

# Graph Mining and Graph Representation Learning

Prof. O-Joun Lee

Dept. of Artificial Intelligence,  
The Catholic University of Korea  
*ojlee@catholic.ac.kr*



네트워크 과학 연구실  
NETWORK SCIENCE LAB



가톨릭대학교  
THE CATHOLIC UNIVERSITY OF KOREA



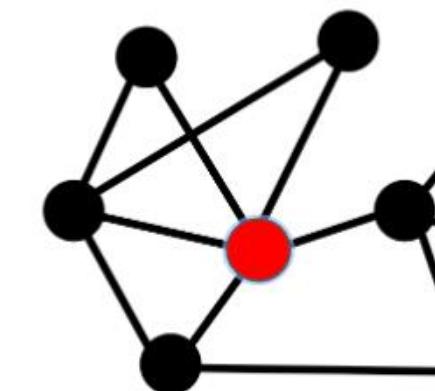
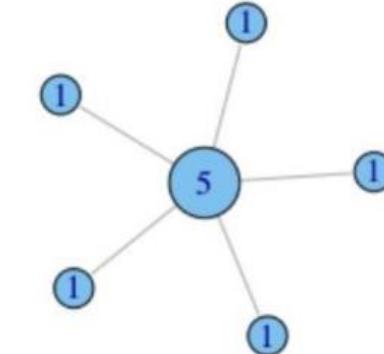
# Contents



- Centrality Measurements:
  - Betweenness, Closeness, PageRank Centrality
- Graph Measurements:
  - Clustering Coefficients, Conductance, Modularity
- Additional Metrics:
  - Jaccard's Coefficient, Katz Index
- Graph Kernels:
  - Shortest Path Kernel, Weisfeiler-Lehman Kernel
- Graph Representation Learning (GRL):
  - From traditional ML for graph to GRL
  - WalkLet, Node2Vec, LINE, HOPE, Struc2Vec, Role2Vec, metapath2vec
  - Dynnode2vec, Continuous-Time Dynamic Network Embeddings

- Graphs efficiently model relationships, perfect for addressing questions like "What's the lowest-cost path from A to B?"
- We need a data structure that represents graphs
- Determining the "Best" Data Structure can depend on:
  - Properties of the graph (dense vs. sparse)
  - Common queries
    - For example: "is  $(u, v)$  an edge?" vs "what are the neighbors of node  $u$ ?"
- There are two standard graph representations:
  - Adjacency Matrix
  - Adjacency List

- **Node degree:** the degree  $k_v$  of node  $v$  is the number of edges (neighboring nodes) the node has
  - In-degree: Number of incoming edges to the node
  - Out-degree: Number of outgoing edges from the node
  - Total degree: Sum of in-degree and out-degree
- Knowing the network structure, we can calculate various useful quantities or measures that capture important features of network topology
- Centrality measures represent the most important nodes in graphs:
  - The most influential person in a social network
  - The most critical nodes in an infrastructure
  - The highest spreaders of disease



## ➤ Betweenness Centrality:

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

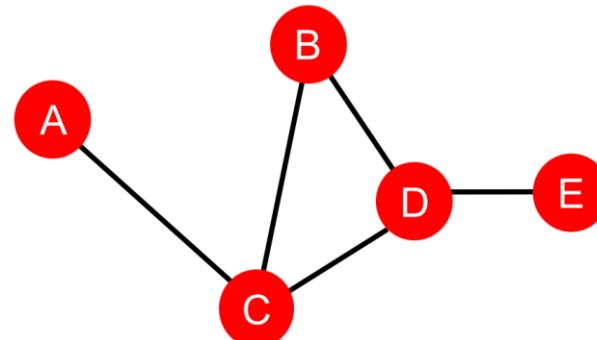
$$c_v = \sum_{s \neq v \neq t} \frac{\#(\text{shortest paths between } s \text{ and } t \text{ that contain } v)}{\#(\text{shortest paths between } s \text{ and } t)}$$

- Usually normalized by:

$$\bar{B}(v_i) = B(v_i) / [(n-1)(n-2)/2]$$

No. of nodes in the graph

- For example:



- $C_A = C_B = C_E = 0$
- $C_C = 3$  (A-C-B, A-C-D, A-C-D-E)
- $C_D = 3$  (A-C-D-E, B-D-E, C-D-E)

- Nodes with high betweenness centrality have influence in the network by virtue of their control over information passing between others
  - They get to see the messages as they pass through
  - They could get paid for passing the message along
- Thus, they get a lot of power: their removal would disrupt communication

## ➤ Closeness Centrality:

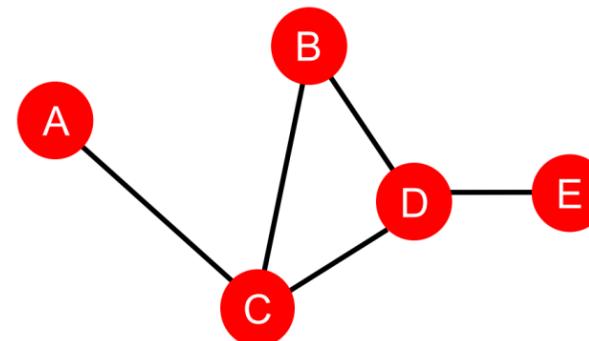
- A node is important if it has **small shortest path lengths** to all other nodes

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

- Usually normalized by:

$$\bar{C}(v_i) = \frac{n - 1}{\sum_{j=1}^n |d(v_i, v_j)|}$$

- For example:

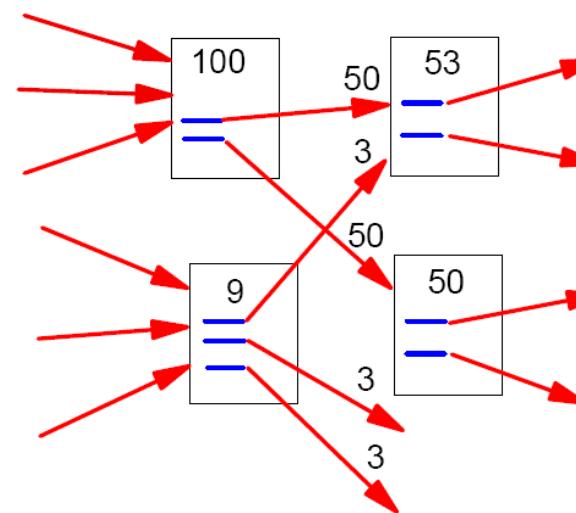


- $C_A = 1/(2 + 1 + 2 + 3) = 1/8$   
(A-C-B, A-C, A-C-D, A-C-D-E)
- $C_D = 1/(2 + 1 + 1 + 1) = 1/5$   
(D-C-A, D-B, D-C, D-E)

- PageRank is a numeric value that represents how important a page is on the web
- Webpage importance
  - One page links to another page = A vote for the other page A link from page A to page B is a vote on A to B
  - If page A is more important itself, then the vote of A to B should carry more weight
  - More votes = More important the page must be

## ➤ Importance Computation

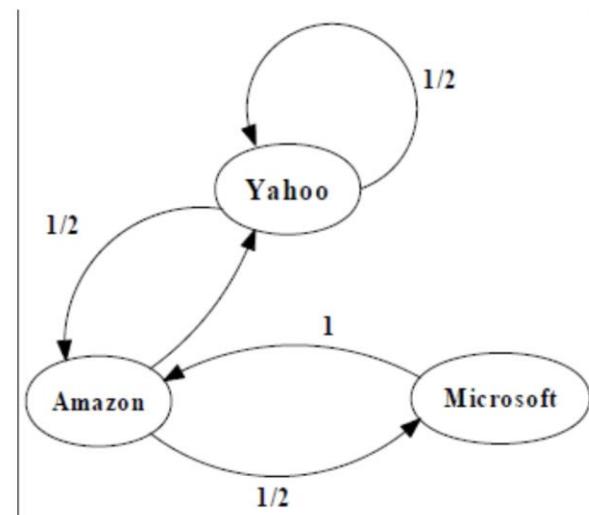
- The importance of a page is distributed to pages that it points to
- The importance of a page is the aggregation of the importance shares of the pages that points to it
  - If a page has 5 outlinks, the importance of the page is divided into 5 and each link receives one fifth share of the importance



$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

- $u$ : a web page
- $B_u$ : the set of  $u$ 's backlinks
- $N_v$ : the number of forward links of page  $v$
- $c$ : the normalization factor to make  $\|R\|_{L1} = 1 (\|R\|_{L1} = |R_1 + \dots + R_n|)$

➤ For example:



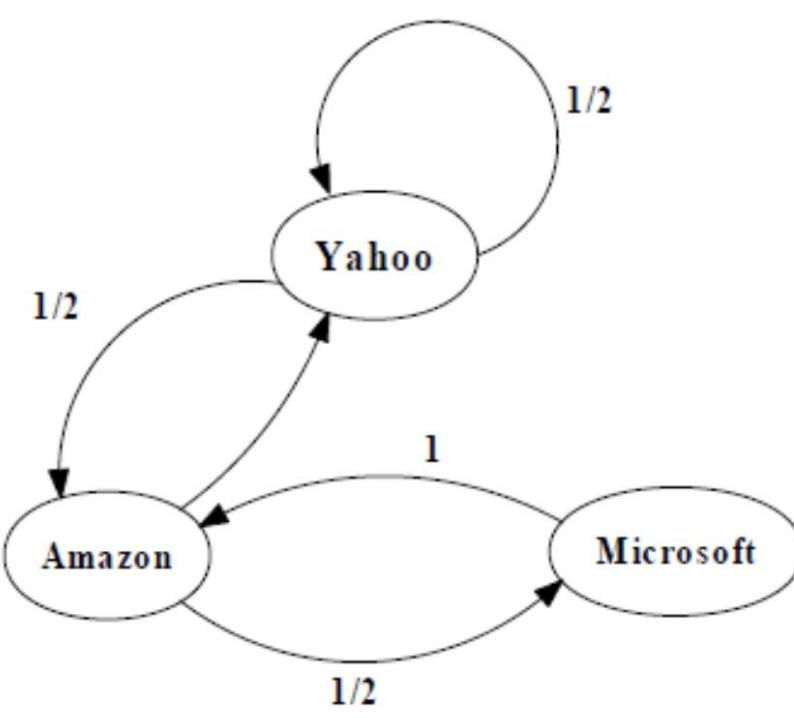
$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

# A Simple Version of PageRank

11



$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}.$$

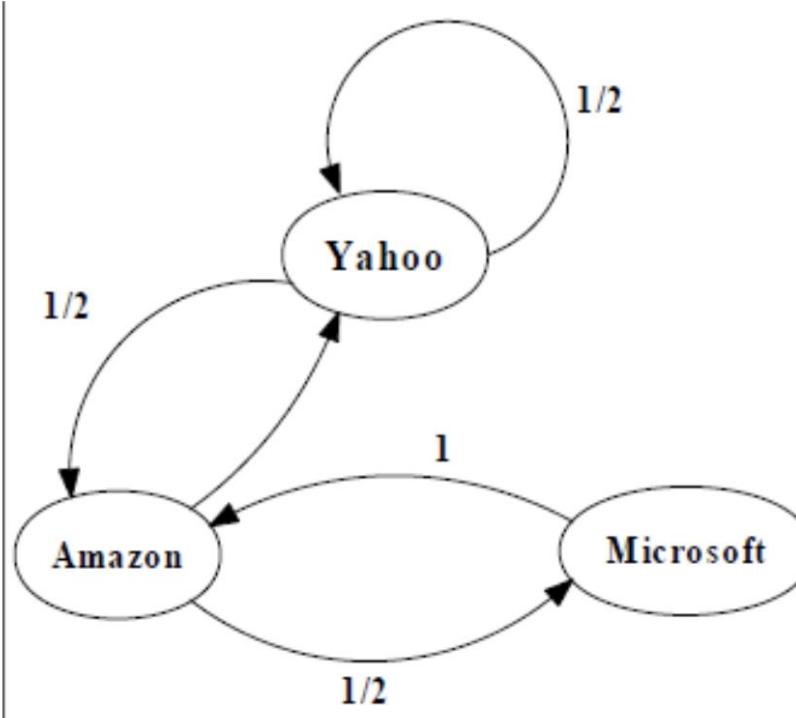
$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 5/12 \\ 1/3 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix}$$

PageRank Calculation: second iteration

# A Simple Version of PageRank

12



$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}.$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

$$\begin{bmatrix} 3/8 \\ 11/24 \\ 1/6 \end{bmatrix} \begin{bmatrix} 5/12 \\ 17/48 \\ 11/48 \end{bmatrix} \dots \begin{bmatrix} 2/5 \\ 2/5 \\ 1/5 \end{bmatrix}$$

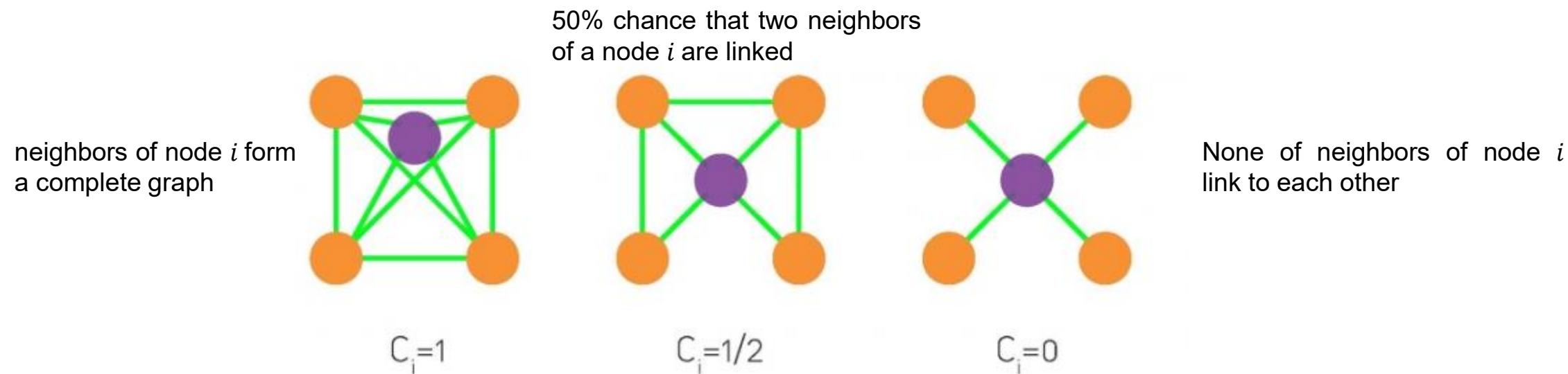
Convergence after some iterations

- The clustering coefficient measures how connected a node's neighbours are to one another
- The range is from 0 to 1 (from non-neighbour are connected to each other to all neighbours are fully connected)
- There are three types of clustering coefficient:
  - Local clustering coefficient
  - Average clustering coefficient
  - Global clustering coefficient

- How close its neighbours are to being a clique (complete graph)
- For a node  $i$  with degree  $d_i$  and  $L_i$  represents the number of edges between neighbours of node  $i$

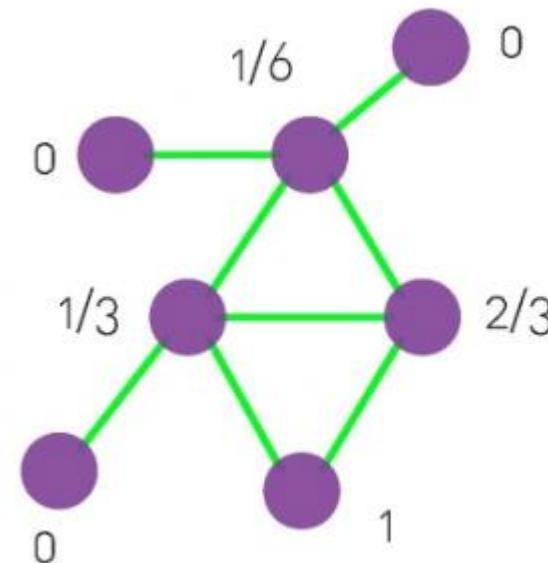
The local clustering coefficient  $C_i$  for a node  $i$  is defined as:

$$C_i = \frac{2L_i}{d_i(d_i - 1)}$$



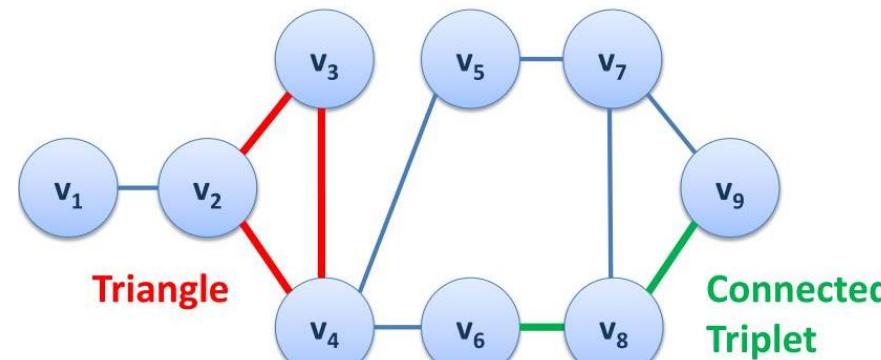
- The degree of clustering of a whole network is captured by the average clustering coefficient, namely  $\langle C \rangle$ , representing the average of all the local clustering coefficient  $C_i$  over all nodes  $i = 1, \dots, N$ .

$$\langle C \rangle = \frac{1}{N} \sum_{i=0}^N C_i$$



$$\langle C \rangle = \frac{1}{7} * \left( 0 + \frac{1}{6} + \frac{1}{3} + \frac{2}{3} + 1 + 0 + 0 \right) = 0.333$$

- The global clustering coefficient is based on triplets of nodes
- A triplet consists of three connected nodes. A triangle therefore includes three closed triplets, one centered on each of the nodes

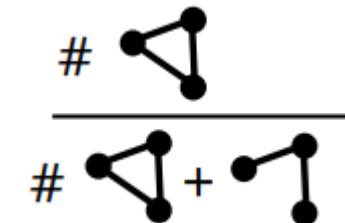


- The global clustering coefficient is the number of closed triplets over the total number of triplets (both open and closed)

Closed triplets:  $(v_2, v_3, v_4), (v_7, v_8, v_9)$

Connected triplets:  $(v_6, v_8, v_9), \dots$

$$C(G) = \frac{\# \text{ of closed triplets}}{\# \text{ of connected triplets}}$$

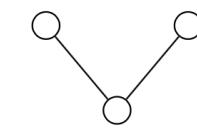


- The clustering coefficient can be extended to higher order structures with  $k$ -cliques

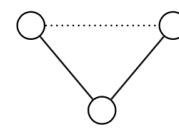
1. Start with  
an  $\ell$ -clique



2. Find an adjacent edge  
to form an  $\ell$ -wedge



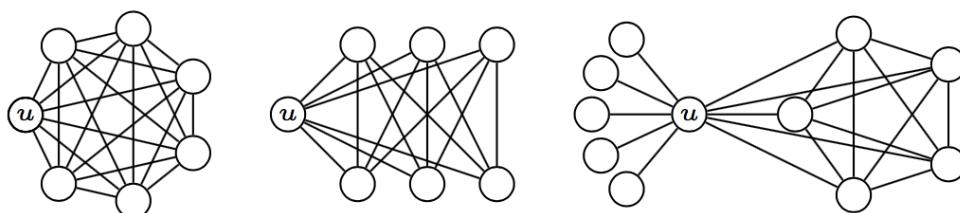
3. Check for an  
 $(\ell + 1)$ -clique



$C_2$

$C_3$

$C_4$



$C_2(u)$

1

$$\frac{d}{2(d-1)} \approx \frac{1}{2}$$

$$\frac{d-2}{4d-4} \approx \frac{1}{4}$$

$C_3(u)$

1

$$\frac{d-4}{2d-4} \approx \frac{1}{2}$$

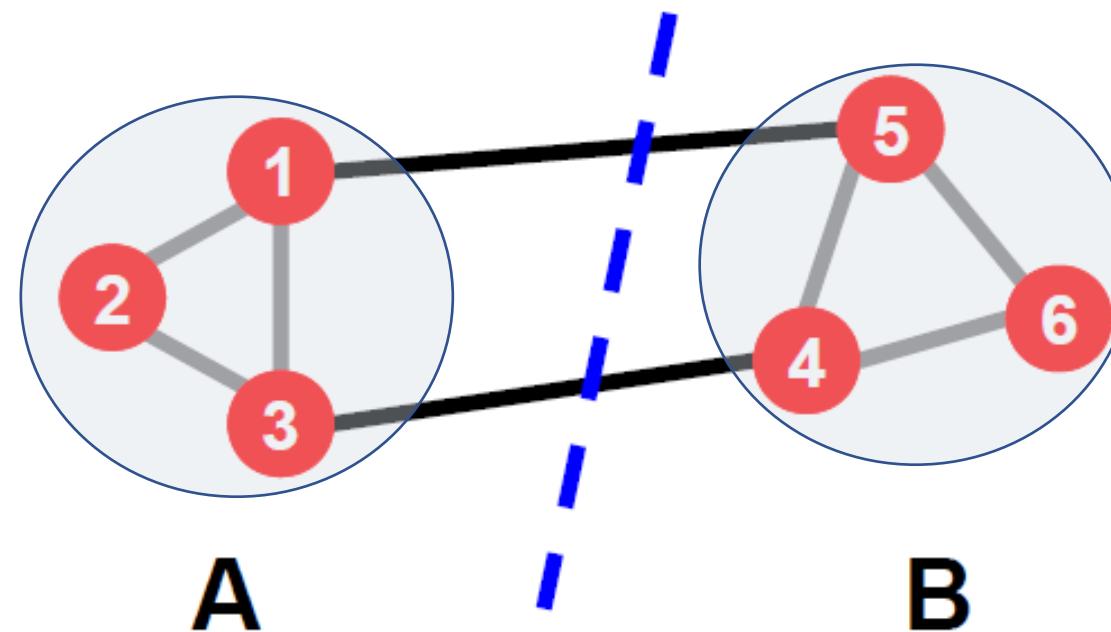
$C_4(u)$

1

$$\frac{d-6}{2d-6} \approx \frac{1}{2}$$

➤ Basic principle for graph partitioning:

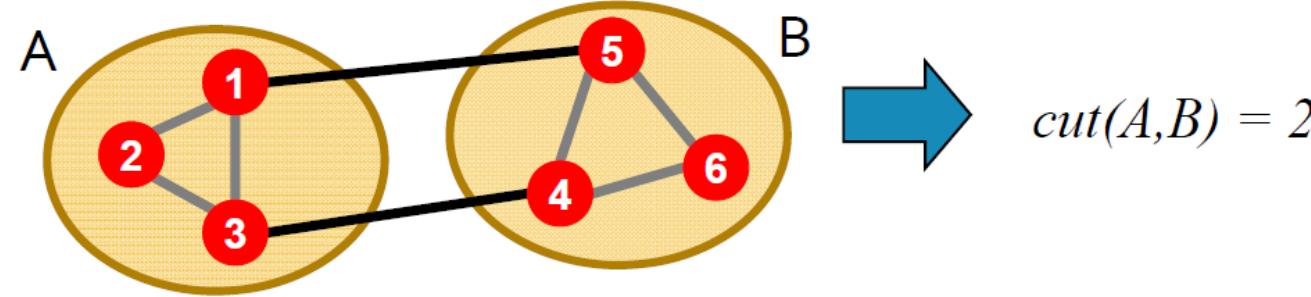
- Minimize the number of between-group connections
- Maximize the number of within-group connections



Graph partitioning : A & B

# Mathematical expression: Cut (A,B)

➤ For considering between-group:



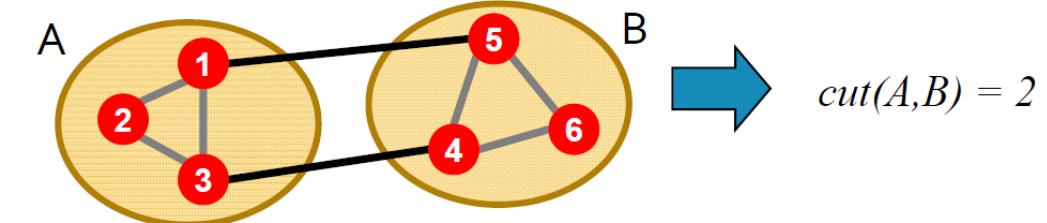
Cut: Set of edges with only one vertex in a group:  
$$cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$$

- For considering within-group
- $\text{Assoc}(A, V)$ : the total connection from nodes in A to all nodes in the graph

$$\text{assoc}(A, V) = \sum_{u \in A, t \in V} w(u, t)$$

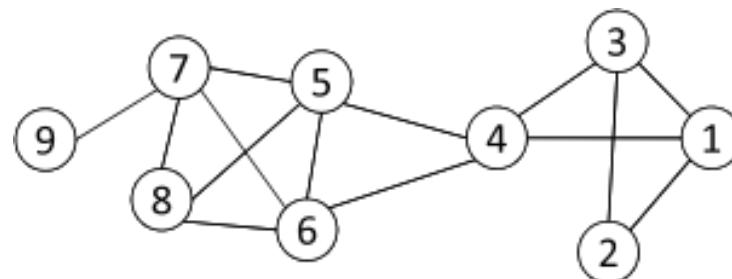
$$Ncut(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)}$$

$$\text{Conductance}(A, B) = \frac{\text{cut}(A, B)}{\min(\text{assoc}(A, V), \text{assoc}(B, V))}$$



- Modularity measures the strength of a community partition by considering the degree distribution
- Given a graph with  $m$  edges, the expected number of edges between two nodes with degree  $d_i$  and  $d_j$  is

$$d_i d_j / 2m$$



The expected number of edges between nodes 1 and 2 is  $3 \cdot 2 / (2 \cdot 14) = 3/14$

- Modularity:

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - \gamma \frac{d_i d_j}{2m}) \delta(C_i, C_j)$$

Adjacency matrix of graph

probability a random edge would go between  $i$  and  $j$

$\delta(C_i, C_j)$ : 1 if  $i$  and  $j$  are in the same community else 0

- We have modularity matrix:

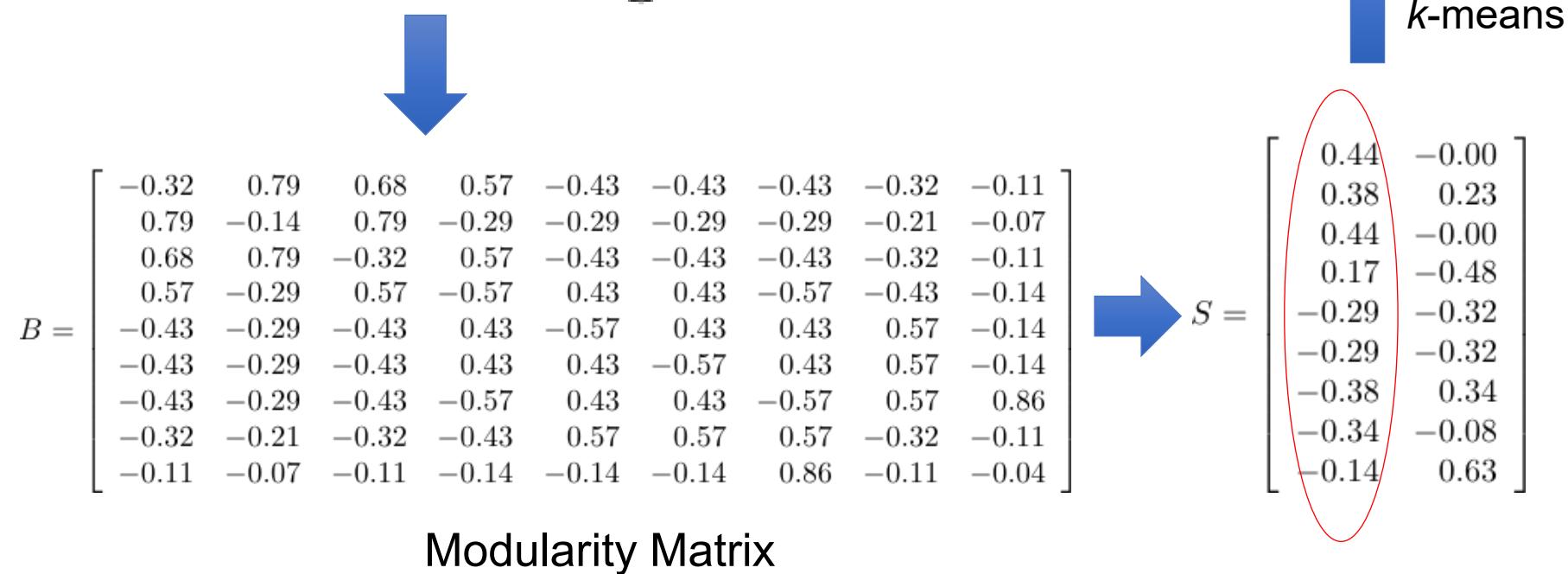
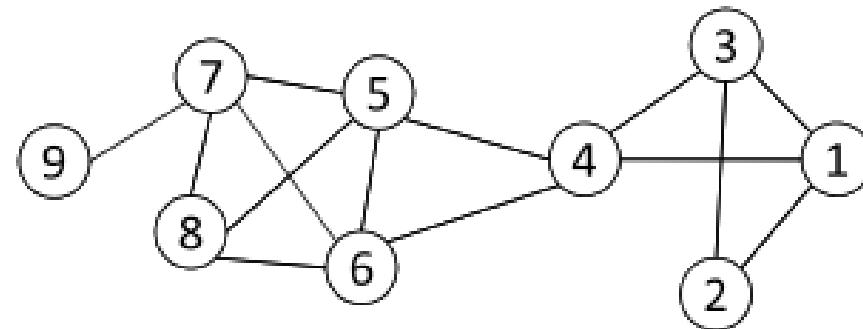
$$B = A - \mathbf{d}\mathbf{d}^T/2m \quad (B_{ij} = A_{ij} - d_i d_j / 2m)$$

Where: d is node degree

- Similar to spectral clustering, modularity maximization can be reformulated as

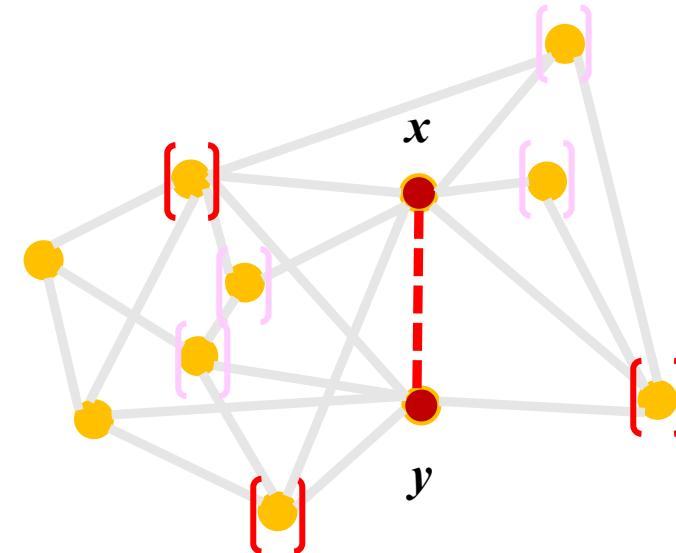
$$\max Q = \frac{1}{2m} \text{Tr}(S^T B S) \quad \text{s.t. } S^T S = I_k$$

- Optimal solution: top eigenvectors of the modularity matrix
- Apply k-means to S as a post-processing step to obtain community partition



➤ Jaccard coefficient:

- How likely a neighbour of  $x$  is also a neighbour of  $y$
- Same as common neighbours, adjusted for degree



$$JC = \frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|} = \frac{CN}{d_x + d_y - CN}$$

## ➤ Katz index:

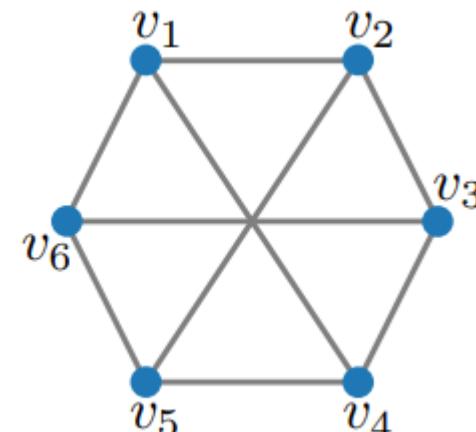
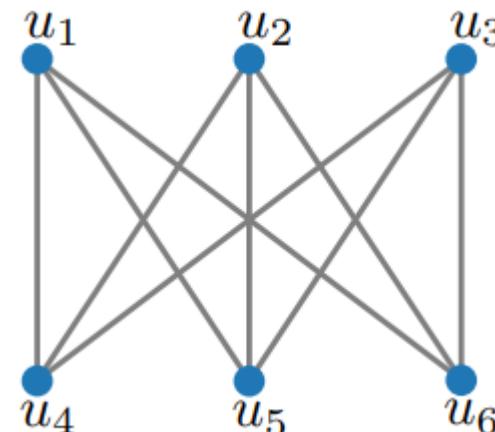
$$score(x, y) = \sum_{l=1}^{\infty} \beta^l |paths_{xy}^{(l)}| = \beta A_{xy} + \beta^2 A_{xy}^2 + \dots$$

Element  $(x,y)$  in the adjacency matrix

- Sum over ALL paths of length  $\ell$
- $0 < \beta < 1$  is a parameter of the predictor, exponentially damped to count short paths more heavily
- Small damped parameter = predictions much like common neighbours

➤ Graph isomorphism:

- Find a mapping  $f$  of the nodes of  $G_1$  to the nodes of  $G_2$  such that  $G_1$  and  $G_2$  are identical
- i.e.  $(u_i, u_j)$  is an edge of  $G_1$  iff  $(f(u_i), f(u_j))$  is an edge of  $G_2$ . Then  $f$  is an isomorphism, and  $G_1$  and  $G_2$  are called isomorphic
- No polynomial-time algorithm is known for graph isomorphism
- Neither is it known to be NP-complete



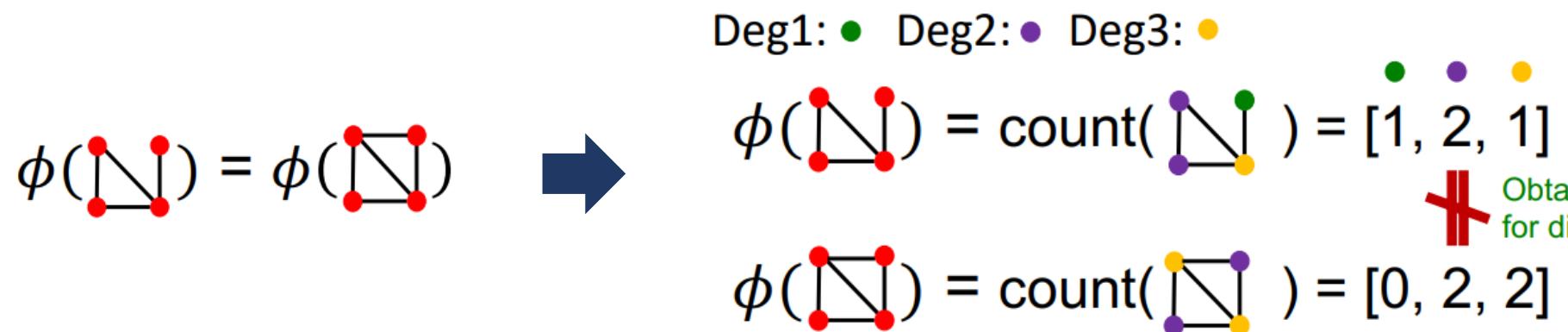
$V^{G_1} \quad V^{G_2}$

$v_1 \rightarrow u_1$   
 $v_2 \rightarrow u_4$   
 $v_3 \rightarrow u_2$   
 $v_4 \rightarrow u_5$   
 $v_5 \rightarrow u_3$   
 $v_6 \rightarrow u_6$

$\psi :$

- Kernel is a type of measures of similarity
- Mapping two objects  $x$  and  $x'$  via mapping  $\phi$  into feature space  $H$
- Measure their similarity in  $H$  as  $\langle \phi(x), \phi(x') \rangle$ .
- **Kernel Trick:** Compute inner product in  $H$  as kernel in input space

$$k(x, x') = \langle \phi(x), \phi(x') \rangle.$$



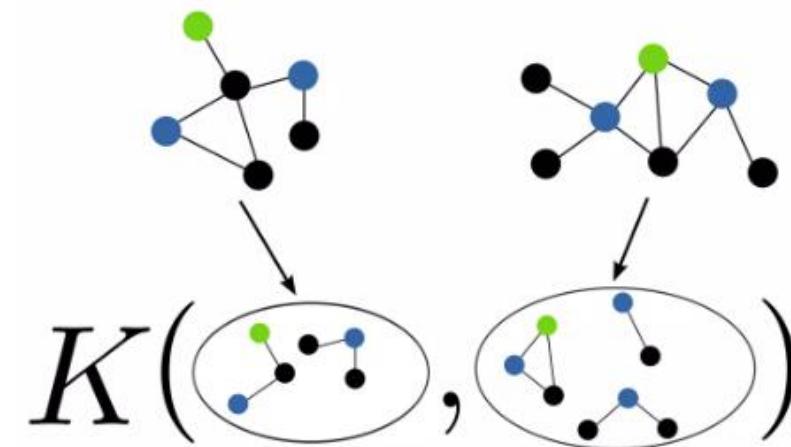
- Instance of R-convolution kernels by Haussler (1999):
  - R-convolution kernels **compare decompositions of two structured objects**

$$k_{convolution}(x, x') = \sum_{(x_d, x) \in R} \sum_{(x'_d, x') \in R} k_{parts}(x_d, x'_d)$$

- Decompose graphs into their substructures and add up the pairwise similarities between these substructures
- Concept:
  - Kernel function to measure the similarity of pairs of graphs by computing an inner product on graphs
  - Compare substructures of graphs that are computable in polynomial time

- Graph kernels based on **bags of patterns**:

- Extraction of a set of patterns from graphs
- Comparison between patterns
- Comparison between bags of patterns



- Graph kernels are one of the most recent approaches to graph comparison
- Interestingly, graph kernels employ concepts from all three traditional branches of graph comparison:
  - Measure similarity in terms of **isomorphic substructures of graphs**
  - **Allow for inexact matching of nodes, edges, and labels**
  - Treat graphs as **vectors** in a Hilbert space of graph features

## Floyd-transformation:

- Given an input graph G, outputs a **shortest-path graph S**
  - S contains the **same set of nodes** as the input graph G
  - There exists an edge between all nodes in S which are **connected by a path** in G
  - Every edge in S between two nodes is **labelled by the shortest distance** between these two nodes
  - This transformation can be done in  $O(n^3)$  time

- Compute all-pairs-shortest-paths for  $G$  and  $G'$  via Floyd-Warshall
- Define a kernel by comparing all pairs of shortest path lengths from  $G$  and  $G'$ :

$$k(G, G') = \sum_{v_i, v_j \in G} \sum_{v'_k, v'_l \in G'} k_{length}(d(v_i, v_j), d(v'_k, v'_l))$$

where  $d(v_i, v_j)$  is the length of the shortest path between node  $v_i$  and  $v_j$

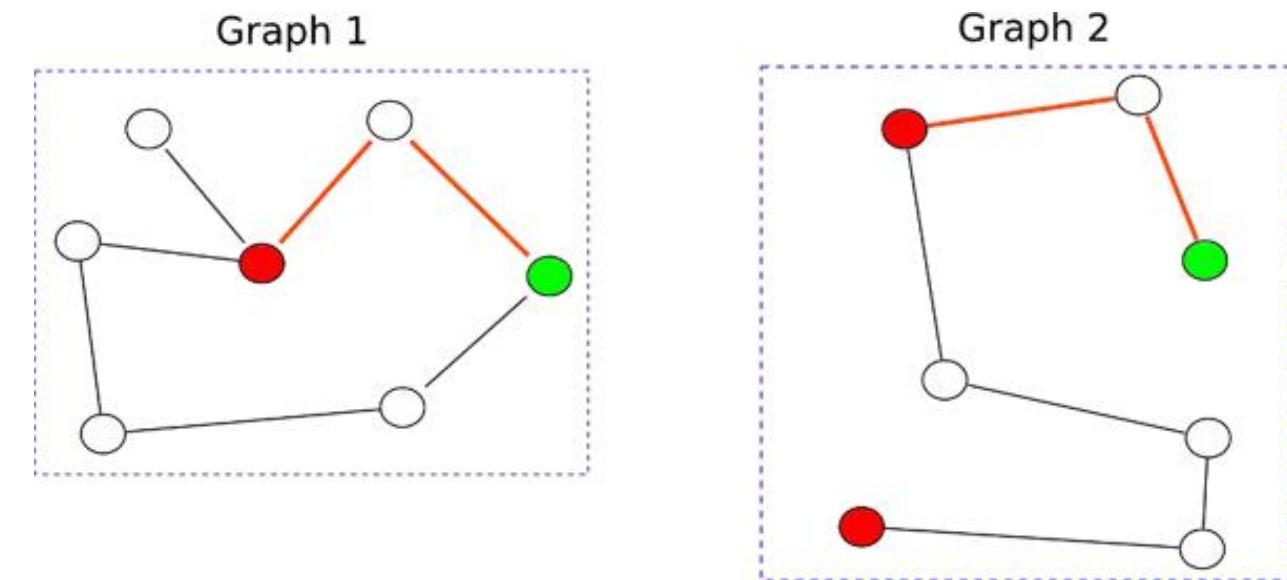
- $k_{length}$  is a kernel that compares the lengths of two shortest paths:

- a linear kernel

$$k(d(v_i, v_j), d(v'_k, v'_l)) = d(v_i, v_j) * d(v'_k, v'_l)$$

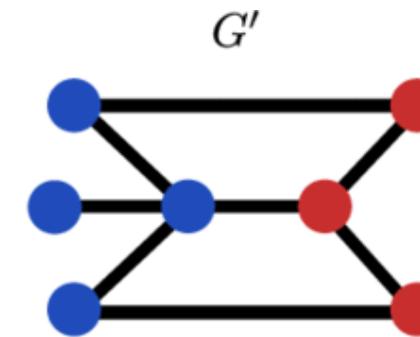
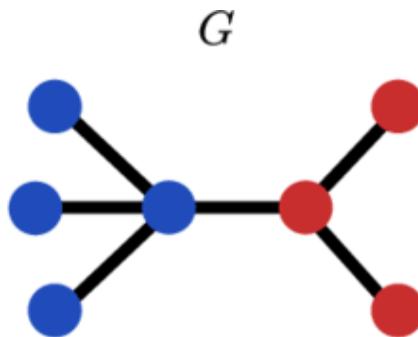
- a delta kernel

$$k(d(v_i, v_j), d(v'_k, v'_l)) = \begin{cases} 1, & \text{if } d(v_i, v_j) = d(v'_k, v'_l) \\ 0, & \text{otherwise} \end{cases}$$



## Shortest-path graph kernel:

$$k(G, G') = \sum_{v_i, v_j \in G} \sum_{v'_k, v'_l \in G'} k_{length}(d(v_i, v_j), d(v'_k, v'_l))$$



$$\phi_{\text{path}}(G) = [ 3 \ 2 \ 1 \ 3 \ 1 \ 5 \ 0 \ 0 \ 6 ]$$

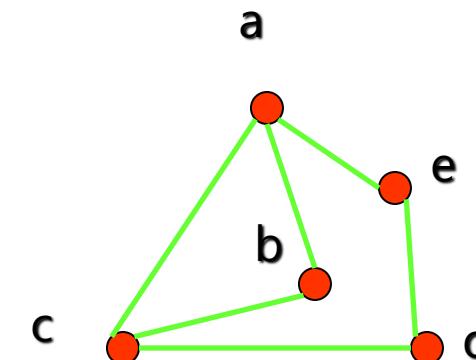
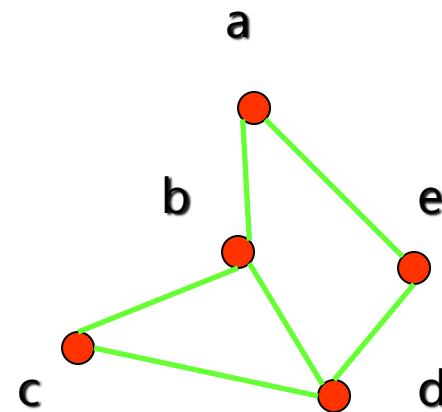
$$\phi_{\text{path}}(G') = [ 3 \ 2 \ 3 \ 3 \ 1 \ 5 \ 0 \ 0 \ 4 ]$$

**Definition:**

- The simple graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  are isomorphic if there is a bijection (a one-to-one and onto function)  $f$  from  $V_1$  to  $V_2$  with the property that  $a$  and  $b$  are adjacent in  $G_1$  if and only if  $f(a)$  and  $f(b)$  are adjacent in  $G_2$ , for all  $a$  and  $b$  in  $V_1$
- Such a function  $f$  is called an isomorphism
- In other words,  $G_1$  and  $G_2$  are isomorphic if their nodes can be ordered in such a way that the adjacency matrices  $M(G_1)$  and  $M(G_2)$  are identical

- For this purpose, we can check invariants, that is, properties that two isomorphic simple graphs must both have
- For example, they must have
  - The same number of nodes,
  - The same number of edges,
  - And the same degrees of nodes
- Note that two graphs that differ in any of these invariants are not isomorphic, but two graphs that match in all of them are not necessarily isomorphic

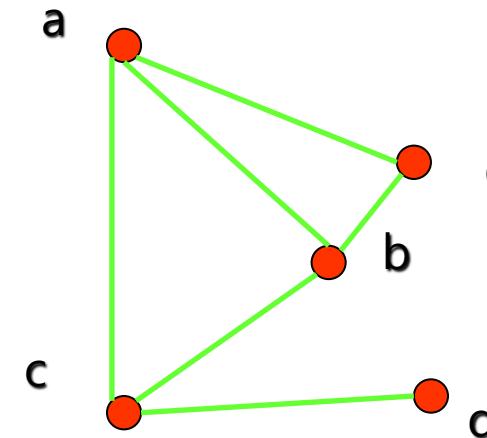
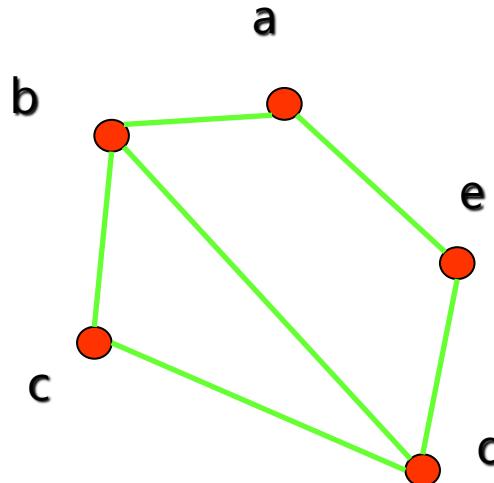
- Are the following two graphs isomorphic?



- **Solution:** Yes, they are isomorphic, because they can be arranged to look identical
- You can see this if in the right graph you move vertex b to the left of the edge {a, c}. Then the isomorphism  $f$  from the left to the right graph is

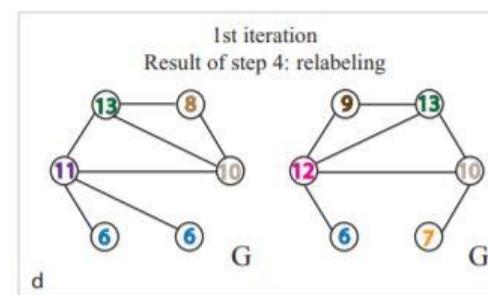
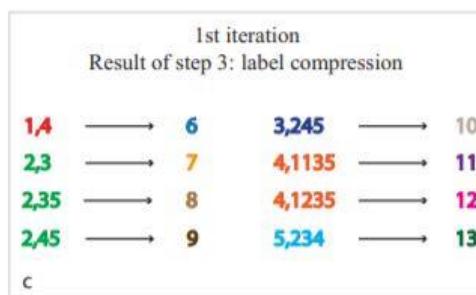
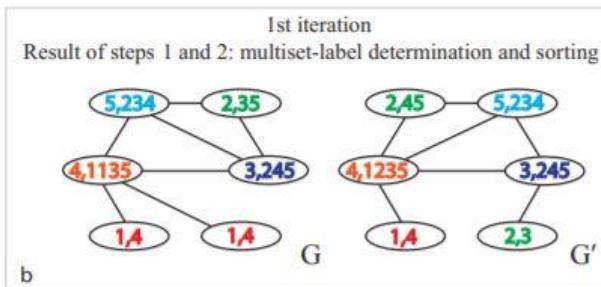
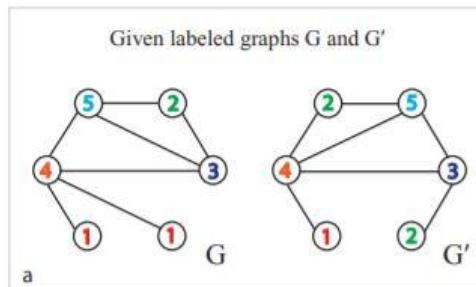
$$f(a) = e, f(b) = a, f(c) = b, f(d) = c, f(e) = d$$

- Are the following two graphs isomorphic?



- **Solution:** No, they are not isomorphic, because they differ in the degrees of their nodes
- Node d in right graph is of degree one, but there is no such node in the left graph

## ➤ Weisfeiler-Lehman Isomorphism Testing:



End of the 1st iteration  
Feature vector representations of  $G$  and  $G'$

$$\phi_{WL\text{subtree}}^{(1)}(G) = (2, 1, 1, 1, 1, 2, 0, 1, 0, 1, 1, 0, 1)$$

$$\phi_{WL\text{subtree}}^{(1)}(G') = (\underbrace{1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1}_{\text{Counts of compressed node labels}}, \underbrace{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}_{\text{Counts of original node labels}})$$

$$k_{WL\text{subtree}}^{(1)}(G, G') = \langle \phi_{WL\text{subtree}}^{(1)}(G), \phi_{WL\text{subtree}}^{(1)}(G') \rangle = 11.$$

e

---

### Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

---

**Input:** Initial node coloring  $(h_1^{(0)}, h_2^{(0)}, \dots, h_N^{(0)})$

**Output:** Final node coloring  $(h_1^{(T)}, h_2^{(T)}, \dots, h_N^{(T)})$

$t \leftarrow 0;$

**repeat**

**for**  $v_i \in \mathcal{V}$  **do**

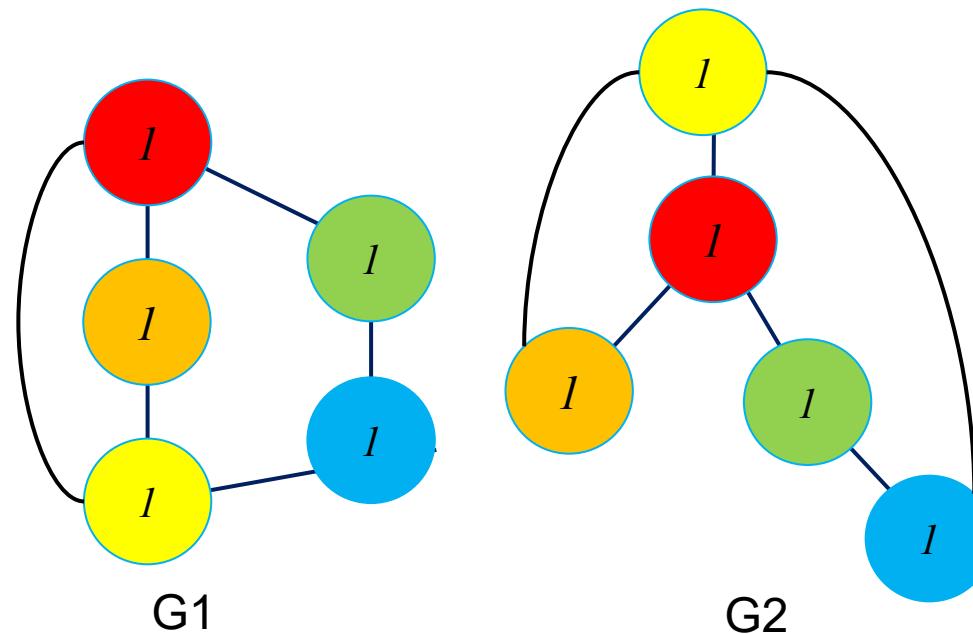
$h_i^{(t+1)} \leftarrow \text{hash} \left( \sum_{j \in \mathcal{N}_i} h_j^{(t)} \right);$

$t \leftarrow t + 1;$

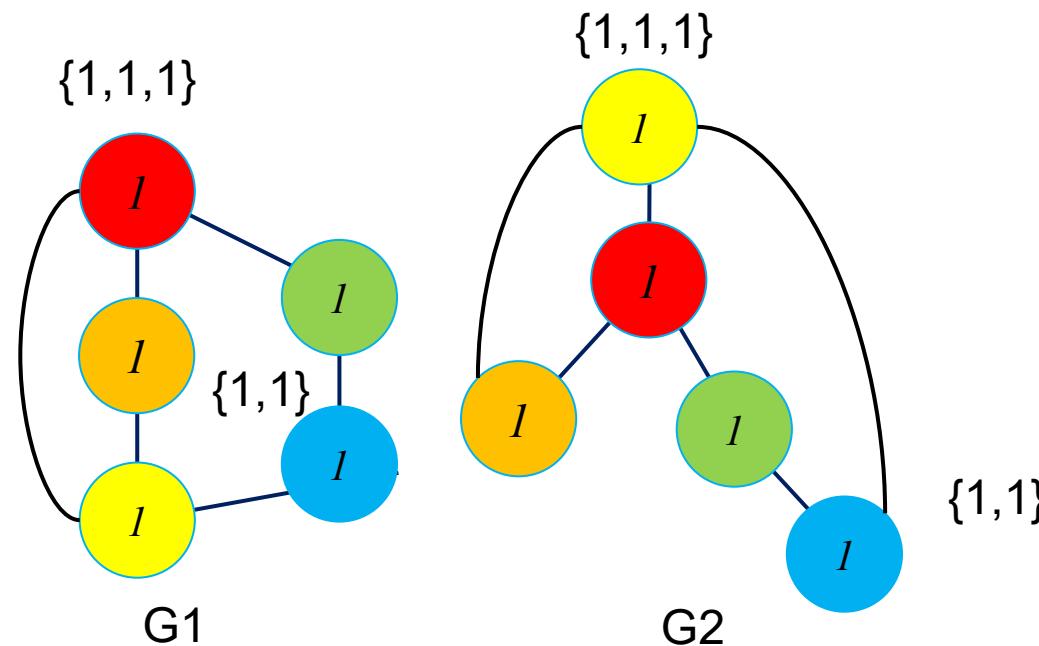
**until** stable node coloring is reached;

---

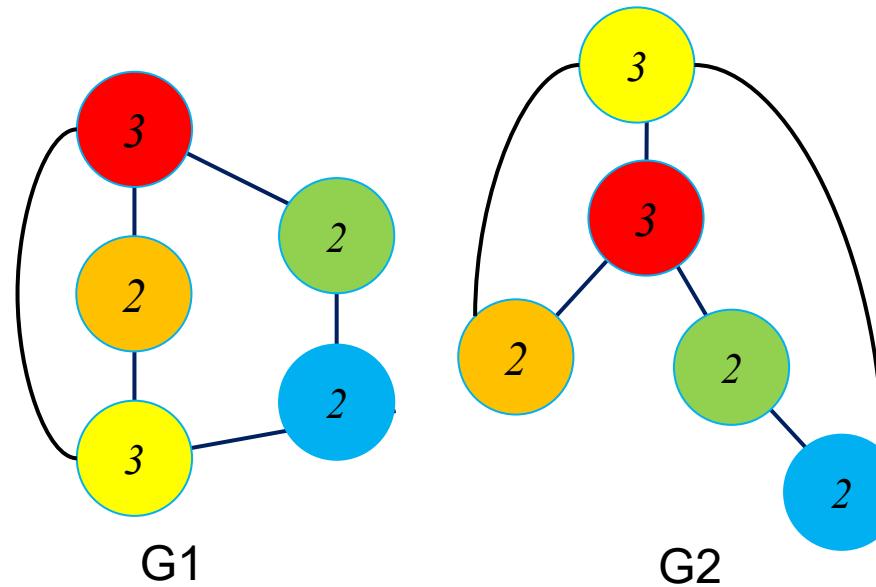
- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test
- Step 1: Set node label =1 for all nodes



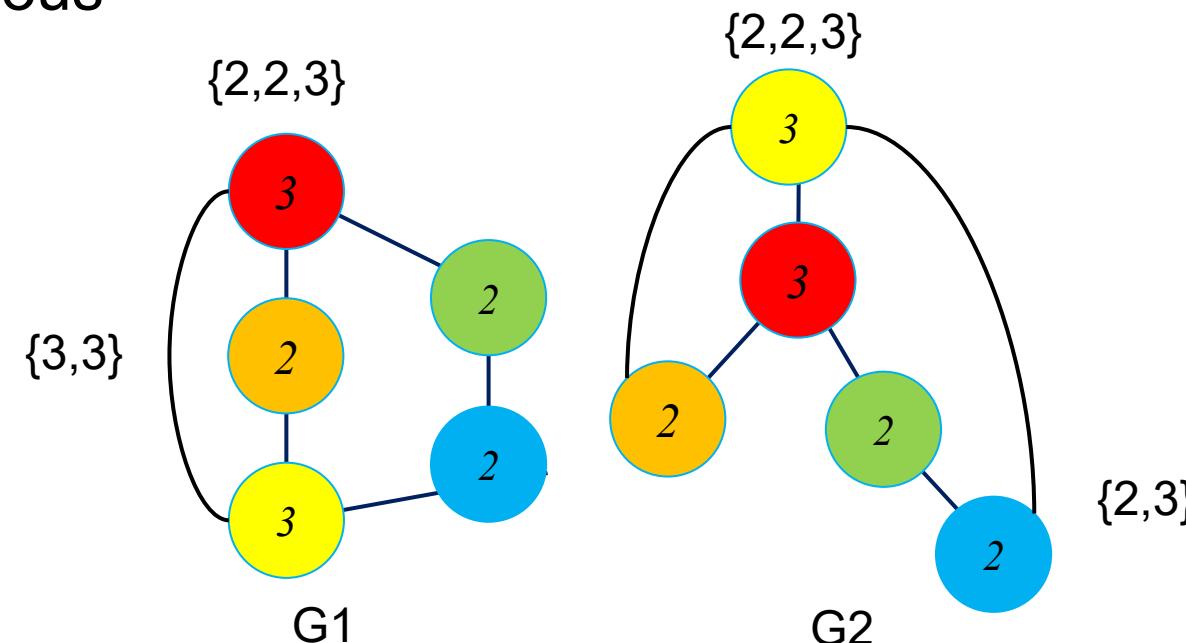
- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighbouring nodes' compressed labels



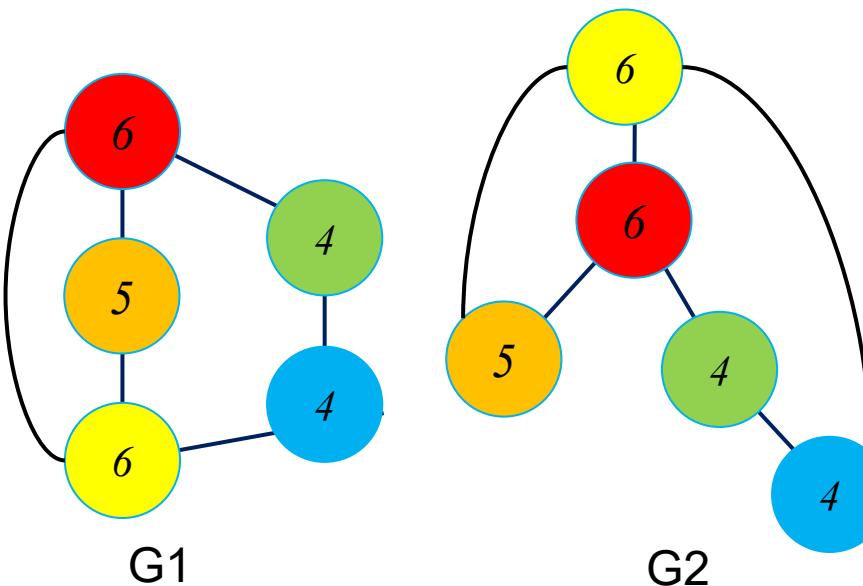
- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighbouring nodes' compressed labels



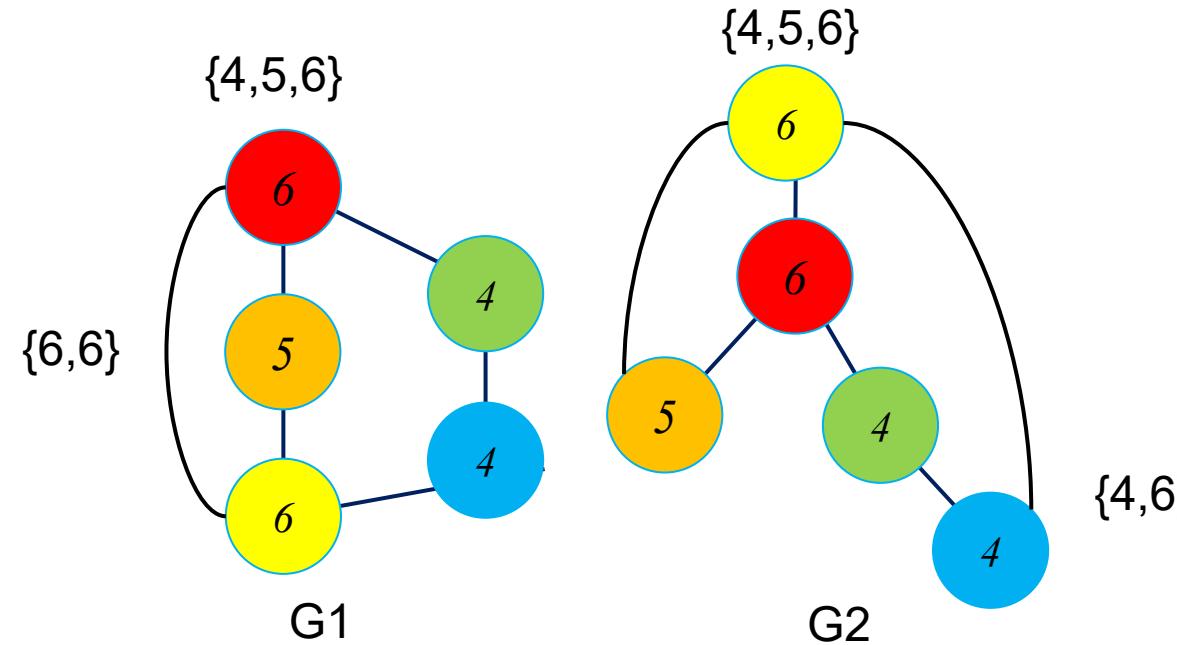
- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test
- Step 1: Set node label = 1 for all nodes
- Step 2: Compute multiset of the neighbouring nodes' compressed labels
- Step 3: Continuous



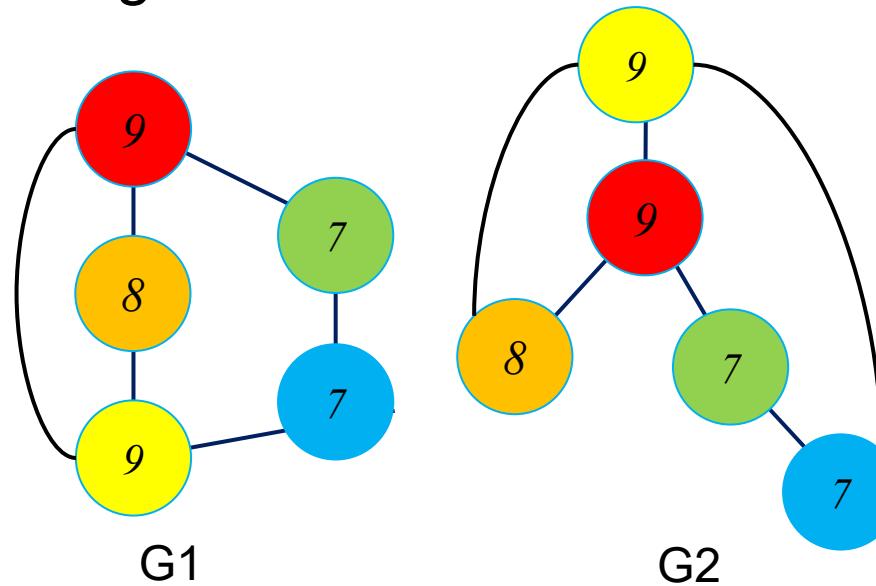
- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighbouring nodes' compressed labels
- Step 3: Continuous....



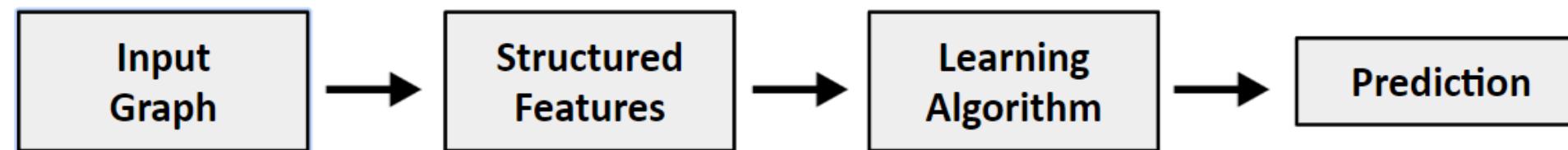
- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighbouring nodes' compressed labels
- Step 3: Continuous...



- We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test
- Step 1: Set node label =1 for all nodes
- Step 2: Compute multiset of the neighboring nodes' compressed labels
- Step 3: Continuous....
- Step 4: Since the partition of nodes by compressed label has not changed, we may terminate the algorithm here



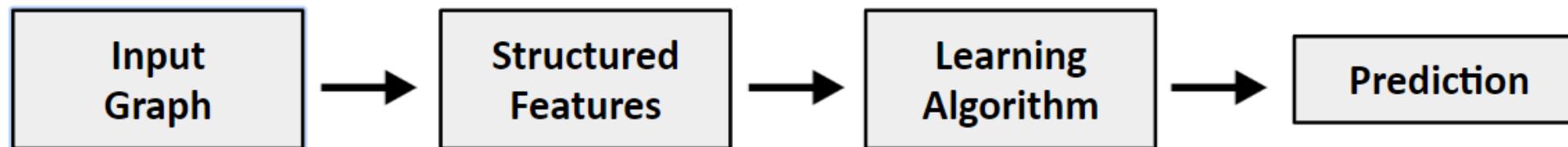
- Given a graph, we can extract node, edge, graph-level features from the graph, then learn a model to map the features to the desired labels



## Feature Engineering

- Node feature
- Edge feature
- Graph feature
- SVM
- Random Forest
- XGBoost
- DNN
- Node-level
- Edge-level
- Graph-level

- GRL aims to generate graph representation vectors that describe graph's structure
- We don't need to do feature engineering **every single time**



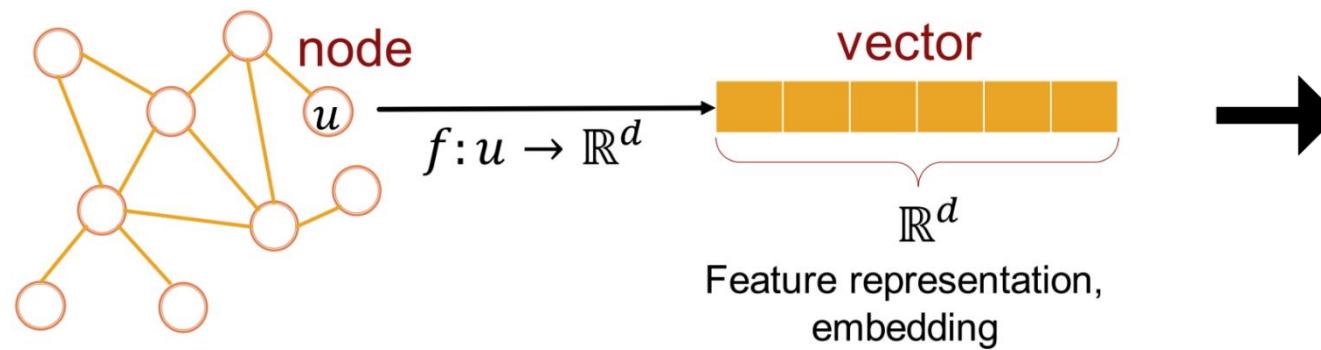
✗ **Feature Engineering**

- SVM
- Random Forest
- XGBoost
- DNN
- Node-level
- Edge-level
- Graph-level

✓ **Representation Learning**

learn the features by itself

- GRL's **goal**: Learn efficient task-independent feature for ML with graphs
  - Map nodes into an **embedding space**
- Similarity of embeddings between nodes indicates their similarity in the network
  - Example: both nodes are close to each other (connected by an edge)
- For simplicity, no node features or extra information is used



## Tasks

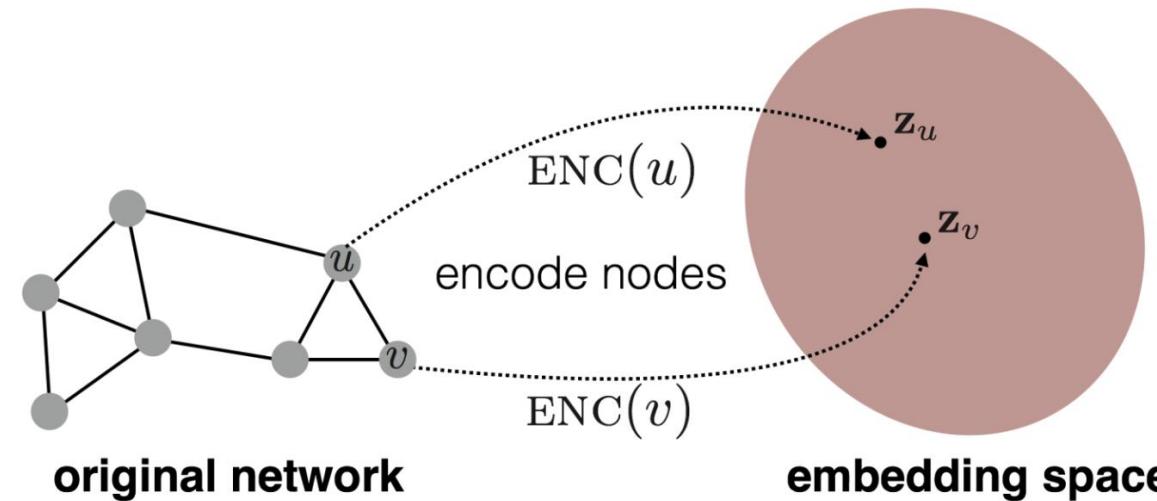
- Node classification/regression
- Edge prediction
- Graph classification/regression.
- Graph clustering
- ...

- Goal: Encode nodes → similarity in embedding space (dot product)  $\approx$  similarity in the original graph

$$\text{similarity}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$$

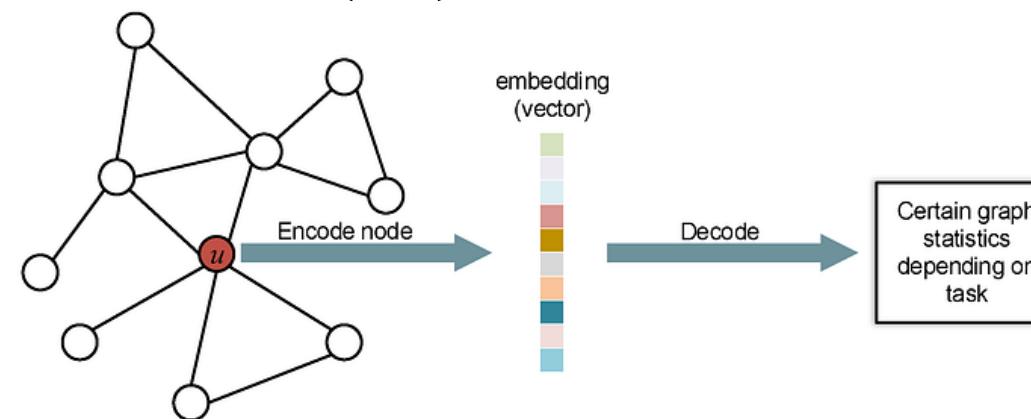
in the original network      Similarity of the embedding

We need to define:  
 $\text{ENC}(u)$   
 $\text{Similarity}(u, v)$

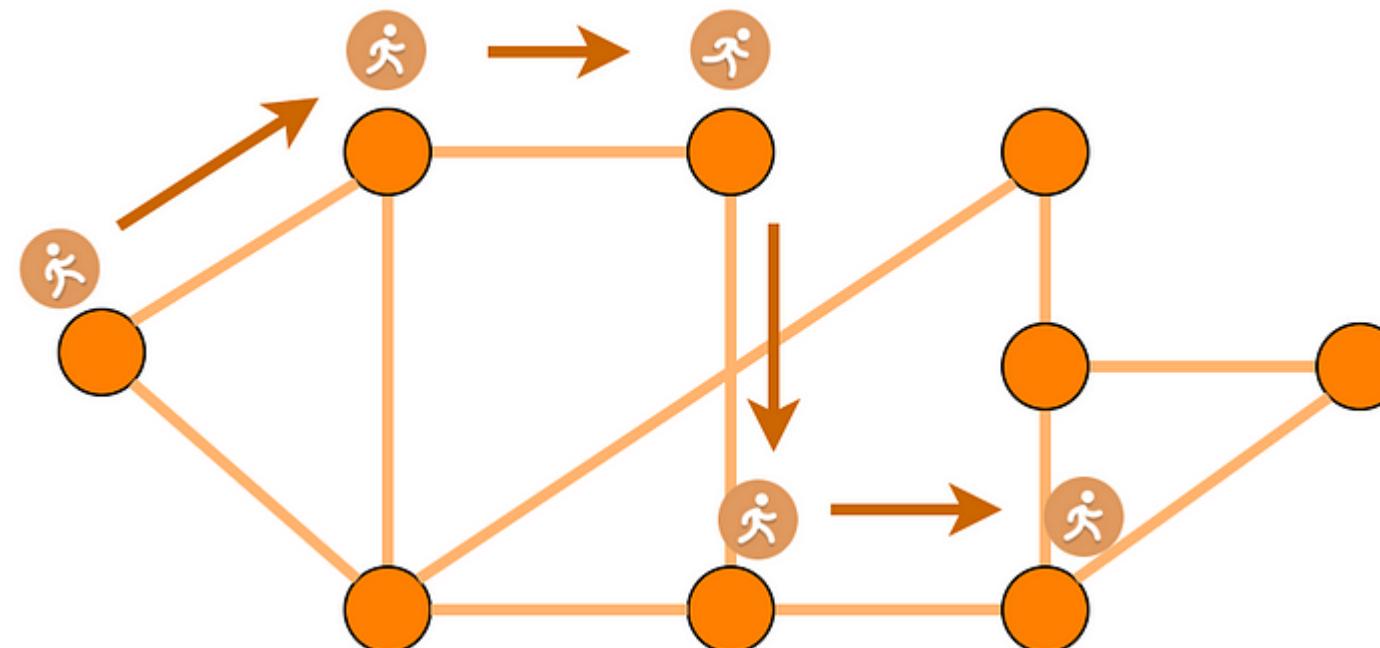


- Encoder: Node → embedding (d-dimensional vector,  $\text{ENC}(v) = Z_v$ )
- Decoder: Map embedding → similarity score (dot product)
- Define a node similarity function:
  - Whether 2 nodes are close in the graph, and how close are they?

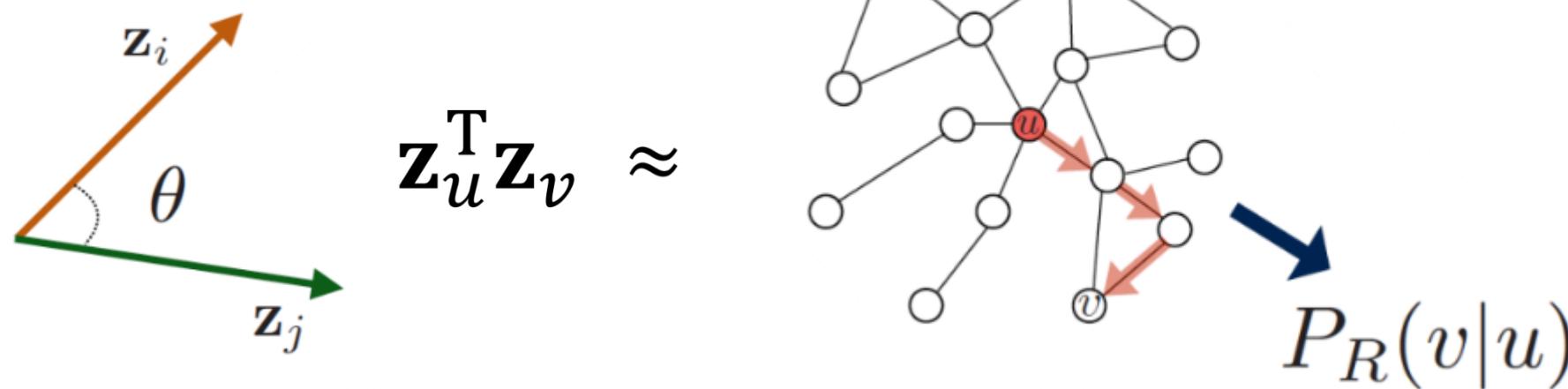
- Optimize →  $\text{similarity}(u, v) \approx z_v^T z_u$   
in the original network
- Dot product between node embedding  
Similarity of the embedding
- Maximize  $z_v^T z_u$  for node pairs  $(u, v)$  that are **similar**



- Given a graph and a starting point, we select a neighbour of it at random, and move to this neighbour
- Then, we select a neighbor of this point at random, and move to it,...
- The random sequence of nodes visited this way is a **random walk on the graph**



- Estimate probability of visiting node  $v$  on a random walk starting from node  $u$  using some random walk strategy  $R$
- Optimize embeddings to encode these random walk statistics



- Run short **random walks starting from each node** on the graph using some strategy R
- For each node  $u$  collect  **$N(u)$ , the multiset\* of nodes visited on random walks starting from  $u$**
- Optimize embeddings to according to:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- \*  $N(u)$  can have repeat elements since nodes can be visited multiple times on random walks

- Employ short random walks on the graph to discover the structure

$v_4 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5 \rightarrow v_1 \rightarrow v_{46}$

**Random walks in Network**  
 =  
**Sentences in NLP**

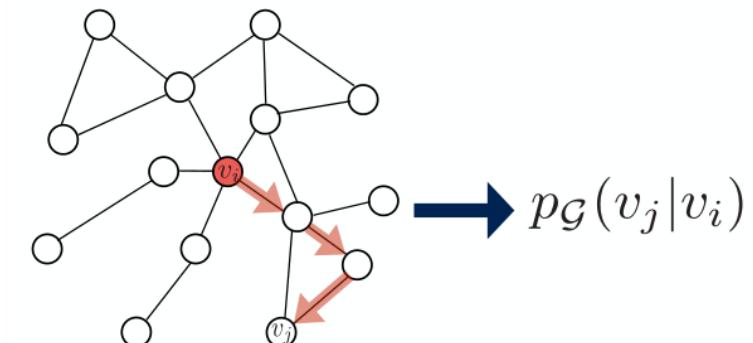
The quick brown fox jumps over the lazy dog.

$$u_k \begin{bmatrix} 3 \\ 1 \\ 5 \\ 1 \\ \vdots \end{bmatrix} v_j \xrightarrow{\Phi} \begin{matrix} d \\ j \end{matrix}$$

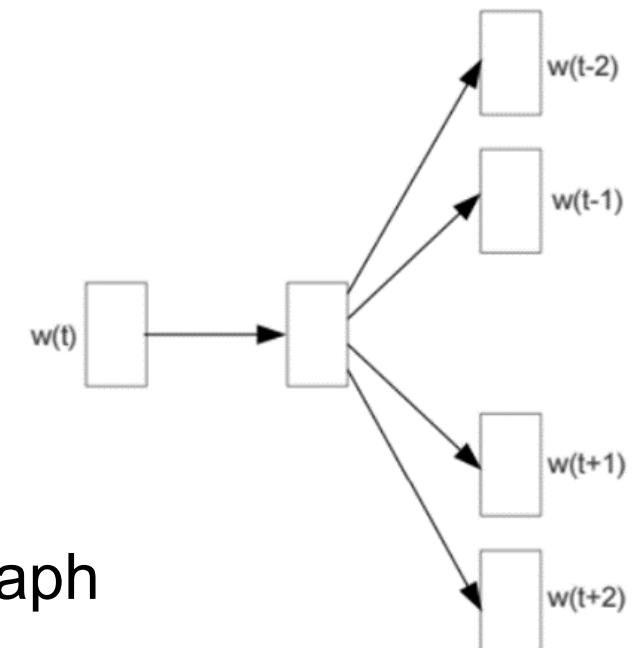
**Node sequence as the input of word2vec models**

$$\Phi: v \in V \mapsto \mathbb{R}^{|V| \times d}$$

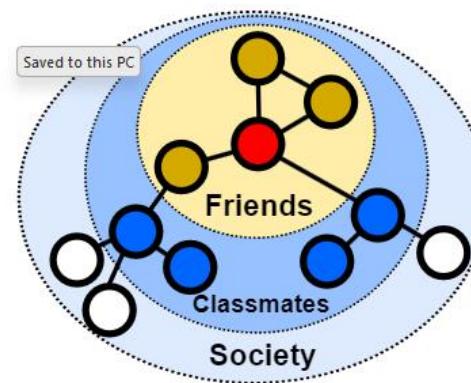
**Short random walk** is not sufficient in large scale graph



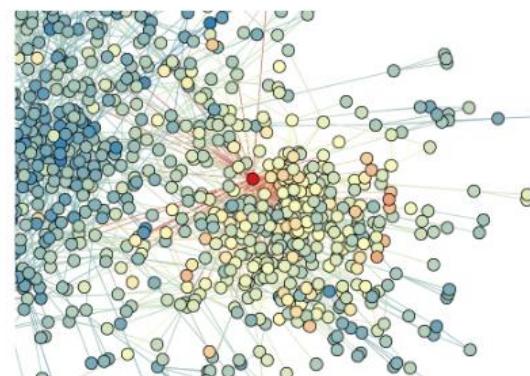
1. Run random walks to obtain co-occurrence statistics.



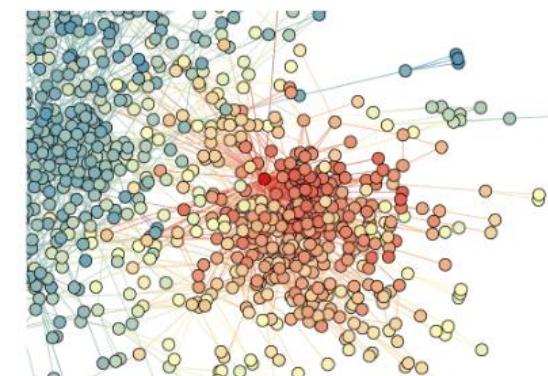
- **Problem:** DeepWalk learn a single global representation that conflates information across all scales
  - Not sufficient in large scale graph
- **Idea:** Multiple Scales of Random Walks at different scales to explicitly deal with large graph
- **Solutions:** Propose WalkLets algorithm with skipping mechanism



(a) A student (in red) is a member of several increasing larger social communities.

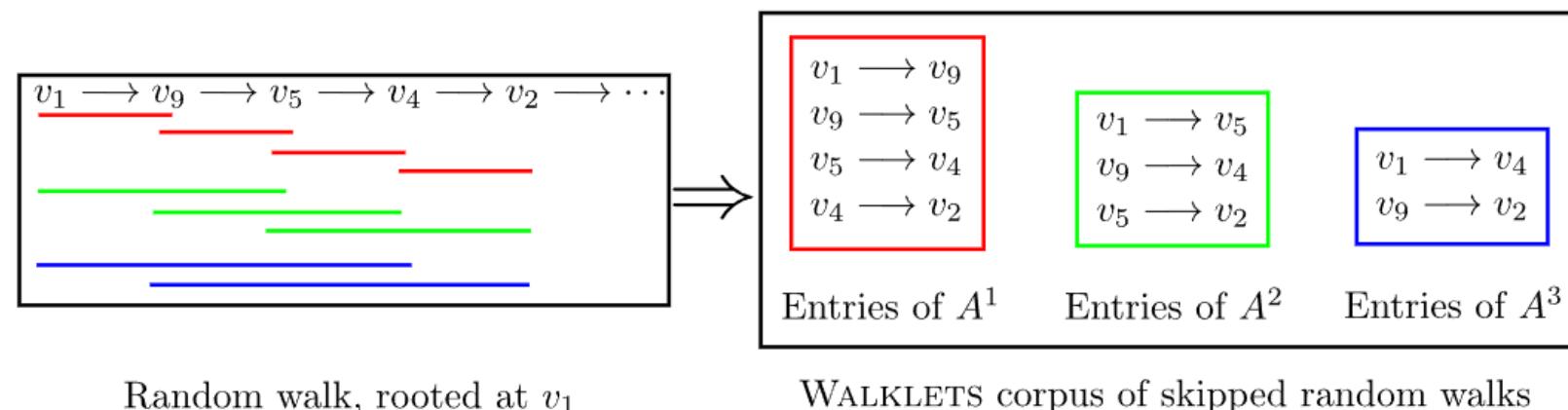


(b) WALKLETS Fine Representation



(c) WALKLETS Coarse Representation

- Generates truncated random walks on the graph, but "skips" over nodes at different scales
  - Results in multiple "corpora" of node co-occurrences at different scales
- Using skip-gram models to learns separate embeddings from each of these corpora
  - Learns multiple embeddings capturing relationships at different scales explicitly
- Optimization by Stochastic Gradient Descent



- Optimize embeddings to **maximize likelihood of random walk co-occurrences**

$$J = - \sum_{v_i, v_j \in C_k} \log \Pr(v_i | v_j)$$

- (Stochastic) Gradient Descent: a simple way to minimize  $J$
- **SGD Algorithm:** evaluate it for each individual training example

- Initialize  $z_u$  at some randomized value for all nodes  $u$
- Iterative until  $J$  converges:

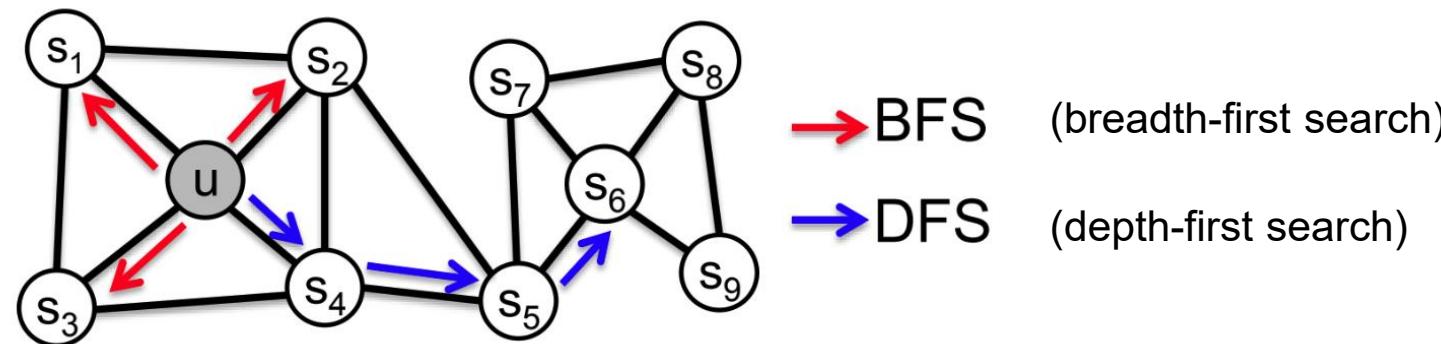
- Sample a node  $u$ , for all  $v$  calculate the derivative  $\frac{\partial J^{(u)}}{\partial z_v}$

- For all  $v$ , update:

$$z_v \leftarrow z_v - \eta \frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$$

Learning rate

- **Idea:** Use flexible, **biased random walks** that can trade-off between local and global views of the network (Grover and Leskovec, 2016)
- Two classic strategies to define a neighbourhood  $N_R(u)$  of a given node  $u$

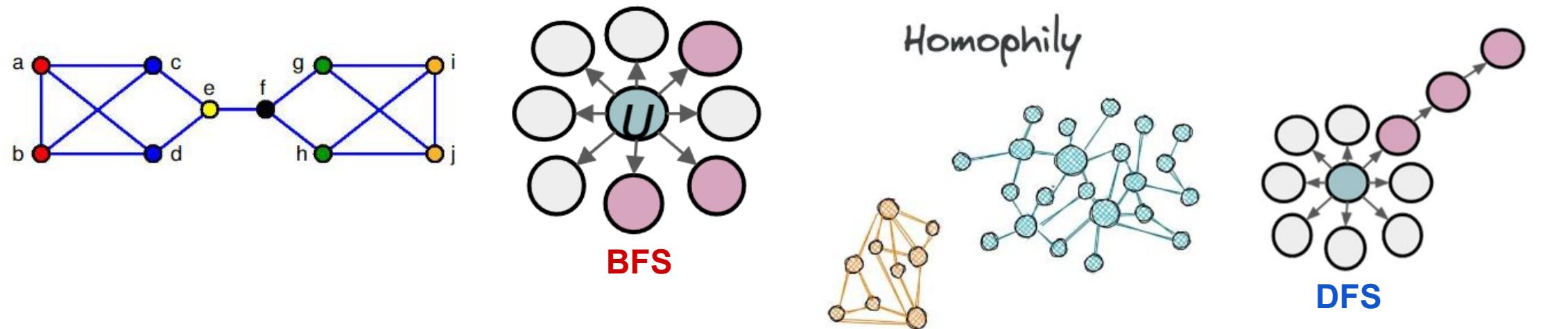


- Walk of length ( $N_R(u)$ ) of size 3:

$$N_{BFS}(u) = \{s_1, s_2, s_3\} \quad \text{Local view}$$

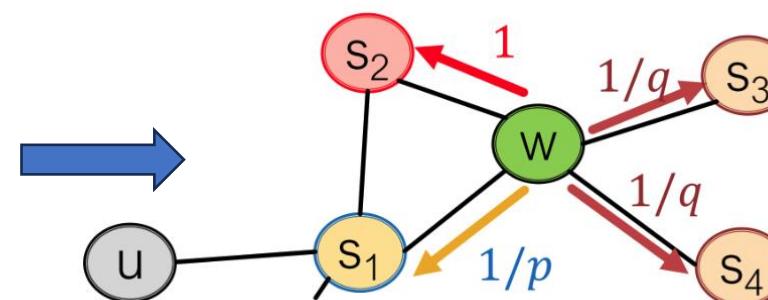
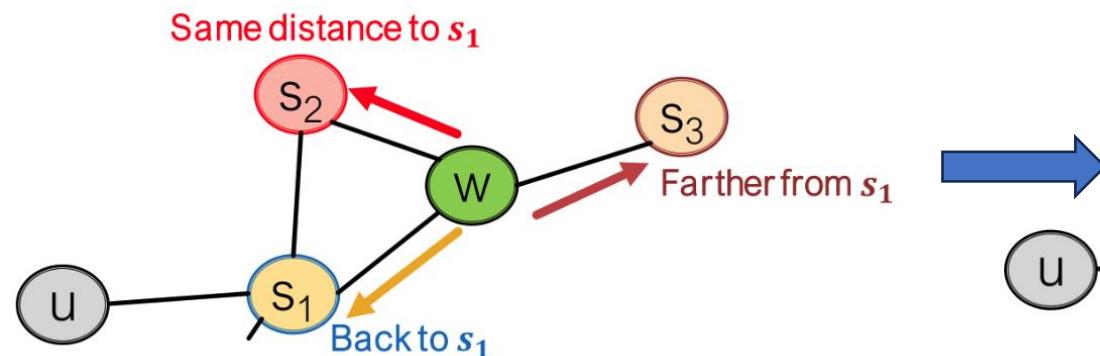
$$N_{DFS}(u) = \{s_4, s_5, s_6\} \quad \text{Global view}$$

- **Structural Equivalence:** if and only if they have the same structure
- **Homophily:** tendency of individuals to associate and bond with similar others
  - Neighborhoods sampled by **BFS** corresponding to **structural** equivalences
  - **DFS** explore macro-view of the network, which infers communities: **homophily** equivalence



- Biased fixed-length random walk  $R$  that given a node  $u$  generates neighborhood  $N_R(u)$ 
  - Two parameters of random walks:
    - Return parameter  $p$ : Return to the previous node
    - In-out parameter  $q$ : Moving outwards (DFS) vs. inwards (BFS) from the previous node

- Biased 2nd-order random walks explore network neighbourhood
- Walker just traversed edge  $(s_1, w)$  and is at  $w$ , now he/she can go using a bias probability:



$1/p, 1, 1/q$  are unnormalized  
probs  
 $p$ : return parameter  
 $q$ : “walk away” parameter

- BFS-like walk: Low value of  $p$
- DFS-like walk: Low value of  $q$

Target $t$	Prob.	Dist. $(s_1, t)$
$s_1$	$1/p$	0
$s_2$	1	1
$s_3$	$1/q$	2
$s_4$	$1/q$	2

➤ **Motivation:**

- Preserving network proximities: LINE seeks to preserve both the **first-order proximity** (direct connections between nodes) and **second-order proximity** (shared neighborhood structures) in the learned embeddings
- Scalable objective functions:
  - Consider first-order and second-order proximities separately
  - Different network types: directed, undirected, weighted or unweighted
- An edge-sampling algorithm is proposed to help stochastic gradient descent on weighted edges

- First-order proximity: local pairwise proximity between two connected nodes
- For each node pair  $(v_i, v_j)$ 
  - If  $(v_i, v_j) \in E$ , the first-order proximity between  $v_i$  and  $v_j$  is  $w_{ij}$
  - Otherwise, the first-order proximity between  $v_i$  and  $v_j$  is 0
- Given an undirected edge  $(v_i, v_j)$ , the joint probability of  $v_i$  and  $v_j$ :

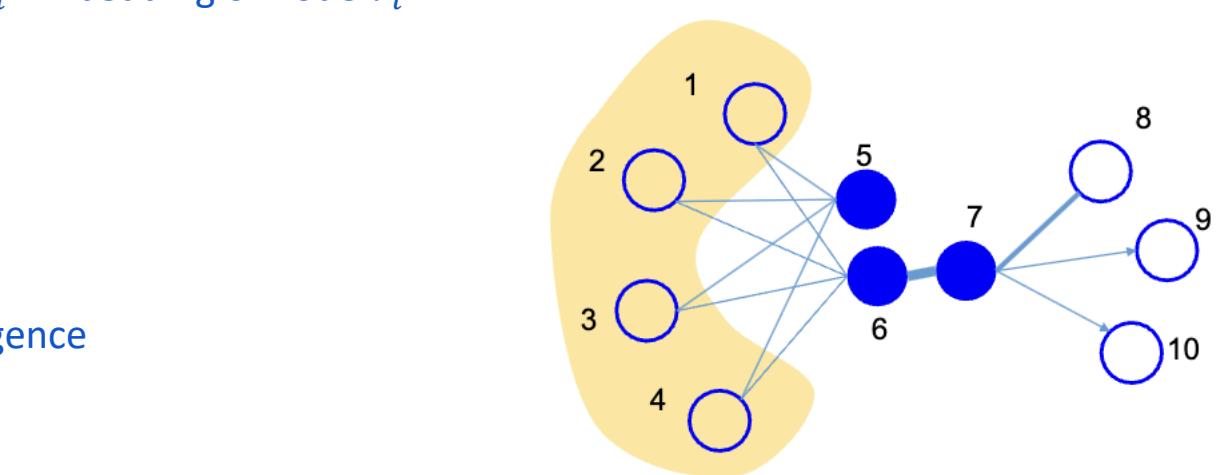
$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}$$

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(i', j')} w_{i' j'}}$$

- Objective:

$$O_1 = d(\hat{p}_1(\cdot, \cdot), p_1(\cdot, \cdot))$$

$$\propto - \sum_{(i,j) \in E} w_{ij} \log p_1(v_i, v_j)$$

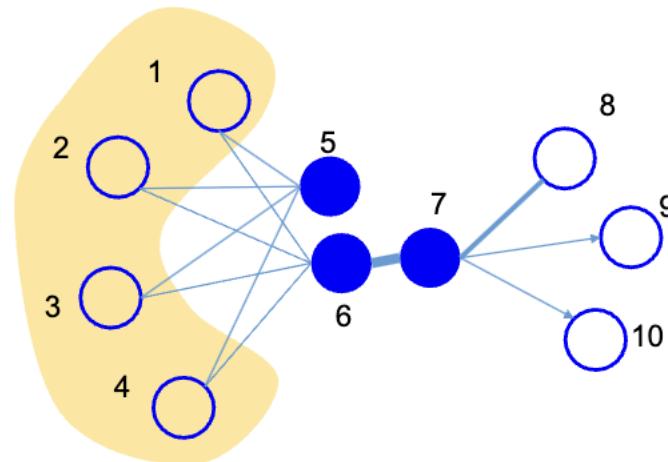


First-order: node 6 and 7

- Second-order proximity: captures the 2-step relations between each pair of nodes
- For each node pair  $(v_i, v_j)$ 
  - Determine by the number of common neighbours shared by the two nodes

$$\hat{p}_u = (w_{u1}, w_{u2}, \dots, w_{u|V|})$$

$$\hat{p}_v = (w_{v1}, w_{v2}, \dots, w_{v|V|})$$



Node 5 and 6 have a large second-order proximity

$$\hat{p}_5 = (1, 1, 1, 1, 0, 0, 0, 0, 0, 0)$$

$$\hat{p}_6 = (1, 1, 1, 1, 0, 0, 1, 0, 0, 0)$$

- Given an undirected edge  $(v_i, v_j)$ , the joint probability of  $v_i$  and  $v_j$ :

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)},$$

$\vec{u}_i$ : Embedding of node  $v_i$  when i is a source node.

$\vec{u}'_i$ : Embedding of node  $v_i$  when i is a target node.

$$\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{\sum_{k \in V} w_{ik}}$$

- Objective:

$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i)),$$

$$\propto - \sum_{(i,j) \in E} w_{ij} \log p_2(v_j|v_i).$$

$\lambda_i$ : Prestige of node in the network  $\lambda_i = \sum_j w_{ij}$

- Concatenate the embeddings individually learned by the two proximity:

First-order:



Second-order:



- Stochastic Gradient Descent + Negative Sampling

- Randomly sample an edge and multiple negative edges

- The gradient w.r.t the embedding with edge (i,j)

$$\frac{\partial O_2}{\partial \vec{u}_i} = w_{ij} \cdot \frac{\partial \log p_2(v_j|v_i)}{\partial \vec{u}_i}$$

Multiplied by the weight of the edge  $w_{ij}$

- Problematic when the weights of the edges diverge

- The scale of the gradients with different edges diverges

- **Solution:** Edge sampling

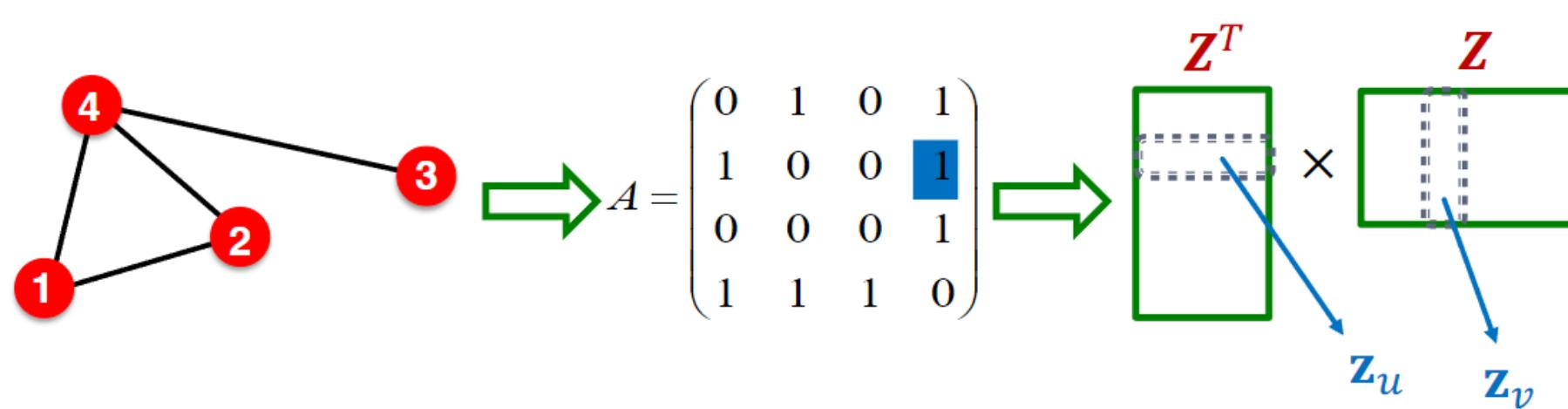
- Sample the edges according to their weights and treat the edges as binary

- Complexity:  $\Theta(dK|E|)$

- Linear to the dimension d, number of negative samples K, and number of edges E

- One of the simplest and most intuitive approaches to defining similarity:
  - Adjacency between two nodes v and u
- Similarity: Two nodes are adjacent to one another within the structure of the graph

$$\mathbf{z}_v^T \mathbf{z}_u = A_{u,v} \text{ or } \mathbf{Z}^T \mathbf{Z} = A$$



- Encoding: Find the embedding matrix  $\mathbf{Z}$  that minimizes the loss function  $\mathbf{L}$

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}\|^2$$

loss (what we want to minimize)

sum over all node pairs

embedding similarity

(weighted) adjacency matrix for the graph

- Solution:

- Option 1: Use stochastic gradient descent (SGD) as a general optimization method
- Option 2: Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines)
  - Matrix Factorization-based

- Let  $A$  be an  $m \times n$  adjacency matrix. The factorization of  $A$  takes the form

$$A = USV^T$$

where  $U$  is a  $m \times m$  orthogonal matrix,  $V^T$  is a  $n \times n$  orthogonal matrix and  $S$  is a  $m \times n$  diagonal matrix

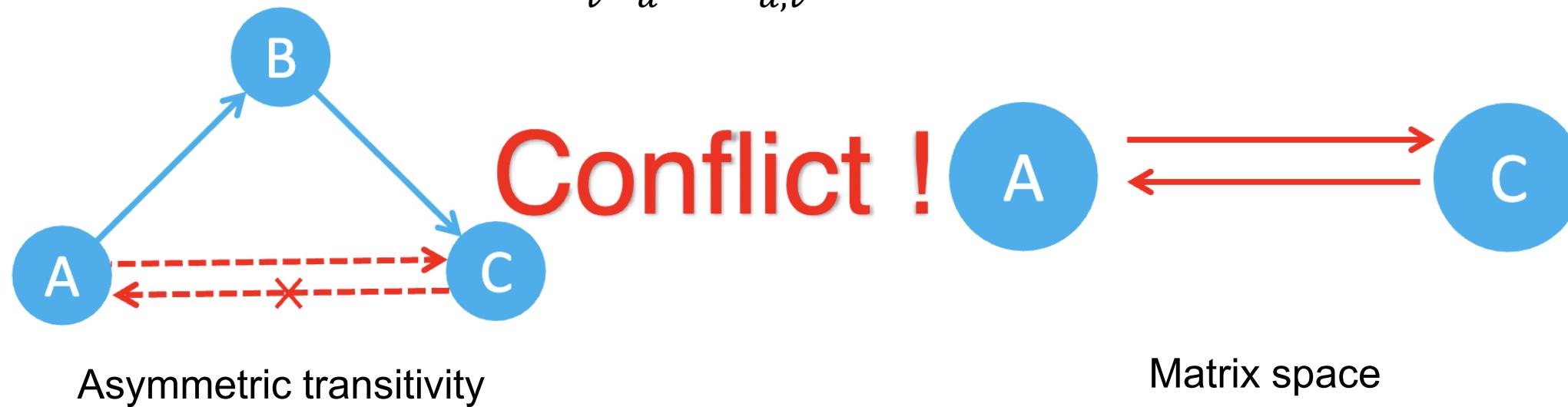
- Factorizing to low-rank approximations based on minimizing the sum-squared distance using **Singular Value Decomposition**
- To decompose:
  - Evaluate the  $n$  eigenvectors  $v_i$  and eigenvalues  $\lambda_i$  of  $A^T A$
  - Make a matrix  $V$  from the normalized vectors  $v_i$
  - Make a diagonal matrix  $S$  from the square roots of the eigenvalues

$$\sigma_i = \sqrt{\lambda_i} \quad \text{and} \quad \sigma_1 \geq \sigma_2 \geq \sigma_3 \dots$$

- Find  $U$ :  $A = USV^T \Rightarrow US = AV \Rightarrow U = AVS^{-1}$

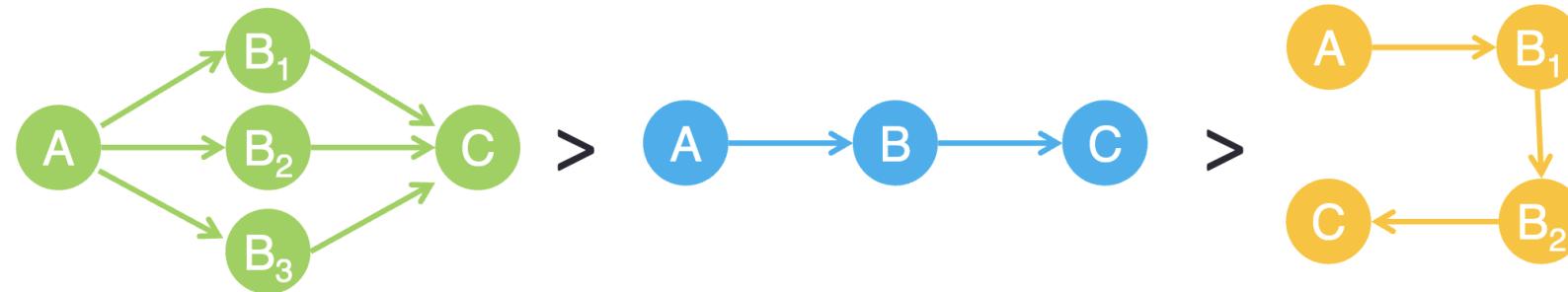
- **Ideas:** Critical property in directed graph - **Asymmetric Transitivity**
- Transitivity is **Asymmetric** in directed graph
- **Challenge:** incorporate asymmetric transitivity in graph embedding
- **Problem:** metric space is **symmetric**

$$\mathbf{z}_v^T \mathbf{z}_u = A_{u,v} \text{ or } \mathbf{Z}^T \mathbf{Z} = A$$



➤ **High-order Proximity with asymmetric transitivity:**

- Asymmetry: not symmetric in directed graph
- Transitivity:
  - More directed paths, larger similarity
  - Shorter paths, larger similarity



- Compare A → C similarity: Katz Index

$$\mathbf{s}^{Katz} = \sum_{l=1}^{+\infty} (\beta \cdot \mathbf{A})^l$$

➤ **Preserve high-order proximity embedding:**

- Katz Index modified:

$$\mathbf{S}^{Katz} = \sum_{l=1}^{+\infty} (\beta \cdot A)^l = (I - \beta \cdot A)^{-1} \cdot (\beta \cdot A)$$

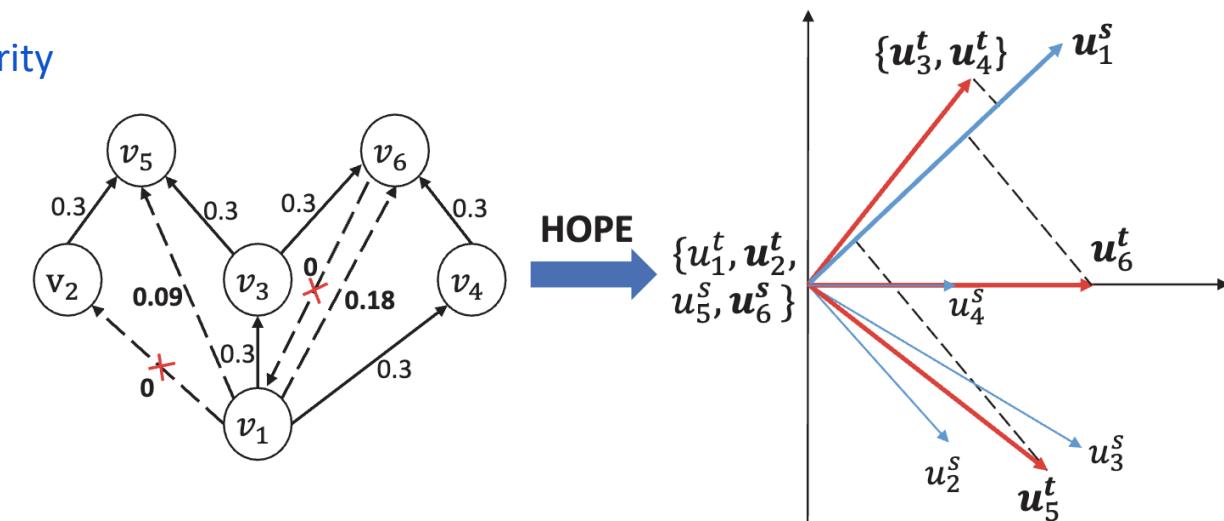
- Objective:

$$\min_{U_s, U_t} \| \mathbf{S} - U_s \cdot U_t^T \|_F^2$$

$$\mathbf{S} = M_g^{-1} \cdot M_l$$

Embedding similarity

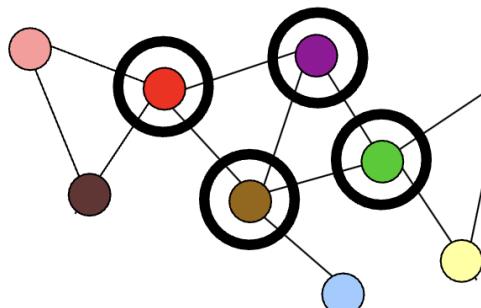
where  $M_g, M_l$  are polynomial of adjacency matrix or proximity measurements: Katz, Adamic Adar, PageRank, Common Neighbours



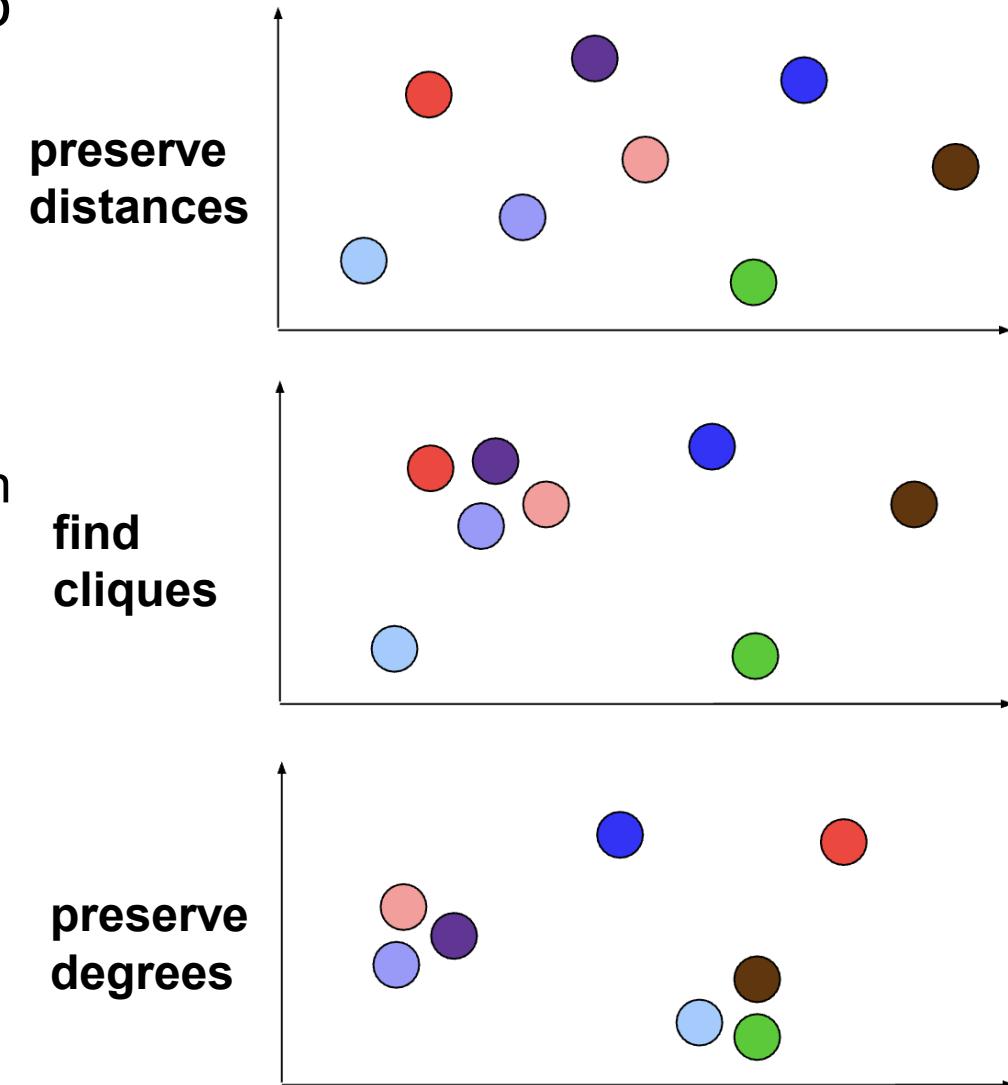
- Solving by **Generalized Singular Value Decomposition**: decompose  $\mathbf{S}$  without calculating it

- **Linear complexity** w.r.t. the volume of data (i.e. edge number)

- Network embedding: map network nodes into Euclidean space
- Structural Identity:
  - Nodes in networks have specific roles
    - E.g., individuals, web pages, proteins, etc
  - Structural identity: identification of nodes based on network structure (no other attribute)
    - often related to role played by node
  - **Automorphism**: strong structural equivalence



**Red, Green:** automorphism  
**Purple, Brown:** structurally similar



- **Ideas:** Based on structural identity
- Structural similarity does not depend on hop distance
  - Neighbour nodes can be different, far away nodes can be similar
- Structural identity as a hierarchical concept
  - Depth of similarity varies
- Flexible four step procedure
  - Operational aspect of steps are flexible

➤  $g(D_1, D_2)$ : distance between two ordered sequences

- Cost of pairwise alignment:  $\max(a, b) / \min(a, b) - 1$
- Optimal alignment by Dynamic Time Warping (DTW) in our framework

$$s(R_0(u)) = 4$$

$$s(R_0(v)) = 3$$

$$g(\cdot, \cdot) = 0.33$$

$$s(R_1(u)) = 1, 3, 4, 4$$

$$s(R_1(v)) = 4, 4, 4$$

$$g(\cdot, \cdot) = 3.33$$

$$s(R_2(u)) = 2, 2, 2, 2$$

$$s(R_2(v)) = 1, 2, 2, 2, 2$$

$$g(\cdot, \cdot) = 1$$

➤  $f_k(u, v)$ : structural distance between nodes  $u$  and  $v$  considering first  $k$  rings

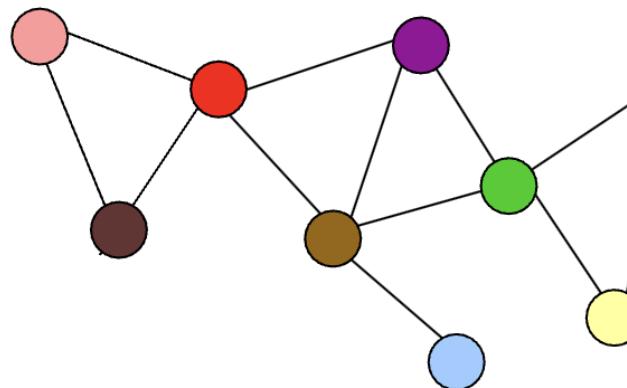
$$\triangleright f_k(u, v) = f_{k-1}(u, v) + g(s(R_k(u)), s(R_k(v)))$$

$$f_0(u, v) = 0.33$$

$$f_1(u, v) = 3.66$$

$$f_2(u, v) = 4.66$$

- Encodes structural similarity between all node pairs



$$w(u_k, u_{k+1}) = \log(\Gamma_k(u) + e), \quad k = 0, \dots, k^* - 1$$

$$w(u_k, u_{k-1}) = 1, \quad k = 1, \dots, k^*$$

- Each layer is weighted complete graph

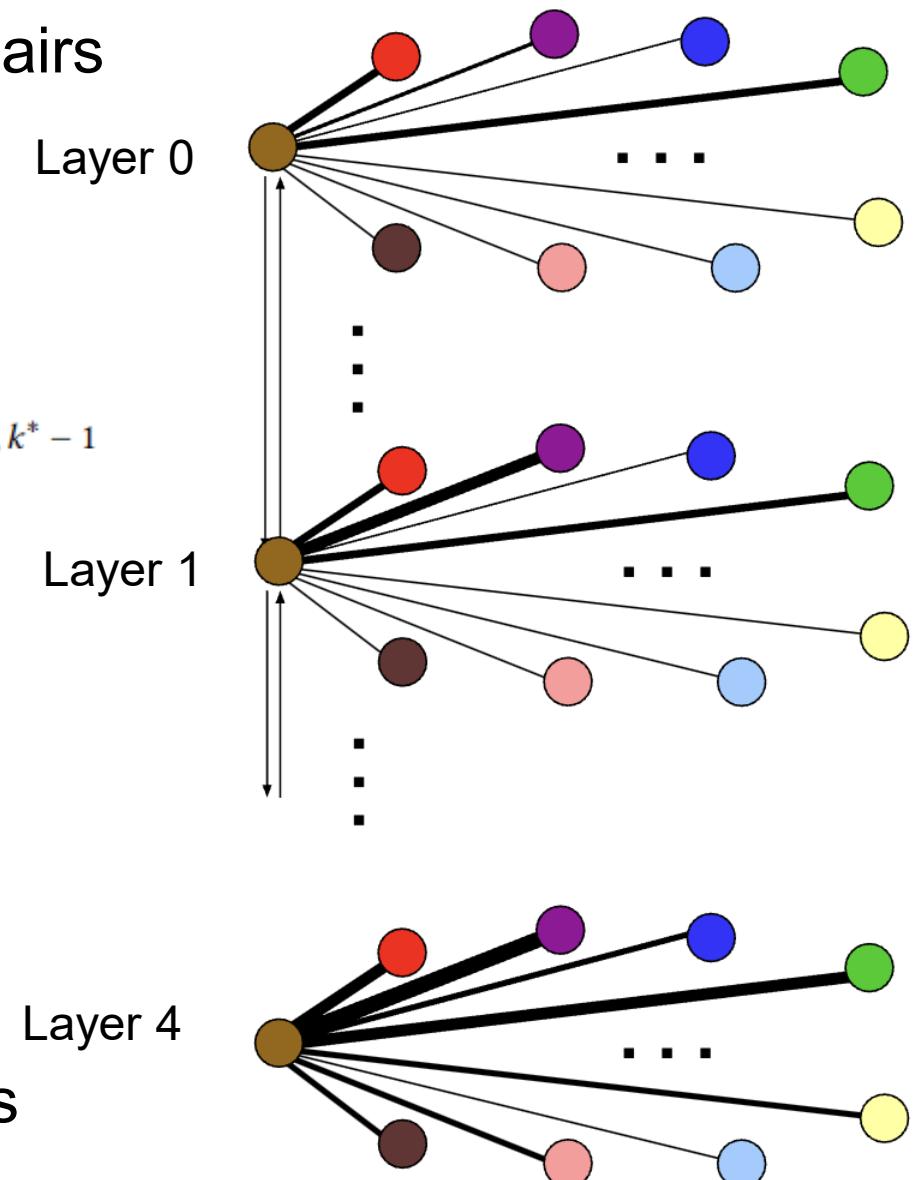
➤ Corresponds to similarity hierarchies

- Edge weights in layer  $k$

$$\Gamma_k(u) = \sum_{v \in V} \mathbb{1}(w_k(u, v) > \bar{w}_k)$$

$$w_k(u, v) = e^{-f_k(u, v)}, \quad k = 0, \dots, k^*$$

- Connect corresponding nodes in adjacent layers



- Context generated by biased random walk (same as Node2vec)

- Walking on multi-layer graph

- Walk in current layer with probability p:  $p_k(u, v) = \frac{e^{-f_k(u, v)}}{Z_k(u)}$

➤ Choose neighbour according to edge weight where  $Z_k(u)$  is the normalization factor for vertex  $u$  in layer  $k$ ,

- RW prefers more similar nodes

$$Z_k(u) = \sum_{\substack{v \in V \\ v \neq u}} e^{-f_k(u, v)}$$

- Change layer with probability 1-p

- Choose up/down according to edge weight

- RW prefer layer with less similar neighbours

$$p_k(u_k, u_{k+1}) = \frac{w(u_k, u_{k+1})}{w(u_k, u_{k+1}) + w(u_k, u_{k-1})}$$

$$p_k(u_k, u_{k-1}) = 1 - p_k(u_k, u_{k+1})$$

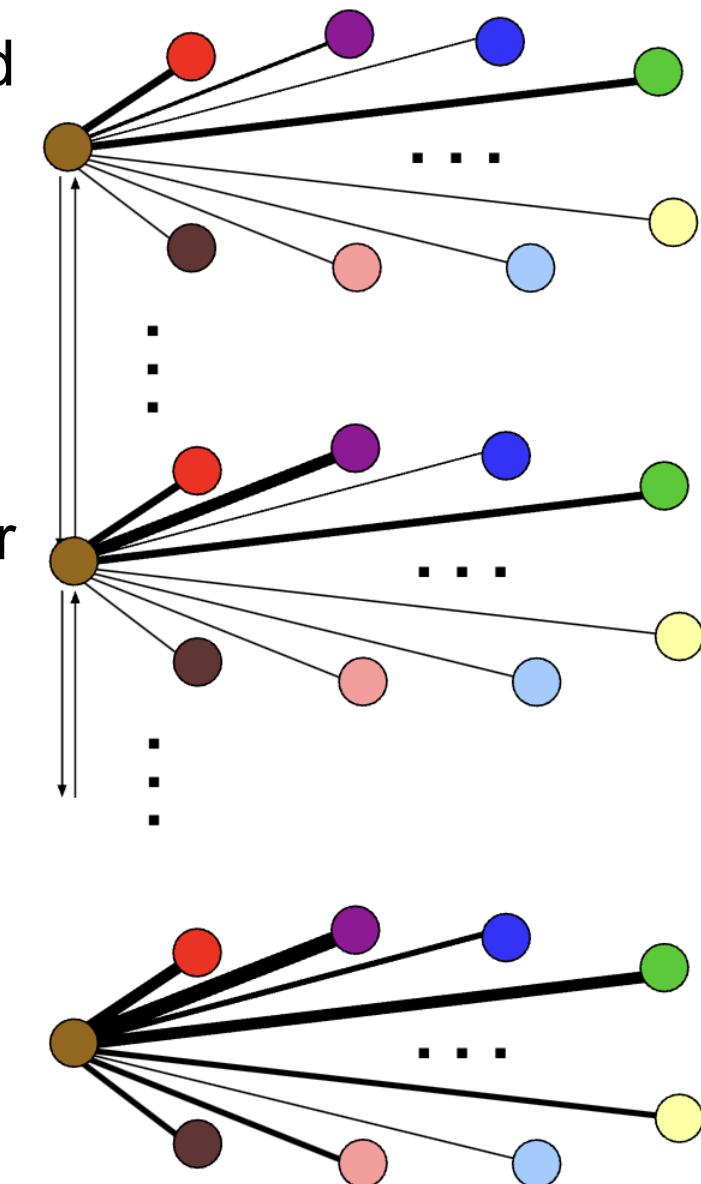
- For each node, generate set of independent and relative short random walks

- Context for node; sentences of a language



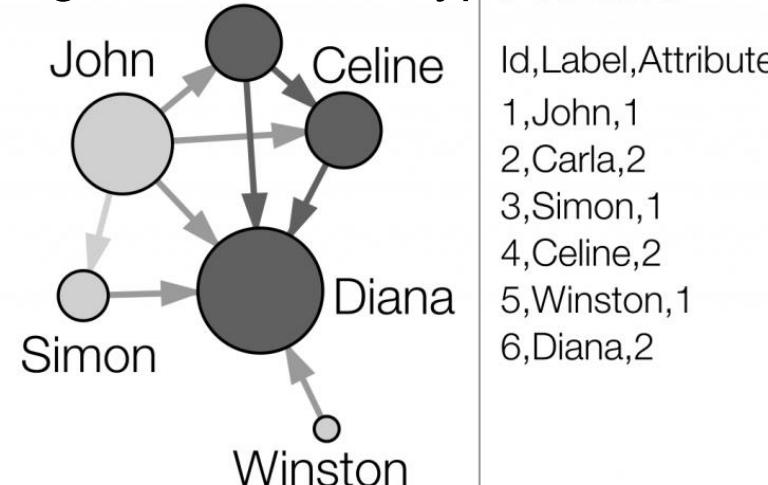
- Train a neural network to learn latent representation for nodes

- maximize probability of nodes within context
  - Skip-gram (Hierarchical Softmax) adopted

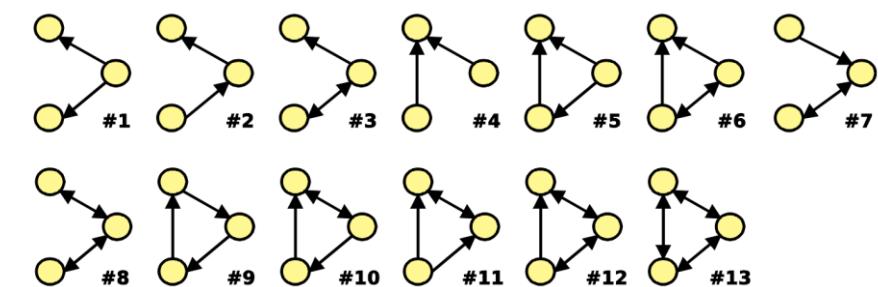


- Reduce time to generate/store multi-layer graph and context for nodes:
  - Option 1: Reduce length of degree sequences
    - Use pairs (degree, number of occurrences)
  - Option 2: Reduce number of edges in multi-layer graph
    - Only  $\log n$  neighbours per node
  - Option 3: Reduce number of layers in multi-layer graph
    - Fixed (small) number of layers
- Scales quasi-linearly
  - Over 1 million nodes

- **Ideas:** struc2vec and conventional methods (DeepWalk, Node2vec) learn in node representation
  - Role2Vec learns embeddings for "node types/roles" determined by node attributes/features
- **Solution:** "attributed random walks"
  - Maps nodes to node types/roles using node attributes like **motif counts, degrees** etc
  - Generates attributed random walks over the node types instead of node IDs
  - Finally, it learns embeddings for ~~the node types~~ **Nodes** other than individual nodes



- Given a network, most of the time, some subgraphs are “overrepresented”
- A connected graph that has many occurrences in a network is called a **motif** of the network
- Assume set of occurrences  $G'$  in  $G$  is  $occ_G(H)$ 
  - Cardinality of  $occ_G(H)$  in  $G$  is **frequent**
  - How to know if  $G'$  is **frequent** in  $G$ ?

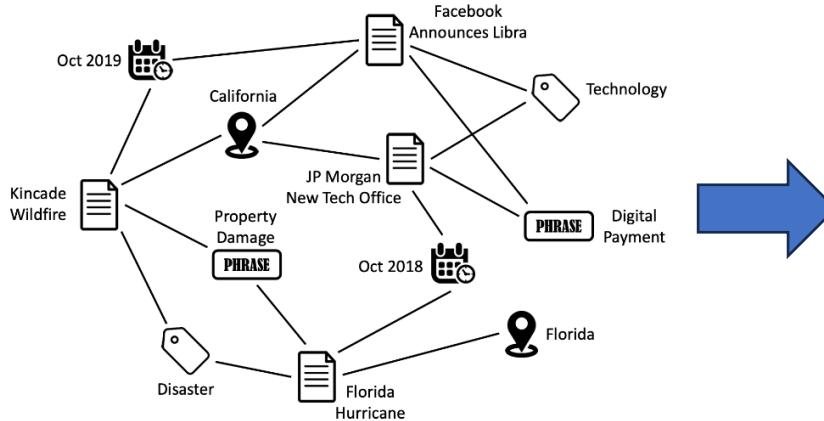


→ Compute the probability that  $occ_N(G') \geq occ_G(G')$  for a random network  $N$   
 $G'$  is said to be frequent in  $G$  if this probability is small enough  
To compute this probability, we need to have a distribution over networks

- **Input:** Graph  $G$ , node attribute matrix  $X$ , embedding dimension  $D$ , number of walks per node  $R$ , walk length  $L$ , context window size  $\omega$
- If  $X$  is not available, extract structural features like motif counts from the graph structure itself and use those as node attributes in  $X$
- Apply logarithmic binning to the node attribute values in  $X$ . Map nodes to node types/roles using a function  $\phi(x)$  that takes the node attribute vector  $x$  as input
  - Two types of  $\phi$  functions:
    - a) Simple functions like concatenation of attribute values
    - b) Low-rank matrix factorization of  $X$
- Precompute random walk transition probabilities  $\pi$ . Generate  $R$  attributed random walks of length  $L$  for each node, using the node type mapping  $\phi$  instead of node IDs
- Learn embeddings using stochastic gradient descent on the attributed random walks, optimizing the probability of observing the context node types in the walks
- **Output:** Learned embeddings for each node type  $w \in W$ , where  $W$  is the set of node types found by  $\phi$

- Mostly same as Node2Vec, except their work are considered in terms of node attribute, not node representation
- There are many advantage from node attributed embedding:
  - Embeddings generalize to new nodes/graphs (inductive learning)
  - Better captures structural node roles
  - Space-efficient as it learns fewer embeddings for node types instead of all nodes
  - Supports attributed graphs

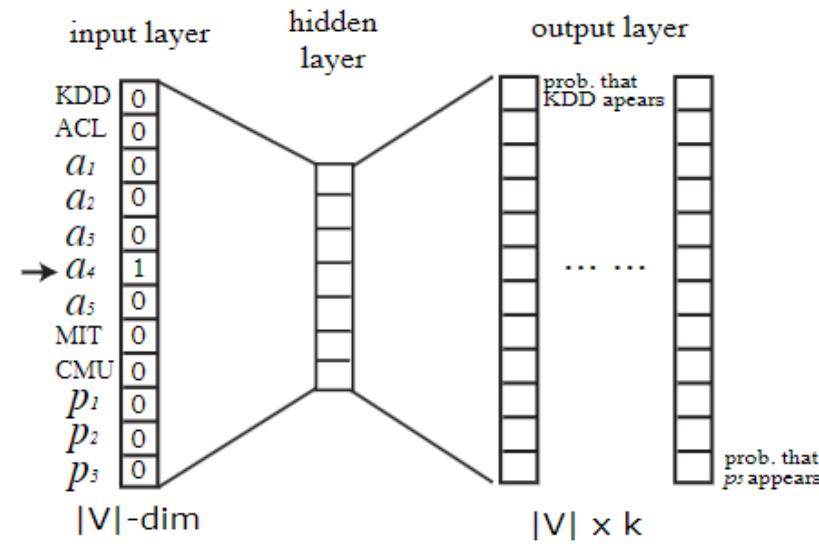
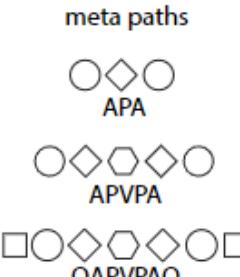
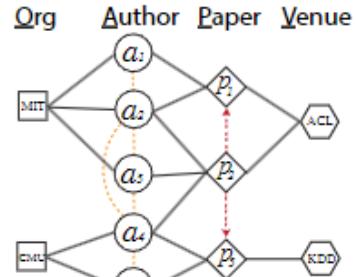
- Previous works are considering **homogeneous network**
  - Only **one type** of **nodes** and **edges**
  - Real world is ubiquitous. For example: social media websites like Facebook contain a set of node types, such as users, posts, groups and tags
  - Special case of heterogeneous network
- Input: A heterogeneous information network  $G = (V, E, T)$ .  $T$  is object relation type
- Output:  $X \in R^{|V| \times d}$ ,  $d \ll |V|$ ,  $d$ -dim vector  $X_v$  for each node  $V$



Latent representation vector

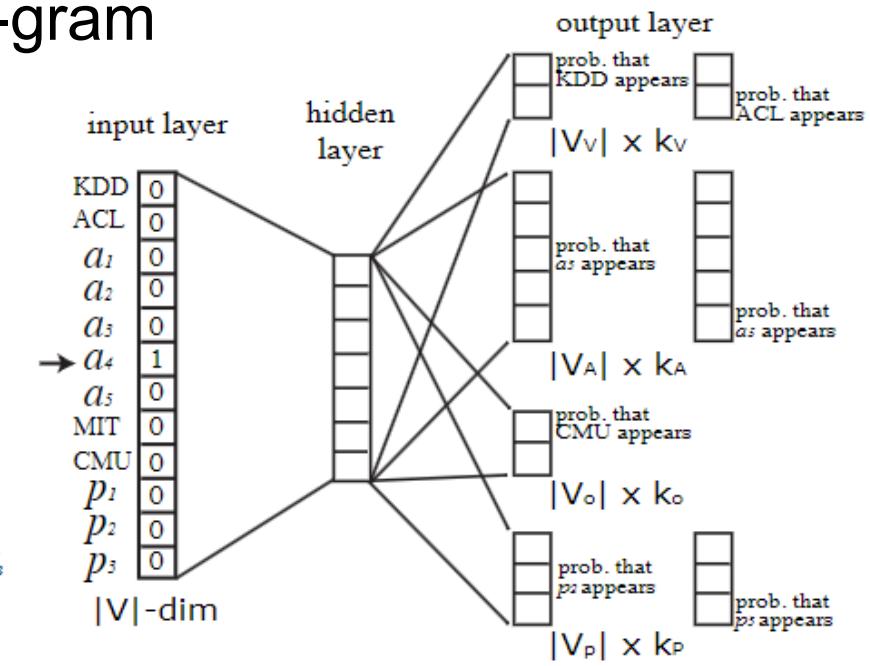
- How do we effectively preserve the concept of “node-context” among multiple types of nodes?
  - E.g., users, posts, groups and, tags in social heterogeneous networks
  - Or authors, papers, & venues in academic heterogeneous networks
- Can we directly apply homogeneous network embedding architectures to heterogeneous networks?
- It is also difficult for conventional meta-path-based methods to model similarities between nodes without connected meta-paths

- **Solution:** meta-path based random walk (inspired by DeepWalk and Node2Vec)
- metapath2vec and metapath2vec++
- **Goal:** to generate paths that can capture both the **semantic** and **structural correlations** between different types of nodes, facilitating the transformation of heterogeneous network structures into skip-gram



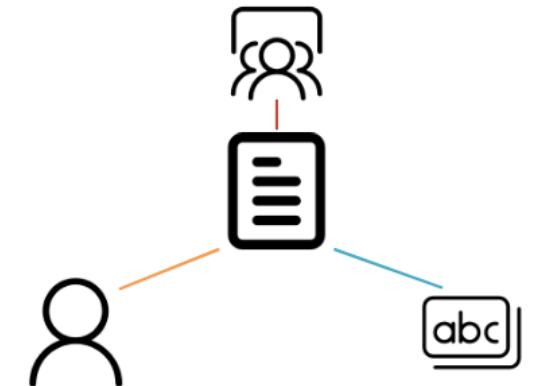
Heterogenous network

Skip-gram in metapath2vec



Skip-gram in metapath2vec++

- Network schema  $S = (V, R)$  of graph G
  - Directed graph defined over node types V and with edges as relations from R
  
- meta-path: based on network schema S
  - Denoted as  $V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \dots V_t \xrightarrow{R_t} V_{t+1} \dots \xrightarrow{R_{l-1}} V_l$
  - Node types  $V_1, V_2, \dots, V_l \in V$  and edge type  $R_1, R_2, \dots, R_{l-1} \in R$



- Each meta-path captures the proximity between the nodes on its two ends from a particular semantic perspective

- **Metapath2Vec:**

- Given a meta-path scheme:

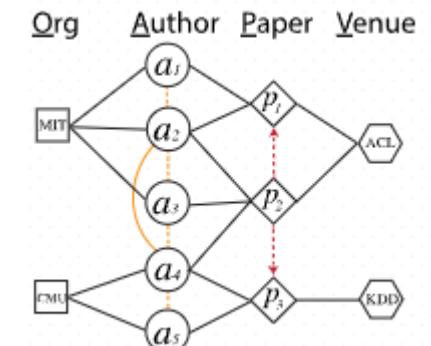
$$V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots V_t \xrightarrow{R_t} V_{t+1} \cdots \xrightarrow{R_{l-1}} V_l$$

- The transition probability at step  $i$  is defined as:

$$p(v^{i+1}|v_t^i, \mathcal{P}) = \begin{cases} \frac{1}{|N_{t+1}(v_t^i)|} & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) = t+1 \\ 0 & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) \neq t+1 \\ 0 & (v^{i+1}, v_t^i) \notin E \end{cases}$$

- Recursive guidance for random walkers, i.e.,

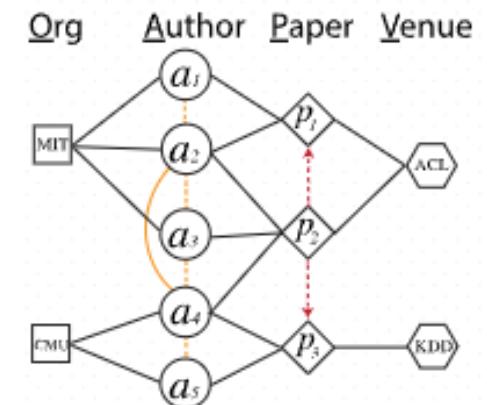
$$p(v^{i+1}|v_t^i) = p(v^{i+1}|v_1^i), \text{ if } t = l$$



- Metapath2Vec:
- Given a meta-path scheme (Example):

OAPVPAO

- In a traditional random walk procedure, the next step of a walker on node a<sub>4</sub> transitioned from node CMU can be all types of nodes surrounding it - a<sub>2</sub>, a<sub>3</sub>, a<sub>5</sub>, p<sub>2</sub>, p<sub>3</sub>, and CMU
- Under the meta-path scheme ‘OAPVPAO’, for example, the walker is biased towards paper nodes (P) given its previous step on an organization node CMU (O), following the semantics of this meta-path



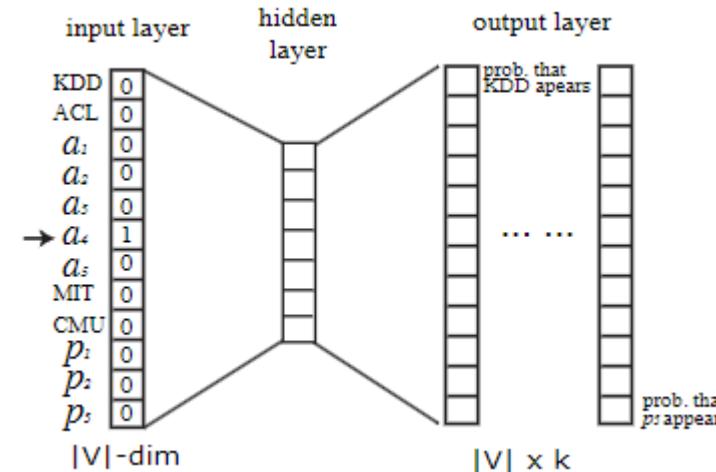
➤ Softmax in Metapath2Vec

$$p(c_t|v; \theta) = \frac{e^{x_{ct} \cdot x_v}}{\sum_{u \in V} e^{x_u \cdot x_v}},$$

→ Not consider node type

➤ The potential issue of skip-gram for heterogeneous network embedding:

- To predict the context node  $c_t$  (type t) given a node v, metapath2vec encourages all types of nodes to appear in this context position



- **Metapath2Vec++: Heterogeneous Skip-Gram**
- Objective function (heterogeneous negative sampling):

$$O(\mathbf{X}) = \log \sigma(X_{c_t} \cdot X_v) + \sum_{m=1}^M \mathbb{E}_{u_t^m \sim P_t(u_t)} [\log \sigma(-X_{u_t^m} \cdot X_v)]$$

- Softmax in Metapath2Vec++

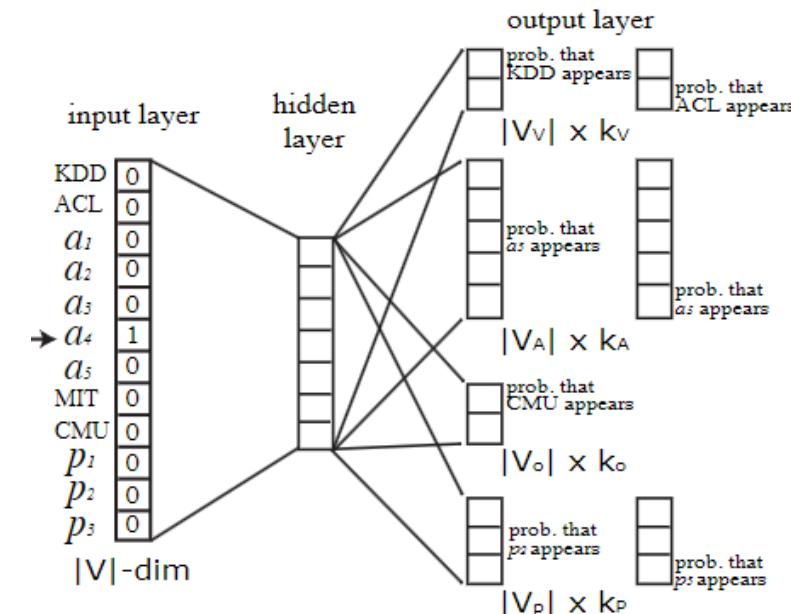
$$p(c_t|v; \theta) = \frac{e^{X_{c_t} \cdot X_v}}{\sum_{u_t \in V_t} e^{X_{u_t} \cdot X_v}}$$

→ Consider node type t

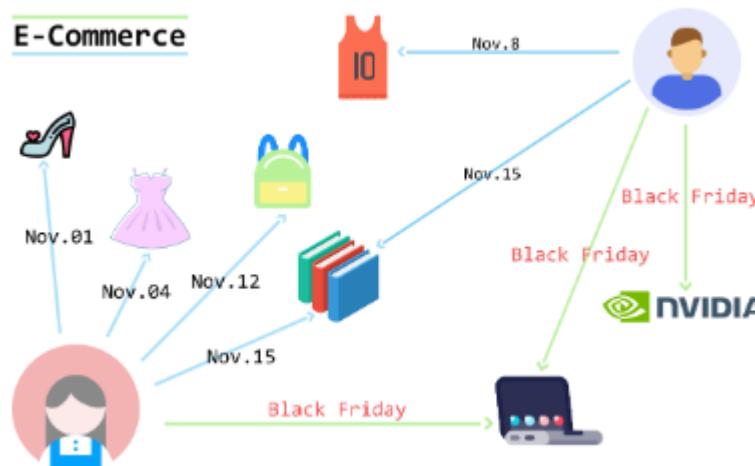
- Stochastic gradient descent

$$\frac{\partial O(\mathbf{X})}{\partial X_{u_t^m}} = (\sigma(X_{u_t^m} \cdot X_v - \mathbb{I}_{c_t}[u_t^m])) X_v$$

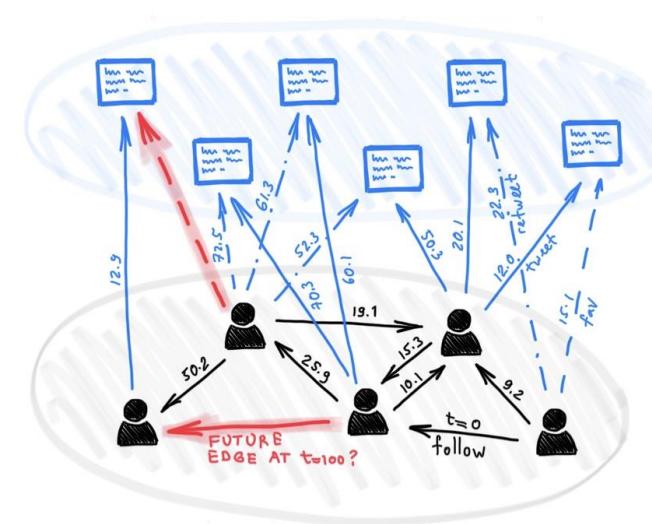
$$\frac{\partial O(\mathbf{X})}{\partial X_v} = \sum_{m=0}^M (\sigma(X_{u_t^m} \cdot X_v - \mathbb{I}_{c_t}[u_t^m])) X_{u_t^m}$$



- Previous random walk methods – Node2Vec, WalkLet, LINE, Struc2Vec, Role2Vec, Metapath2Vec, etc – are relied on static graph
- However, the networks in the real world are dynamic
  - Evolving over time



An illustration of user-item graph.

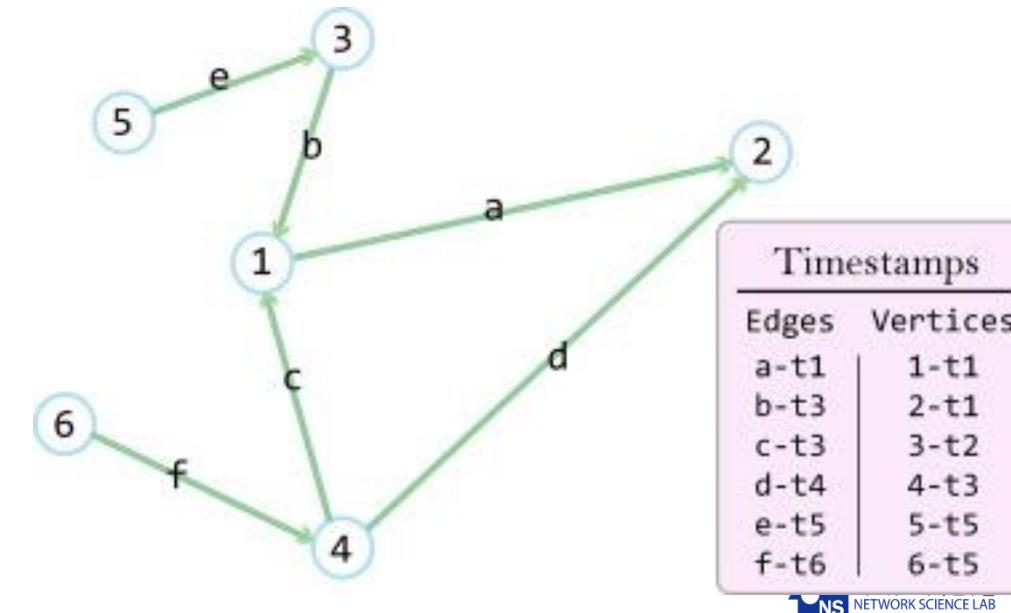
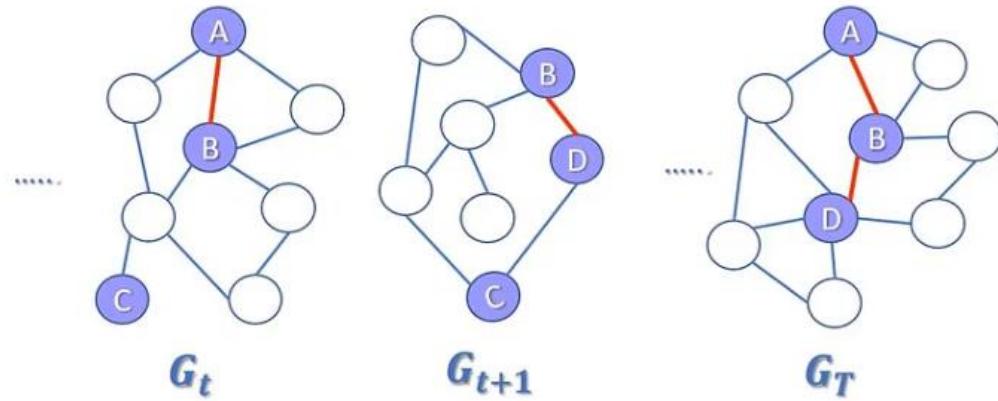


An illustration of social graph.

- **Problem:** Learning dynamic node representations
- **Challenges:**
  - Time-varying graph structures: links and node can emerge and disappear; communities are changing all the time
    - Requires the node representations capture both **structural proximity** (as in static cases) and their **temporal evolution**
    - Time intervals of events are uneven
  - Causes of the change: can come from different aspects, e.g.in co-authorship network, research community & career stage perspectives
    - Requires modeling multi-faceted variations

➤ **2 ways to model dynamic:** Discrete model and continuous model

- Discrete model: Sequence of network snapshots within a given time interval
  - $G = \{G_1, \dots, G_T\}$ , where  $T$  is the number of snapshots. Each snapshot  $G_t = (V_t, E_t)$  is a static network recorded at time  $t$
- Continuous model: a network with edges and nodes annotated with timestamps
  - We have  $G = (V_T, E_T, \mathfrak{T})$  where  $\mathfrak{T} : V, E \rightarrow \mathbb{R}^+$  is a function that maps each edge and node to a corresponding timestamp

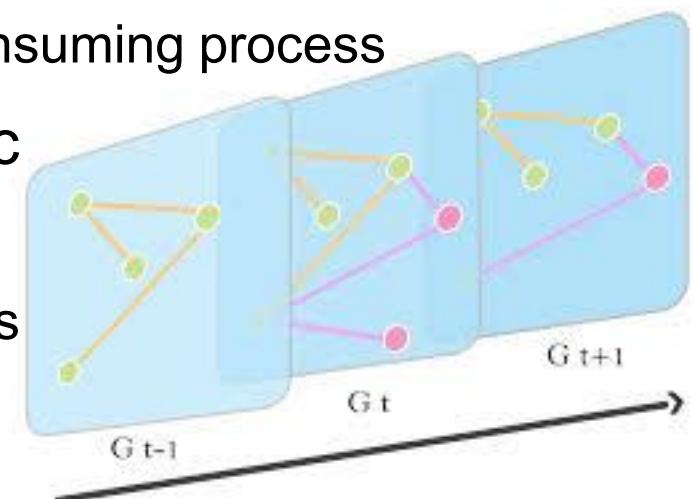


➤ **Motivation:** Networks in the real world are always evolving

- New users (new nodes) in social networks, new citations (new edges) in citation networks
- Users may delete friends (delete edges) or some users may leave the network (delete nodes)
- The static graph embedding methods are not capable of dealing with the critical challenge involved in dynamic networks
  - **Disadvantage:** Embedding vectors for each timestamp are in different spaces
    - Leading to learn embedding vectors separately is a time-consuming process

➤ **Solution:** dynnode2vec method to modify the node2vec method

- Employing the previous learned embedding vectors as initials weights for the skip-gram model



- Given a dynamic graph as a sequence  $G_1, G_2, \dots, G_T$  from timestamp 1 to  $T$
- Each graph at time  $t$  is defined as  $G_t = (V_t, E_t)$
- **Evolving Random Walk Generation:**

- Only generates random walks for the set of "evolving nodes" ( $\Delta V_t$ ) that have changed between consecutive timestamps  $t$  and  $t+1$ :

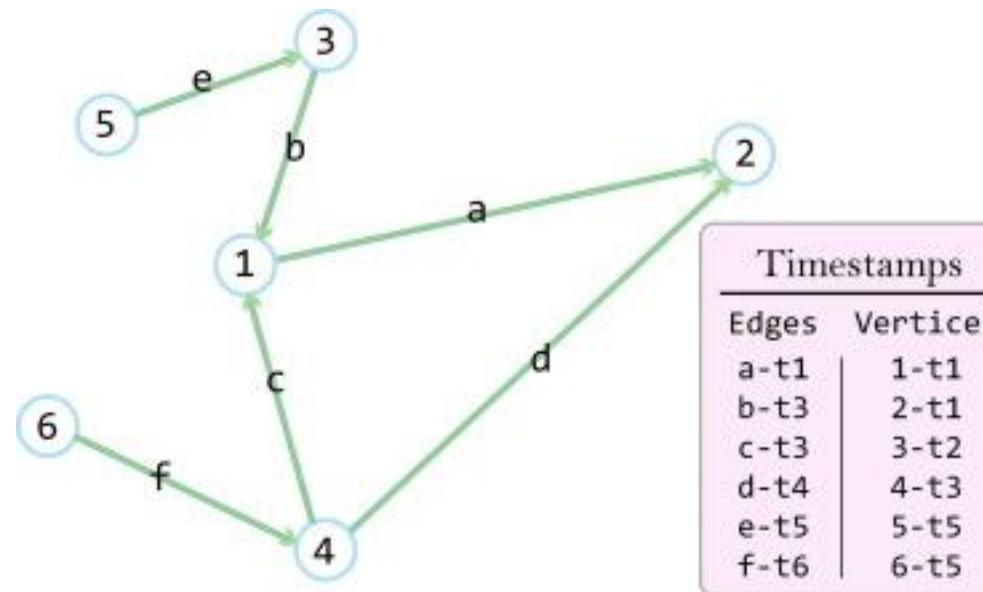
$$\Delta V_t = V_{add} \cup \{v_i \in V_t \mid \exists e_i = (v_i, v_j) \in (E_{add} \cup E_{del})\}$$

- Change: new nodes added, existing nodes deleted, or edges added/removed for existing nodes)

- Dynamic Skip-gram Model:

- Initializes the skip-gram model at timestamp  $t$  with the pre-trained embedding vectors from the previous timestamp  $t-1$
- Vocabulary is updated based on the new evolving random walks, and Skip-gram  $t$  is retrained using only the new evolving random walks on the evolving node set  $\Delta V_t$

- **Motivation:** Traditional methods often treat dynamic network as a sequence of static snapshots, which loss important temporal information
- **Solution:** Treat network as a continuous-time dynamic network, capturing the exact times of interaction

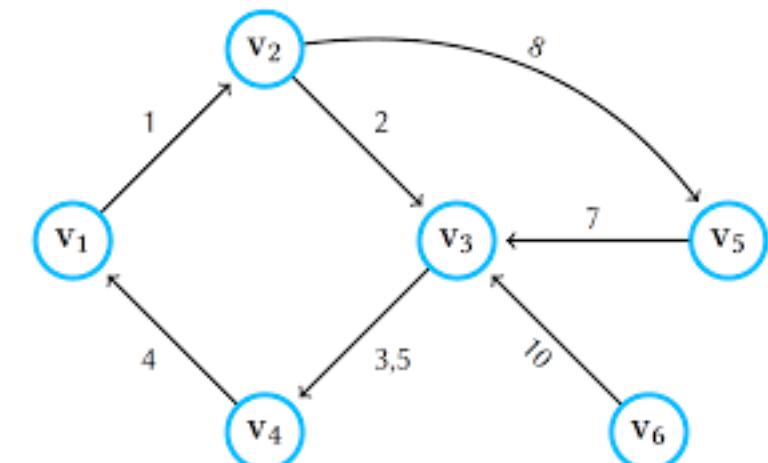


➤ A continuous-time dynamic network is defined as  $G = (V, E_T, \mathfrak{I})$

- $V$ : set of nodes
- $E_T \subseteq V \times V \times R^+$ : set of temporal edges between vertices in  $V$
- $\mathfrak{I}: E \rightarrow R^+$ : a function that maps each edge to a corresponding timestamp

➤ Temporal walk:

- A temporal walk from  $v_1$  to  $v_k$  in  $G$  is a sequence of nodes  $\langle v_1, v_2, \dots, v_k \rangle$  such that
  - $\langle v_i, v_{i+1} \rangle$  for  $1 \leq i < k$  and
  - $\mathfrak{I}(v_i, v_{i+1}) \leq \mathfrak{I}(v_{i+1}, v_{i+2})$  for  $1 \leq i < (k - 1)$



- **Temporal random walk:**
- The set of temporal neighbours of a node  $v$  at time  $t$ :

$$\Gamma_t(v) = \{(w, t') \mid e = (v, w, t') \in E_T \wedge \tau(e) > t\}$$

- Unbiased selection: each temporal neighbour  $w$  of node  $v$  at time  $t$  is selected

$$\Pr(w) = 1/|\Gamma_t(v)|$$

- Biased selection: sampling the next node in a temporal walk via temporally weighted distribution based on

$$\Pr(w) = \frac{\exp[\tau(w) - \tau(v)]}{\sum_{w' \in \Gamma_t(v)} \exp[\tau(w') - \tau(v)]}$$

Exponential decay: selecting a neighbour decreases exponentially with time

$$\Pr(w) = \frac{\delta(w)}{\sum_{w' \in \Gamma_t(v)} \delta(w')}$$

Linear decay: sorts temporal neighbours in descending order time-wise

- Temporal context windows: To handle the temporal nature of walks - a walk to run out of temporally valid edges to traverse
  - A walk must have a minimum length  $\omega$  and can extend up to a maximum length L. The number of context windows is defined

$$\beta = \sum_{i=1}^k |S_{t_i}| - \omega + 1$$

- Learning time-preserving embeddings: maximize the likelihood of observing temporal context windows given the embeddings

$$\max_f \log \Pr(W_T = \{v_{i-\omega}, \dots, v_{i+\omega}\} \setminus v_i \mid f(v_i))$$

→ Utilizing stochastic gradient descent



네트워크 과학 연구실  
NETWORK SCIENCE LAB



가톨릭대학교  
THE CATHOLIC UNIVERSITY OF KOREA

