjacency matrix A)

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

Node Centrality: Eigenvector Centrality

- Rewrite the recursive equation in the matrix form

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

λ is normalizing constant
(largest eigenvalue of A)

- A: adjacency matrix
  $A_{uv} = 1$ if $u \in N(v)$
- $\boldsymbol{c}$: centrality vector
- λ: eigenvalue

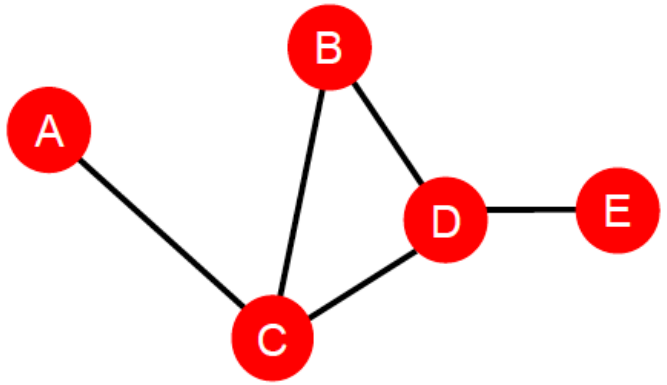
- We can see that centrality $\boldsymbol{c}$ is the eigenvector of A
- The largest eigenvalue λ_{max} is always positive and unique (by Perron-Frobenius Theorem)
- The eigenvector $\boldsymbol{c_{max}}$ corresponding to λ_{max} is used for centrality

Node Centrality: Betweenness Centrality

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

$$c_v = \sum_{s \neq v \neq t} \frac{\#(shortest\ path\ between\ s\ and\ t\ that\ contains\ v)}{\#(shortest\ path\ between\ s\ and\ t)}$$

- Example:

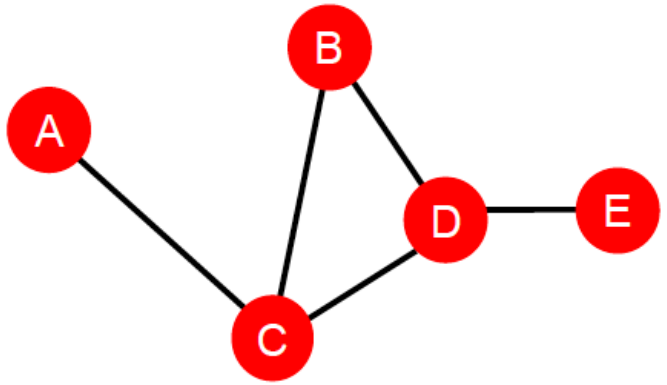


Node Centrality: Closeness Centrality

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

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

- Example:

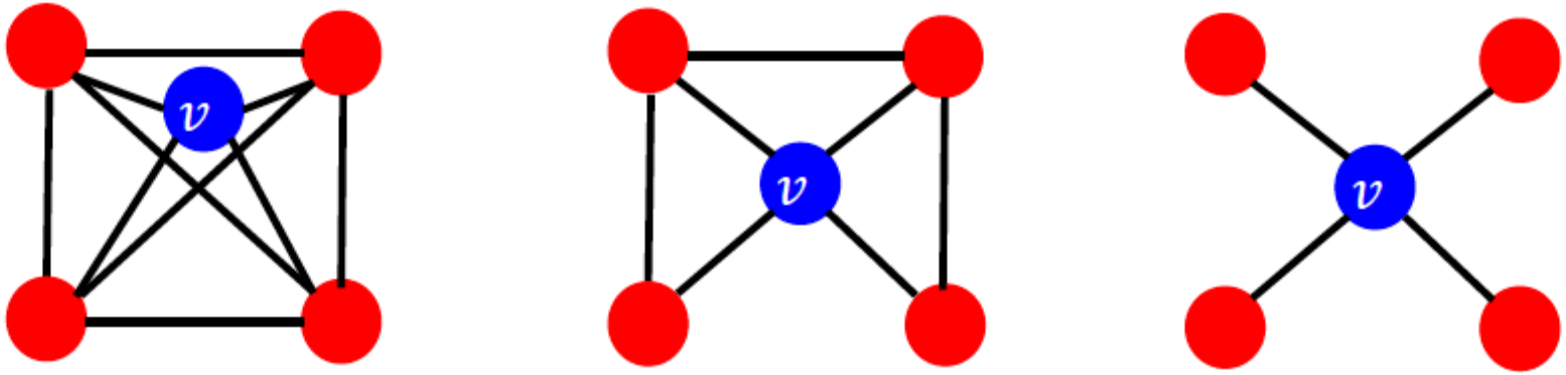


Node Features: Clustering Coefficient

- Measure how connected v's neighboring nodes are:

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

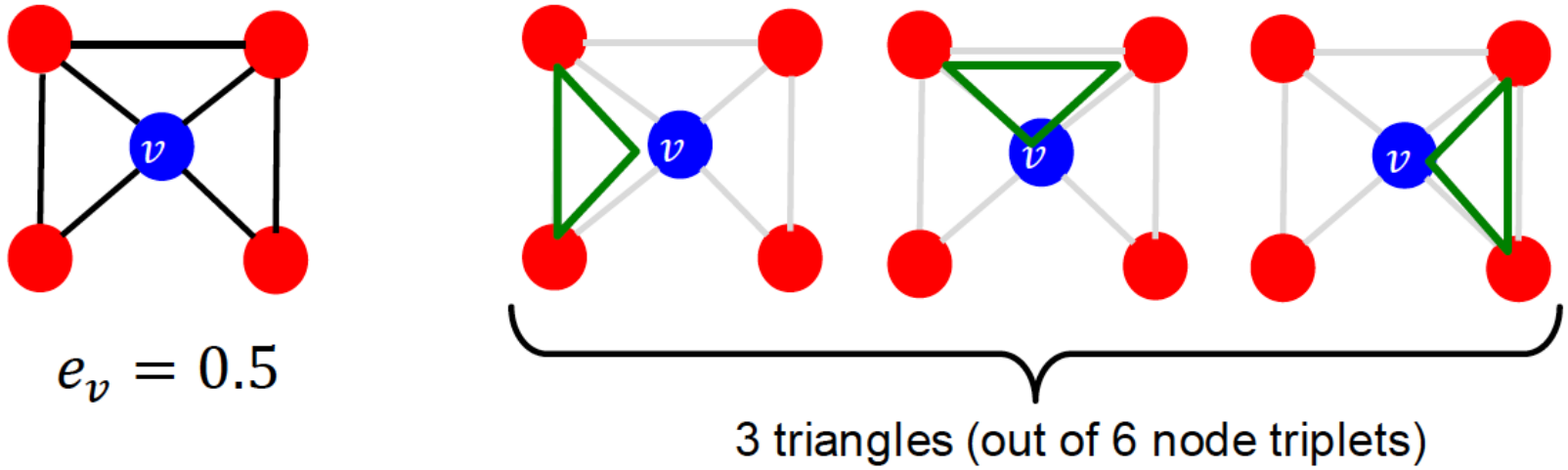
- Example:



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

Node Features: Graphlets

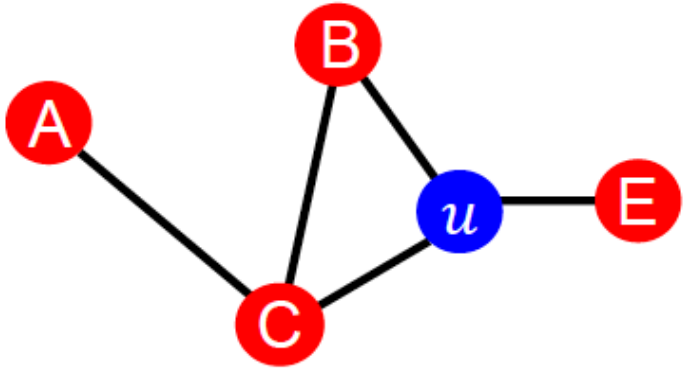
- Observation: Clustering coefficient counts the number of triangles in the ego-network



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

Node Features: Graphlets

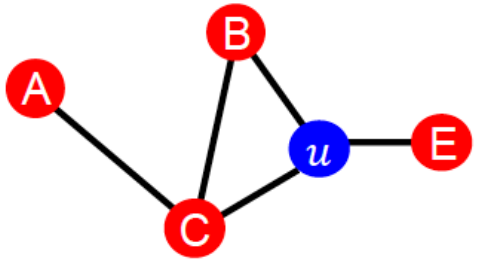
- Goal: describe network structures around a node u
  - Graphlets: small subgraphs that describe the structures of node u's network neighborhood



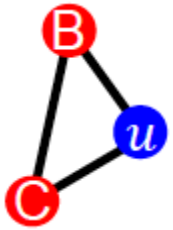
- Analogy:
  - Degree counts #edges that a node touches
  - Clustering coefficient: counts #triangles that a node touches
  - Graphlet Degree Vector (GDV): Graph-based features for nodes
    - Count #graphlets that a node touches

Node Features: Graphlets

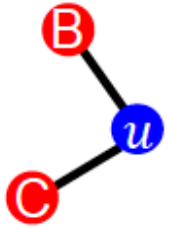
- Induced subgraph: is another graph formed from a subset of vertices and all edges connecting vertices in the subset



Original graph

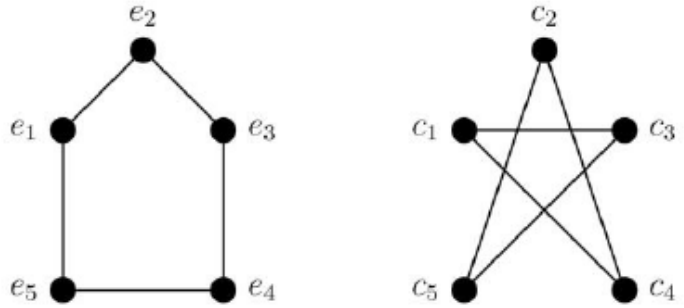


An induced graph

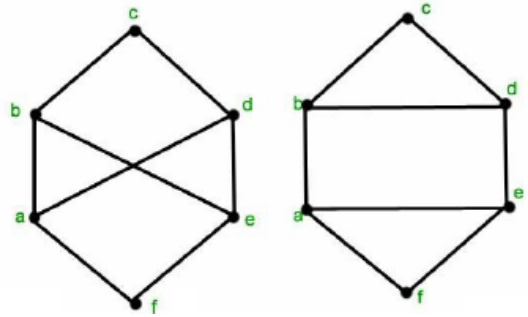


Not an induced graph

- Graph isomorphism: two graphs containing the same number of nodes connected in the same way are said to be isomorphic.



Isomorphic

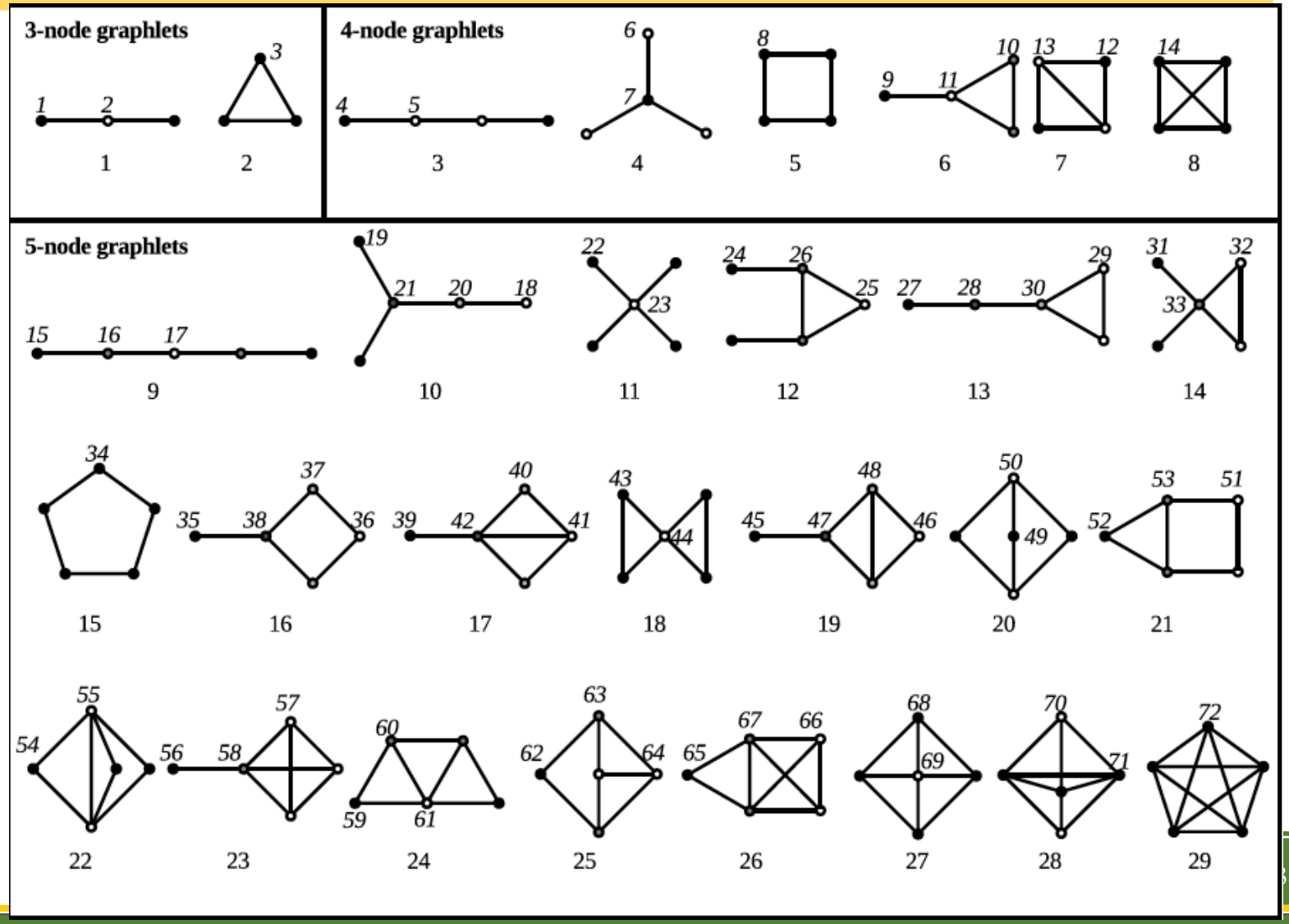


Non isomorphic

Node Features: Graphlets

- Graphlets: Connected induced non-isomorphic subgraphs

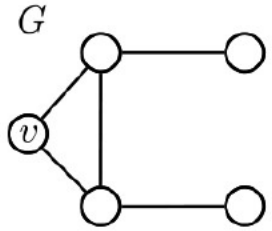
- Orbits: set of vertices which are mapped onto each other by the graphlet's automorphism



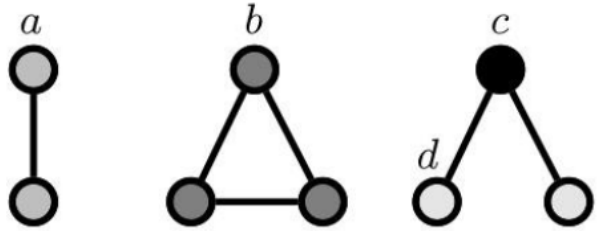
Node Features: Graphlets

- Graphlet Degree Vector (GDV): a vector with the frequency of the node in each orbit position

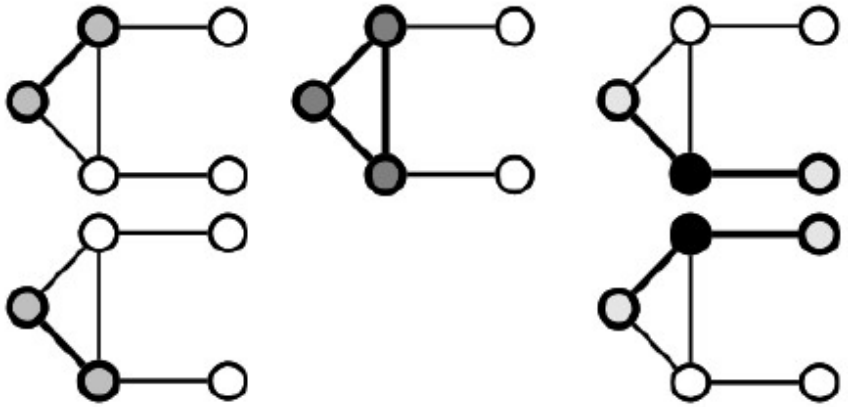
- Example:



Possible graphlets on up to 3 nodes



orbit	a	b	c	d
GDV(v)	2	1	0	2



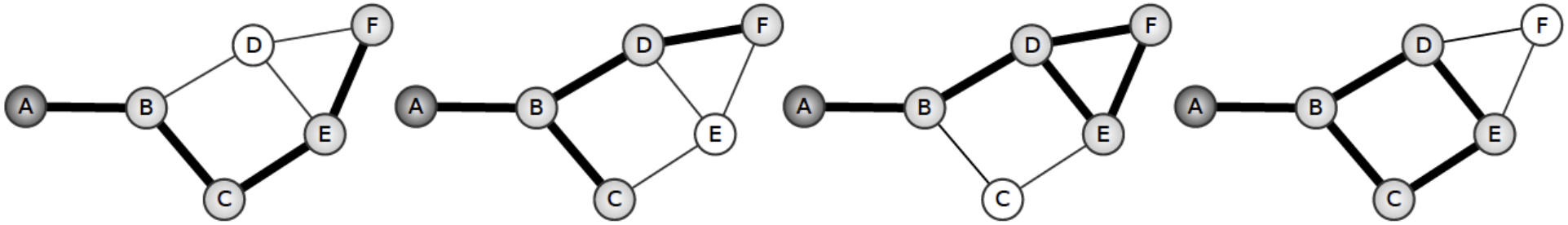
Graphlet Degree Vector (GDV)

- Count #graphlets that a node touches at a particular orbit

- Considering graphlets on 2 to 5 nodes, we get:
  - Vector of 73 coordinates: a signature of a node that describes the topology of the node's neighborhood
  - Capture its interconnectivities up to a distance of 4 hops

- GDV provides a measure of a node's local topology
  - Comparing vectors of two nodes provides a highly constrained measure of local topological similarity between them.

Graphlet Degree Vector: Example



Orbit 15 Orbit 19 Orbit 27 Orbit 35

Orbit	0	1	2...3	4	5	6	7...14	15	16...18	19	20...26	27	28...34	35	36...72
GDV(A)	1	2	0...0	3	0	1	0...0	1	0...0	1	0...0	1	0...0	1	0...0

- GDV of a node A:
  - i-th element of GDV(A): #graphlets that touch A at orbit i
  - Highlighted are graphlets that touch A at orbits 15, 19, 27, 35.

Node-Level Feature: Summary

- Importance based features
  - Node degree
  - Different node centrality measures (eigenvector, betweenness, closeness)

- Structure-based features
  - Node degree
  - Clustering coefficient
  - Graphlet count vector

Node-Level Feature: Summary

- Importance-based features
  - Node degree: count #neighboring nodes
  - Node centrality:
    - Model 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: predict celebrity users in a social network

Node-Level: Summary

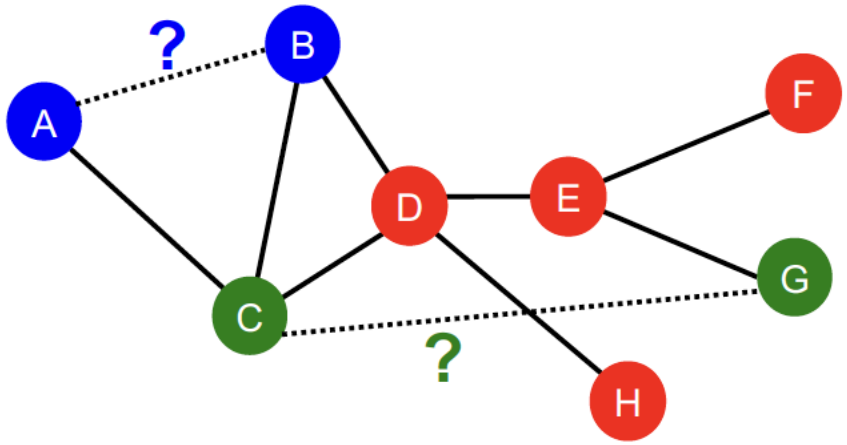
- Structure-based features: capture topological properties of local neighborhood around a node
  - Node degree: count #neighboring nodes
  - Clustering coefficient: measure how connected neighboring nodes are
  - Graphlet count vector: count the occurrences of different graphlets

- Useful for predicting a particular role a node plays in a graph
  - Example: predict protein functionality in a protein-protein interaction network

Link Prediction Task and Features

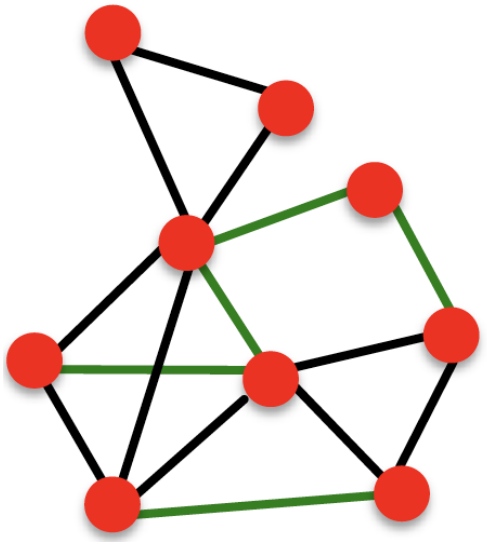
Link-level Prediction Task: Recap

- The task is to predict new/missing/unknown links based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- Task: Make a prediction for a pair of nodes.



Link Prediction as a Task

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

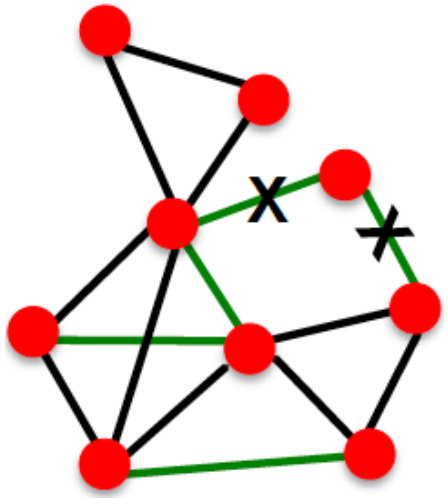


$G[t_0, t_0']$

$G[t_1, t_1']$

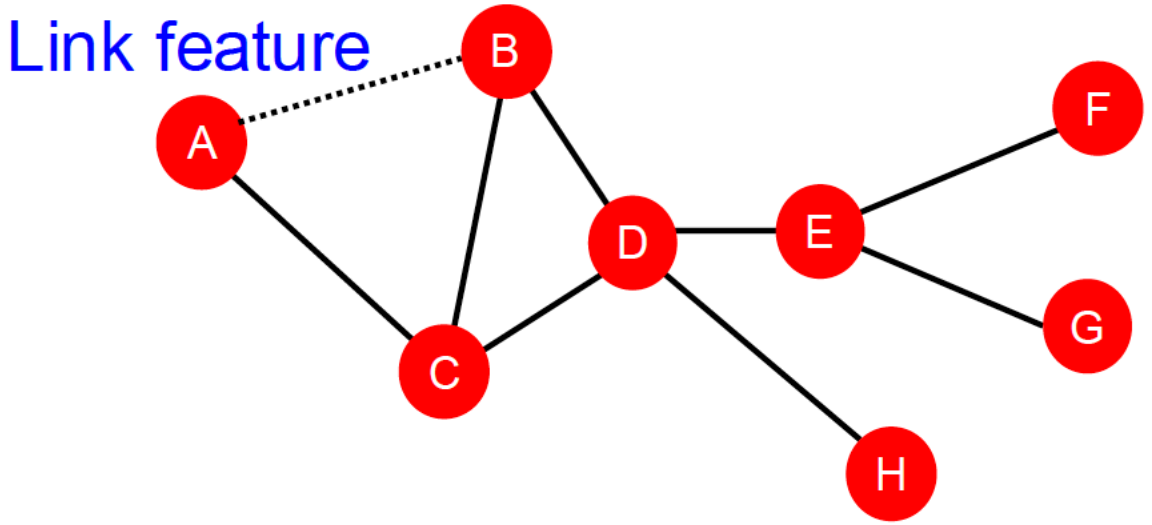
Link Prediction via Proximity

- Methodology:
  - For each pair of nodes (x, y), compute score $c(x, y)$
    - For example: #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 $G[t_1, t_1']$



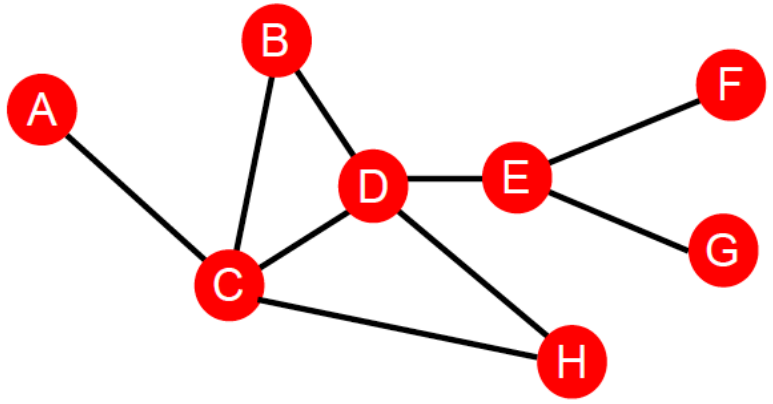
Link-Level Feature: Overview

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



Distance-based Feature

- Shortest path distance between two nodes
  - Example



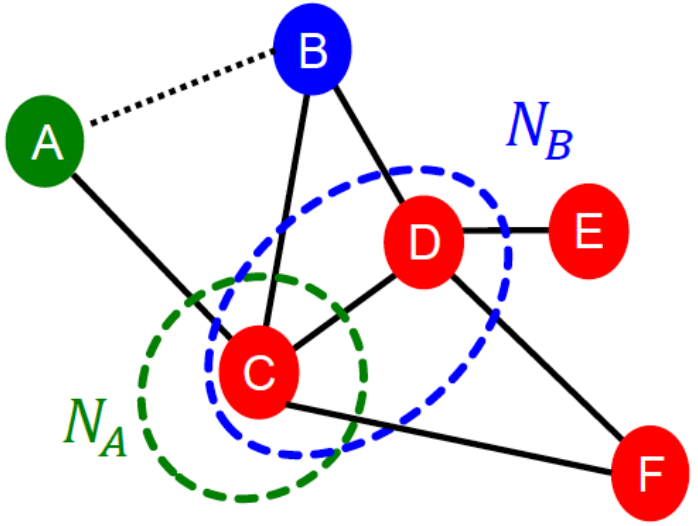
$$S_{BH} = S_{BE} = S_{AB} = 2$$

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

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

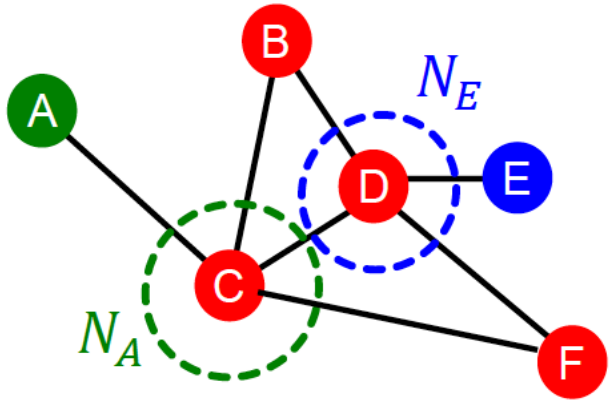
Local Neighborhood Overlap

- Capture #neighboring nodes shared between two nodes
  - Common neighbors: $|N(v_1) \cap N(v_2)|$
    - Example: $|N(A) \cap N(B)| = 1$
  - Jaccard's coefficient: $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$
    - Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{1}{2}$
  - 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 overlap
  - Metric is always zero if the two nodes do not have any neighbors in common



$$|N(A) \cap N(E)| = 0$$

  - However, the two nodes may still potentially be connected in the future
- Global neighborhood overlap: resolves the limitation by considering the entire graph

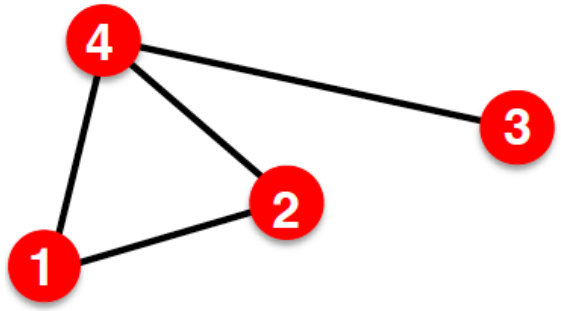
Global Neighborhood Overlap

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

- Compute #walks:
  - Use powers of the graph adjacency matrix

Intuition: Powers of Adj Matrices

- Compute #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 $P^{(k)} = A^k$
  - $P_{uv}^{(1)} = A_{uv} = \#walks$ of length 1 (direct neighborhood) between u and v



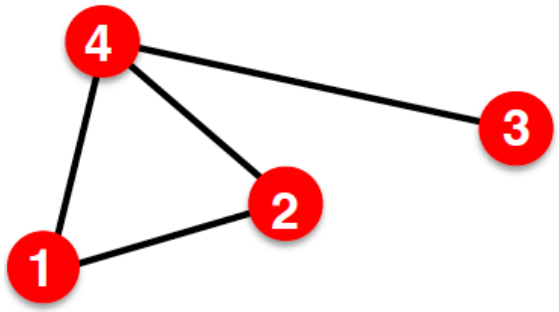
$$\boldsymbol{P_{12}^{(1)} = A_{12}}$$

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

Intuition: Powers of Adj Matrices

- How to compute $P_{uv}^{(2)}$?
  - Step 1: Compute #walks of length 1 between each neighbor of u 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

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

Power of adjacency

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

Global Neighborhood Overlap

- Katz index between v_1 and v_2 is computed as:

Sum over *all walk lengths*

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

$A^l_{v_1 v_2}$: #walks of length l between v_1 and v_2

β^l: $0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

$$\boldsymbol{S} = \sum_{i=1}^{\infty} \beta^i \boldsymbol{A}^i = \underbrace{(\boldsymbol{I} - \beta \boldsymbol{A})^{-1}}_{= \sum_{i=0}^{\infty} \beta^i \boldsymbol{A}^i \text{ by geometric series of matrices}} - \boldsymbol{I},$$

Link-Level Features: Summary

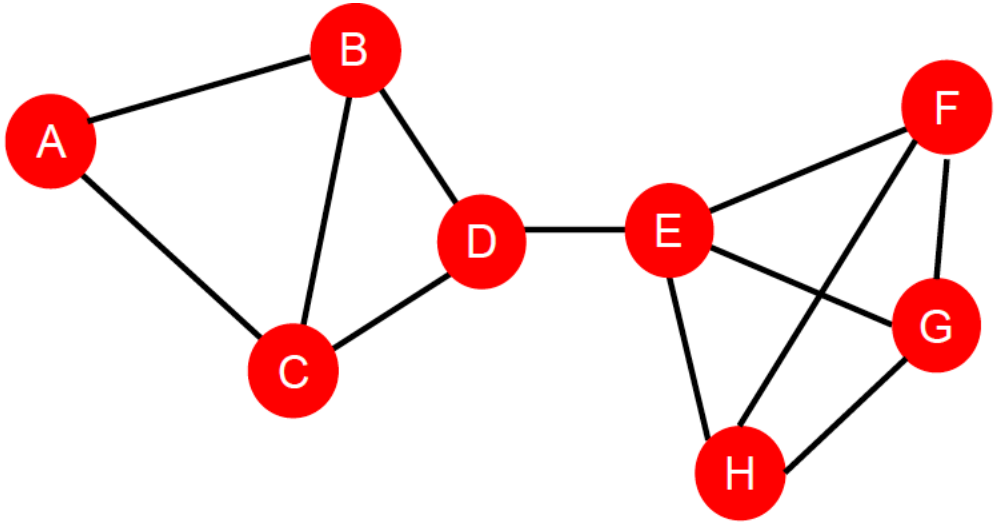
- Distance-based feature
  - Use the shortest path length
  - Does not capture how neighborhood overlaps
- Local neighborhood overlap
  - Capture how many neighboring nodes are shared
  - Become zero when no neighbors are shared
- Global neighborhood overlap
  - Use global graph structure to score two nodes
  - Katz index count #walks of all lengths between two nodes

Graph-Level Features and Graph Kernels

Graph-Level Features

- Goal: characterize structure of an entire graph

- Example:



Background: Kernel Methods

- Definition: A kernel K is a legal definition of legal inner products, i.e., there exists an *implicit* mapping $\Phi: \mathrm{X} \rightarrow \mathbb{R}^N$ such that:

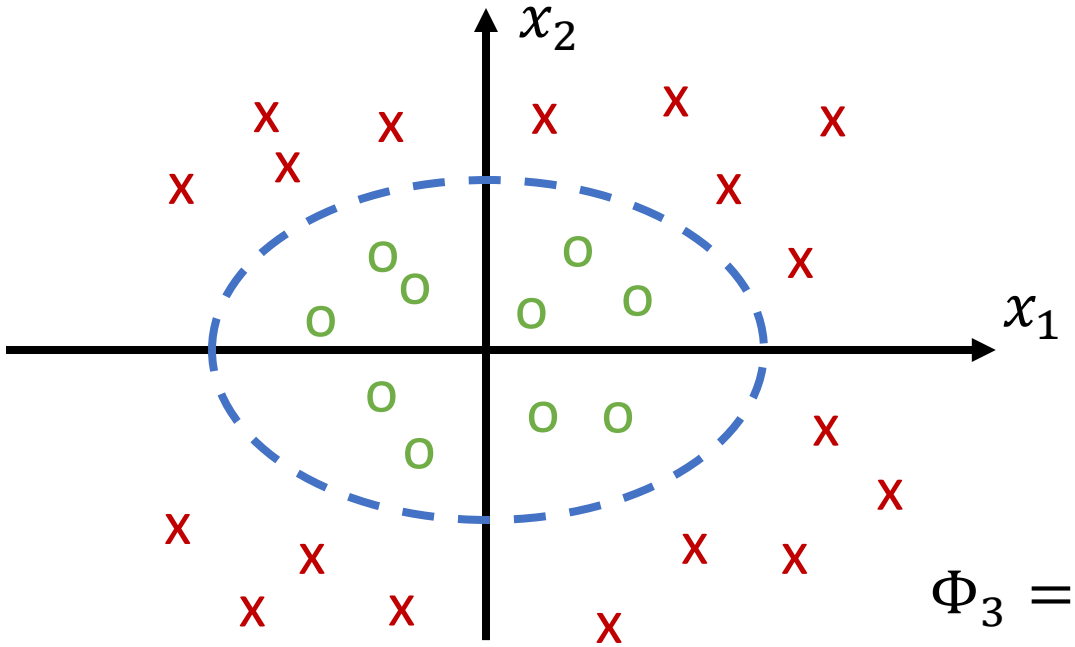
$$K(x, z) = \Phi(x) \cdot \Phi(z)$$

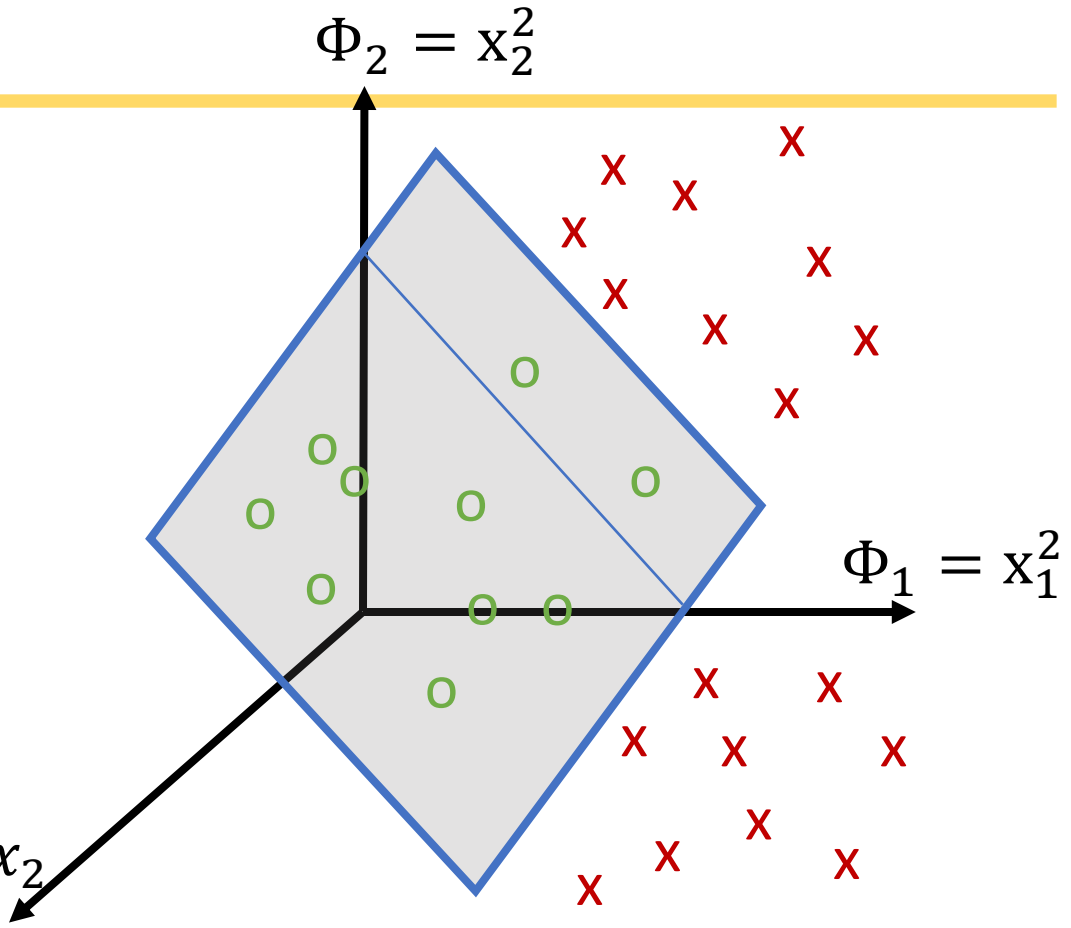
  - N can be very large ($N \gg d$)

- But think of Φ as implicit, not explicit!!!!

Example

- Feature map $\Phi(x) = \left(x_1^2, x_2^2, \sqrt{2}\, x_1 x_2\right)$





- The kernel $K(x, z) = (x \cdot z)^2$ with $x = (x_1, x_2)$ corresponds to:

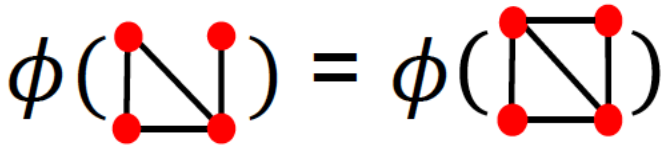
$$K(x, z) = \Phi(x) \cdot \Phi(z)$$

Graph-Level Features: Overview

- Graph kernels: measure similarity between two graphs
  - Graphlet kernel
  - Weisfeiler-Lehman kernel
  - Other kernels (will not be covered in this lecture)
    - Random-walk kernel
    - Shortest-path graph kernel
    - Many more…

Graph Kernel: Ideas

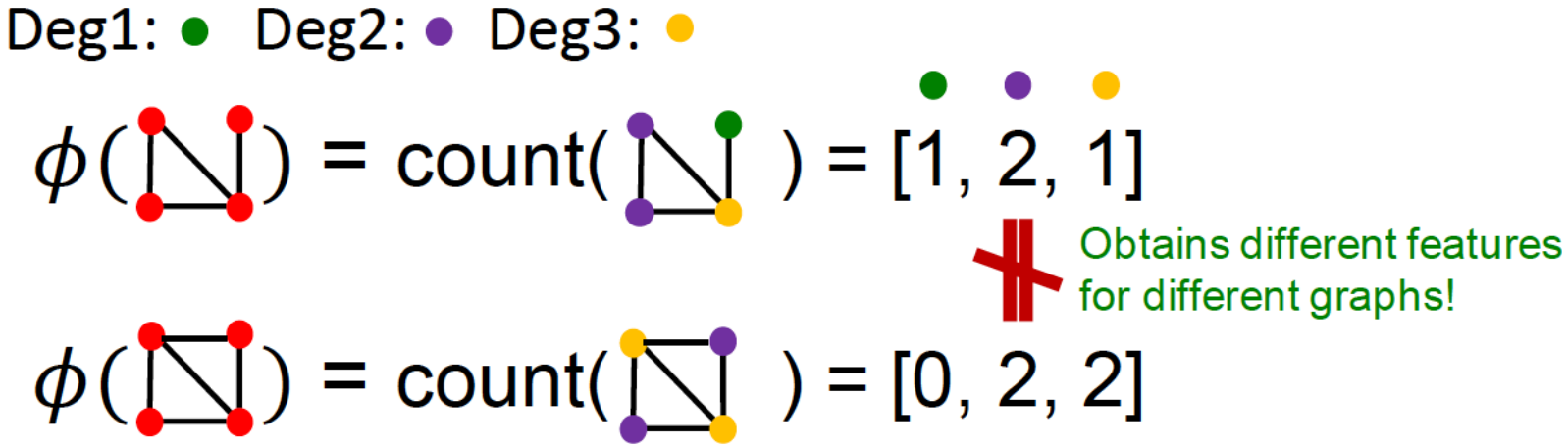
- Goal: design graph feature vector $\phi(G)$
- Key ideas: Bag-of-Words (BoW) for a graph
  - Recall: BoW uses word counts as features for documents (no ordering)
  - Naïve extension to a graph: consider nodes as words
    - Limitation:



    - Since both graphs have 4 nodes, we get the same feature vector for two different graphs

Graph Kernel: Key Ideas

- What if we use Bag of node degrees?



- Both Graphlet kernel and Weisfeier-Lehman (WL) use Bag-of-* representation of graphs.

Graph-Level Graphlet Features

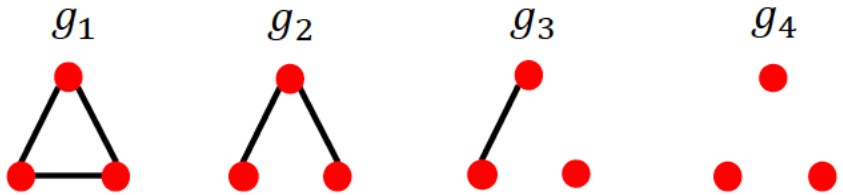
- Key idea: count #different graphlets in a graph

  - Note: definition of graphlets here is slightly different from node-level features

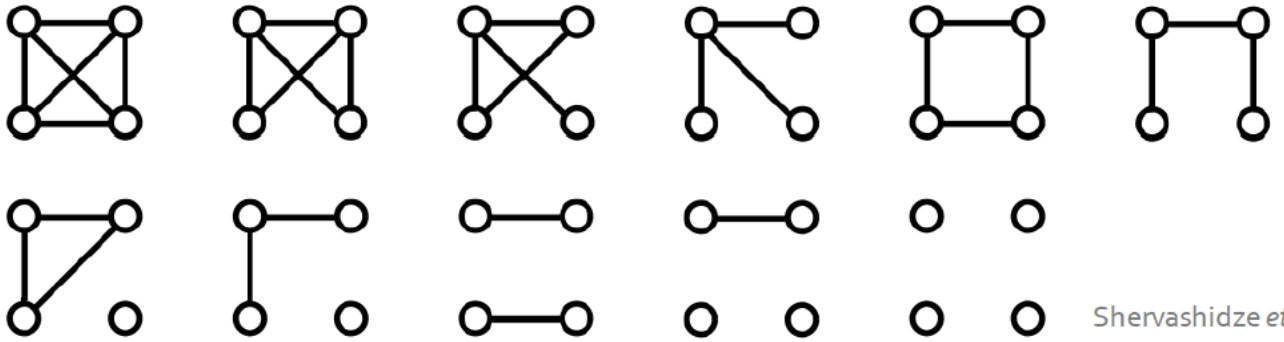
  - Two differences:
    - Nodes in graphlets here do not need to be connected
    - Graphlets here are not rooted

Graph-Level Graphlet Features

- Let $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$ be a list of graphlets of size k.
- For k=3, there are 4 graphlets



- For k = 4, there are 11 graphlets



Shervashidze *et al.*, AISTATS 2011

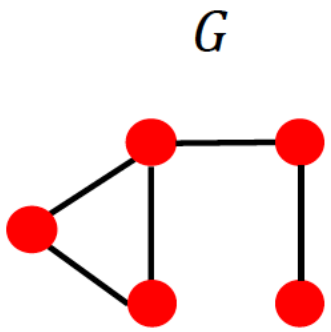
Graph-Level Graphlet Features

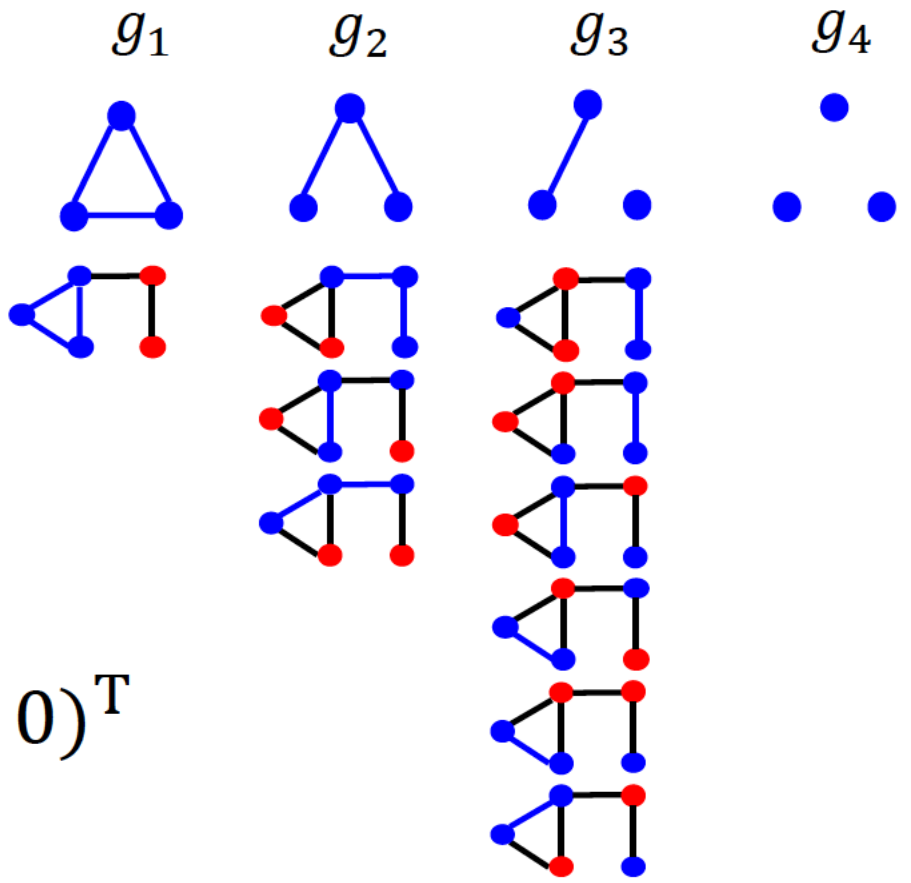
- Given graph G, and a graphlet list $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$, define the graphlet count vector $f_G \in \mathbb{R}^{n_k}$ as:

$$(f_G)_i = \#(g_i \in G), \forall i = 1, 2, \ldots, n_k$$

Graph-Level Graphlet Features

- Example: k = 3





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

Graph-Level Graphlet Kernel

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

$$K(G, G') = f_G^T f_{G'}$$

- Problem:
  - If G and G' have different sizes, that will greatly skew the value.

- Solution: normalize each feature vector

$$h_G = \frac{f_G}{sum(f_G)} \qquad K(G, G') = h_G^T h_{G'}$$

The Graphlet Kernel

- Limitations: counting graphlets is expensive
  - Counting size-k graphlets for a graph of size n by enumeration takes n^k
  - This is unavoidable in worst case since subgraph isomorphism test (judging if 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 graphlets of size k.

- Can we design a more efficient graph kernel?

Weisfeiler-Lehman Kernel

- Goal: Design an efficient graph feature description $\phi(G)$

- Key idea: Use neighborhood structure to iteratively enrich node vocabulary.
  - Generalized version of Bag of node degree since node degrees are one-hop neighborhood information

- Algorithm: 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

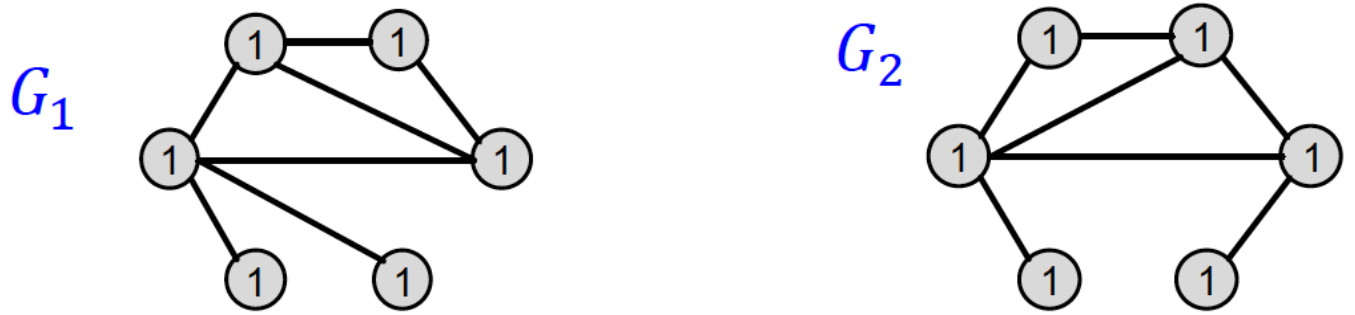
$$c^{(k+1)}(v) = HASH\left(\left\{c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right\}\right)$$

where HASH maps different inputs to different colors

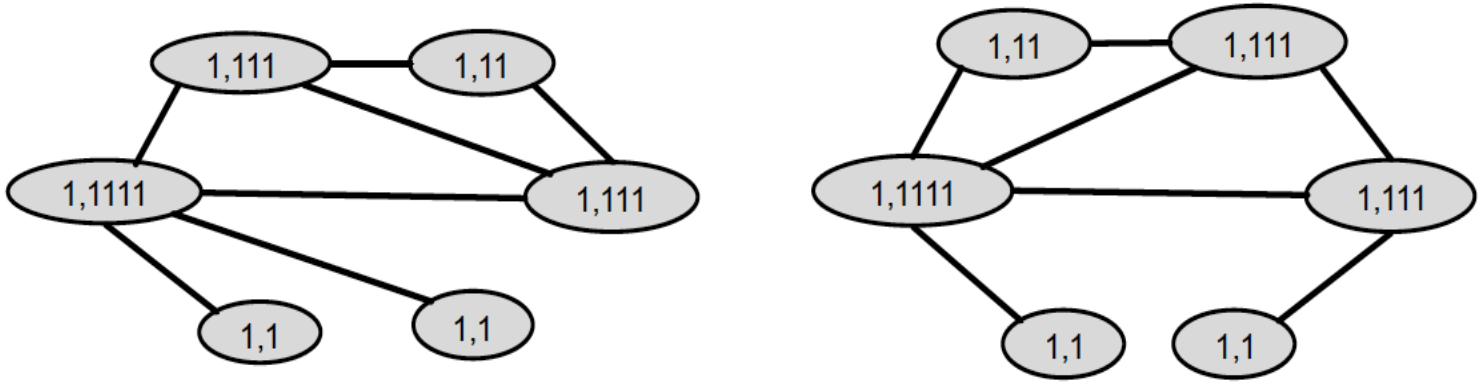
  - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood.

Color Refinement: Example

- Assign initial colors

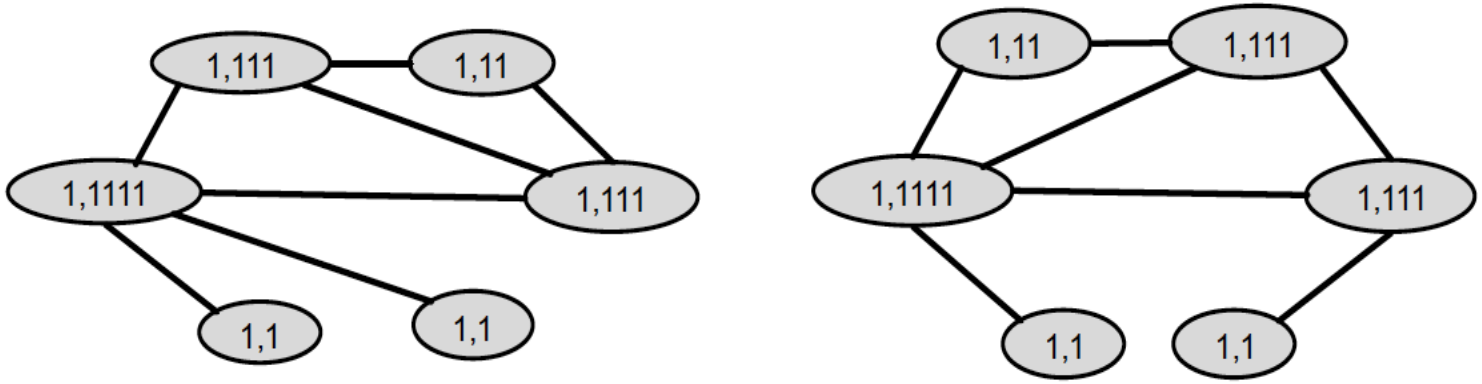


- Aggregate neighboring colors

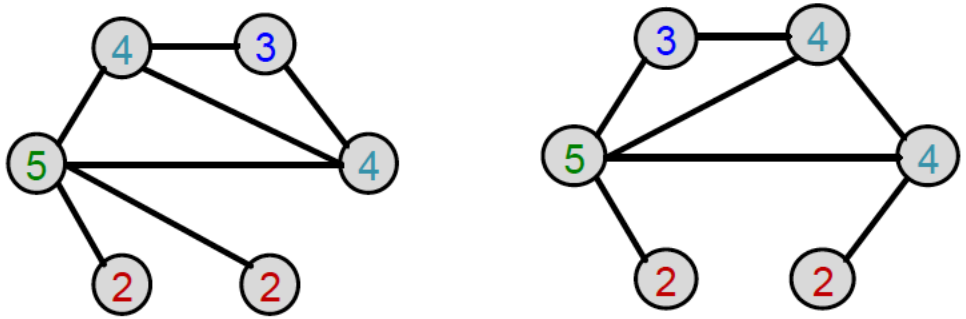


Color Refinement: Example

- Aggregate neighboring colors



- Hash aggregated colors

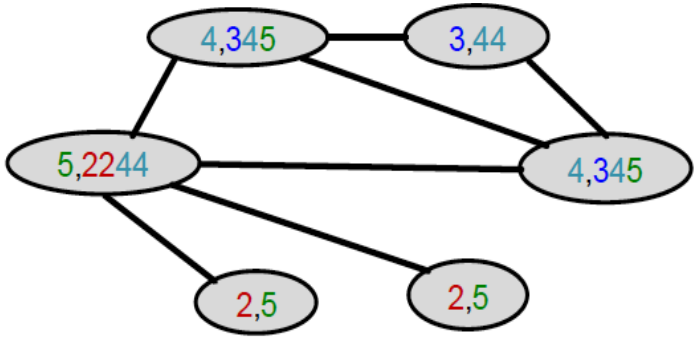


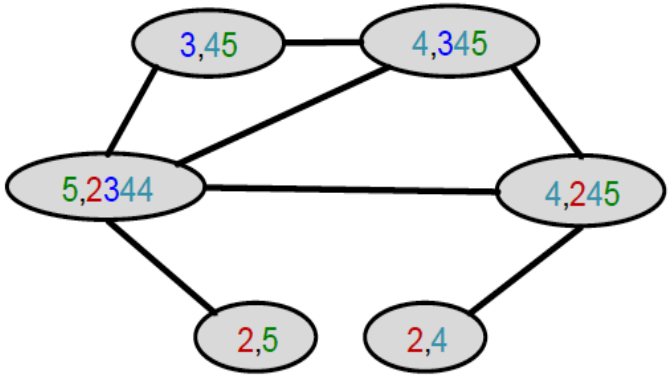
Hash table

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

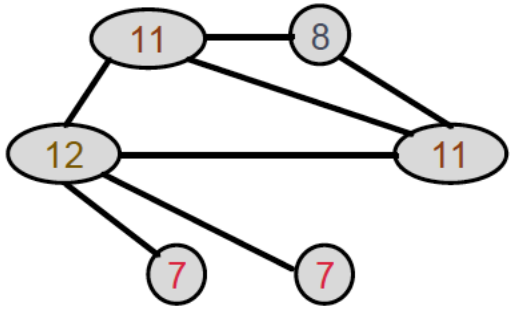
Color Refinement: Example

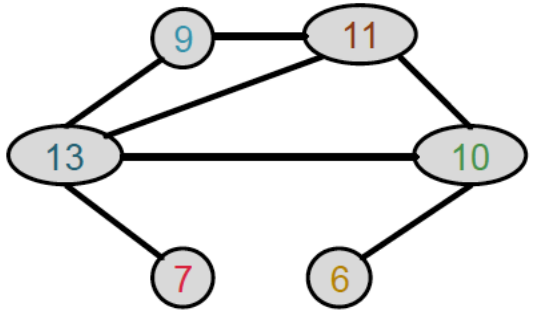
- Aggregate neighboring colors





- Hash aggregated colors



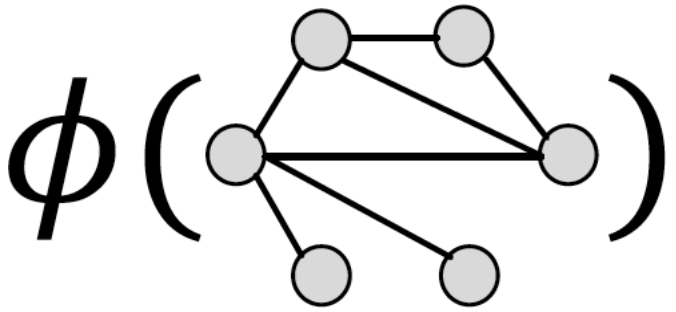


Hash table

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 #nodes with a given color

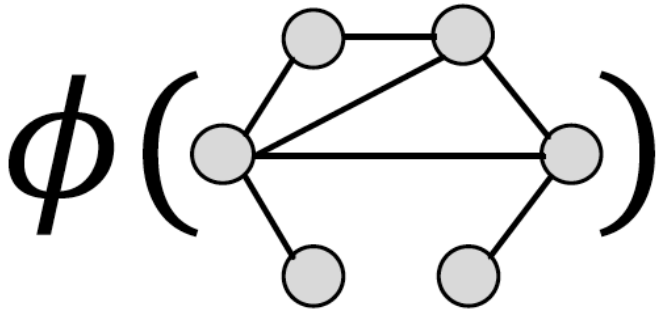


Colors

1,2,3,4,5,6,7,8,9,10,11,12,13

= [6,2,1,2,1,0,2,1,0, 0, 2, 1, 0]

Counts

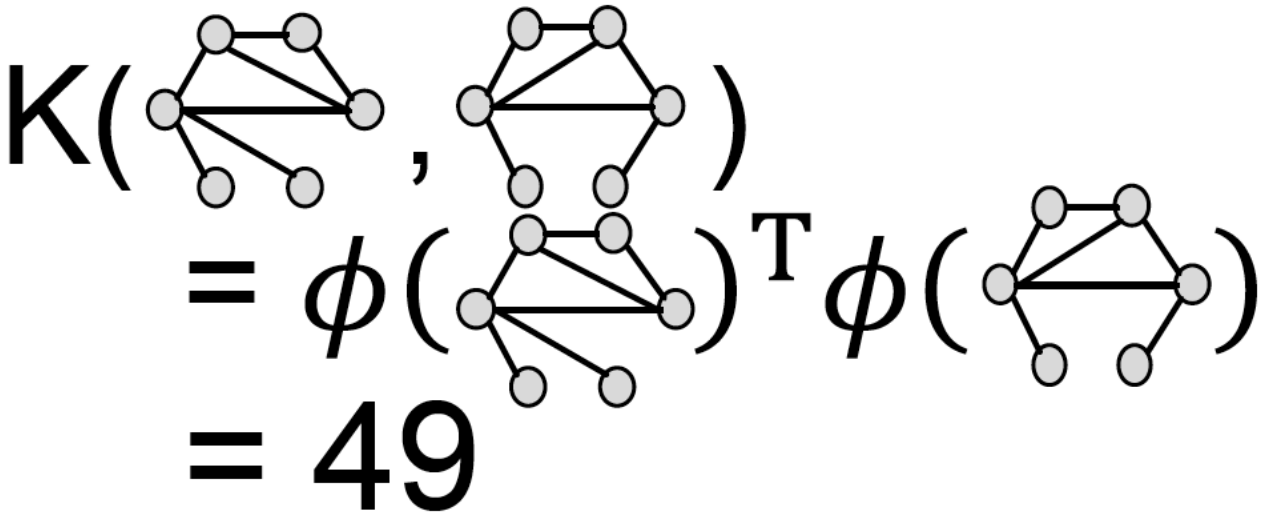


1,2,3,4,5,6,7,8,9,10,11,12,13

= [6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

Weisfeiler-Lehman Kernel

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



Weisfeiler-Lehman Kernel

- Computationally efficient
  - Time complexity for color refinement at each step is linear in #edges.
- When computing a kernel value, only colors appeared in the two graphs need to be tracked
  - Thus, #color 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 Nets (will study later)

Summary

- Traditional ML pipeline
  - Hand-crafted (structural) features + ML models
- Hand-crafted features for graph data
  - Node-level: node degree, centrality, clustering coefficient, graphlets
  - Link-level: distance-based features, local/global neighborhood overlap
  - Graph-level: graphlet kernel, WL kernel
- However, we only considered featurizing the graph structure (but not the attribute of nodes and their neighbors)