Machine Learning on Graphs

COMP9312_24T2

Outline

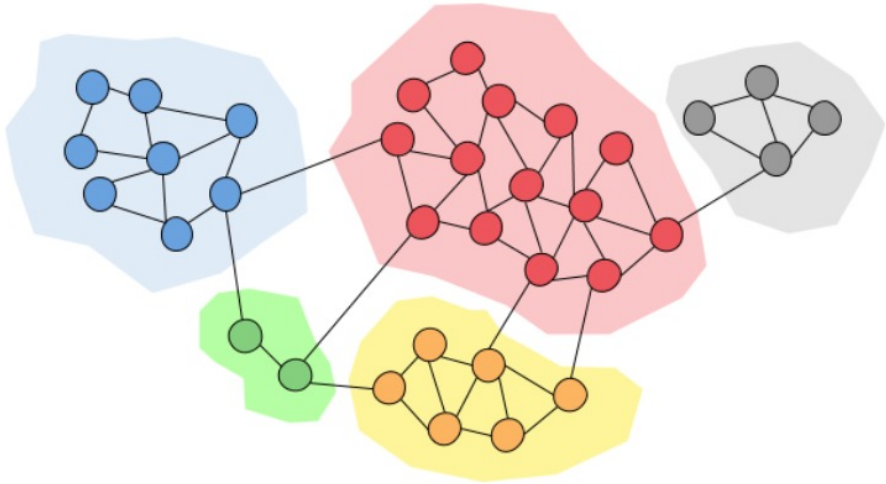
- Machine Learning on Graphs
- Feature Engineering
  - Node Level
  - Edge Level
  - Graph Level

Data Structure & Algorithms

Case studies on Community Detection:

Connected Component, K-Core, K-Truss, Clique, …

Clustering/partition algorithms, …

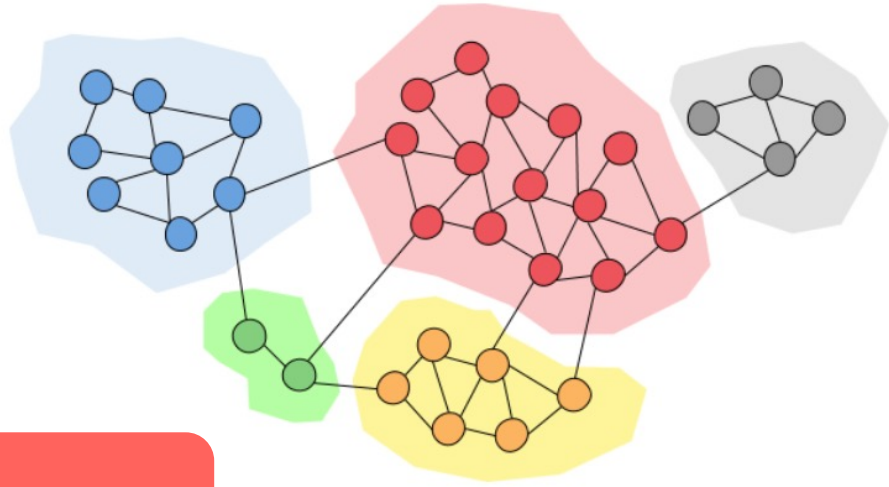


Step 1: Formulate problem → Step 2: Design Algorithm

It is hard in many applications.

Learning-based Algorithms

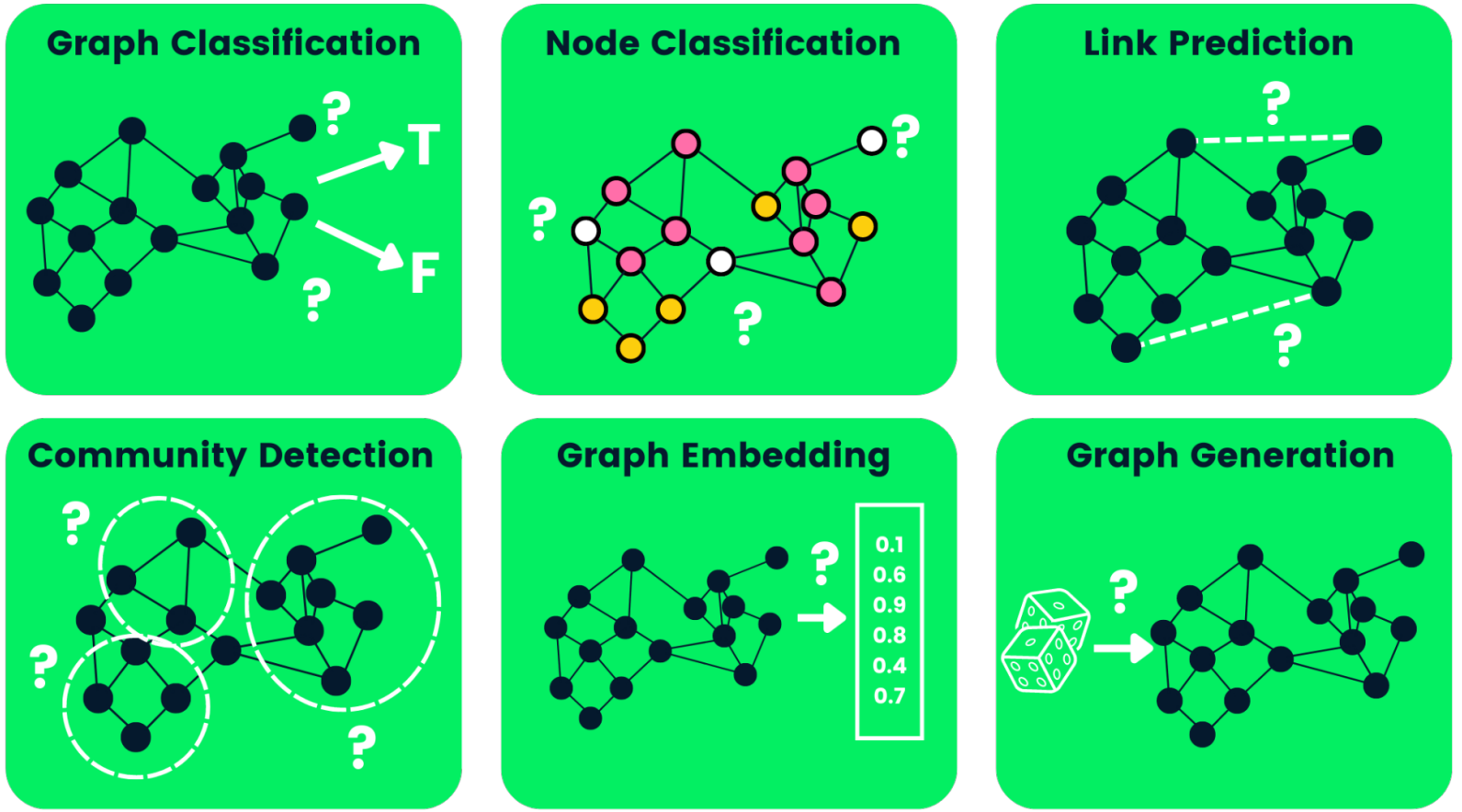
- It is hard to define a good community.
- It is not hard to judge a community.



Step 1: Decide parameters → Step 2: Define loss function → Step 3: Training

Efficiency VS Effectiveness

Application



https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial

Application

Node classification: Predict a property of a node

Example: Categorize online users / items

Link prediction: recommendation

Example: Knowledge graph completion

Graph classification: Categorize different graphs

Example: Molecule property prediction

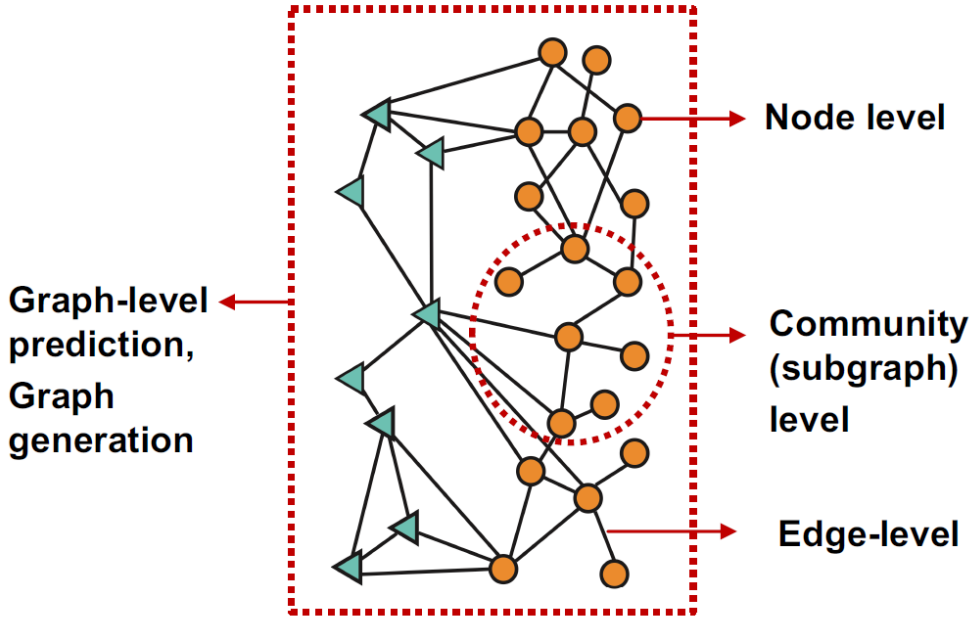
Clustering: Detect if nodes form a community

Example: Social circle detection

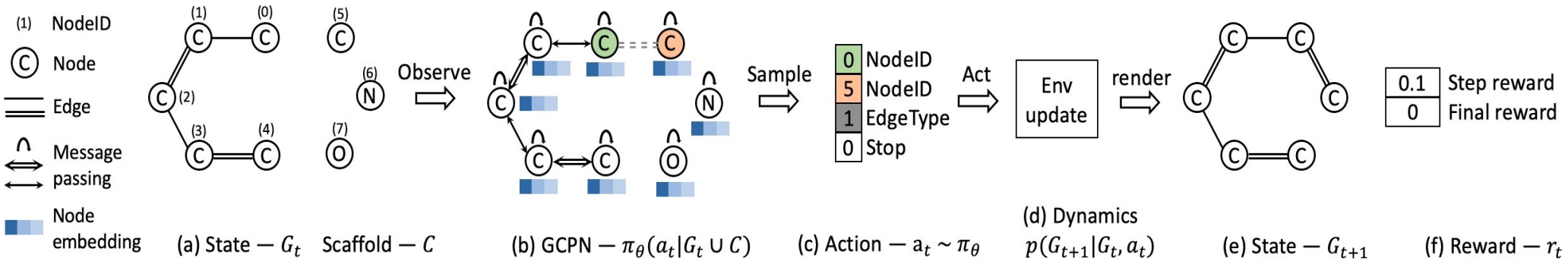
Other tasks:

Graph generation: Drug discovery

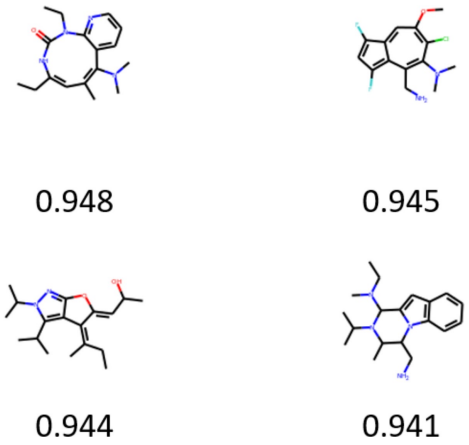
Graph evolution: Physical simulation



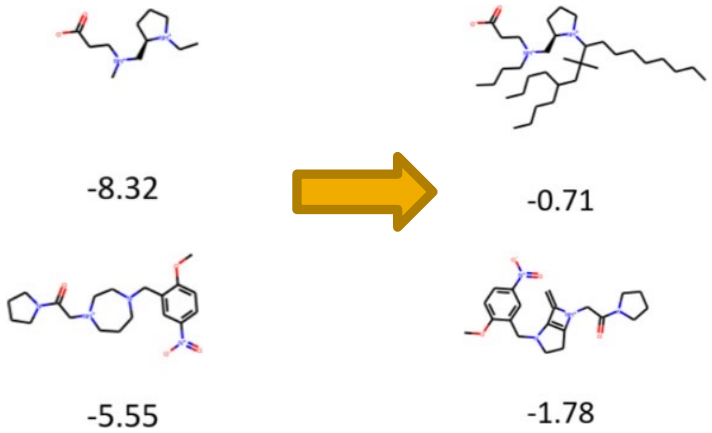
Application: Molecule Generation



Use case 1: Generate novel molecules with high drug likeness



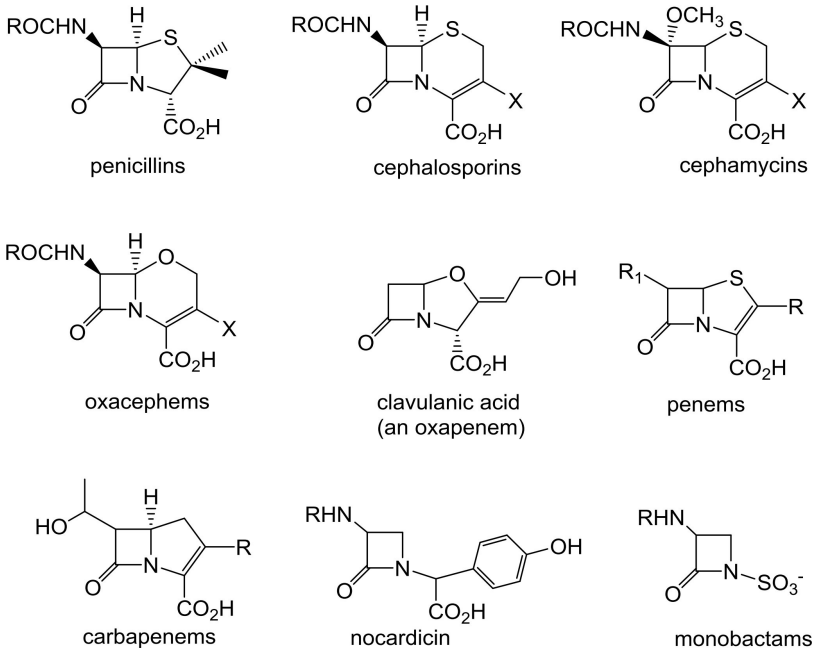
Use case 2: Optimize existing molecules to have desirable properties



Application: Drug Discovery

Antibiotics are small molecular graphs

- Nodes: Atoms
- Edges: Chemical bonds



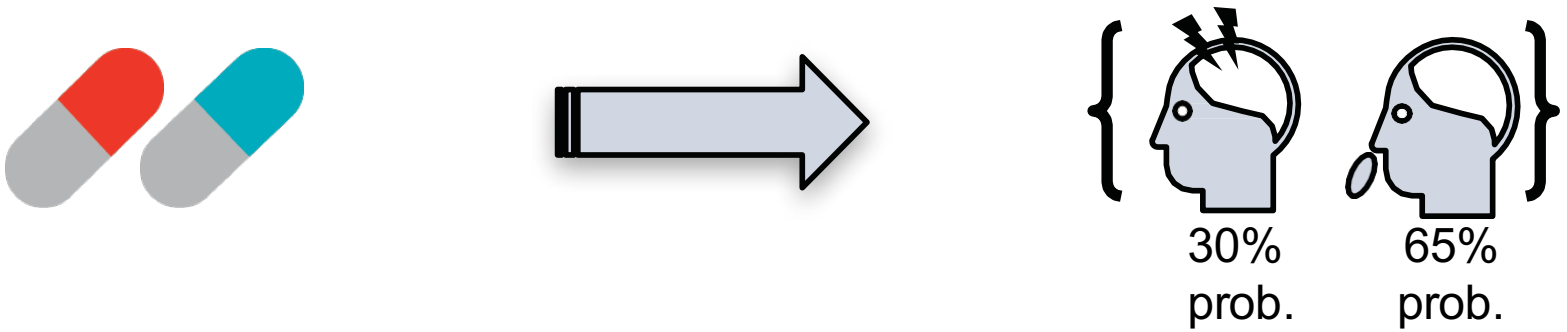
Konaklieva, Monika I. "Molecular targets of β-lactam-based antimicrobials: beyond the usual suspects." Antibiotics 3.2 (2014): 128-142.

Application: Drug Side Effects

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

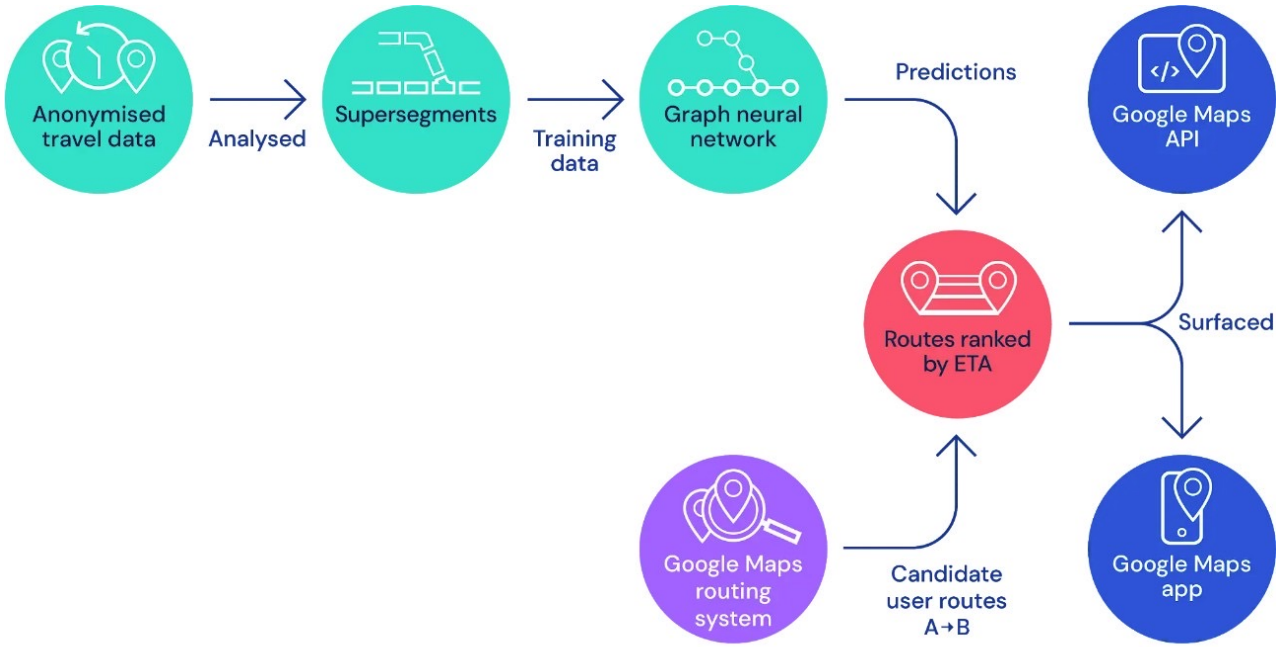
- 46% of people ages 70-79 take more than 5 drugs
- Many patients take more than 20 drugs to treat heart disease, depression, insomnia, etc.

Task: Given a pair of drugs predict adverse side effects



Application: Google Map

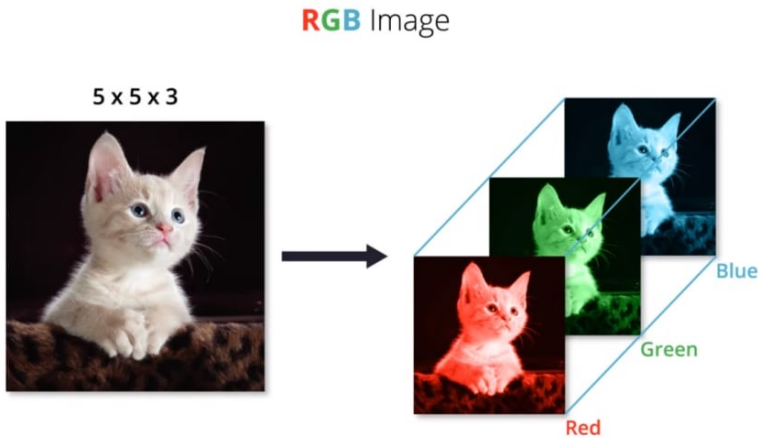
Predict via Graph Neural Networks

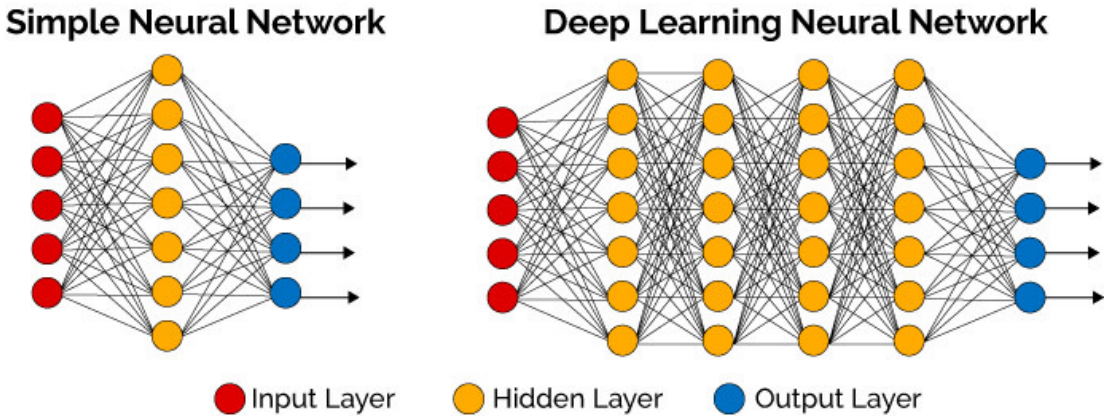


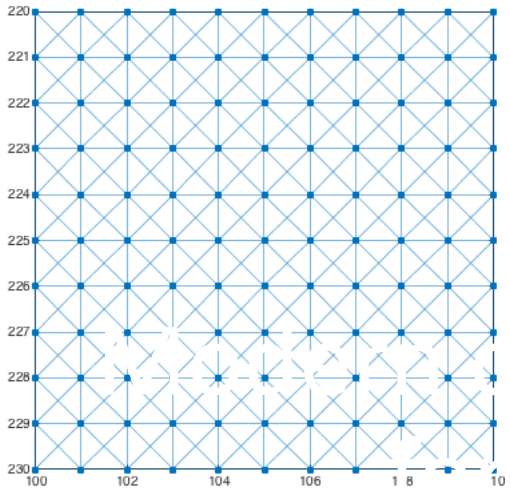
THE MODEL ARCHITECTURE FOR DETERMINING OPTIMAL ROUTES AND THEIR TRAVEL TIME.

Image credit: DeepMind

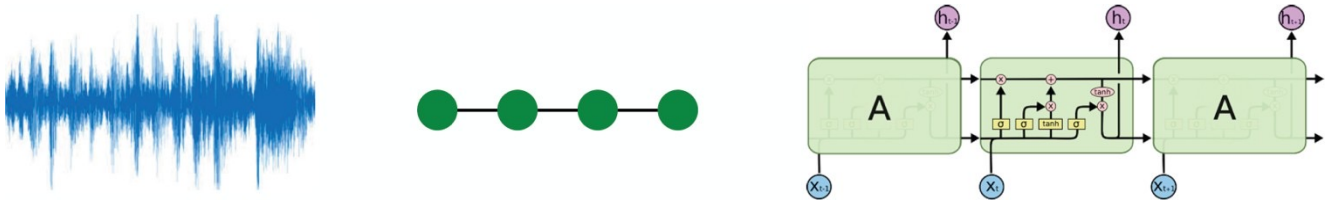
ML/DL on traditional data







Images

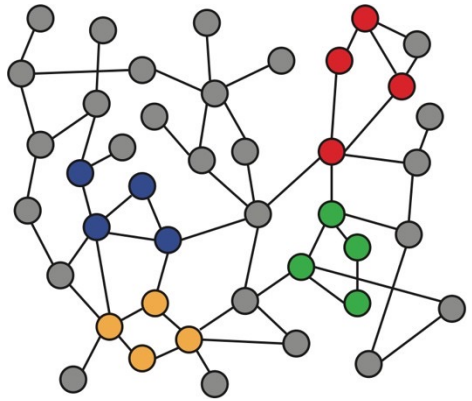


Text/Speech/Audio

Challenges

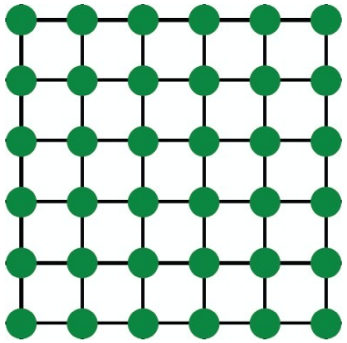
Graphs are complex

- Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)
- No fixed node ordering or reference point
- Often dynamic and have multimodal features



Graph

VS



Image

Text

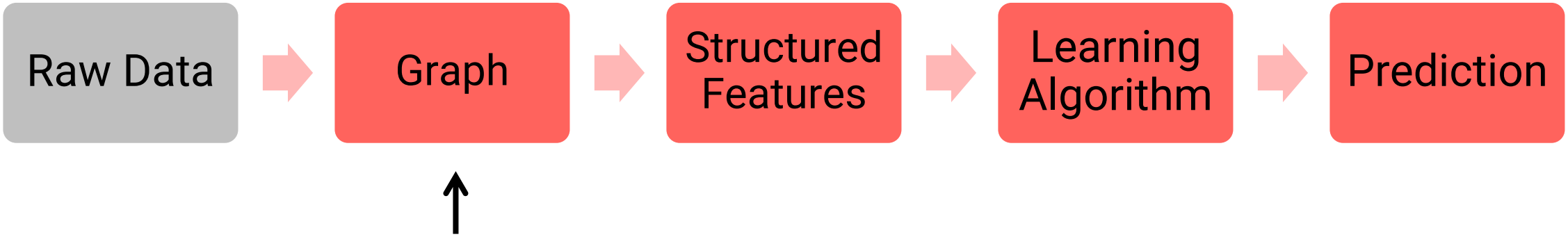
GNN: How to get features

1. Feature Engineering

Covered in this topic

2. Graph Representation Learning

Node embedding

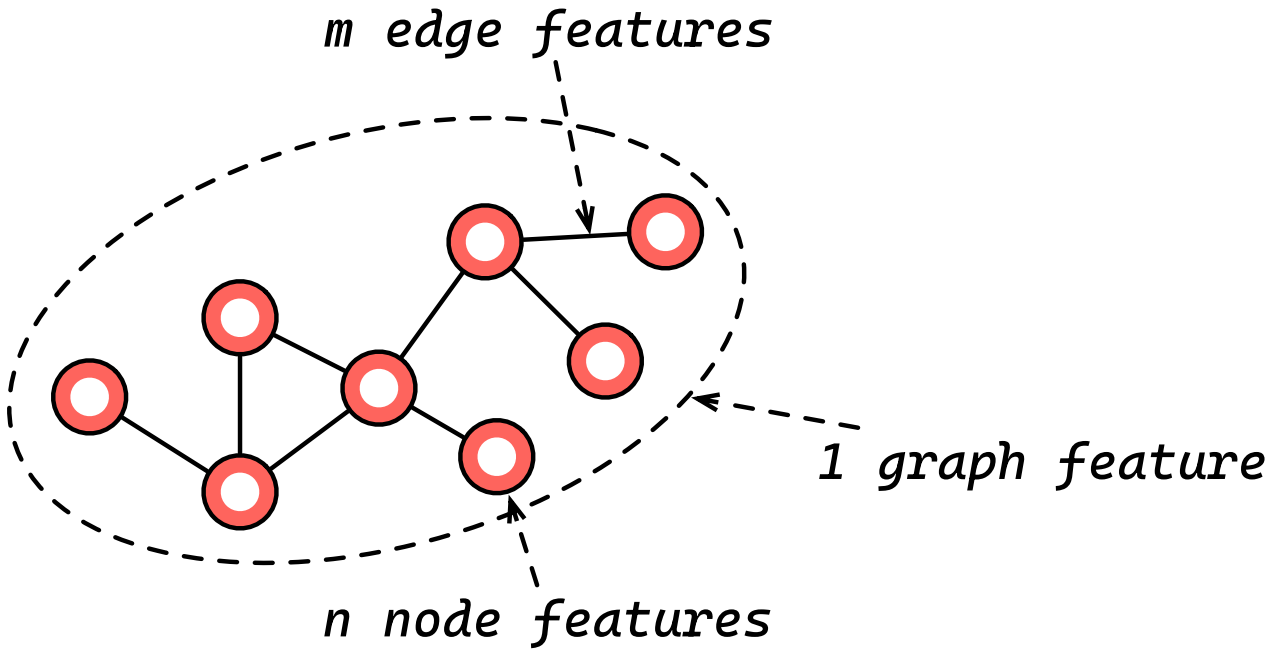


Adjacency Matrix? Adjacency List? CSR?

Machine learning needs features!

Different types of graph features

- Node Level
- Edge Level
- Graph Level



Traditional ML on Graphs

Good features effectively represent the graph structure and achieve good performance.

1. Design features for **nodes/edges/graphs**.
2. Get features additional features from training data.
3. Use features to train parameters.

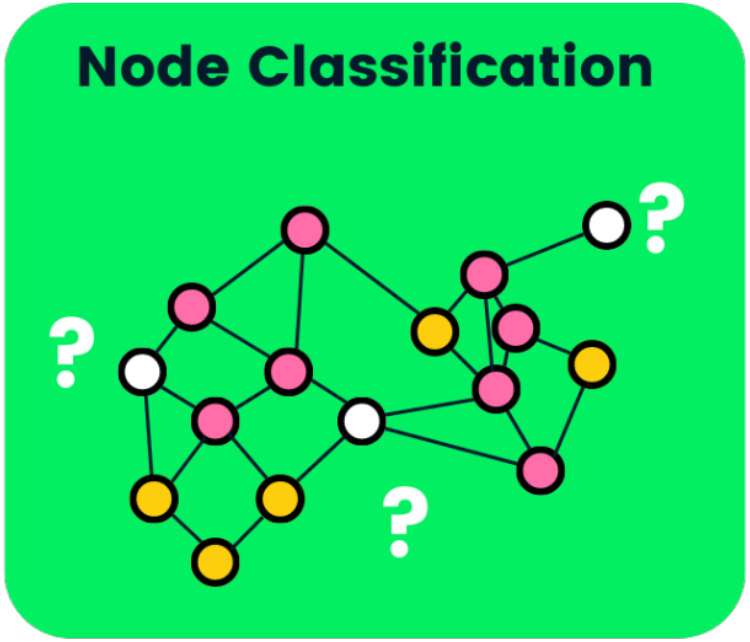
Testing: predict using the feature of query node/link/graph

Node Level Features

Node-Level Features

Goal:

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

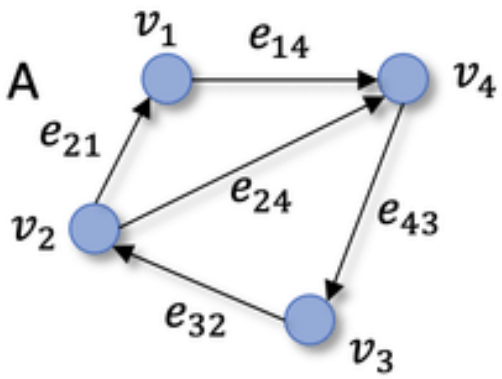


A typical application: node classification

Adjacency Matrix?

Directed graph G(V,E)

A

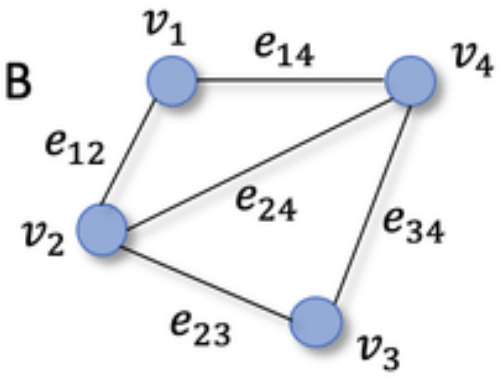


E

	v_1	v_2	v_3	v_4
v_1	0	0	0	1
v_2	1	0	0	1
v_3	0	1	0	0
v_4	0	0	1	0

Undirected graph G(V,E)

B

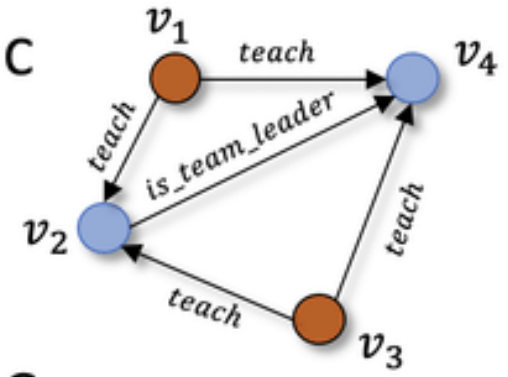


F

	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	1
v_3	0	1	0	1
v_4	1	1	1	0

Knowledge graph G(V,E)

C

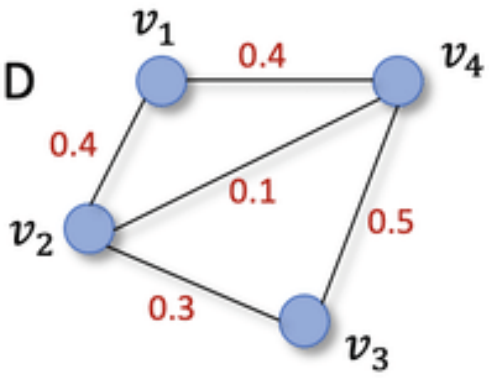


G

	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	0	0	0	1
v_3	0	1	0	1
v_4	0	0	0	0

Weighted graph G(V,E)

D

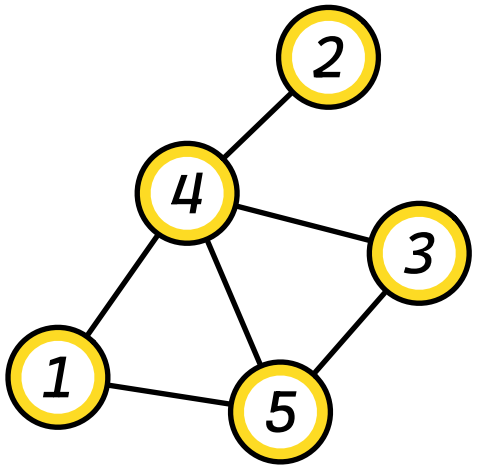


H

	v_1	v_2	v_3	v_4
v_1	0	0.4	0	0.4
v_2	0.4	0	0.3	0.1
v_3	0	0.3	0	0.5
v_4	0.4	0.1	0.5	0

Not working for big graphs!

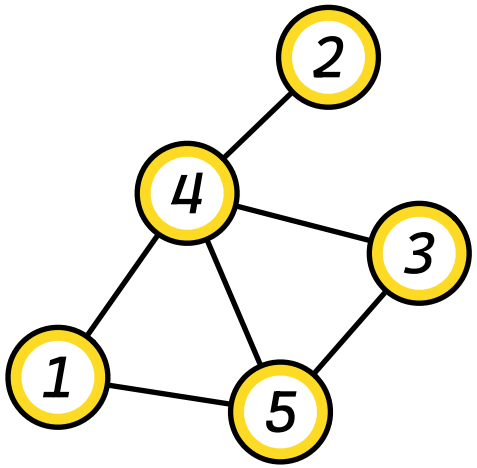
Adjacency List?



v1	4	5		
v2	4			
v3	4	5		
v4	1	2	3	5
v5	1	3	4	

Feature dimension need to be consistent

Adjacency List?



How about this?

v1	4	5	0	0
v2	4	0	0	0
v3	4	5	0	0
v4	1	2	3	5
v5	1	3	4	0

Feature dimension need to be consistent

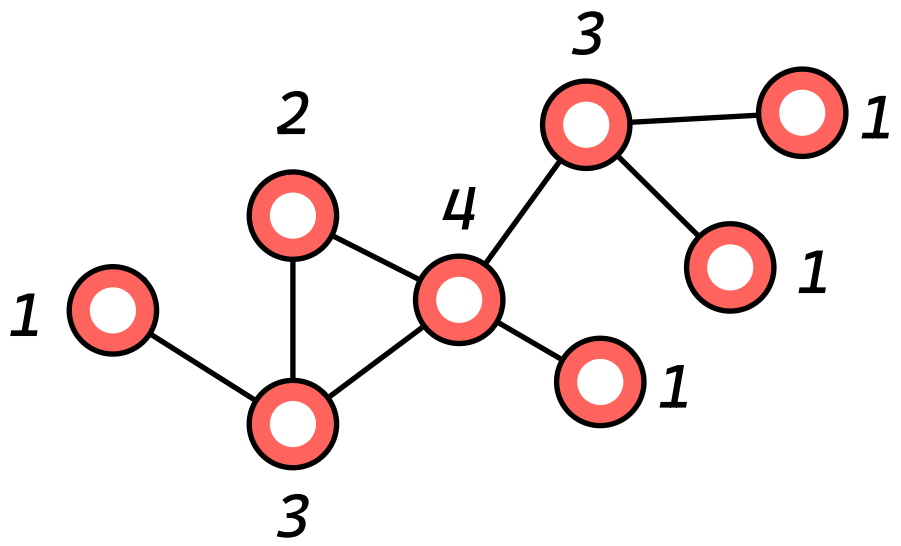
Node-Level Features: Overview

- Node degree
- Clustering coefficient
- Graphlets
- Node centrality
- …

Node Degree

Degree of a node: the number of neighbors.

Treat all neighbors equally.

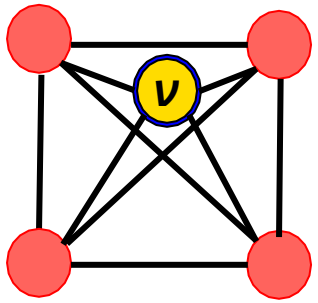


Node Centrality: Clustering Coefficient

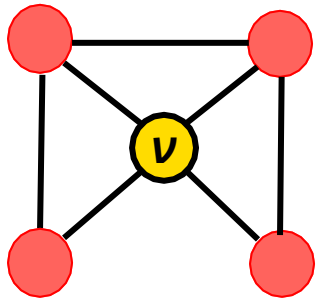
Measures how connected $v's$ neighboring nodes are:

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

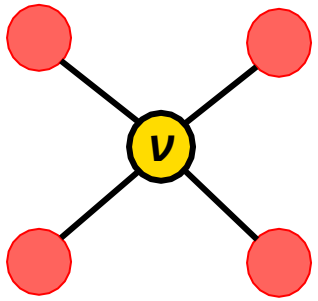
Can be also understand as #triangles/#possible triangles



$e_v = 1$



$e_v = 0.5$



$e_v = 0$

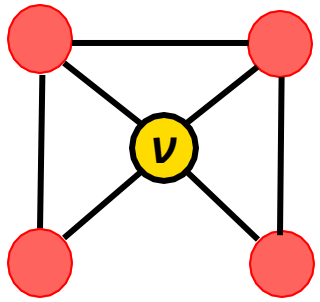
Ego-network:
the induced subgraph of the node and all its neighbors

Computing Clustering Coefficient

Can you design an algorithm to compute the clustering coefficient of all nodes in a graph with n nodes and m edges?

Node Features: Graphlets

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



$$e_v = 0.5$$

Three triangles in 6 possible triplets

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

Node Features: Graphlets

Graphlets are small subgraphs.

We aim to describe network structure around the node based on graphlets.

Analogy:

Degree

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

Clustering coefficient

counts **#(triangles)** that a node is involved.

Graphlet Degree Vector (GDV):

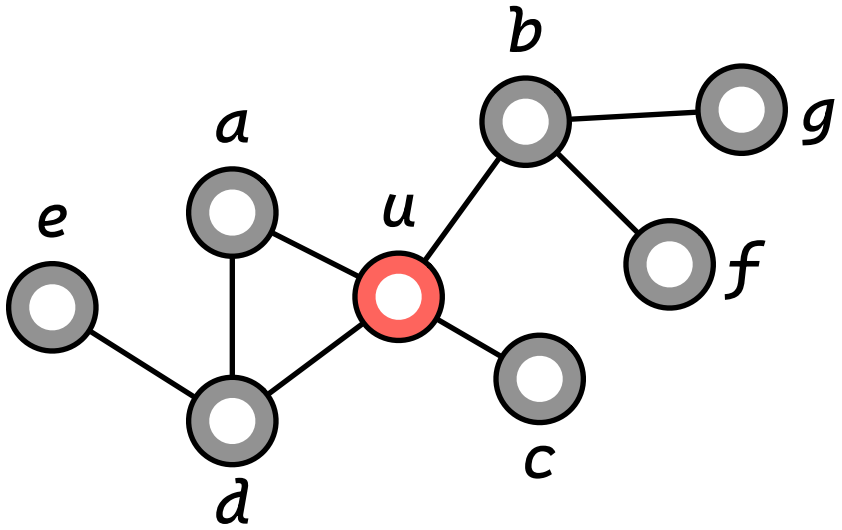
Graphlet-base features for nodes

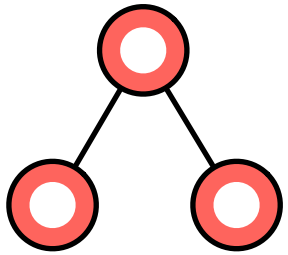
GDV counts **#(graphlets)** that a node is involved.

Node Features: Graphlets

How to represent a node by graphlets?

Let's start by considering (connected) graphlets with three nodes:



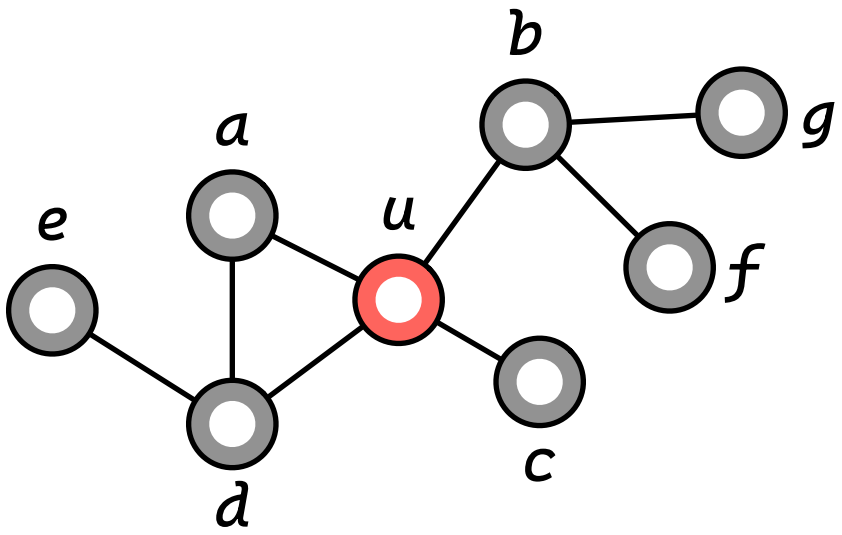


Choose a specific pattern (wedge)

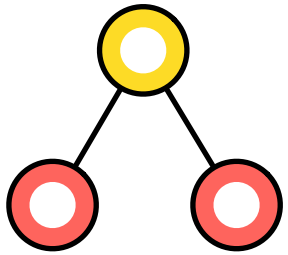
How many subgraphs containing u that are isomorphic to the pattern?

Node Features: Graphlets

How many subgraphs containing u that are isomorphic to the pattern?



11 after removing symmetric cases

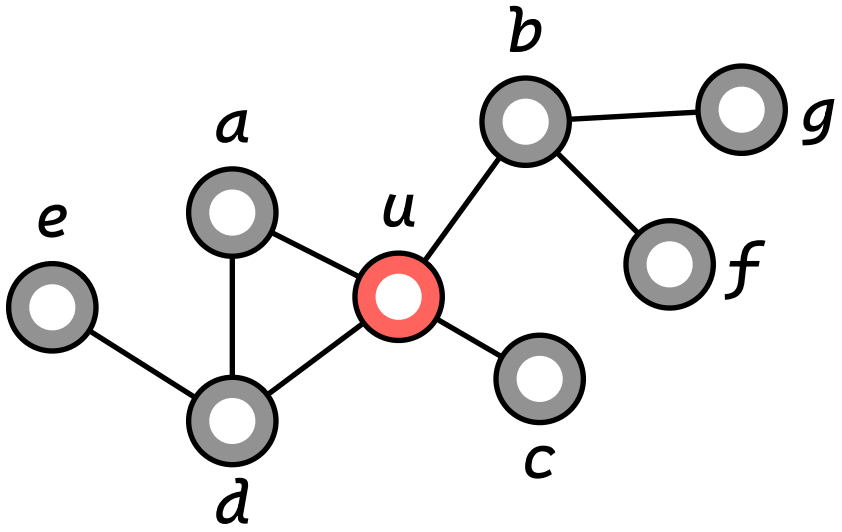


Choose a specific pattern (wedge)

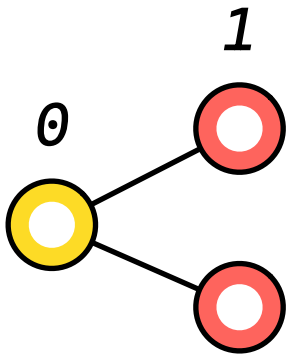
We use 11 as the feature of u

Node Features: Graphlets

Move forward by utilizing different types:



6 for type 0
5 for type 1

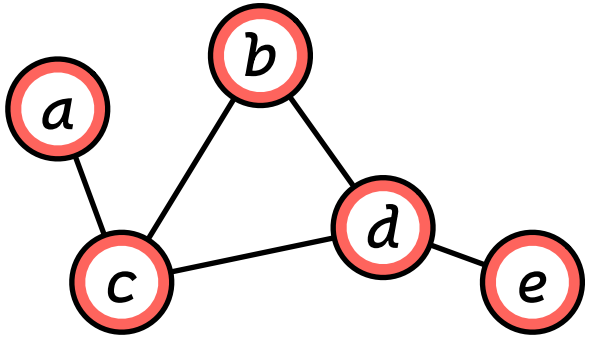


Choose a specific pattern (wedge)

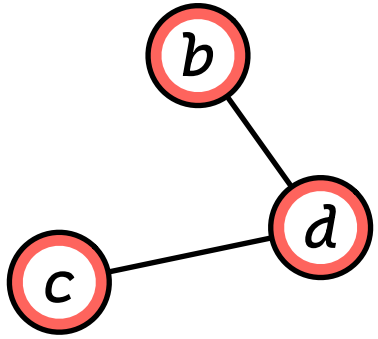
We use **[6,5]** as the feature of **u**

Node Features: Graphlets

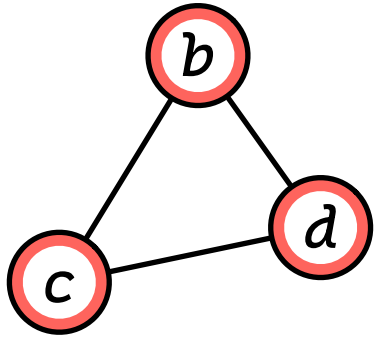
Move forward by only considering induced matching instances:



A graph



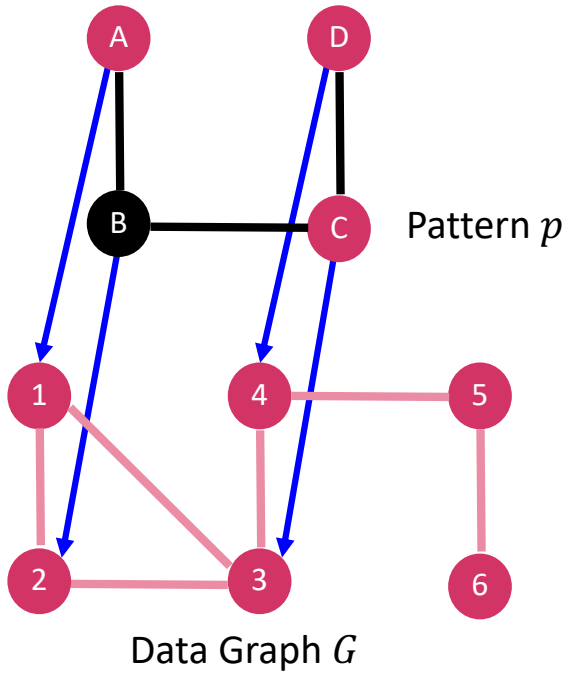
A non-induced subgraph



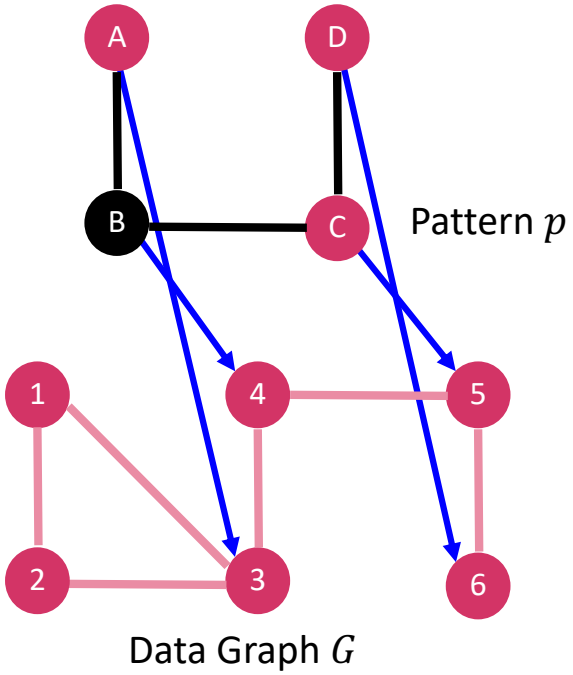
An induced subgraph

Node Features: Graphlets

Move forward by only considering induced matching instances:



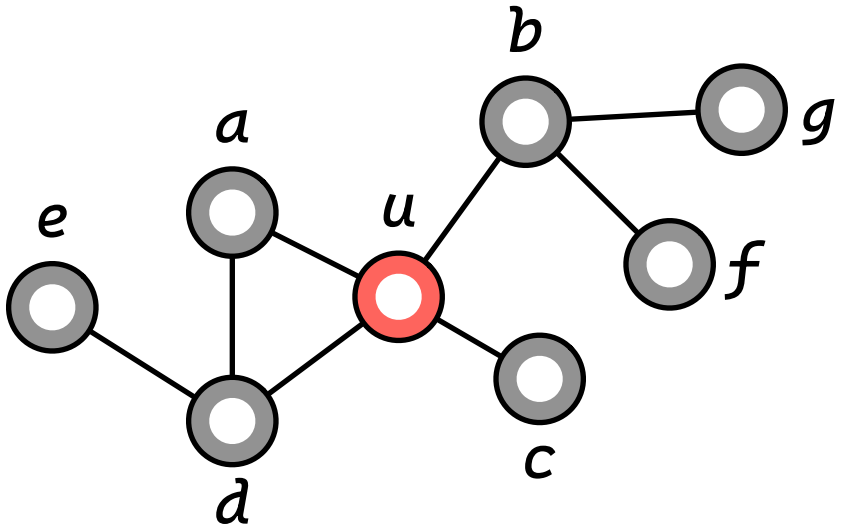
non-induced matching

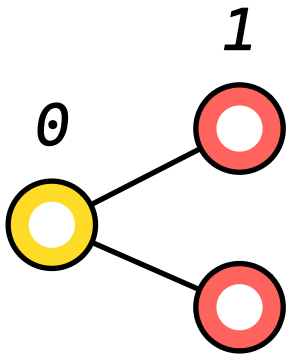


induced matching

Node Features: Graphlets

Move forward by only considering **induced** matching instances:



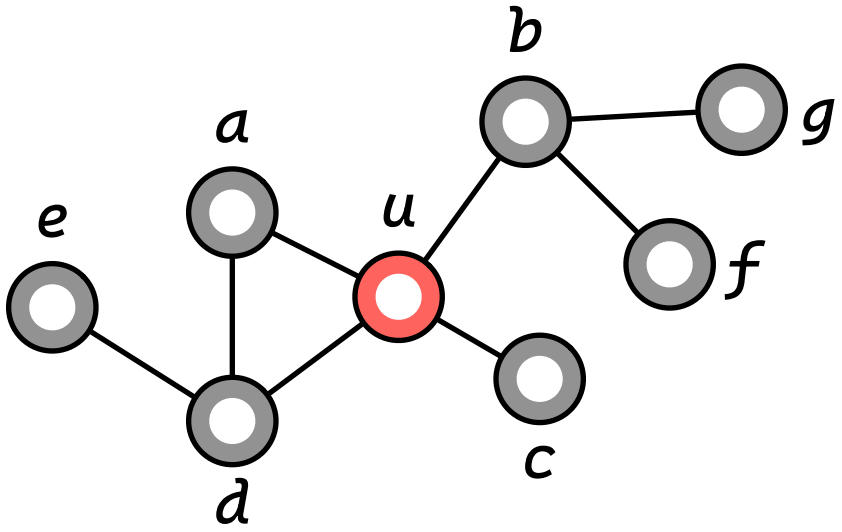


Choose a specific pattern (wedge)

We use $[5,3]$ as the feature of u

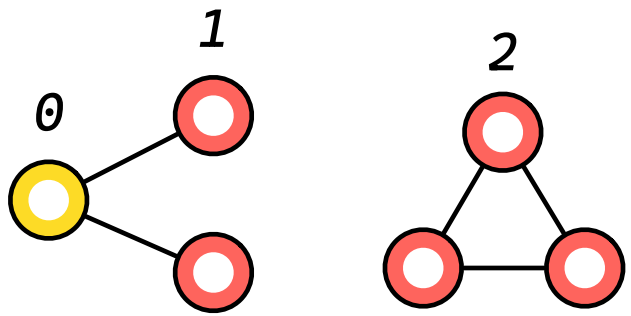
Node Features: Graphlets

Move forward by utilizing all 3-graphlets:



`6 for type 0`
`5 for type 1`
`1 for type 2`

Type 0:



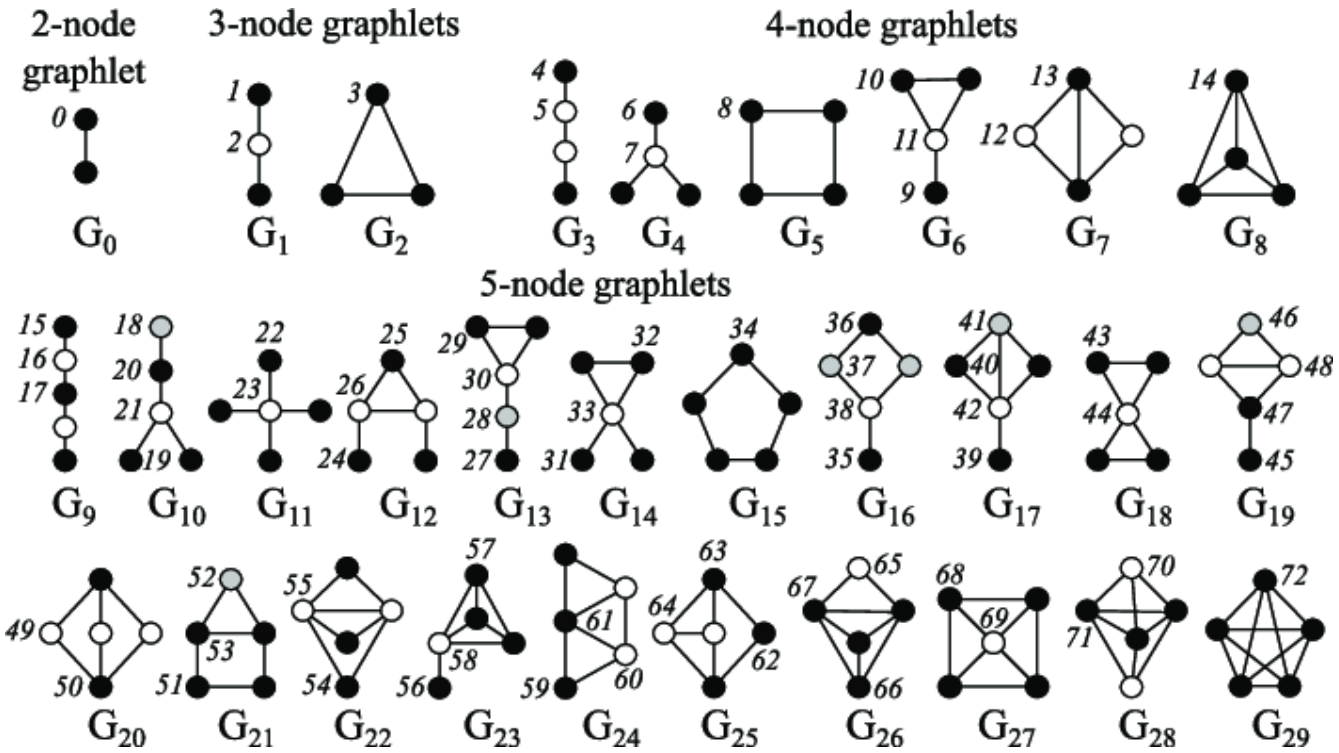
There are three types in all 3-graphlets.

We use `[5,3,1]` as the feature of u

Node Features: Graphlets

Consider all graphlets with <= 5 nodes

How many node roles in all connected non-isomorphic subgraphs?



There are **73** different graphlet roles of up to 5 nodes.

To get the node feature, compute the number of induced matching instances for each role id.

Node Features: Graphlets

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

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:

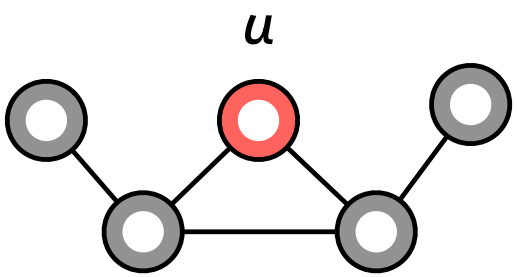
- Comparing vectors of two nodes provides a more detailed measure of local topological similarity than node degrees or clustering coefficient.

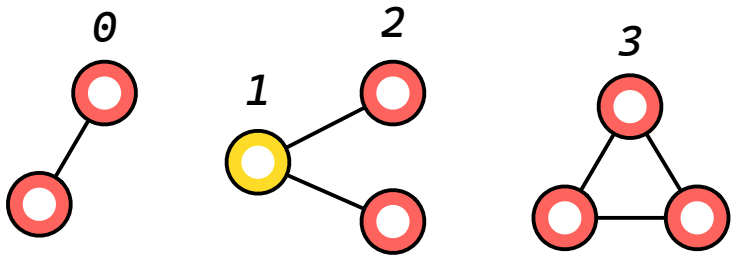
Usually, we only compute up to 4 or 5 nodes . . .

Node Features: Graphlets

More examples:

GFV(u) = [2,0,2,1]





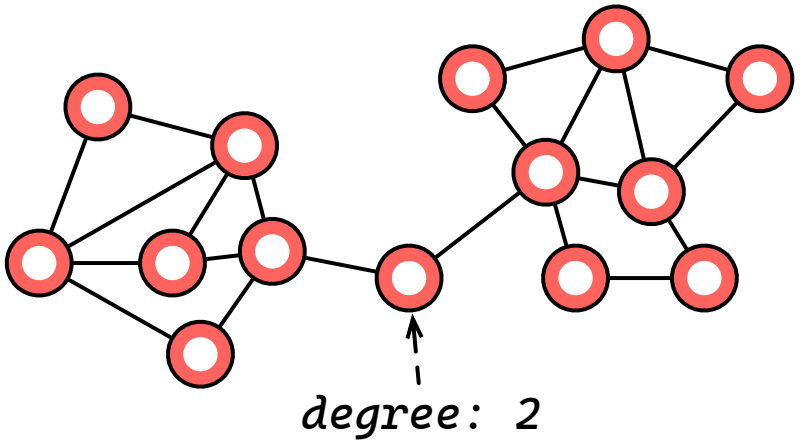
Node Centrality

Node degree counts neighbors without capturing their importance.

Node **centrality** takes the node importance in a graph into account

Different ways to model importance:

- PageRank
- Eigenvector centrality
- Betweenness centrality
- Closeness centrality
- many others…



Node Centrality: Eigenvector

Motivation

A node is important if surrounded by important neighbors.

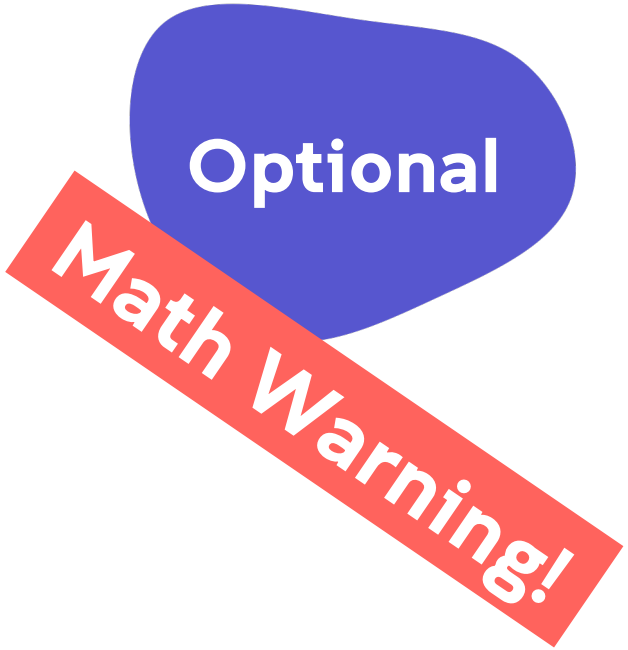
We model the centrality of node v as the sum of the centrality of neighbors:

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

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

The above equation models centrality in a recursive manner.
How do we solve it?

Node Centrality: Eigenvector



Rewrite the recursive equation in the matrix form.

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

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

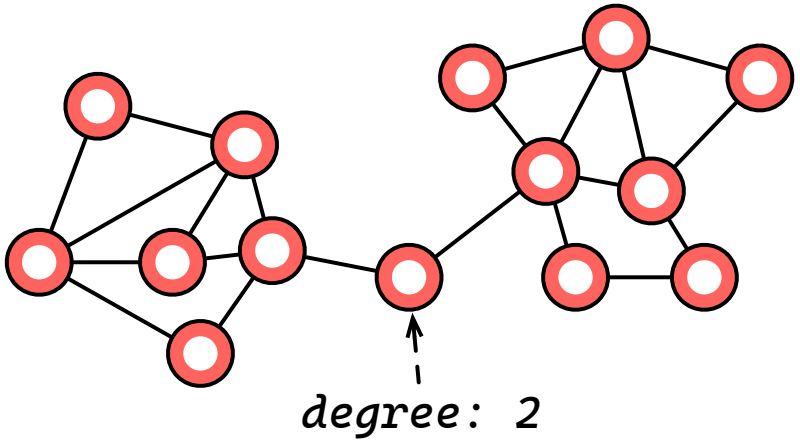
- We see that centrality c is the **eigenvector of $\boldsymbol{A}$**!
- The largest eigenvalue λ_{max} is always positive and unique (by Perron-Frobenius Theorem).
- The eigenvector $\boldsymbol{c}_{max}$ corresponding to λ_{max} is used for centrality.

Node Centrality: Betweenness

Betweenness centrality:

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

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

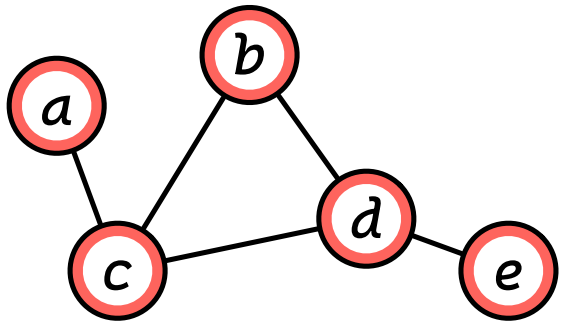


How to identity the bridge node

Node Centrality: Betweenness (cont)

Example:

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



$c_a = c_b = c_e = 0$

$c_c = 3$
(a–c–b, a–c–d, a–c–d–e)

$c_d = 3$
(a–c–d–e, b–d–e, c–d–e)

Computing Betweenness Centrality

Exact solution:

$O(nm)$ for unweighted graphs

$O(nm+n^2 \log n)$ for weighted graphs

https://kops.uni-konstanz.de/server/api/core/bitstreams/420590d1-3010-4eab-a585-6fa3eff46f9e/content

Approximate solution:

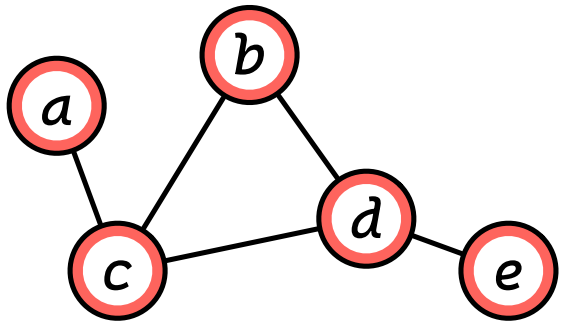
Sampling a set of shortest paths…

Node Centrality: Closeness

Closeness centrality:

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

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



c_a = 1/(2 + 1 + 2 + 3) = 1/8
(a–c–b, a–c, a–c–d, a–c–d–e)

c_a = 1/(2 + 1 + 1 + 1) = 1/5
(d–c–a, d–b, d–c, d–e)

Computing Closeness Centrality

Can you design an algorithm to compute the closeness centrality of all nodes in a graph with n nodes and m edges?

Node-level Feature: Summary

- Importance-based features:
  - Node degree
  - Different node centrality measures

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

Node-level Feature: Summary

Importance-based features: capture the importance of a node in a graph

- ~~Node degree:~~
  - Simply counts the number of neighboring nodes
- Node centrality:
  - Model importance of neighbors 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:

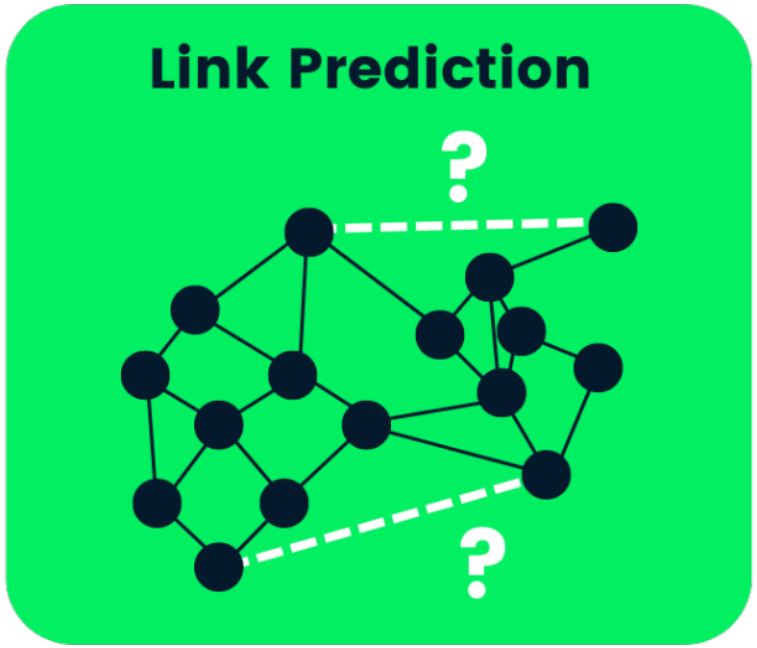
- **Example:** Predicting protein functionality in a protein-protein interaction network.

Edge (Node Pair) Level Features

A Typical Application

The task is to predict **new links** based on the existing links.

We need to design features for **a pair of nodes**



A Little Graph Theory

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.

Methodology of link prediction via proximity/similarity

1. For each pair of nodes u,v, compute sim(u,v)
2. sim can be any metric
3. Compute top-k similar pairs (u,v) such that u and v are not connected

Testing without Ground-Truth

Two formulations of the link prediction task:

1) Links missing at random (simple graphs):

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

2) Links over time (temporal graphs):

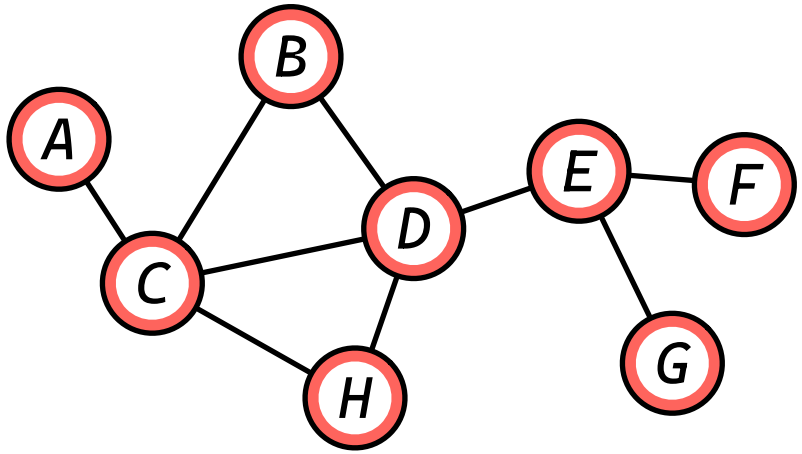
Use the graph snapshot at t_0 to predict edges at t_1

Link-level Features: Overview

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

Distance-based Features

Shortest-path distance between two nodes



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

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

However, this does not capture the degree of neighborhood overlap:

- Node pair *(B, H)* has 2 shared neighbors,
- while pairs *(B, E)* and *(A, B)* only have 1 such node.

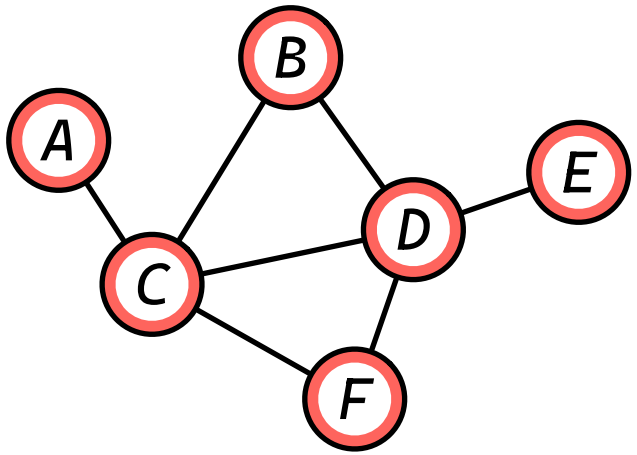
Local Neighborhood Overlap

Captures # neighbors shared between two nodes :

- **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:**

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

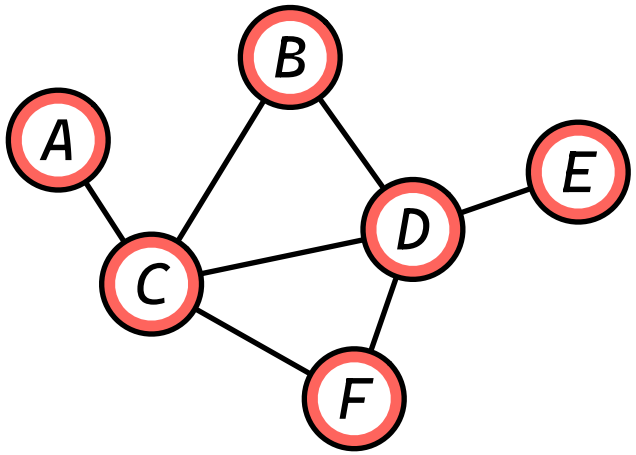
  - Example: $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$



Local Neighborhood Overlap (cont)

Limitation of local neighborhood features:

- Metric is always zero if the two nodes do not have any neighbors in common.
- The two nodes may still potentially be connected in the future.



Global Neighborhood Overlap

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

- Personalized PageRank
- **Katz Index**
- Simrank
- …

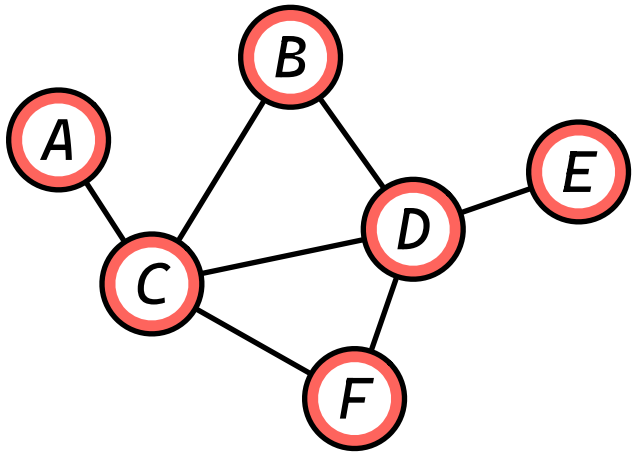
Global Neighborhood Overlap

Katz index:

consider the number of **paths** of all lengths between a given pair of nodes.

$p_{u,v}^{l}$ = #paths between u and v with length l

$Katz(u,v) = \beta * p_{u,v}^{1} + \beta^2 * p_{u,v}^{2} + \beta^3 * p_{u,v}^{3} + \ldots$



$Assume\ \beta = 1/2$

Katz(A,B) = 1/2*0 + 1/4*1 + 1/8*1 + 1/16*6...

4 paths:

ACACB, ACBCB, ACBDB, ACDCB, ACFDB, ACFCB

Global Neighborhood Overlap

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

Sum over *all path lengths*

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

#paths of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

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

by geometric series of matrices

Katz Index: Matrix Power

Optional

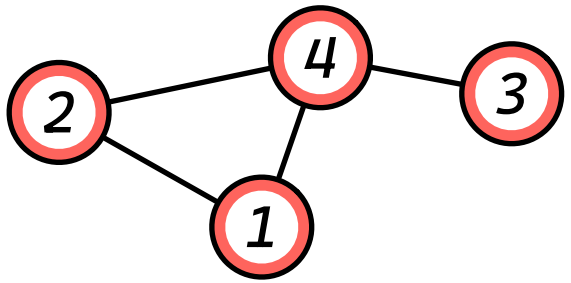
Q: How to compute #paths between two nodes?

Use powers of the graph adjacency matrix!

Recall: $\boldsymbol{A_{uv}} = 1$ if $u \in N(v)$

Let $\boldsymbol{P^{(K)}}$ = #paths of length $\boldsymbol{K}$ between $\boldsymbol{u}$ and $\boldsymbol{v}$

$\boldsymbol{P^{(K)} = A^k}$



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

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

Katz Index: Matrix Power

Optional

How to compute $P_{uv}^{(2)}$?

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

Node 1's neighbors

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

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

Power of adjacency

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

Katz Index: Matrix Power

Optional

How to compute #paths between two nodes?

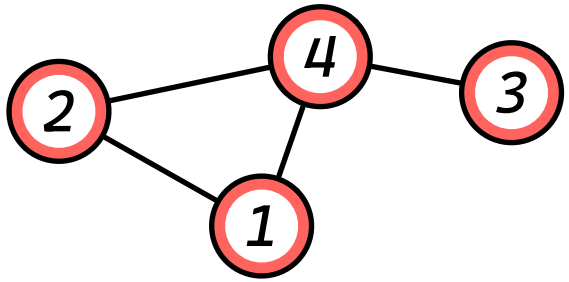
- Use **adjacency matrix powers**!
  - $\boldsymbol{A}_{uv}$ specifies #paths of length 1 (direct neighborhood) between u and v.

$\boldsymbol{A}^2_{uv}$ specifies #paths of **length 2** (neighbor of neighbor) between u and v.

$\boldsymbol{A}^l_{uv}$ specifies #paths of **length $\boldsymbol{l}$**.

Katz Index: Propagation

```
void katz(double decay, int u, int l){
    for each vertex v: katz[v] = 0, cur[v] = 0, next[v] = 0;
    cur[v] = 1;
    for{0 <= i < l}{
        for each vertex v{
            if cur[v] == 0: continue;
            for each neighbor w of v{
                next[w] += decay * cur[v];
                katz[w] += decay * cur[v];
            }
           cur[v]=0
        }
        swap(cur,next);
    }
}
```



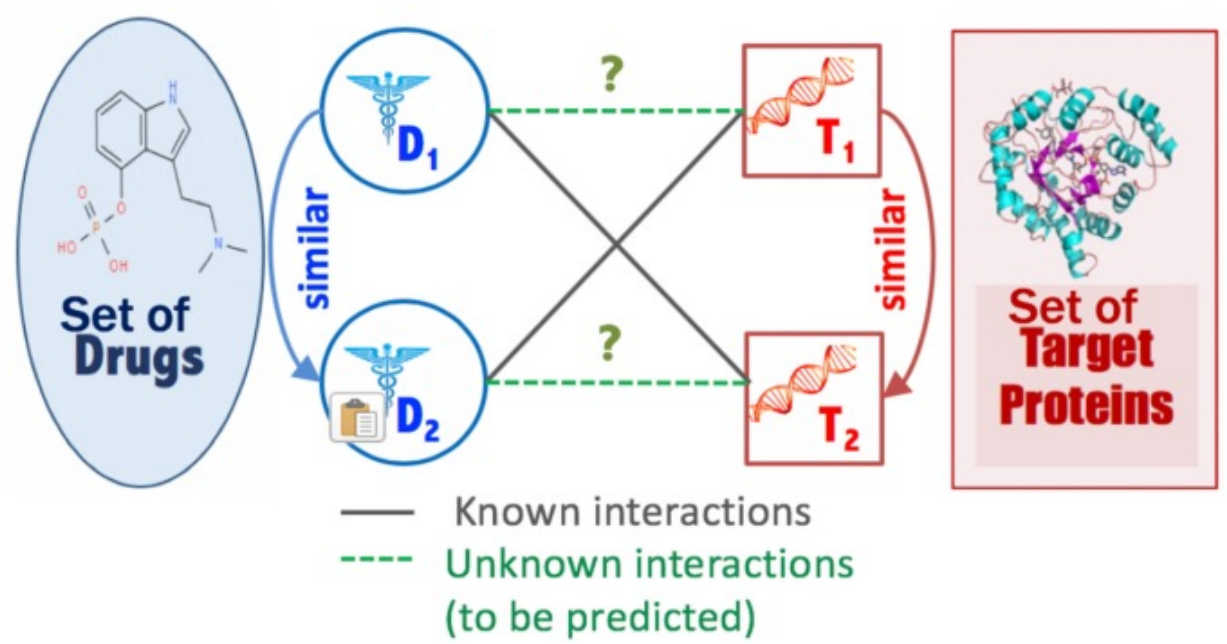
Edge-Level Features: Summary

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

Graph-Level Features & Graph Kernels

Graph-Level Features

Aim to get features to represent the structure of the whole graph.



Kernel Methods

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

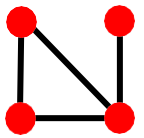
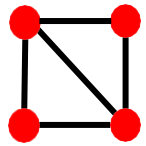
Graph Kernel: Overview

- Graph Kernels: Measure similarity between two graphs:
  - Graphlet Kernel [1]
  - Weisfeiler-Lehman Kernel [2]
  - Other kernels are also proposed in the literature

    (beyond the scope of this lecture)
    - Random-walk kernel
    - Shortest-path graph kernel
    - And many more…

Graph Kernel: Key Idea

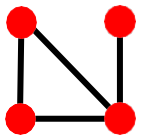
- **Goal**: Design graph feature vector $\phi(G)$
- **Key idea**: Bag-of-Words (BoW) for a graph
  - **Recall:** BoW simply uses the word counts as features for documents (no ordering considered).
  - Naïve extension to a graph: Regard nodes as words.
  - Since both graphs have 4 red nodes, we get the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

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

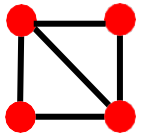
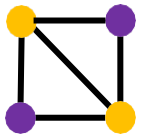
Graph Kernel: Key Idea

Use Bag of node degrees?

Deg1: ● Deg2: ● Deg3: ●

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

Obtains different features for different graphs!

$\phi($  $)$ = count() = [0, 2, 2]

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

Graph Kernel: Graphlet Features

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

Definition of graphlets here is slightly different from node-level features.

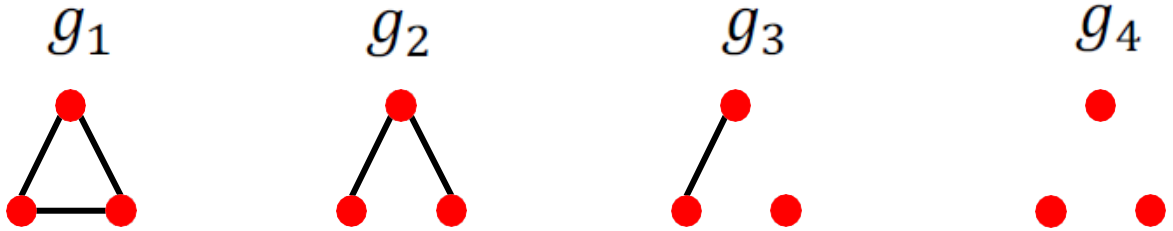
The two differences are:

- Nodes in graphlets here do not need to be connected (allows for isolated nodes)
- The graphlets here are not rooted (i.e., different roles).

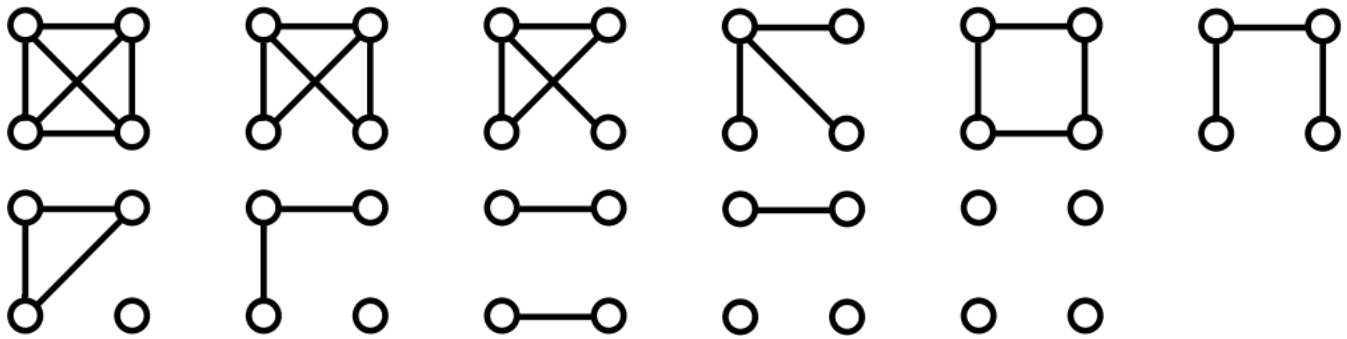
Graph Kernel: Graphlet Features

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.



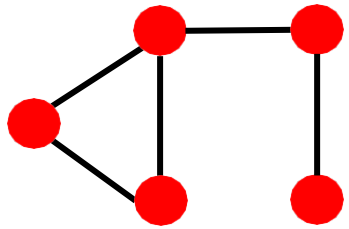
- For k = 4 , there are 11 graphlets.



Graph Kernel: Graphlet Features

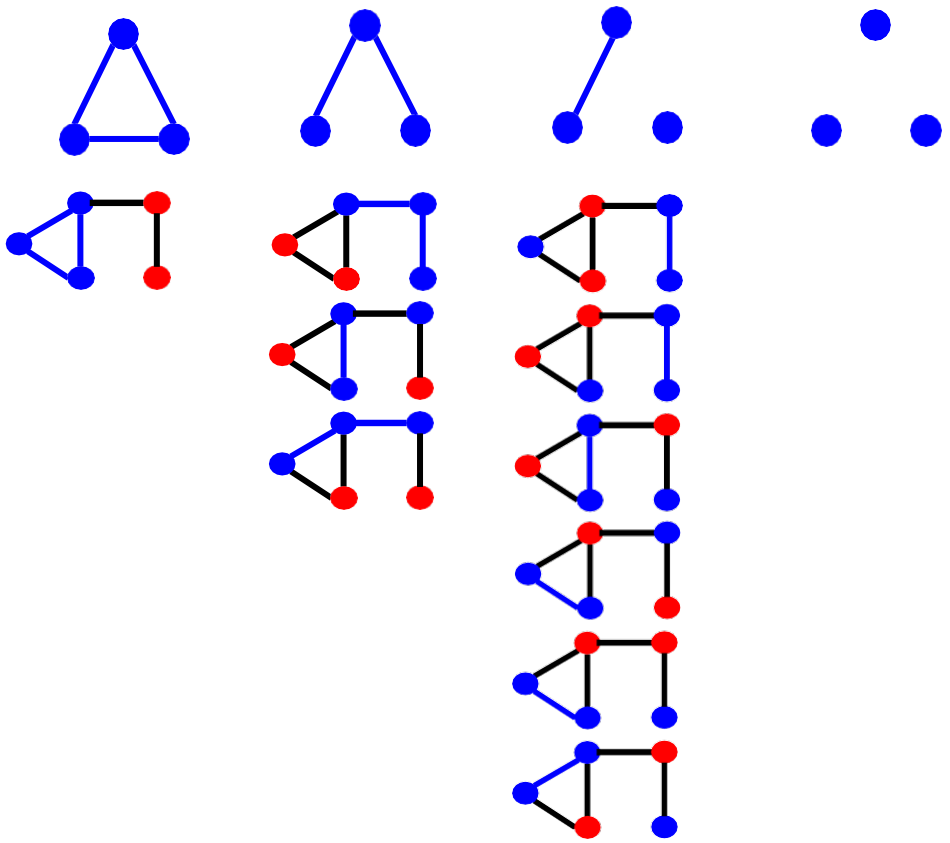
Example for k = 3.

G



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

g_1 g_2 g_3 g_4



Graph Kernel: Graphlet Features

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

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

Graphlet Kernel

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

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

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

Solution: normalize each feature vector

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

Graphlet Kernel

Limitations: Counting graphlets is expensive!

- Counting size-k graphlets for a graph with size n by enumeration takes n^k.
- This is unavoidable in the worst-case since isomorphism test (judging whether a graph is a subgraph of another graph) is **NP-hard**.
- If a graph's node degree is bounded by d, an $O(nd^{k-1})$ algorithm exists to count all the graphlets of size k.

Can we design a more efficient graph kernel?

Weisfeiler-Lehman Graph Kernel

Goal: design an efficient graph feature descriptor $\phi(G)$

Idea: use neighborhood structure to iteratively enrich node vocabulary.

- Generalized version of **Bag of node degrees** since node degrees are one-hop neighborhood information.

Algorithm to achieve this:

Color refinement

Color Refinement

- **Given:** A graph G with a set of nodes V.
  - Assign an initial color $c^{(0)}(v)$ to each node v.
  - Iteratively refine node colors by

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

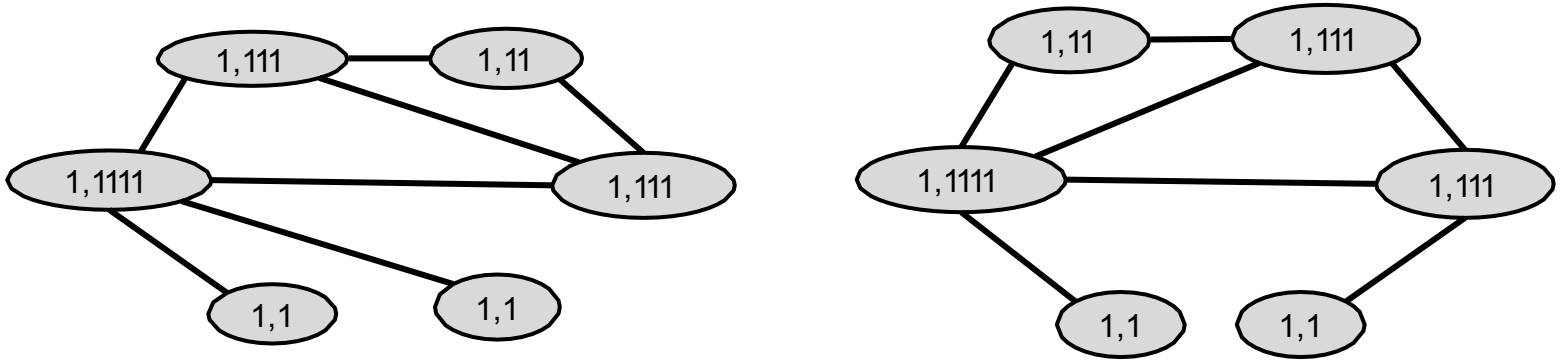
    where HASH maps different inputs to different colors.
  - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood

Color Refinement (1)

Example of color refinement given two graphs

- Assign initial colors

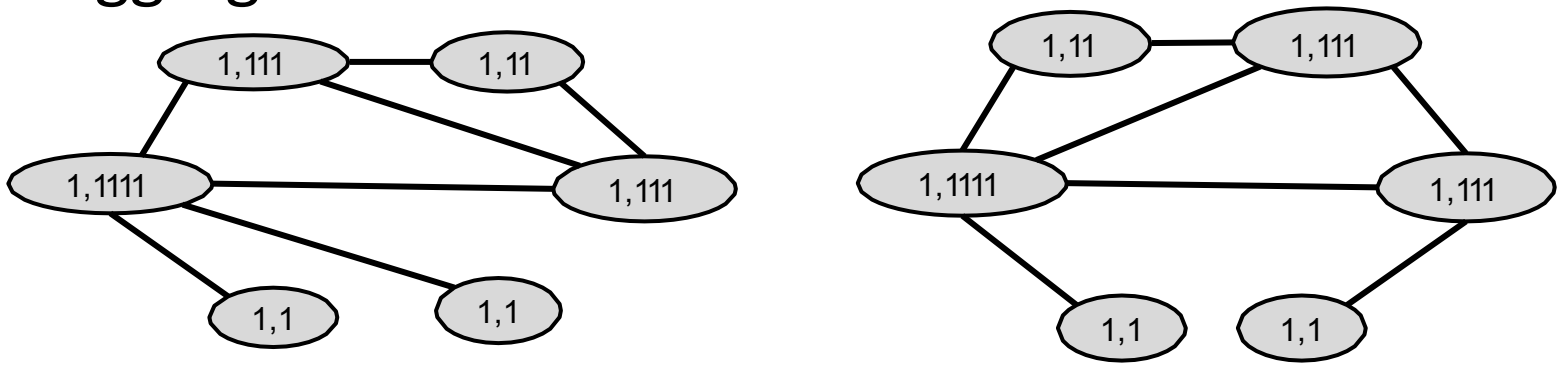
- Aggregate neighboring colors



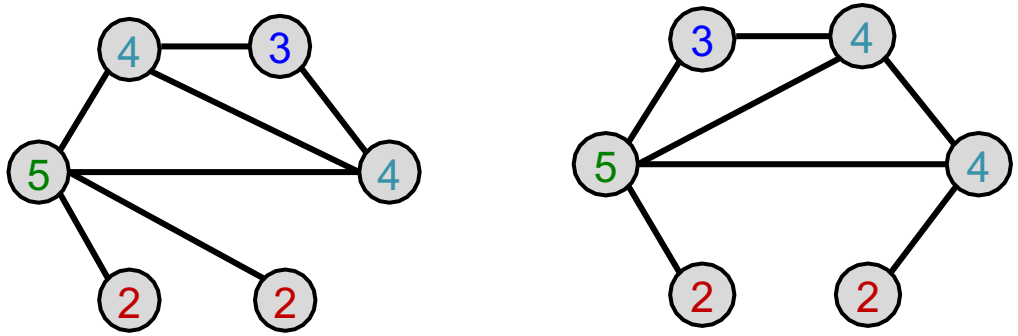
Color Refinement (2)

Example of color refinement given two graphs

- Aggregated colors



- Hash aggregated colors



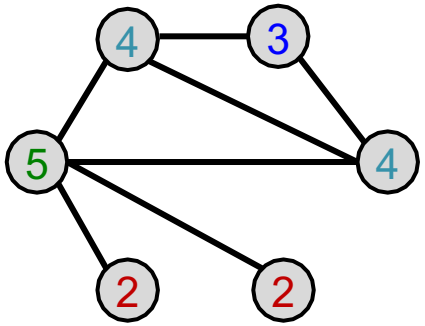
Hash table

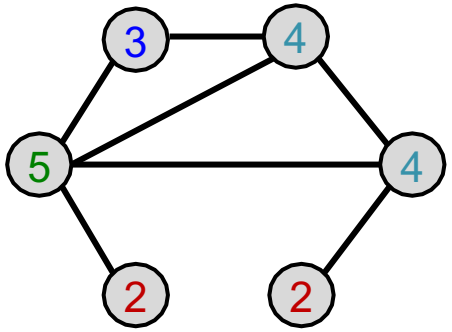
1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Color Refinement (3)

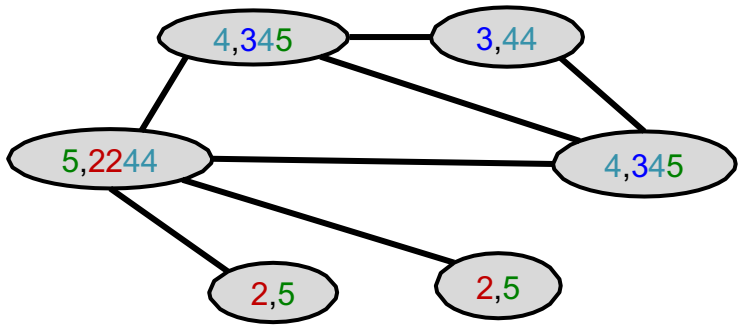
Example of color refinement given two graphs

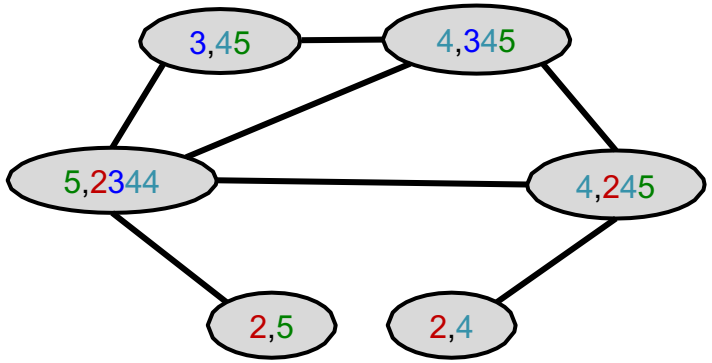
- Aggregated colors





- Hash aggregated colors

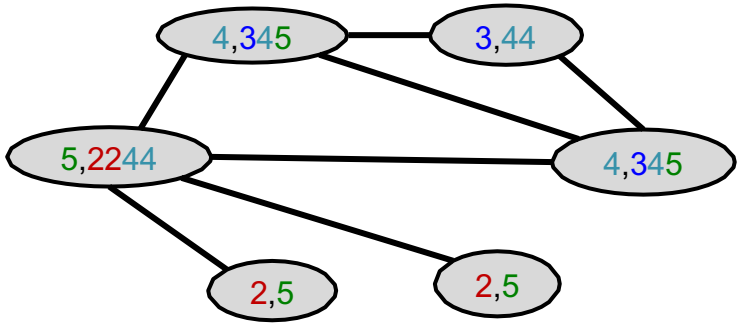


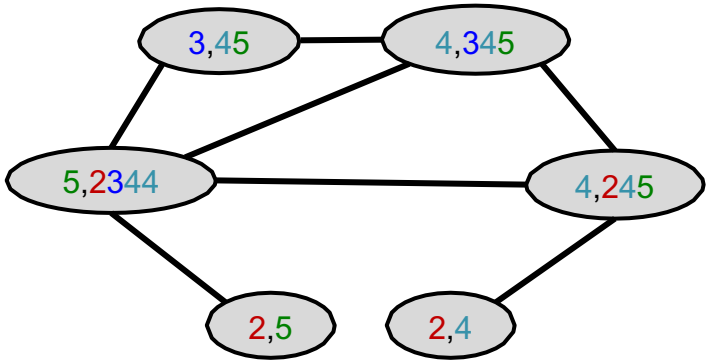


Color Refinement (4)

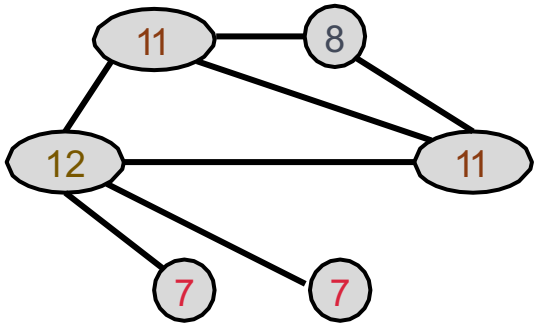
Example of color refinement given two graphs

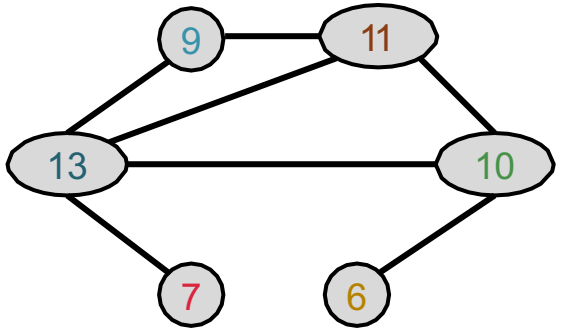
- Aggregated colors





- Hash aggregated colors



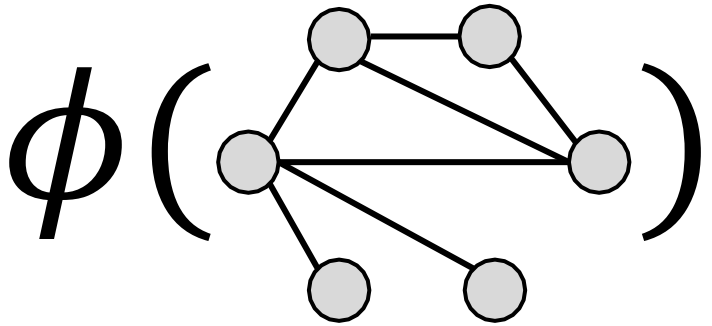


Hash table

2,4	-->	6
2,5	-->	7
3,44	-->	8
3,45	-->	9
4,245	-->	10
4,345	-->	11
5,2244	-->	12
5,2344	-->	13

Weisfeiler-Lehman Graph Kernel

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

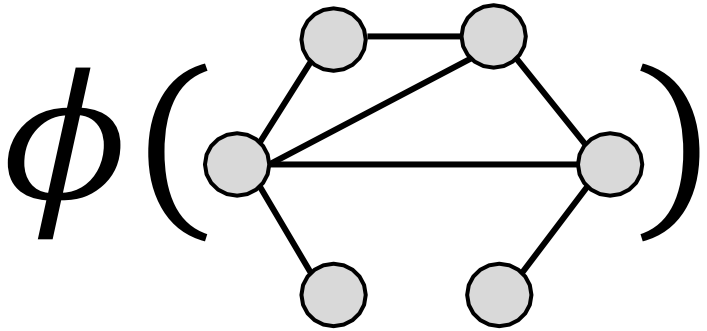


Colors

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

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

Counts

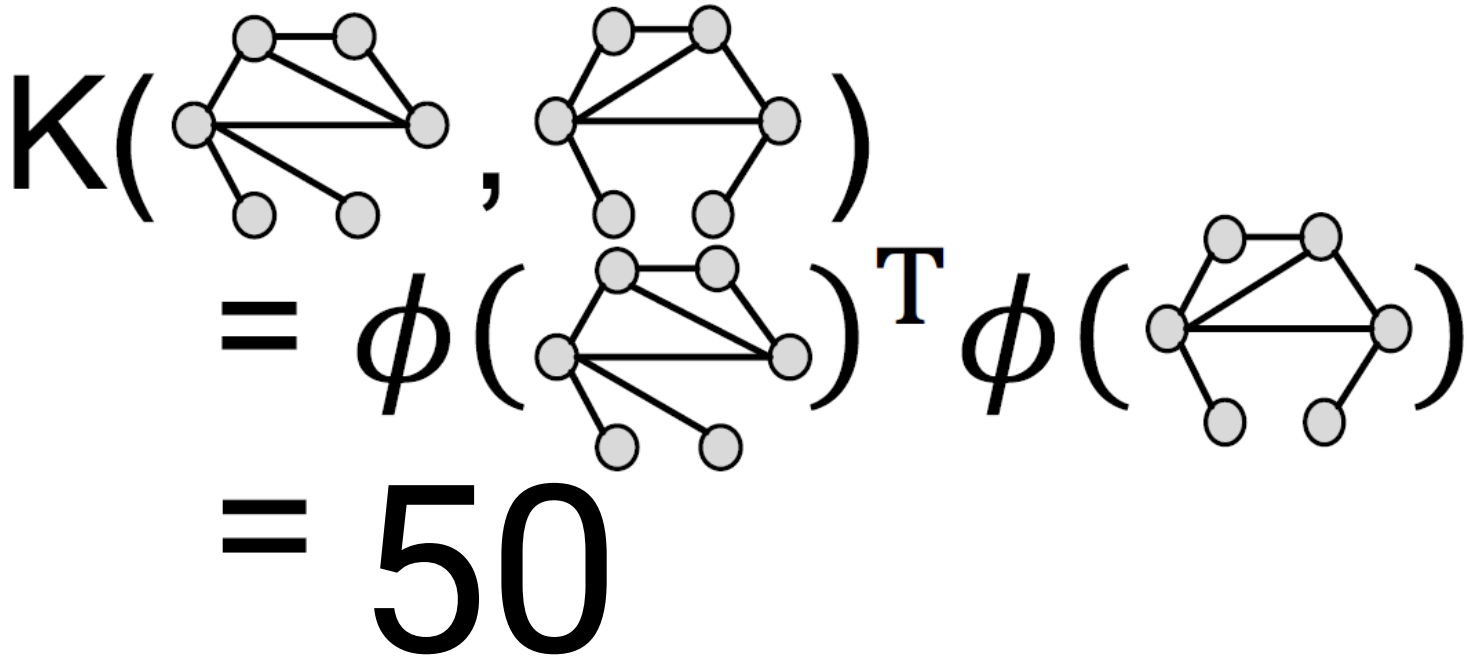


1,2,3,4,5,6,7,8,9,10,11,12,13

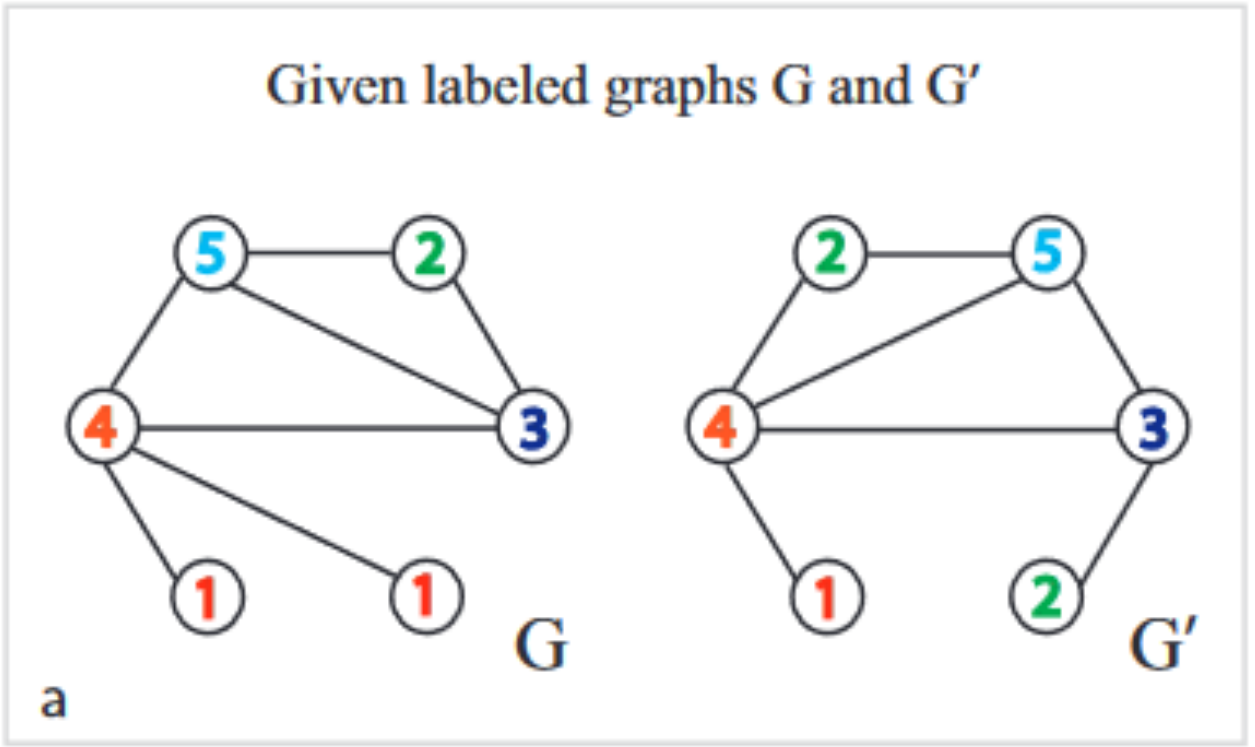
= [6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

Weisfeiler-Lehman Graph Kernel

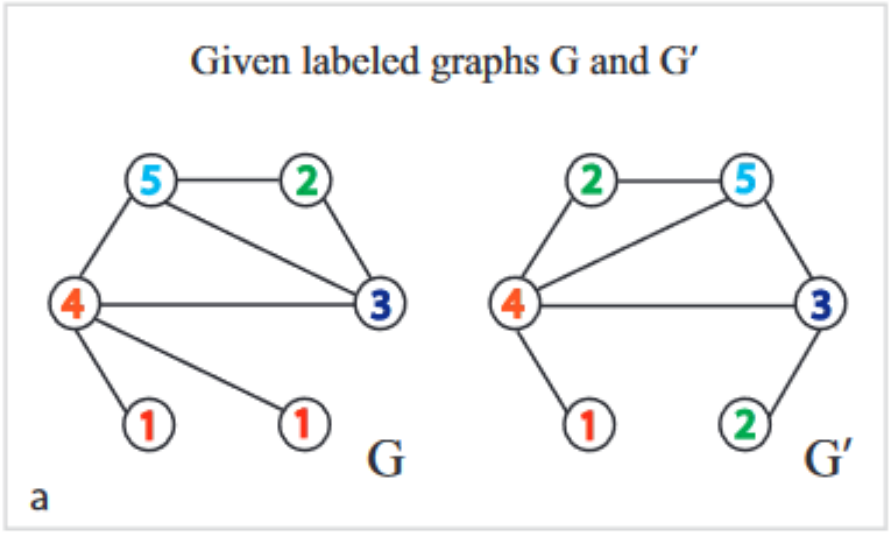
The WL kernel value is computed by the inner product of the color count vectors:

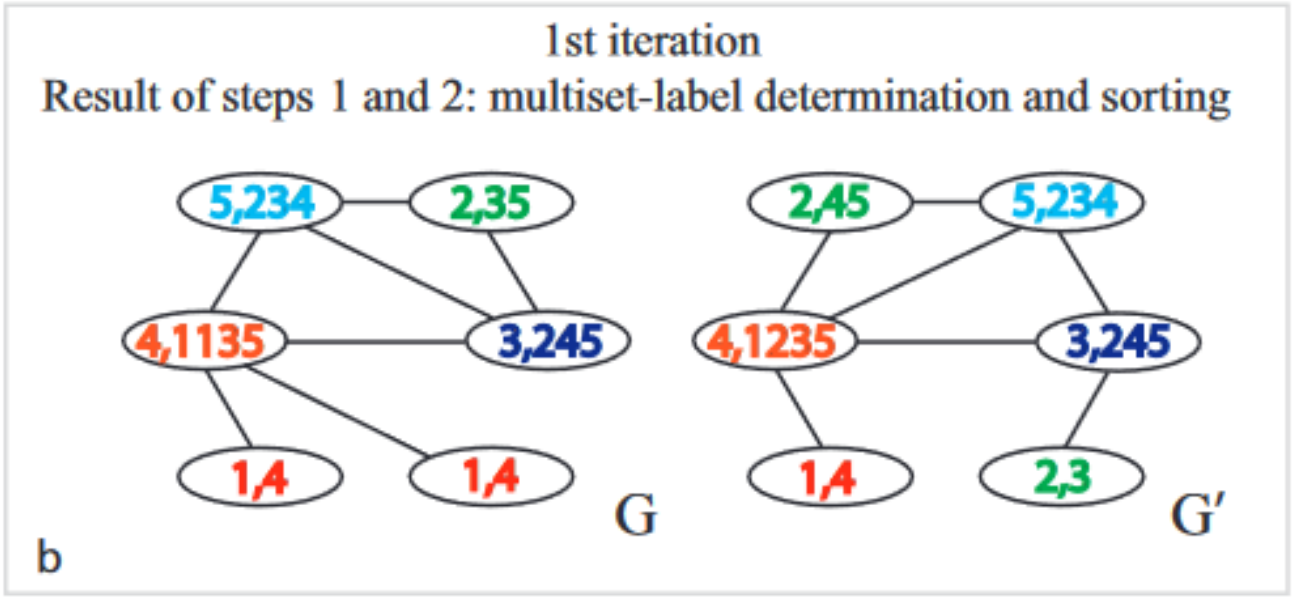


WL features on Labeled Graph (1)

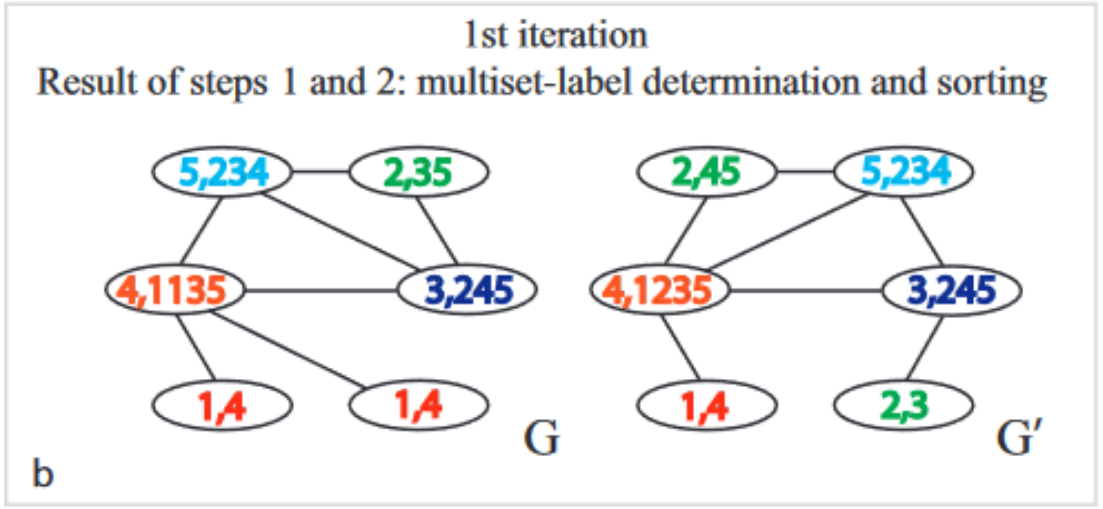


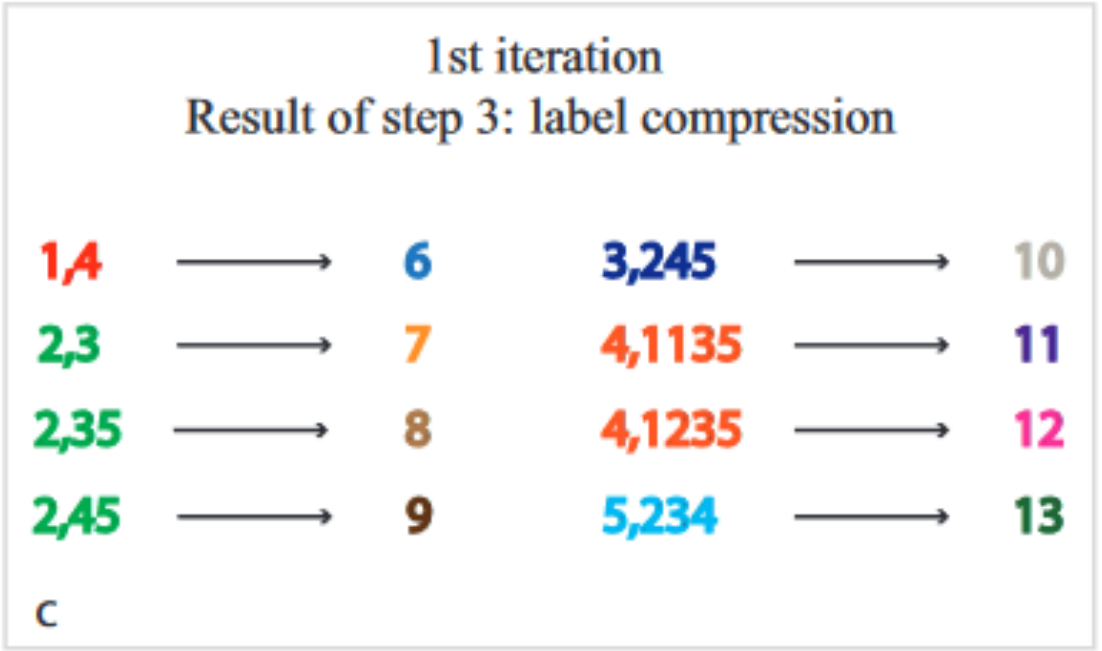
WL features on Labeled Graph (2)





WL features on Labeled Graph (3)



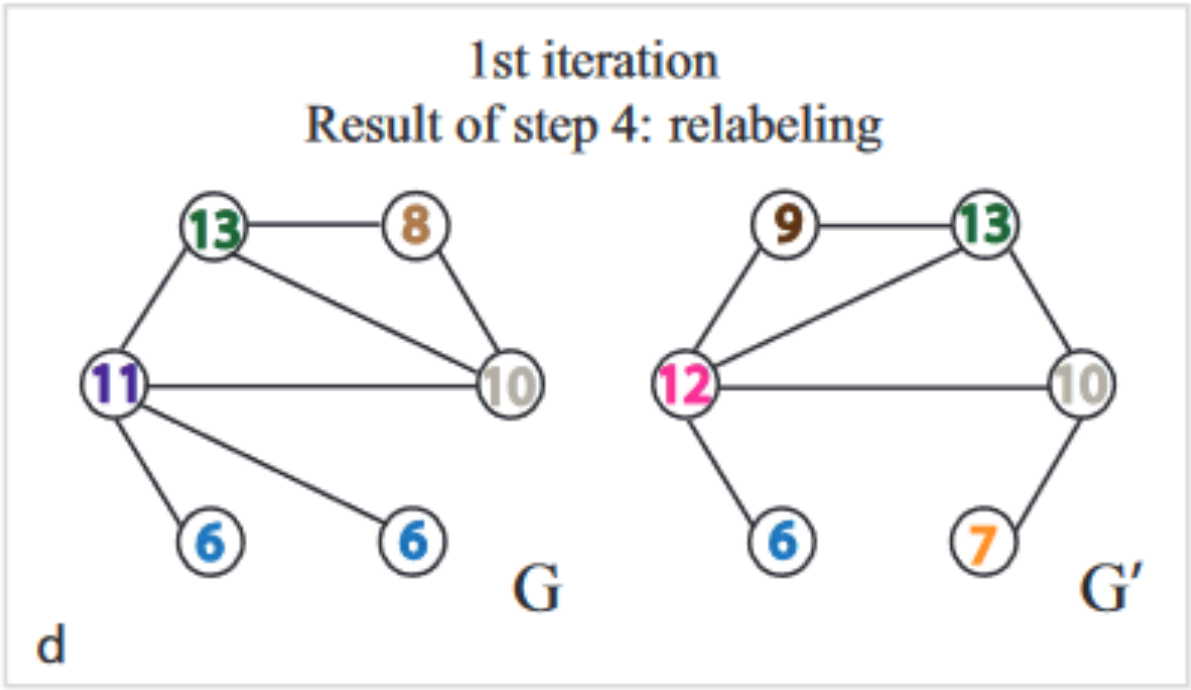


WL features on Labeled Graph (4)

1st iteration
Result of step 3: label compression

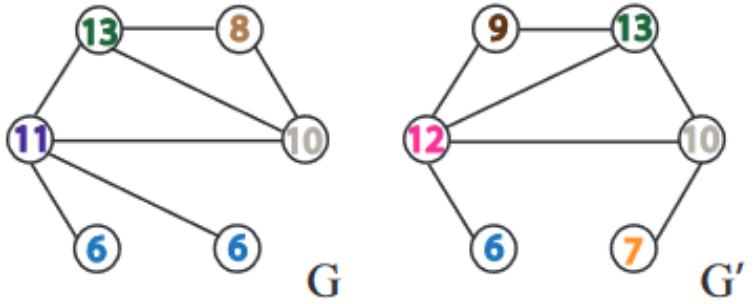
1,4	⟶	6	3,245	⟶	10
2,3	⟶	7	4,1135	⟶	11
2,35	⟶	8	4,1235	⟶	12
2,45	⟶	9	5,234	⟶	13

c



WL features on Labeled Graph (5)

1st iteration
Result of step 4: relabeling



d

End of the 1st iteration
Feature vector representations of G and G′

$$\varphi^{(1)}_{WLsubtree}(G) = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

$$\varphi^{(1)}_{WLsubtree}(G') = (1, 2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1)$$

Counts of original node labels

Counts of compressed node labels

$$k^{(1)}_{WLsubtree}(G,G') = \langle \varphi^{(1)}_{WLsubtree}(G), \varphi^{(1)}_{WLsubtree}(G') \rangle = 11.$$

e

Weisfeiler-Lehman Graph Kernel

- WL kernel is **computationally efficient**
  - The time complexity for color refinement at each step is linear in #(edges), since it involves aggregating neighboring colors.
- When computing a kernel value, only colors appeared in the two graphs need to be tracked.
  - Thus, #(colors) is at most the total number of nodes.
- Counting colors takes linear-time w.r.t. #(nodes).
- In total, time complexity is **linear in #(edges)**.

Graph-Level Features: Summary

Graphlet Kernel

- Graph is represented as Bag-of-graphlets
- **Computationally expensive**

Weisfeiler-Lehman Kernel

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

Learning Outcome

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