POLITECNICO
MILANO 1863

PhD in Data Analytics and Decision Sciences

Advanced Statistical Learning for Complex Data

29th May 2024

Graph Theory and Complex Networks

Michela Carlotta Massi

Postdoc Researcher

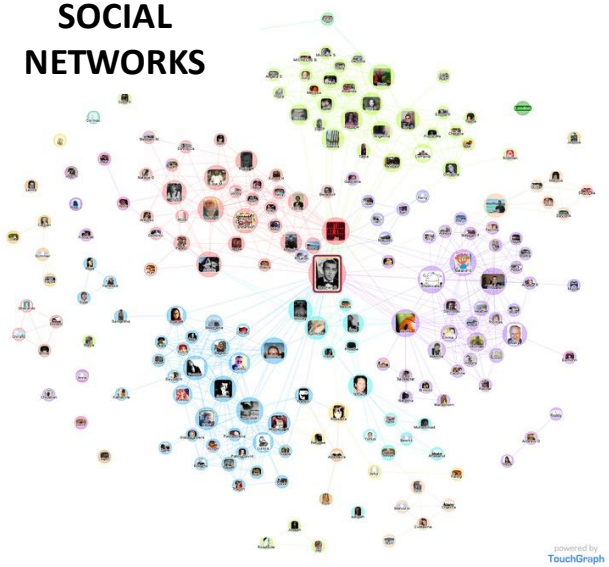
Health Data Science Centre, Human Technopole

PhD DADS

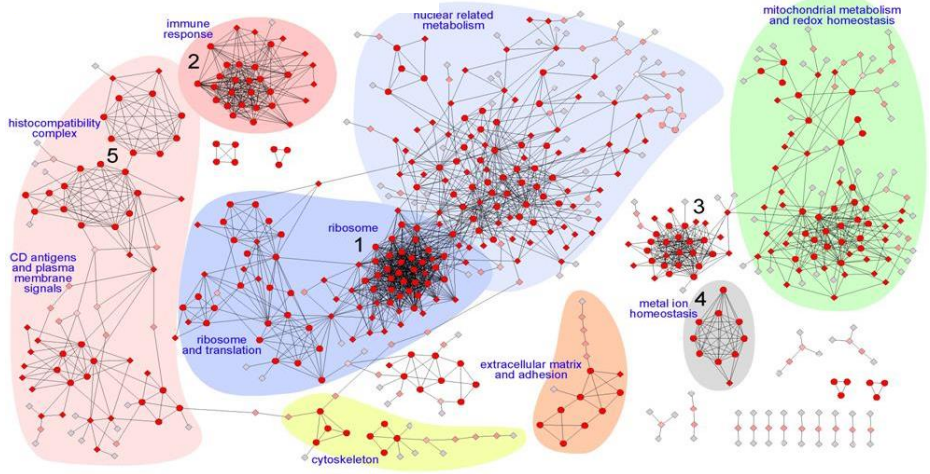
Why Networks?

Networks are a general language to represent **data that describes *complex systems***.

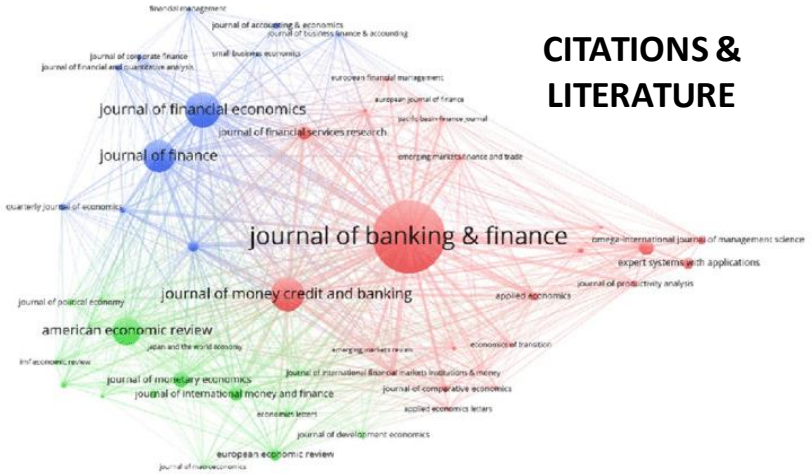
SOCIAL NETWORKS



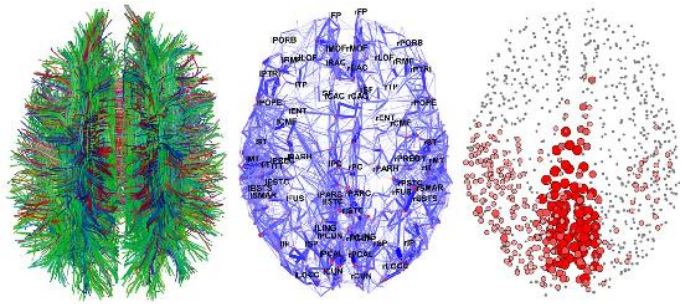
GENE EXPRESSION & FUNCTIONS



CITATIONS & LITERATURE

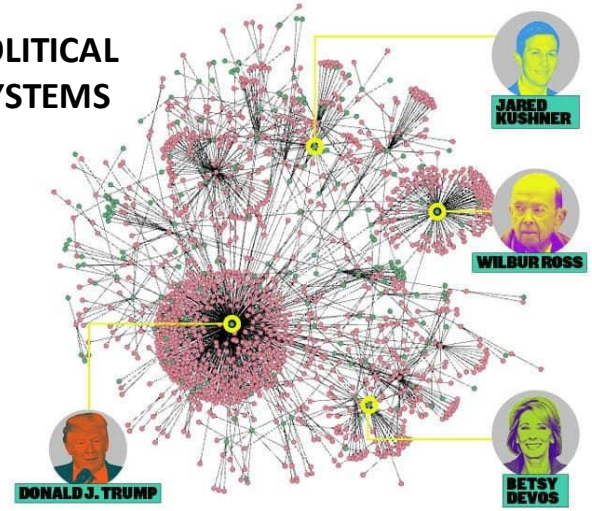


BRAIN CONNECTOME



ROADS & TRANSPORTATION SYSTEMS

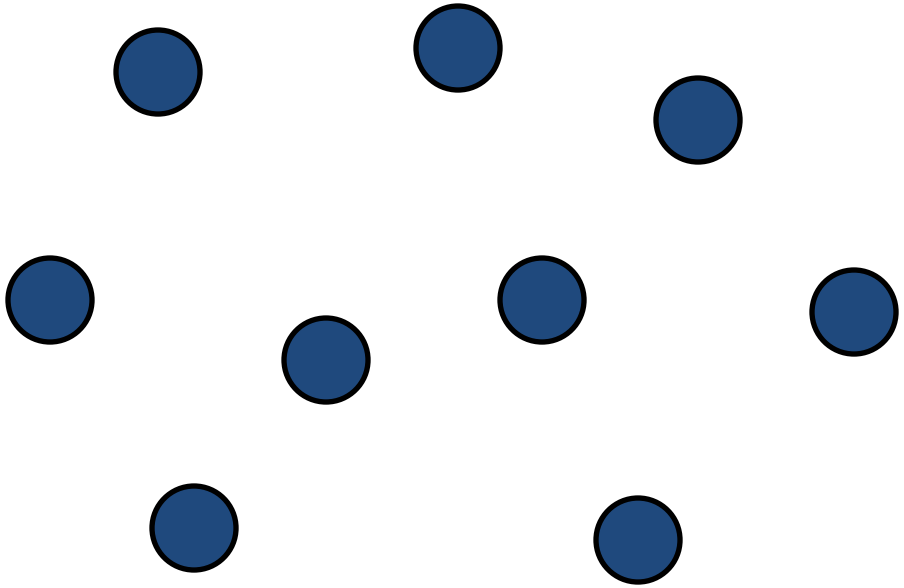
POLITICAL SYSTEMS

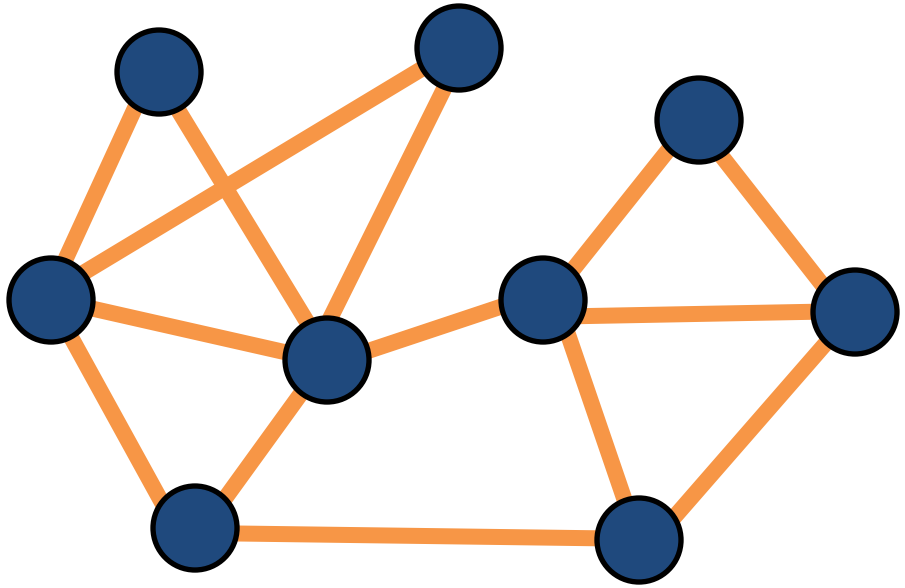


Why Networks?

Networks are a general language to represent **data that describes *complex systems***.

They are useful as they can represent **components** (people, genes, geographical locations, etc.) and their **relationships**.

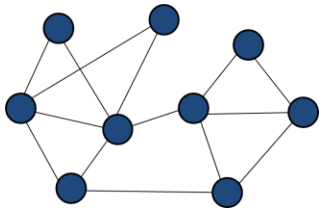




High-level outline of this lecture

PART I - Introduction to Graph Theory

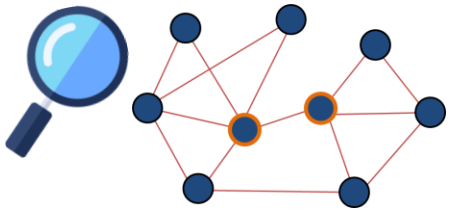
- Notions and definitions
- Graph Data Representation

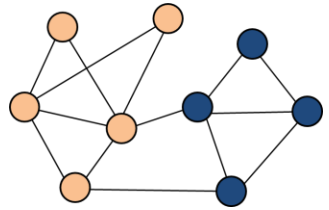


	v	u	w
v	0	1	0
u	0	0	1
w	1	0	0

PART II - Analyzing complex networks

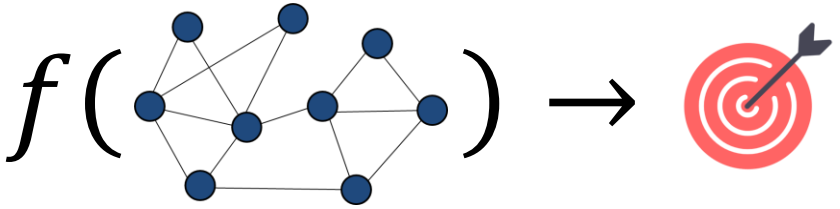
- Network characteristics
- Graph Clustering / Network Community Detection





PART III – Exploiting Graphs as input data

- Graph-based Statistical/Machine Learning Tasks
- Graph Embeddings
- Graph Neural Networks



Introduction to Graph Theory

Introduction to Graph Theory: notions and definitions

Undirected simple graphs

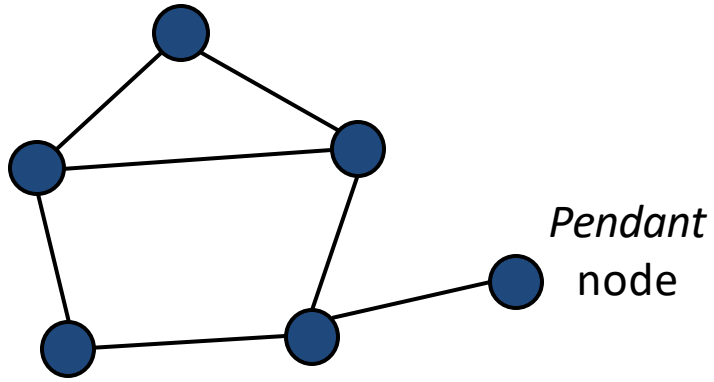
A graph is an ordered pair $G = (V, E)$ comprising

- A set of **vertices** (also called nodes, or points), V
- A set of **edges** (also called links, or lines) $E \subseteq \{\{u, v\} \mid u, v \in V \text{ and } u \neq v\}$
  which are unordered pairs of vertices (i.e. an edge is associated with two distinct vertices)

V is often assumed to be non-empty, while E can be an empty set.

→ This definition corresponds to an **undirected *simple* graph**.
A simple graph is a graph with **no loop** edges or **multiple edges**.

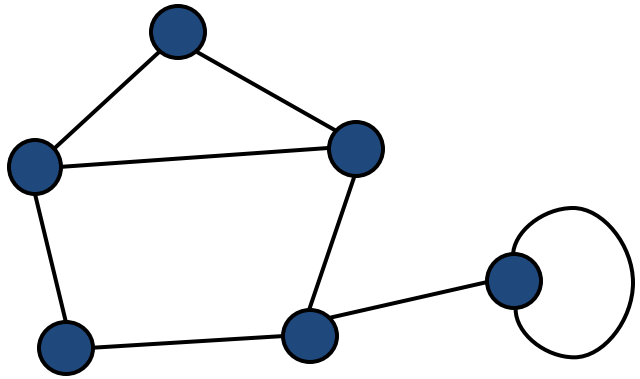
u and v are **adjacent** if $\{u, v\}$ is an edge,
and the edge is called **incident** with u and v.
If $\{u, v\} \in E$, then v is a neighbor of u.



Other graph types

Loop: A loop is an edge whose endpoints are equal i.e., an edge joining a vertex to it self is called a loop.

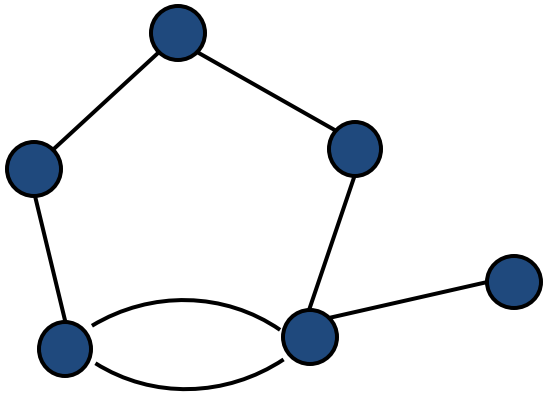
Represented as {u, u} = {u}



If we allow loops we obtain pseudographs.

Multiple Edges: Two or more edges joining the same pair of vertices.

If we allow multi-sets of edges, i.e. multiple edges between two vertices, we obtain a multigraph.



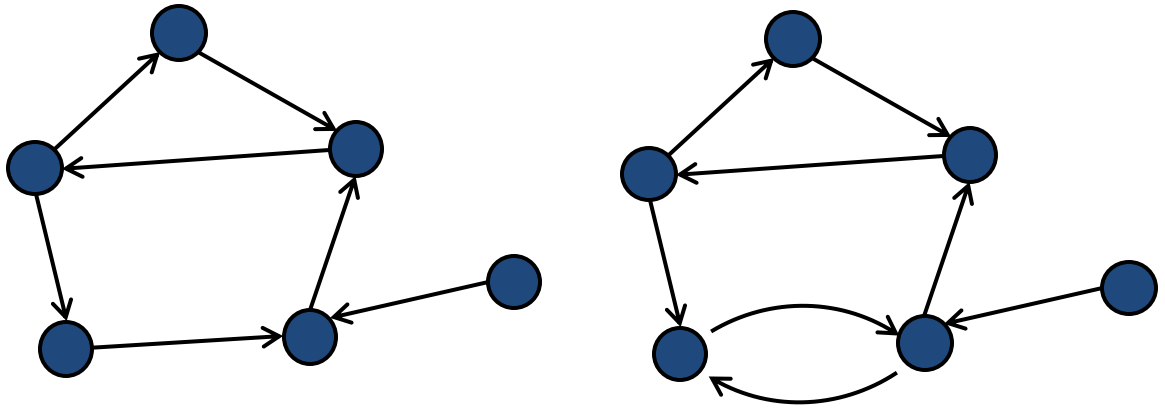
Other graph types

In directed graphs (or digraphs), edges have direction

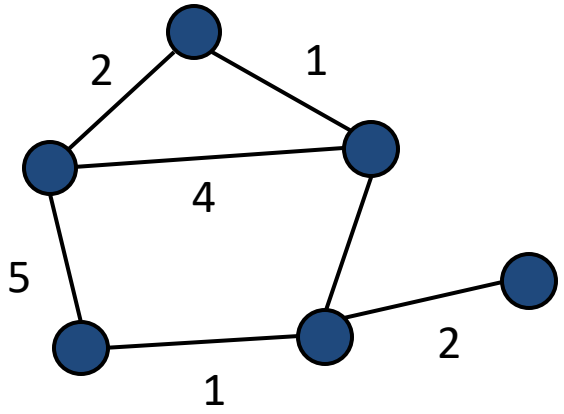
$\{u, v\} \in E$ implies $u \to v$

Thus, $\{u, v\} \in E$ does not imply $\{v, u\} \in E$.

Call u the source and v the destination



A weighted graph is a graph where each edge is associated with a weight w.



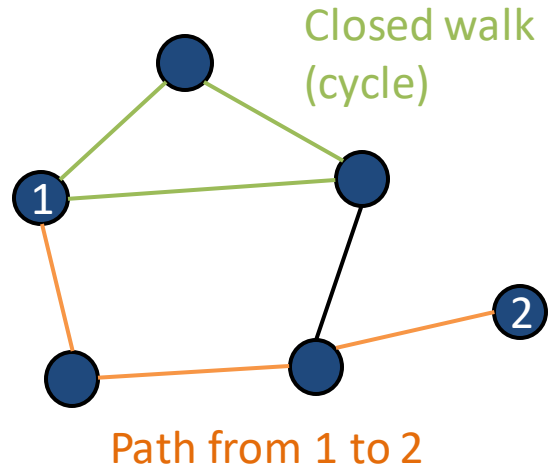
Walks, paths and distances

A **walk** is a sequence of vertices $v_1, v_2, \dots, v_n$ such that $\forall i \in \{1,2,\dots,N-1\}$, $v_i \sim v_{i+1}$ (adjacent).

A **path** is a walk where $v_i \neq v_j, \forall\, i \neq j$.
In other words, a path is a walk that visits each vertex at most once.

A **closed walk** is a walk where $v_i = v_n$.
A cycle is a closed path.
A graph is connected if there exists a path between each pair of vertices.

A graph is called **connected** if any two vertices u and v are connected by a walk.



A **distance** $\boldsymbol{dist(v,u)}$ between two vertices u and v of a connected graph is *the length of the shortest path* connecting them.

For a connected graph G:
$E(v) = max_{x \in V(G)} dist(v,x)$ is the eccentricity of v in G;
$D(v) = max_{v \in V(G)} E(v)$ the diameter of a G;
$R(v) = min_{v \in V(G)} E(v)$ the radius of G .

Graph Representations (data structures)

Adjacency matrix

An adjacency matrix is a $N \times N$ (where $N = |V|$) binary matrix $\boldsymbol{A}$.

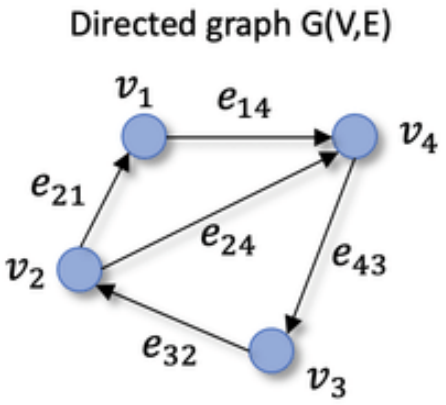
Where

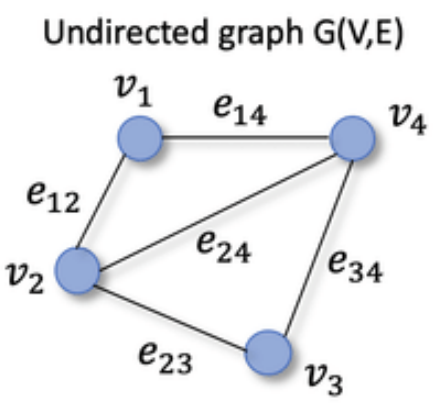
Undirected Graph

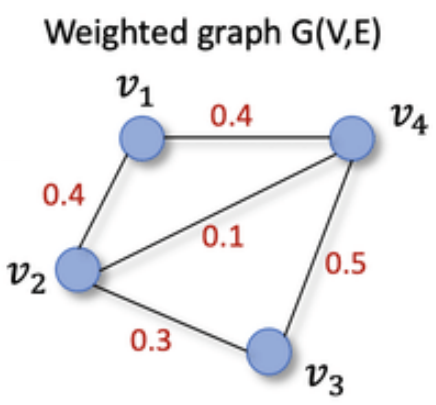
$$a_{ij} = \begin{cases} 1 \text{ if } \{v_i, v_j\} \text{ is an edge of } G \\ 0 \text{ otherwise} \end{cases}$$

Directed Graph

$$a_{ij} = \begin{cases} 1 \text{ if } (v_i, v_j) \text{ is an edge of } G \\ 0 \text{ otherwise} \end{cases}$$







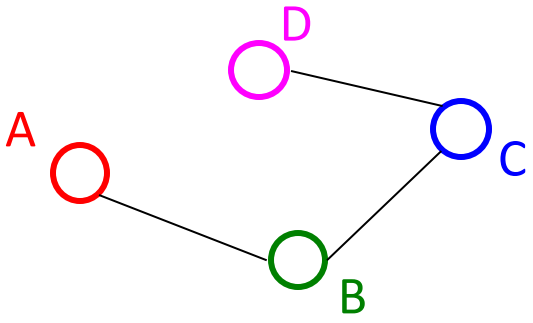
	v_1	v_2	v_3	v_4
v_1	0	0	0	1
v_2	1	0	0	1
v_3	0	1	0	0
v_4	0	0	1	0

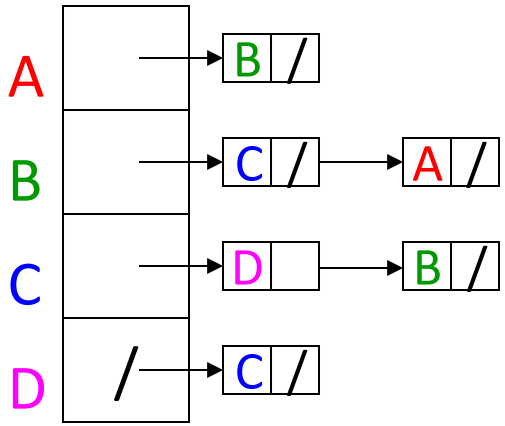
	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	1
v_3	0	1	0	1
v_4	1	1	1	0

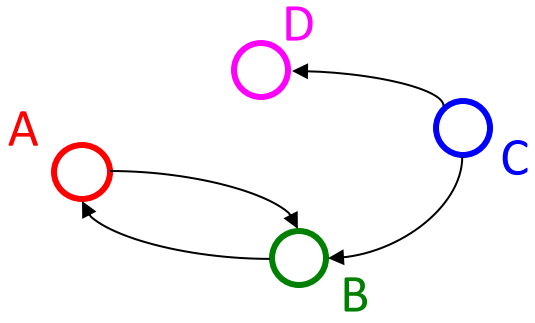
	v_1	v_2	v_3	v_4
v_1	0	0.4	0	0.4
v_2	0.4	0	0.3	0.1
v_3	0	0.3	0	0.5
v_4	0.4	0.1	0.5	0

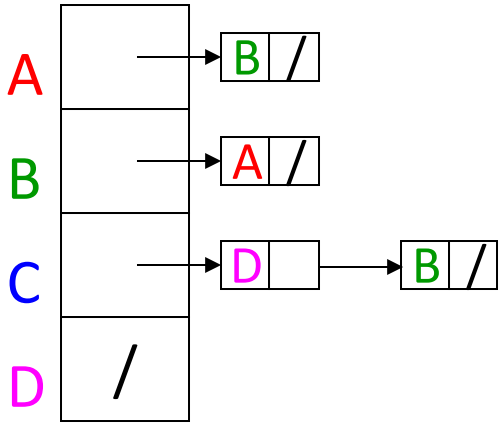
Adjacency list

An array of length $|V|$ in which each entry stores a list of all adjacent vertices (e.g., linked list)









NB: This is a very useful representation in case we have sparse Adjacency matrices, i.e. it reduces memory requirements

Graph Representations (data structures)

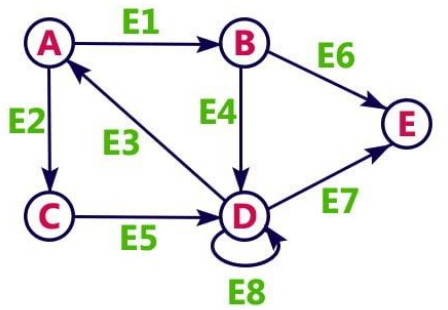
Incidence Matrix

G = (V, E) be an undetected graph. Suppose that $v_1, v_2, v_3, \dots, v_n$ are the vertices and $e_1, e_2, \dots, e_m$ are the edges of G. Then the **incidence matrix** with respect to this ordering of V and E is the $n \times m$ matrix $M = [mij]$, where

$$m_{ij} = \begin{cases} 1 & \text{when } e_j \text{ is incident with } v_j \\ 0 & \text{otherwise} \end{cases}$$

Which has n rows (corresponding to n-vertices) and m e columns (corresponding to the edges)

Can also be used to represent :
Multiple edges: by using columns with identical entries, since these edges are incident with the same pair of vertices
Loops: by using a column with exactly one entry equal to 1, corresponding to the vertex that is incident with the loop



	E1	E2	E3	E4	E5	E6	E7	E8
A	1	1	-1	0	0	0	0	0
B	-1	0	0	1	0	1	0	0
C	0	-1	0	0	1	0	0	0
D	0	0	1	-1	-1	0	1	1
E	0	0	0	0	0	-1	-1	0

A quick note on Building Graphs and Networks

Sometimes data is naturally graph-structured, i.e. you have clear **entities** and **relationships**

- Social media users and their connections (friends, followers, …);
- Geographical locations and their connecting transportation system; …

However…

Most networks are not explicitly and unambiguously defined.

For example,

- **Biological networks** often measure correlations between the biological processes of cells, genes, or proteins.
- **Human contact and social networks** are often inferred from repeated interactions.
- **Brain interactome** networks can vary according to granularity of the observations and meaning of the connections among them.

…Moreover,

You might want to represent your data as a network only to enrich the effectiveness of your downstream classification/regression model (i.e. performing feature selection, accounting for interaction terms in you classifiers, etc.).

A quick note on Building Graphs and Networks

*How to **construct** (or to estimate) their **intrinsic** yet **implicit graph structure?***

1. Define the entities: genes, users, words, cells, patients, …
2. Construct the connections between entities (edges)
3. Assign weights to edges

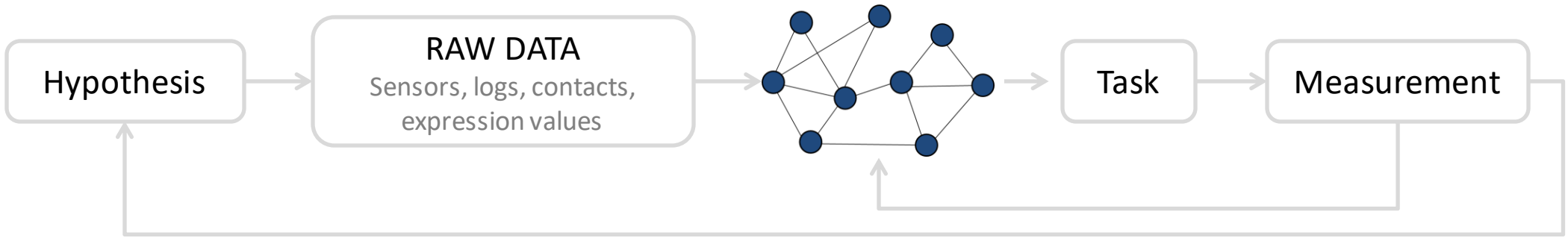
} These two may sometimes happen jointly

Network structure learning can be

Supervised: construct a network representation of your data that maximizes the effectiveness of a given downstream task.

"Is this network a useful representations for measuring and studying the behavior of the system it models?"

i.e.: feature clustering, network clustering, classification, feature selection, …



→ **Unsupervised**: only analyze original features and their inter-relationships

A quick note on Building Graphs and Networks

Unsupervised Graph Structure Learning

(dis)similarity + threshold methods: choose a metric to estimate similarity between entities, and set a lower-bound threshold

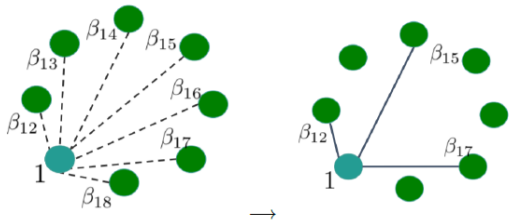
- **co-occurrence**-based association measures
  common in text-mining, and all networks from Boolean/categorical data
- **correlation**-based association measures, i.e. Pearson correlation coefficient (mostly for linear interactions among nodes)
- **Information theory-based association measures**, i.e. Mutual Information, Pairwise Mutual Information

(dis)similarity + clustering methods

- **k-nearest neighbor**: define a metric and for each entity define links as the k closest other entities

Sparsity-based methods:

- **Graphical LASSO**: perform penalized (L1) regression with each node as target and all others as predictors.



Model-based approaches:

- **Bayesian Network Learning**: estimated Directed Acyclic Graphs (DAG) from data, considering conditional dependencies between features

A very large review and many more examples here [1]:

Qiao, Lishan, et al. "Data-driven graph construction and graph learning: A review." *Neurocomputing* 312 (2018): 336-351.

Characterizing & Analyzing Networks

Node Characteristics

Global Structure Characteristics

Graph Clustering: discovering communities

Network Topological Properties

A network can be an exceedingly complex structure, as the connections among the nodes can exhibit complicated patterns.

One challenge in studying complex networks is to develop *simplified measures that capture some elements of the structure* in an understandable way.

We can distinguish two types of **topological properties** when analyzing graphs:

- Properties that describe the **role of single nodes**
- Global properties that describe of the **whole graph structure**

Node characteristics: centrality measures

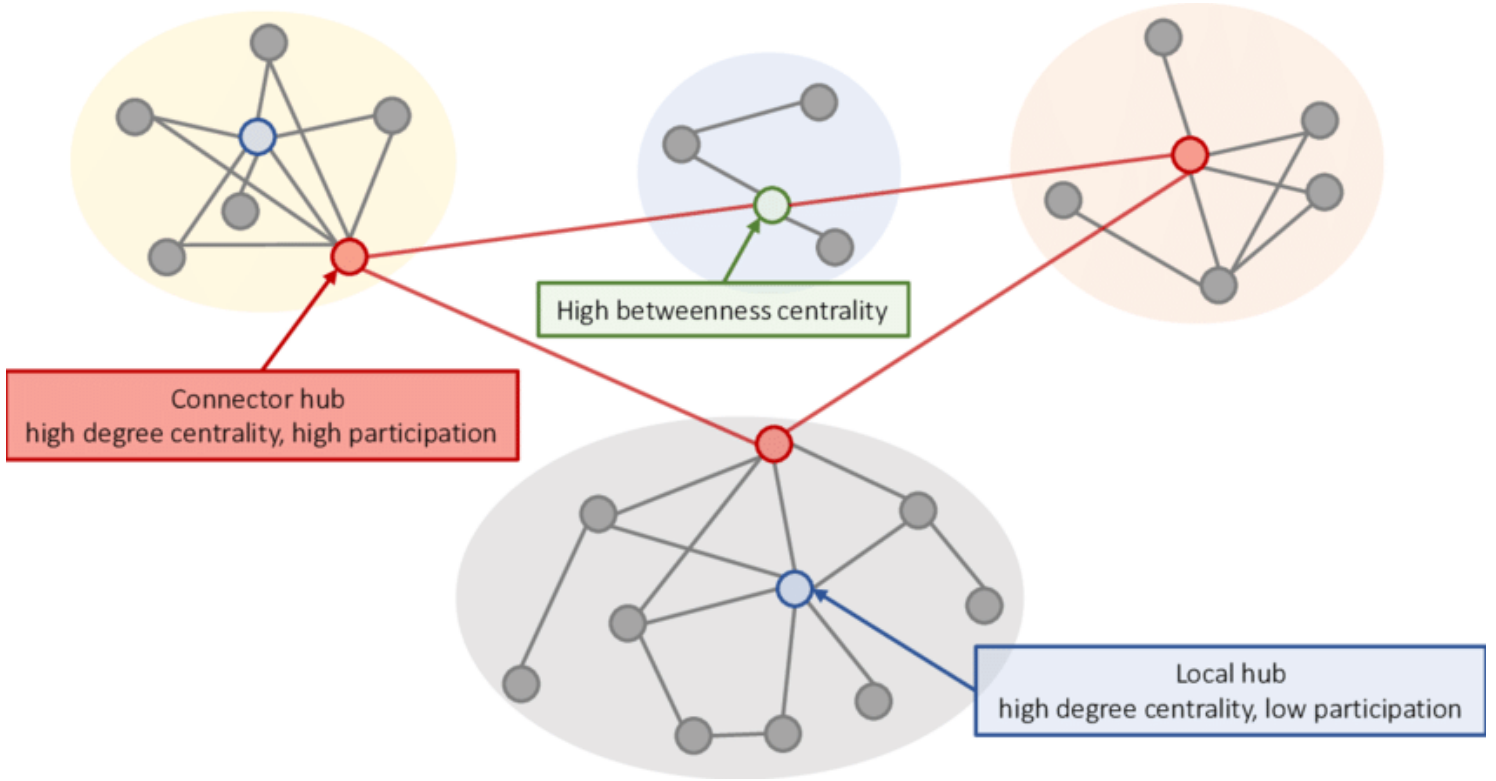
"What characterizes an important vertex?"

The answer is given in terms of a real-valued function on the vertices of a graph

→ The values produced are expected to provide a **ranking** which identifies the **most important nodes**.

Why should we care?

- **Feature selection**
- **Hub nodes** and **influential spreaders** identification
  - Influential subjects in social networks
  - Disease spreaders (epidemic analysis),
  - Web/softwars vulnerability analysis
  - Power grid protection,
  - Discovery of candidate drug targets
  - Essential genes/proteins identification
  - Discovery of important species



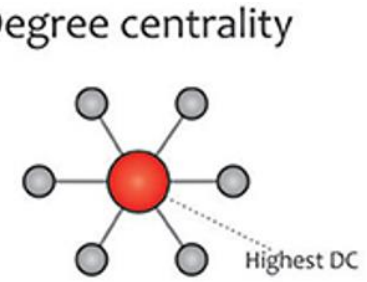
Node characteristics: degree centrality

Degree centrality is **equal to the number of node neighbors**.

Thus the more neighbors a node have the more it's central and highly connected, thus have an influence on the graph.

NB: node degree as a measure doesn't take neighbors connectivity in consideration.

For a graph G, with adjacency matrix A and N nodes, degree centrality can be computed as:



$$k_i = \sum_{j=1}^{N} A_{ij}$$

For directed graphs we can distinguish

In-Degree of a vertex: number of in-bound edges (edges where the vertex is the destination)

Out-Degree of a vertex: number of out-bound edges (edges where the vertex is the source)

The **Maximum degree** of a graph G is the maximum of the degrees of its vertices, often denoted $\Delta(G)$;

The **Minimum degree** of G is the minimum of its vertex degrees, often denoted $\delta(G)$.

The **Total degree** of G is the sum of the degrees of all vertices

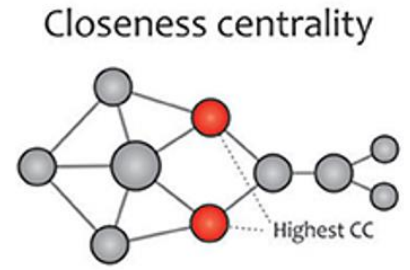
Node characteristics: closeness/betweeness centrality

Closeness centrality, is defined in terms of the average distance of each node to all others.

The normalized closeness centrality (or closeness) of a node is the average length of the shortest path between the node and all other nodes in the graph. Thus the more central a node is, the closer it is to all other nodes.

$$C_C(u)\frac{1}{\sum_v d(u,v)}$$

Where $d(u,v)$ is the shortest path between u and v.



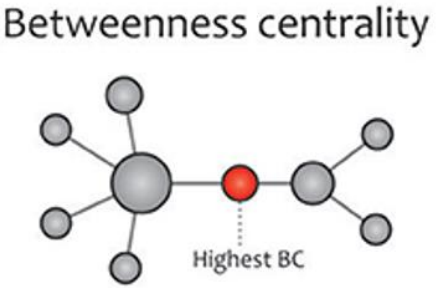
Betweeness centrality quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.

It was introduced as a measure for quantifying the control of a human on the communication between other humans in a social network (L. Freeman)

The betweenness of a vertex s in a graph G:=(V,E) with V vertices is computed as follows:

- For each pair of vertices (u,v), compute the shortest paths between them.
- For each pair of vertices (u,v), determine the fraction of shortest paths that pass through the vertex in question (here, vertex s).
- Sum this fraction over all pairs of vertices (u,v).

More compactly the betweenness can be represented as: $C_B(s) = \sum_{u \neq v \neq s \in V} \frac{|SP(u,v) \text{ that pass through } s|}{|SP(u,v)|}$

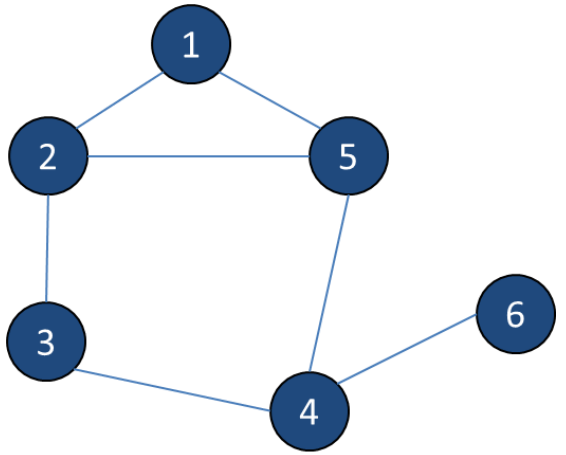


Global topological properties: Degree Distribution

Degree distribution: A frequency count of the occurrence of each degree

Node	Degree
1	2
2	3
3	2
4	3
5	3
6	1

Degree	Frequency
1	1/6
2	2/6
3	3/6



Average Degree: A frequency count of the occurrence of each degree
Let N = |V| be the number of nodes, and L = |E| be the number of edges:

$$\langle K \rangle = \frac{\sum_{i=1}^{n} deg(i)}{N} = \frac{2L}{N}$$

…Why should we care?

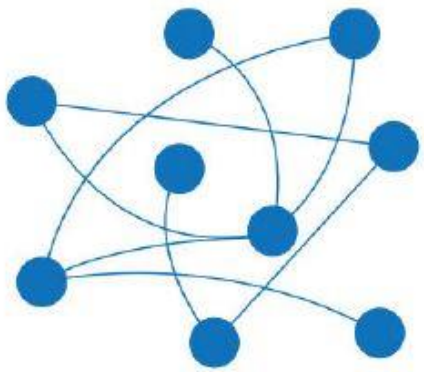
Global topological properties: Degree Distribution

Real world networks usually have very different degree distributions.

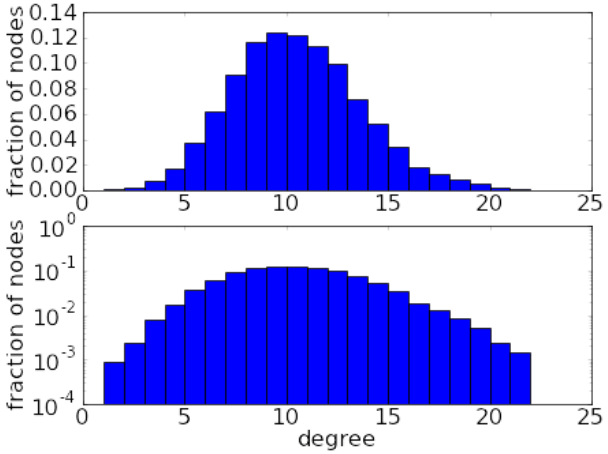
In a real world network, most nodes have a relatively small degree, but a few nodes will have very large degree, being connected to many other nodes.

→ These large-degree nodes are often referred to as *hubs*.

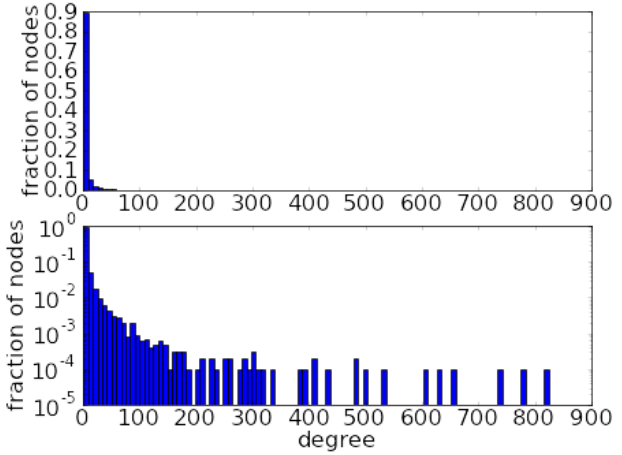
Binomial Degree Distribution

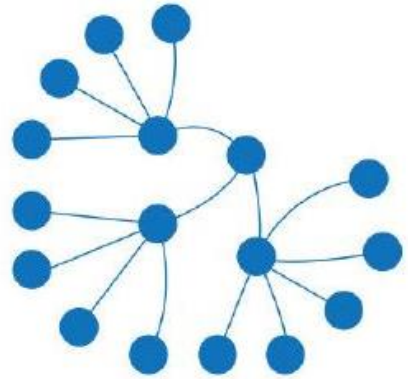


RANDOM NETWORK



Power-Law Degree Distribution





SCALE-FREE NETWORK

→ Studying the degree distribution gives an idea of the topological structure of the graph/network.

→ It can be exploited to compare networks

Network Structure Characteristics: Assortativity

Is there a tendency of nodes with the same magnitude of degree to connect to each other, or are large-degree nodes primarily connected to low-degree nodes?

This question is answered by the **assortativity (or assortative mixing) coefficient *r*** [2].

The *assortativity coefficient* is the Pearson correlation coefficient of degree between pairs of linked nodes.

$$r = \frac{\sum_{jk} jk(e_{jk} - q_j q_k)}{\sigma_q^2}$$

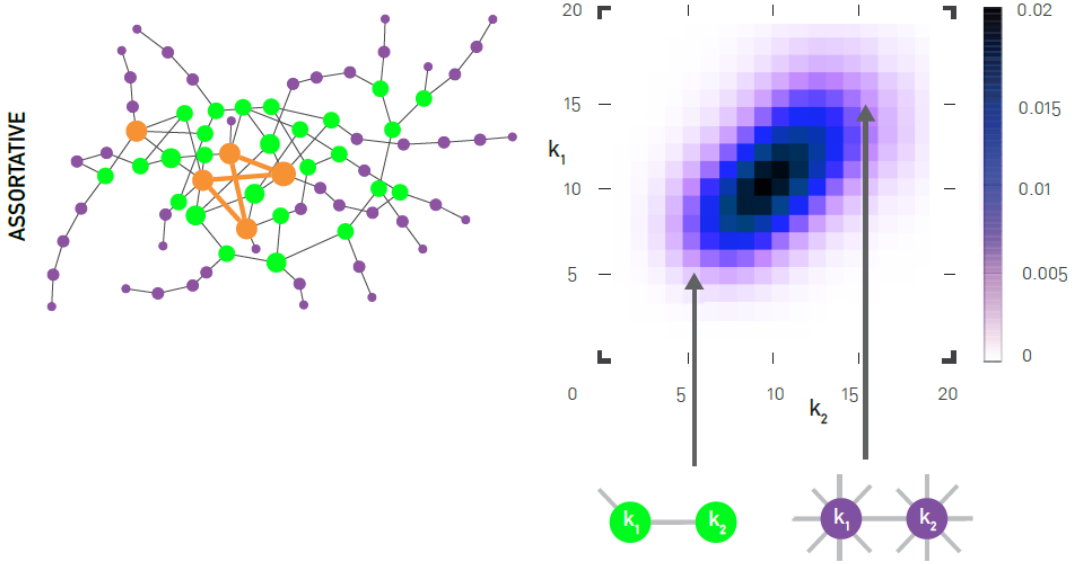
Where

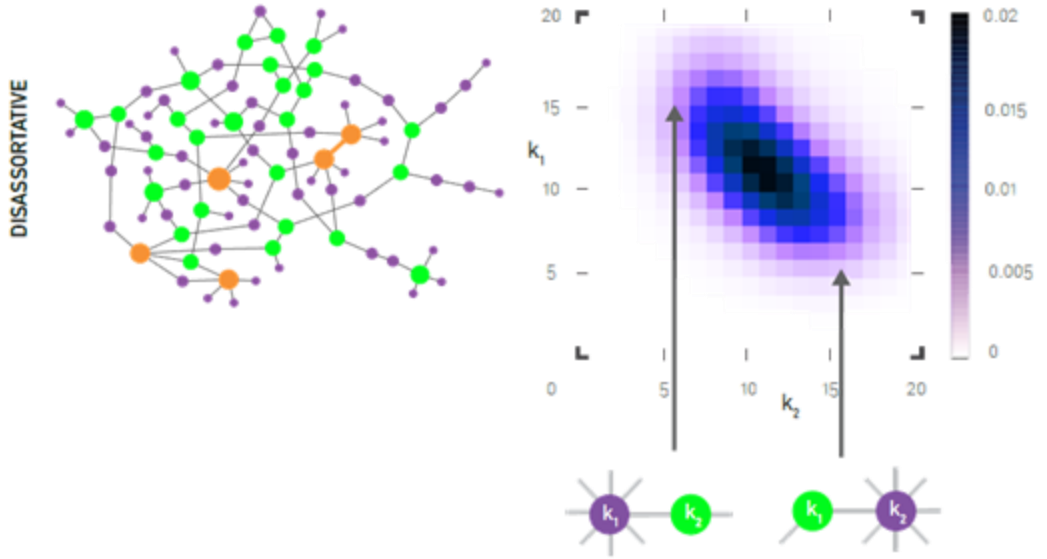
e_{jk} is the joint-excess degree probability for excess degree j and excess degree k (the excess degree, also known as remaining degree, of a node is equal to the degree of that node minus one), and

$$q_k = \frac{(k+1)p_{k+1}}{\sum_j jp_j}$$

is the normalized distribution of the excess degree of a randomly selected node.

The assortativity coefficient measures the extent to which a network can resist failures in its main components (i.e., its vertices and edges). Notably, communication between hubs in assortative networks leads to covering each other's activities when a particular hub crashes, but the performance in disassortative networks will drop sharply due to the presence of vulnerable hubs. [3]





Analyzing Graph Structure: community discovery

Real-life graphs are not random.

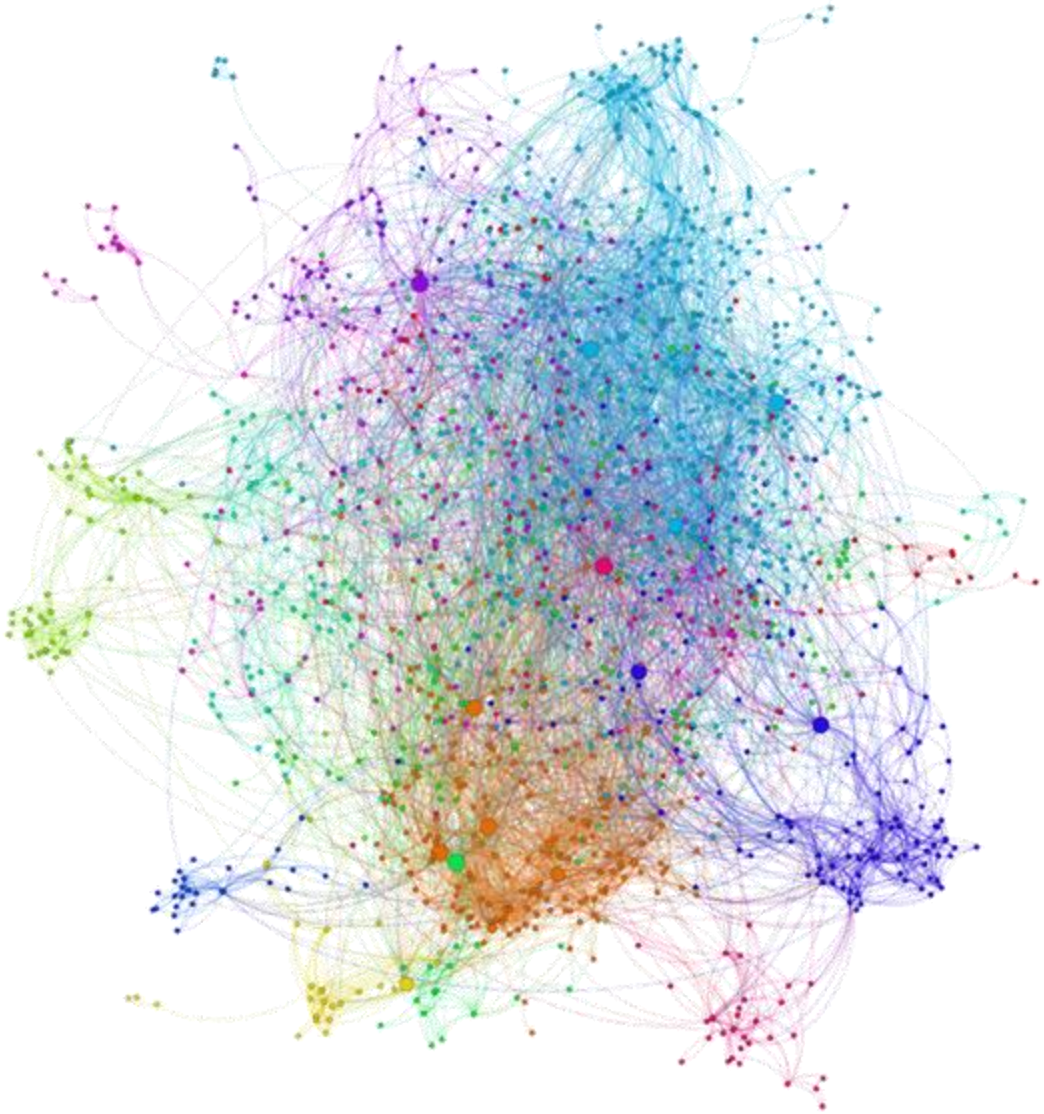
→ We expect nodes in graphs to be somehow internally grouped

In network/graph analysis, we call these groups **communities.**

For instance:

- in social networks, communities represent groups of individuals that are socially tight
- in a citation network among academic journals, communities might correspond to topics or disciplines
- communities of nodes in a web graph might indicate groups of related web pages
- communities of nodes in a metabolic network might indicate functional units within the network

→ Nodes in communities will be densely connected to each other, and sparsely connected with other communities.



Community Detection

In many ways **community detection** is just clustering on graphs.

→ We could apply traditional clustering algorithms on the adjacency matrix, such as k-means

→ We could define a distance or similarity measure between nodes in the graph and apply other algorithms such as hierarchical clustering.

….But there are also algorithms that are specific to graphs

Communities have several definitions, which lead to many different algorithms to identify them.

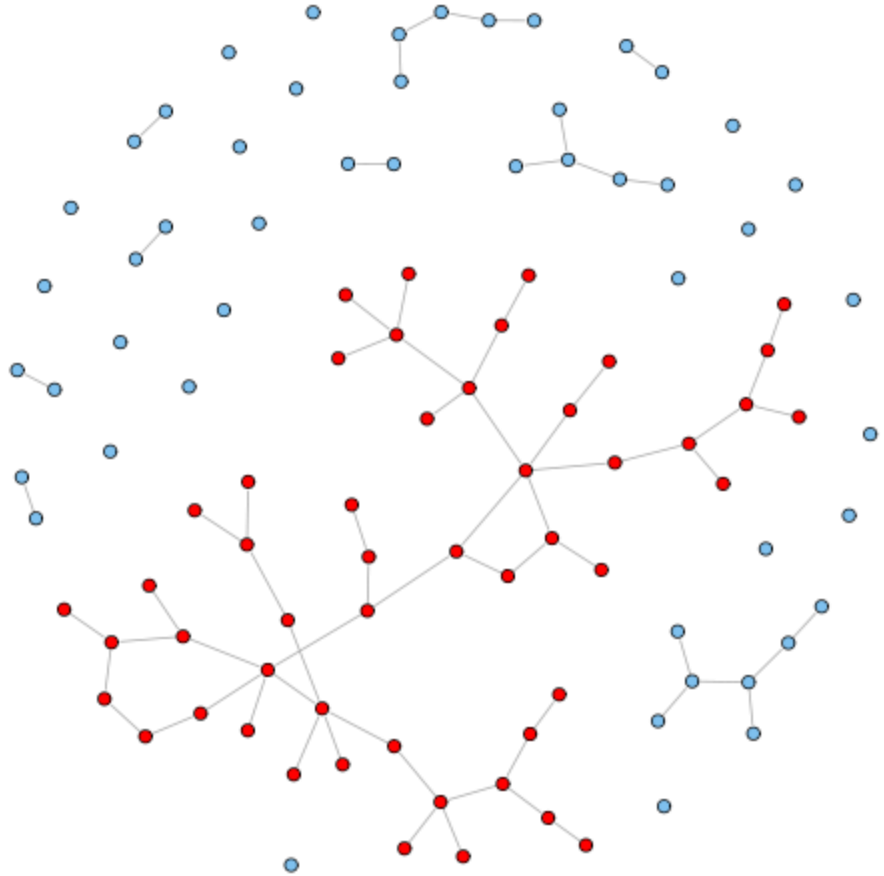
Community Detection: Connected Components

DEFINITION 1: Community members should be connected

→ connected components, i.e. sets of nodes which are reachable from each other

Very rough notion of community

Not useful for (most) real graphs: there is usually a “giant component” containing almost all nodes.



Community Detection: Path-based algorithms

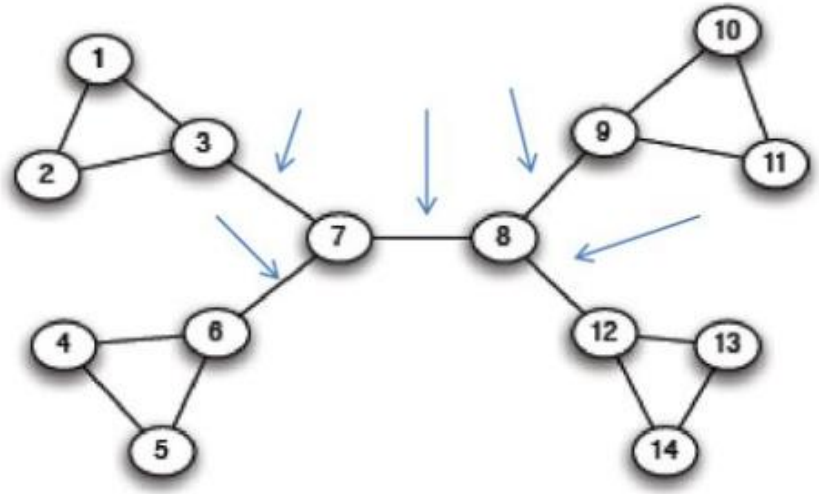
DEFINITION 2: Few edges between communities

→ **Hierarchical divisive method: the Girvan-Neuman Algorithm [4]**

1. Start with the whole graph

2. Find edges whose removal partitions the graph

3. Repeat with each subgraph until single vertices.



Community Detection: Path-based algorithms

DEFINITION 2: Few edges between communities

→ **Hierarchical divisive method: the Girvan-Neuman Algorithm [4]**

1. Start with the whole graph

2. **Find edges whose removal partitions the graph**

   we need a measure of how important an edge is in keeping the graph connected

   → **Edge Betweeness: number of shortest paths passing through that edge**

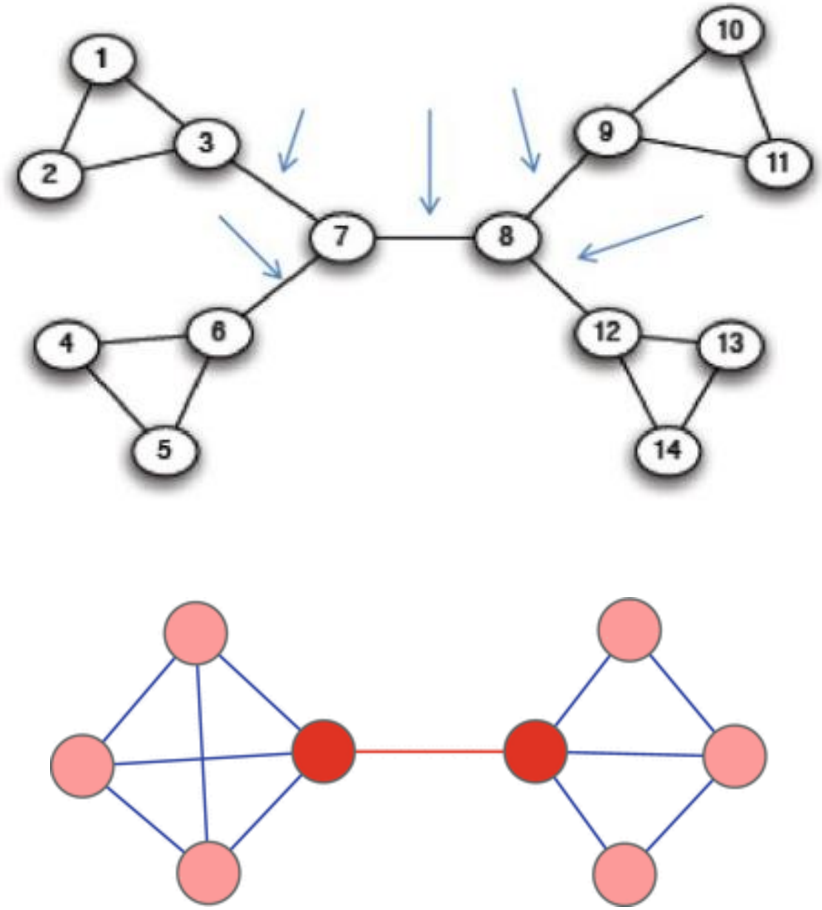
   For each node pair u, v there may be multiple shortest paths (SP)

$$B(e) = \sum_{u,v \in V} \frac{|SP(u,v) \text{ that include } e|}{|SP(u,v)|}$$

   A. Compute the **Edge Betweeness** for all edges

   B. Remove the edge with the highest betweeness

3. Repeat with each subgraph until single vertices.



Community Detection: Path-based algorithms

DEFINITION 2: Few edges between communities

→ **Hierarchical divisive method: the Girvan-Neuman Algorithm [4]**

1. Start with the whole graph

2. **Find edges whose removal partitions the graph**

   we need a measure of how important an edge is in keeping the graph connected

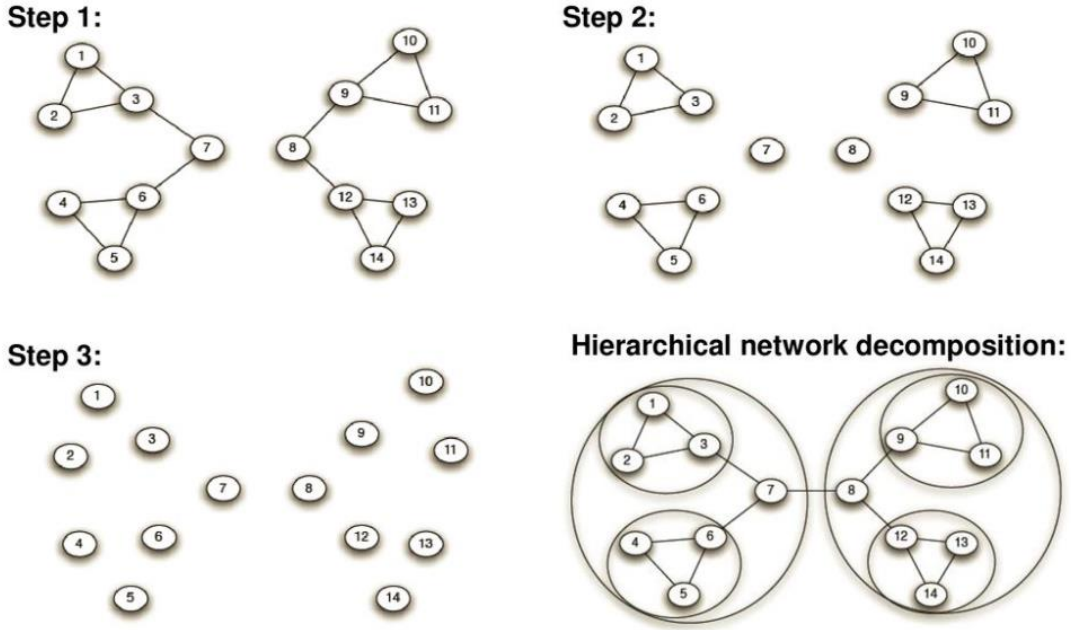
   → **Edge Betweeness: number of shortest paths passing through that edge**

   For each node pair u, v there may be multiple shortest paths (SP)

$$B(e) = \sum_{u,v \in V} \frac{|SP(u,v) \text{ that include } e|}{|SP(u,v)|}$$

   A. Compute the **Edge Betweeness** for all edges

   B. Remove the edge with the highest betweeness



3. **Repeat with each subgraph until single vertices.**

→ At each step of the algorithm, the connected components are the communities

→ Gives a hierarchical decomposition of the graph into communities

Community Detection: Modularity

DEFINITION 3: fewer edges between groups than expected on the basis of chance

→ **Modularity-based community detection**

Modularity of a network is the number of edges that run between vertices of the same community minus the number of such edges we would expect to find if edges were positioned at random while preserving the vertex degrees

Let us denote by c_i the community of vertex i and $\delta(c_i, c_j) = 1$ if $c_i = c_j$ and $\delta(c_i, c_j) = 0$ otherwise.

Hence, the actual number of edges that run between vertices in the same group is

$$\sum_{(i,j)\in E} \delta(c_i, c_j) = \frac{1}{2} \sum_{i,j} A_{i,j} \delta(c_i, c_j)$$

Where E is the set of edges in the graph and A_{ij} is the actual number of edges between i and j.

The expected number of edges that run between vertices of the same group is:

$$\frac{1}{2} \sum_{i,j} \frac{k_i k_j}{2m} \delta(c_i, c_j)$$

Community Detection: Modularity

DEFINITION 3: fewer edges between groups than expected on the basis of chance

→ **Modularity-based community detection**

Modularity of a network is the number of edges that run between vertices of the same community minus the number of such edges we would expect to find if edges were positioned at random while preserving the vertex degrees

Therefore modularity is given by:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{i,j} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) = \frac{1}{2m} \sum_{i,j} B_{i,j} \delta(c_i, c_j)$$

Where the modularity matrix is

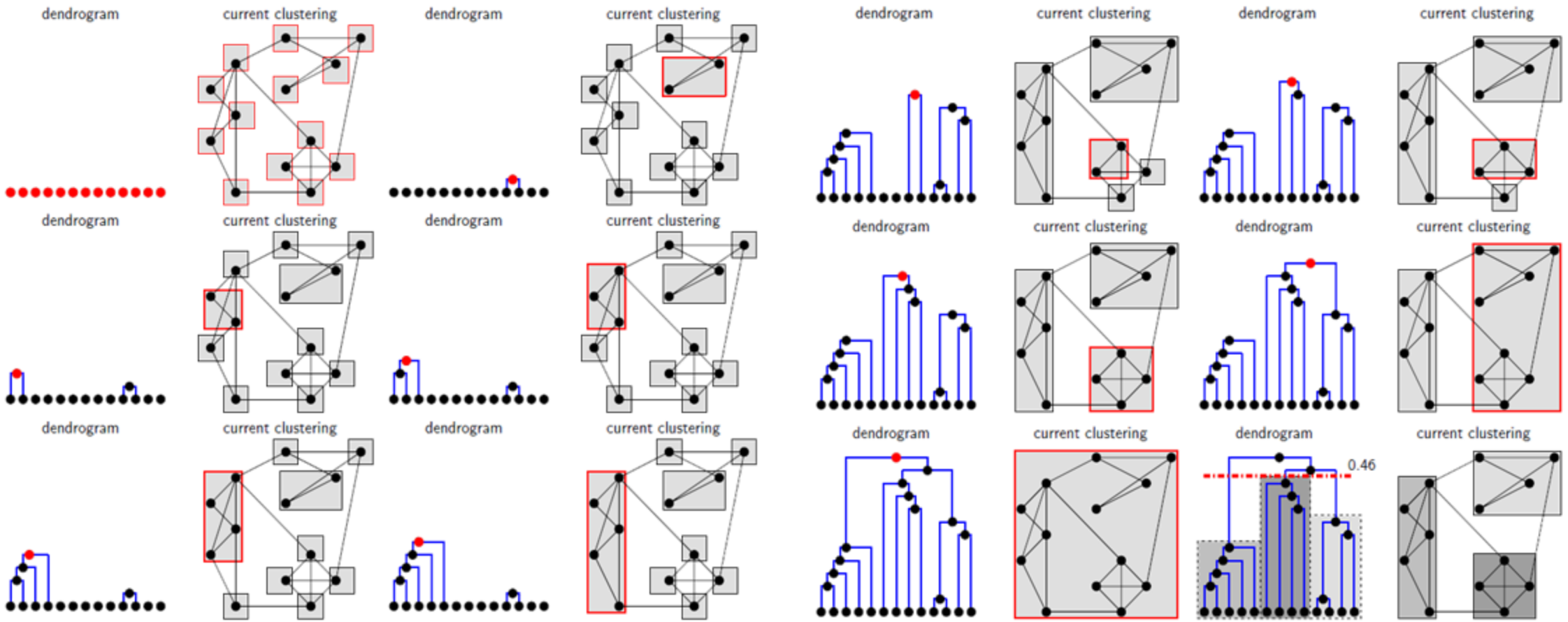
$$B_{i,j} = A_{i,j} - \frac{k_i k_j}{2m}$$

The modularity Q takes positive values if there are more edges between same-group vertices than expected, and negative values if there are less.

Community Detection: Modularity

Finding the best value of Q is NP-hard.

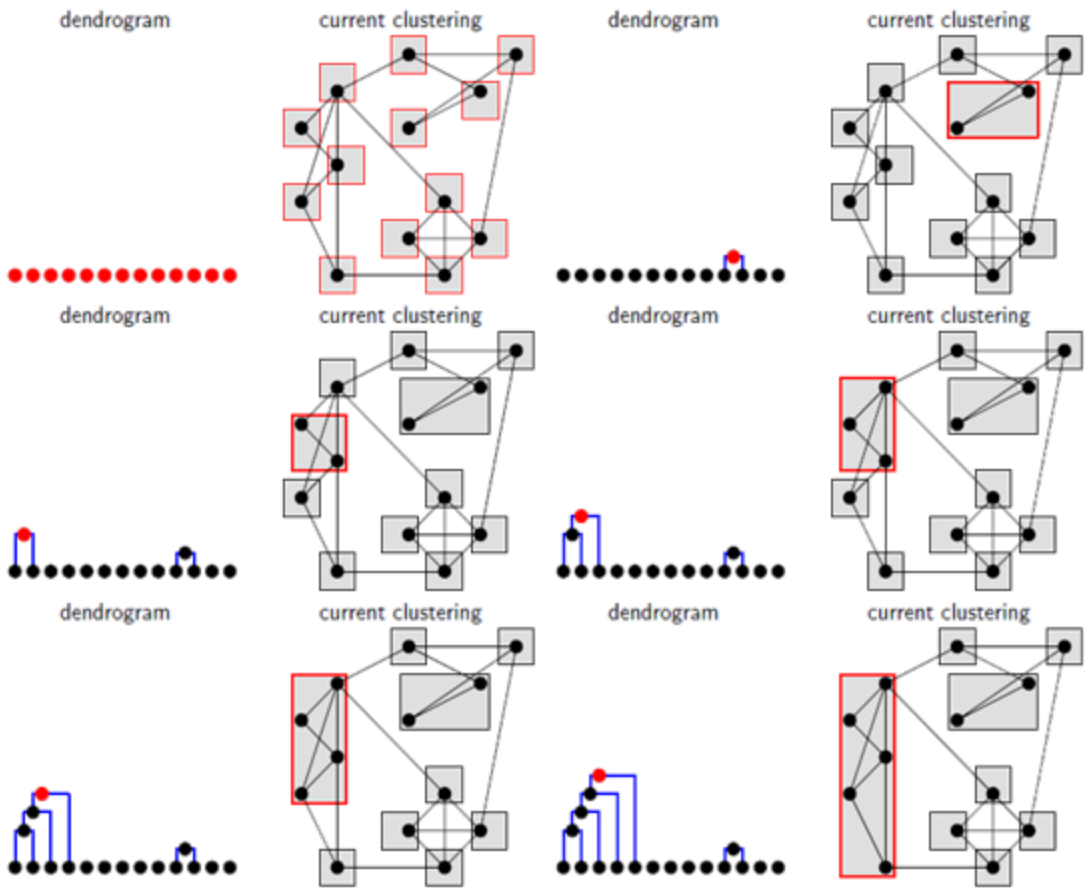
Newman proposed a greedy method to maximize the modularity [5]. It is an agglomerative hierarchical clustering algorithm, in which edges are joined to shape larger communities such that modularity is increased.

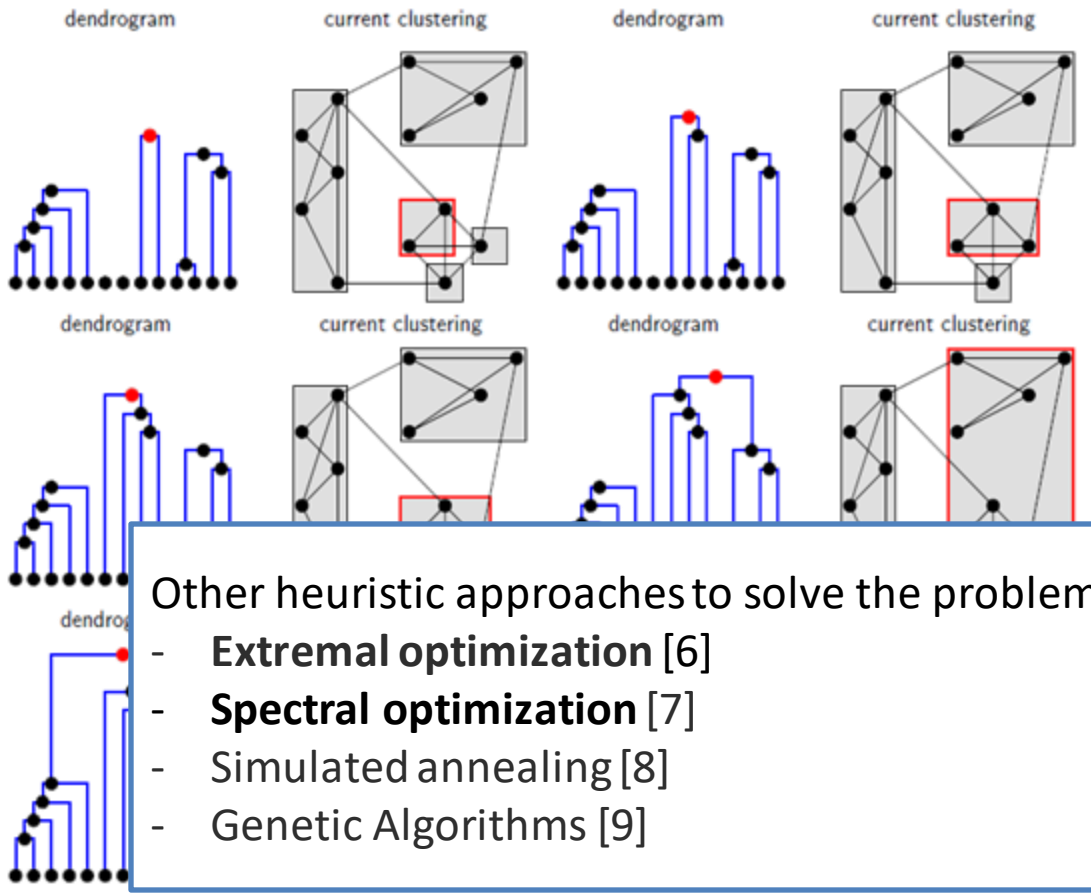


Community Detection: Modularity

Finding the best value of Q is NP-hard.

Newman proposed a greedy method to maximize the modularity [5]. It is an agglomerative hierarchical clustering algorithm, in which edges are joined to shape larger communities such that modularity is increased.





Other heuristic approaches to solve the problem

- **Extremal optimization** [6]
- **Spectral optimization** [7]
- Simulated annealing [8]
- Genetic Algorithms [9]

Community Detection: Clique Percolation

DEFINITION 4: highly connected groups of nodes that might overlap

→ **Clique percolation method**

A k-clique is a complete graph with k vertices.

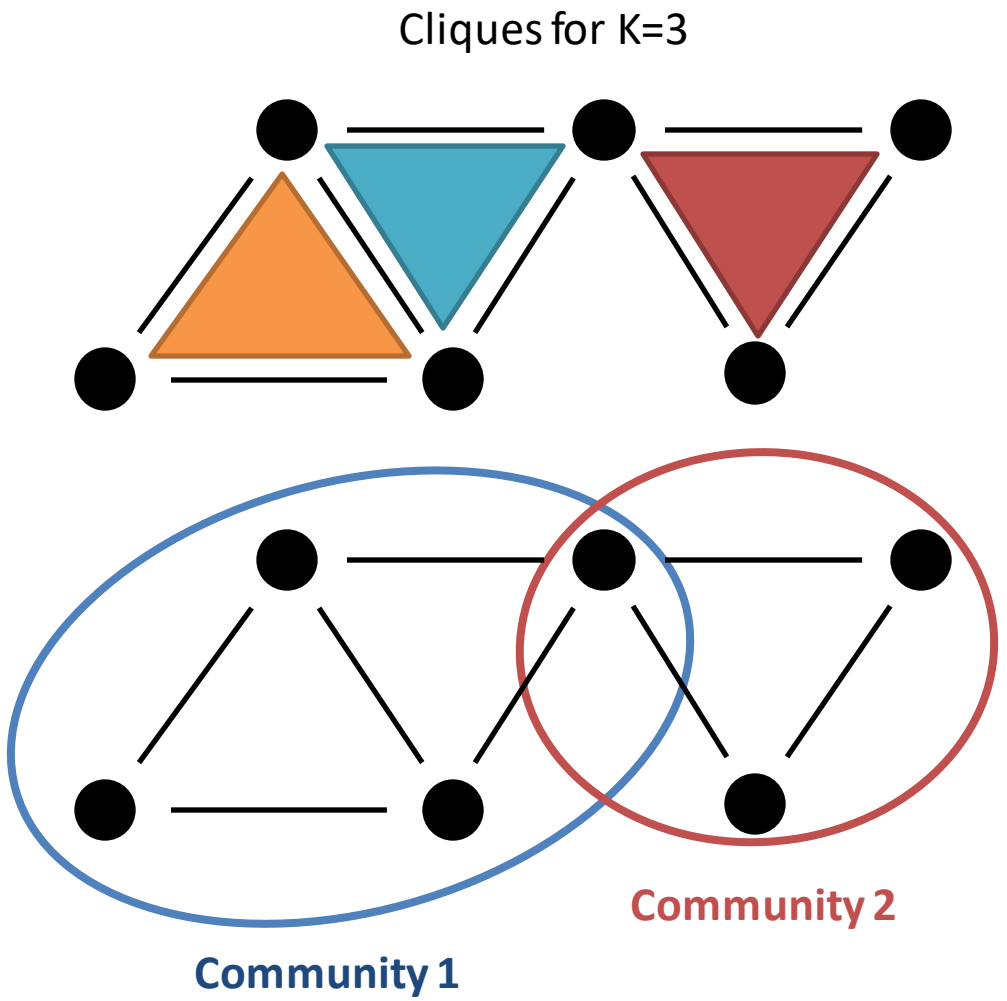
Two k-cliques are adjacent if they share k-1 vertices.

→ The union of adjacent k-cliques is called k-clique chain. Two k-cliques are connected if they are part of a k-clique chain.

→ A **k-clique community is the largest connected subgraph** obtained by the union of a k-clique and all k-cliques which are connected to it.

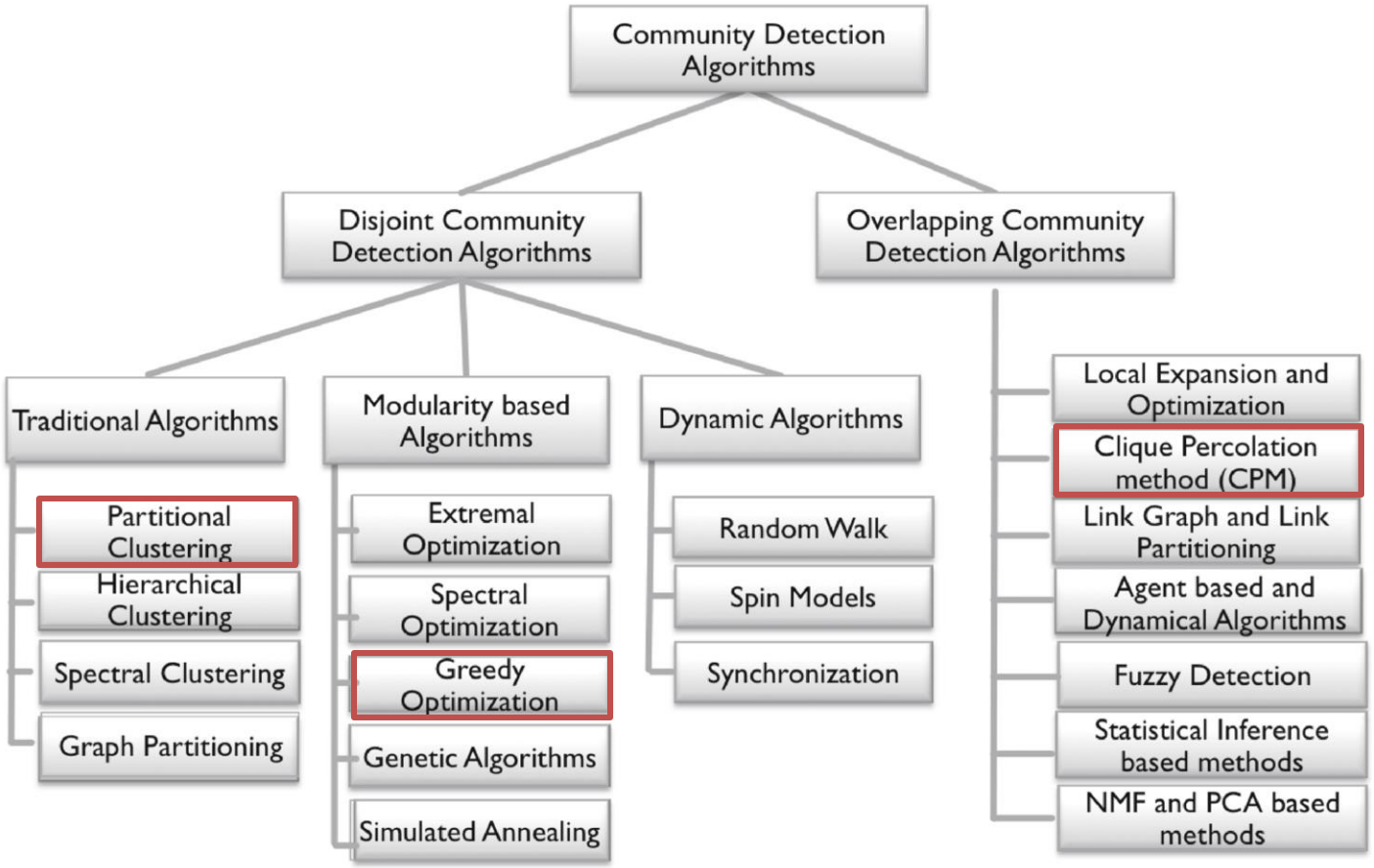
Clique Percolation Method was first introduced in 2008 by Palla et al. [10]

The details of the efficient algorithm they developed in order to identify cliques in large networks can be found in the supplementary material of their original paper [download from here]



Community Detection: a vibrant research field

We just provided some examples of algorithms associated to each community definition, but many more can be found in this huge research field.



A very large review and many more examples here [11]:

Muhammad Aqib Javed et al., Community detection in networks: A multidisciplinary review, Journal of Network and Computer Applications, 2018

Graphs as data: Representations & Machine Learning

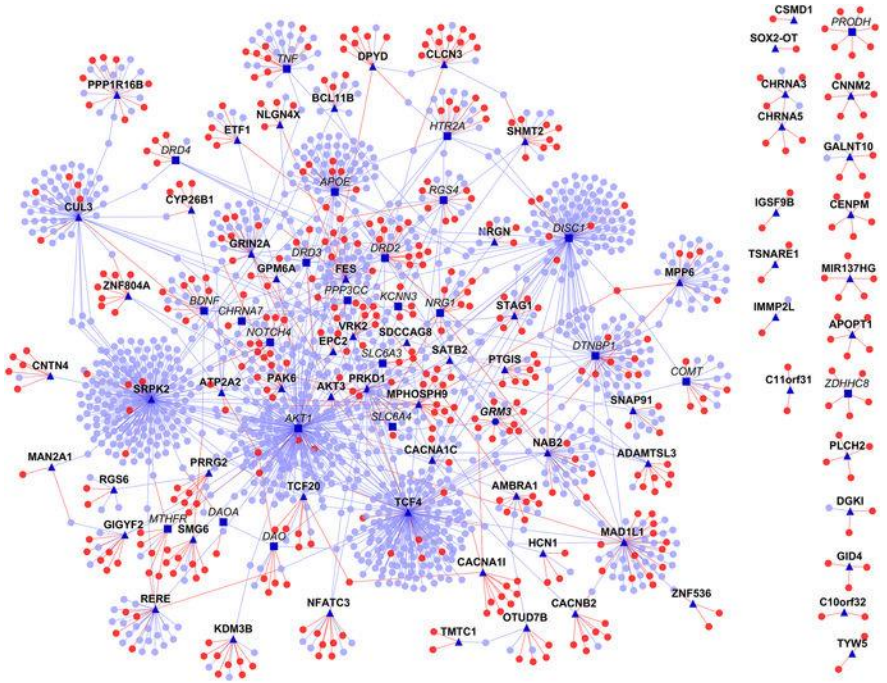
Machine Learning tasks on graphs

Node embeddings methods

Learning on networks

We have discussed how to analyze networks to *describe* them, or to *partition/cluster* them into subnetworks.
However, we can exploit the network structure of our data for more than that…

Classifying proteins' function in the interactome

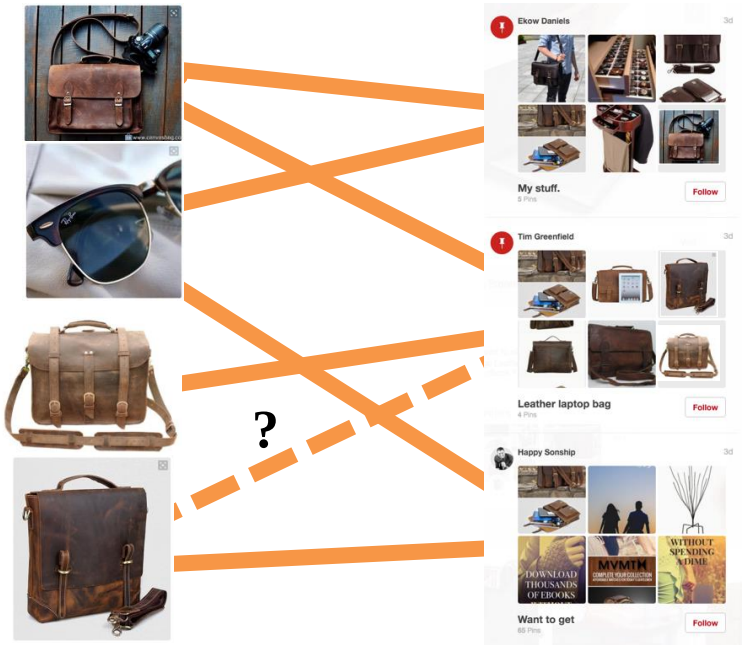


NODE CLASSIFICATION
(or graph/network labeling problem)

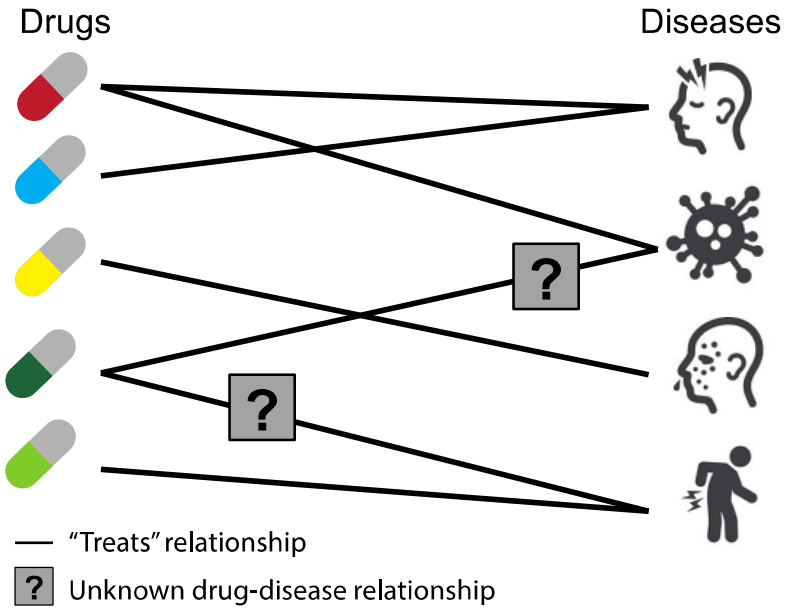
Learning on networks

We have discussed how to analyze networks to *describe* them, or to *partition/cluster* them into subnetworks.
However, we can exploit the network structure of our data for more than that…

Content Recommendation



Treatment Recommendation / Side effect prediction



LINK PREDICTION

Graph classification

We have discussed how to analyze networks to *describe* them, or to *partition/cluster* them into subnetworks.

However, we can exploit the network structure of our data for more than that…

- Bioinformatics and Cheminformatics: to predict the function of a protein structure, if cells are cancerous or not, if a protein is enzyme or not, checking the toxicity of a chemical compound
- Social network analysis: to provide online recommendations for a page or user account, implement newsfeed and calculate page rank - users/pages are nodes and the interaction between them are edges
- Natural Language Processing: to categorize different documents based on the structure of the text.
- Neuroscience: to analyse brain networks. Neurons are represented using nodes and the connection between neurons are represented by edges.

Learning on networks

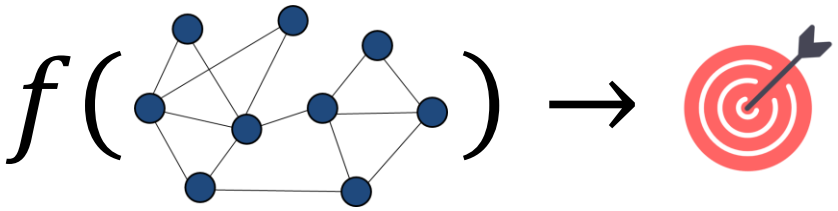
We have discussed how to analyze networks to *describe* them, or to *partition/cluster* them into subnetworks.

However, we can exploit the network structure of our data for more than that…

- Node Classification
- Link prediction
- Graph Classification
- *(Graph Generation)*

All these tasks require us to inform our classification or regression models with ***representations* of graph data**.

In other words, as with many **highly complex data** types, we wish to **extract features** that can then be exploited for classification, regression or clustering.

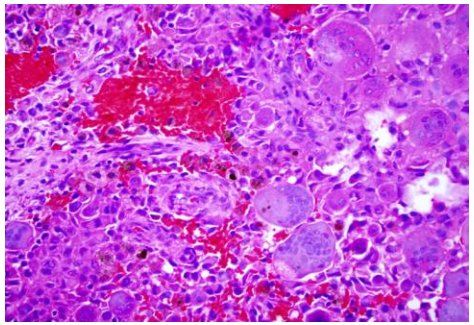


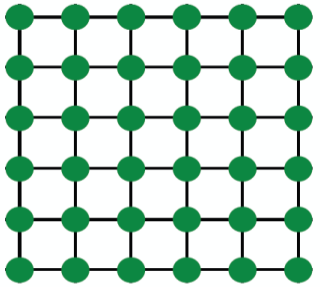
Representation Learning

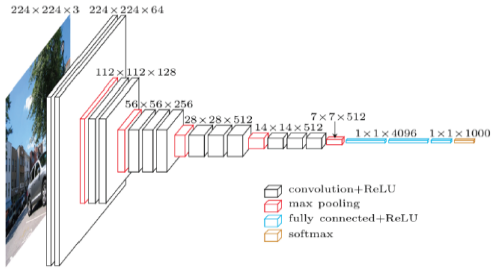
Representation (feature) learning is a typical Machine Learning task.

Several methods exist to automatically extract features from either structured or unstructured data.

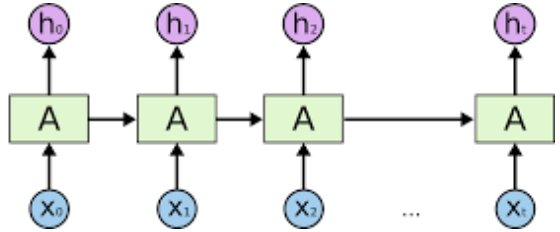
Images have 2D grid structure and we can define convolutions (CNN)



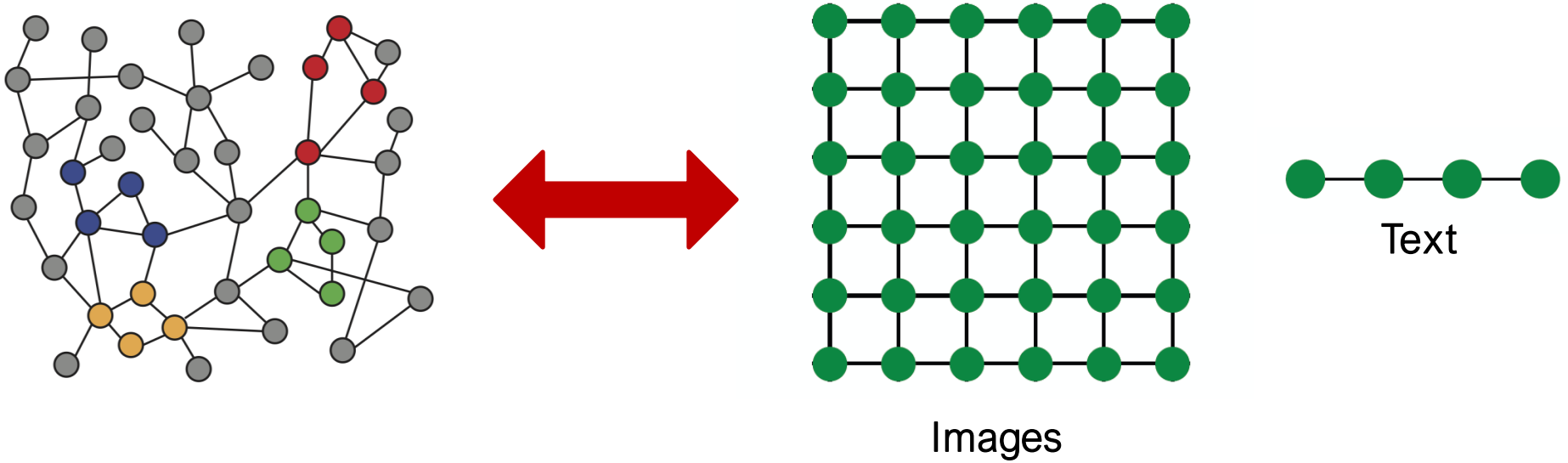




Text and sequences have linear 1D structure. Therefore, we can define sliding window, RNNs, word2vec, etc.



Networks are far more complex!

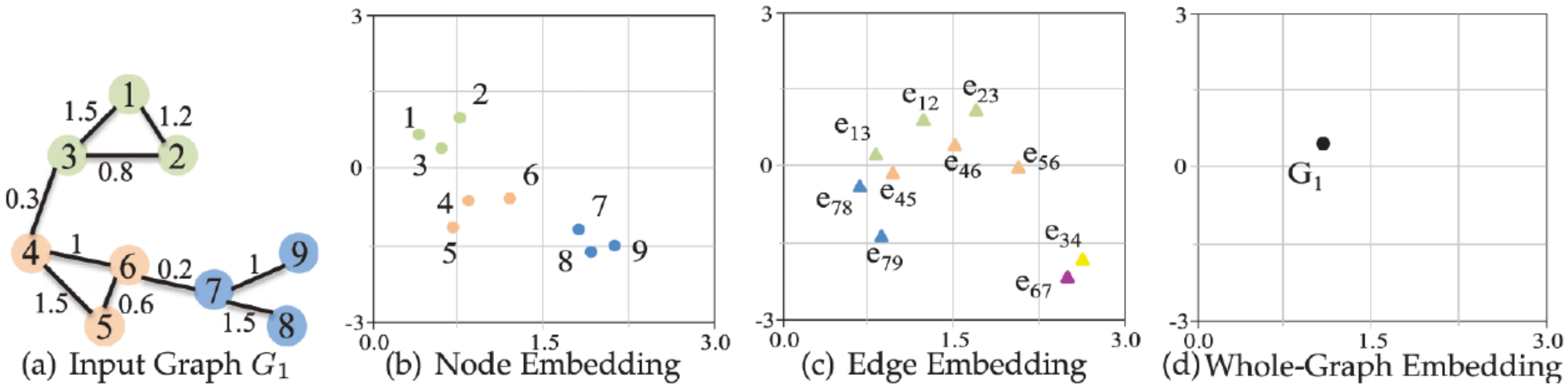


- Arbitrary size and complex topological structure (i.e., no spatial locality like grids)
- No fixed node ordering or reference point (i.e. isomorphism problem)
- Often dynamic and have multimodal features

Embedding types for graphs

What aspects of the graph are we trying to represent:

- vertex embeddings: describe connectivity of each node. It targets node prediction, reconstruction, and graph clustering
- edge/path embeddings: describe traversals across the graph. It targets edge prediction, reconstruction, and graph clustering
- graph embeddings: encode the entire graph into a single vector. It targets graph classification, graph matching



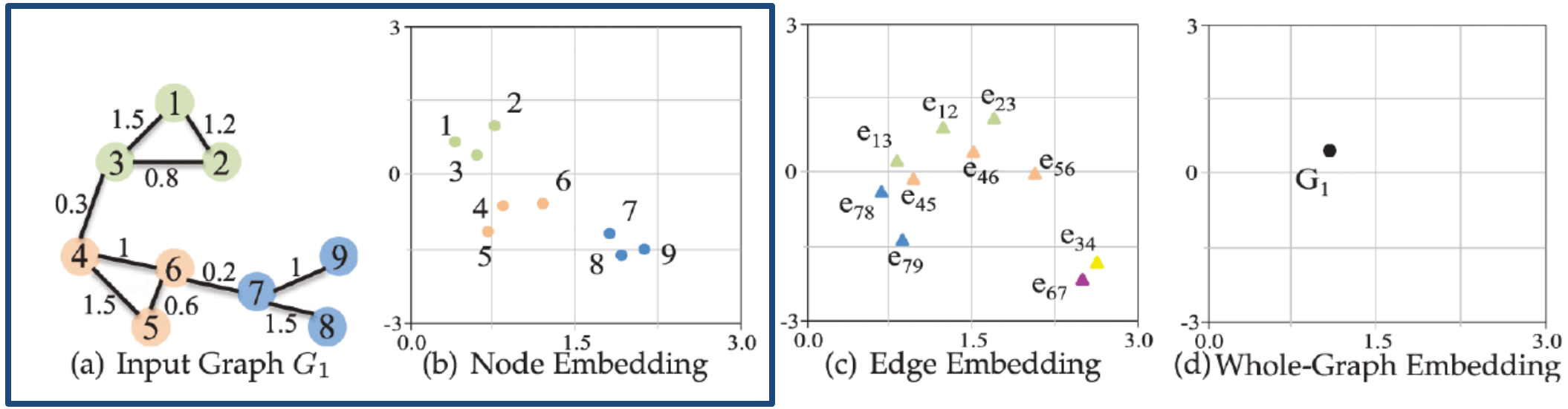
(a) Input Graph G_1 (b) Node Embedding (c) Edge Embedding (d) Whole-Graph Embedding

Embedding types for graphs

What aspects of the graph are we trying to represent:

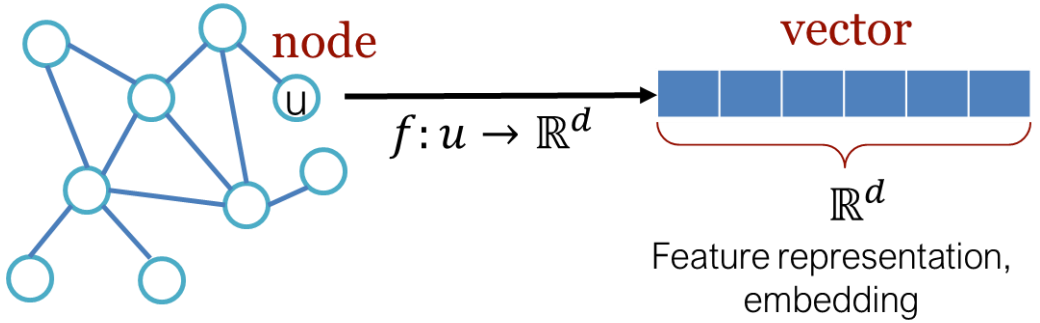
TODAY'S LECTURE

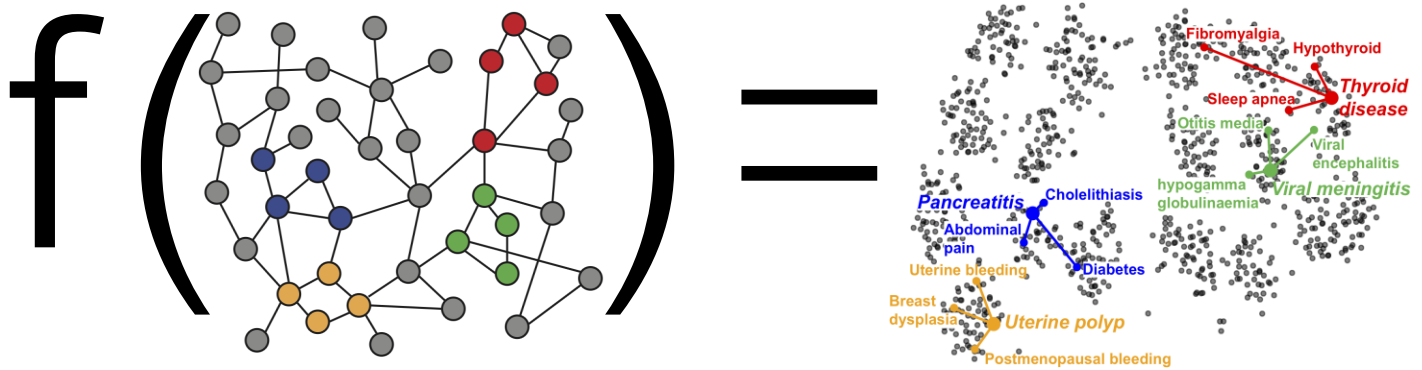
- vertex embeddings: describe connectivity of each node. It targets *node prediction, reconstruction, and graph clustering*
- edge/path embeddings: describe traversals across the graph. It targets edge prediction, reconstruction, and graph clustering
- graph embeddings: encode the entire graph into a single vector. It targets graph classification, graph matching



Unsupervised node representation (feature) Learning on graphs

Goal: Efficient **task-independent** feature learning for machine learning in networks



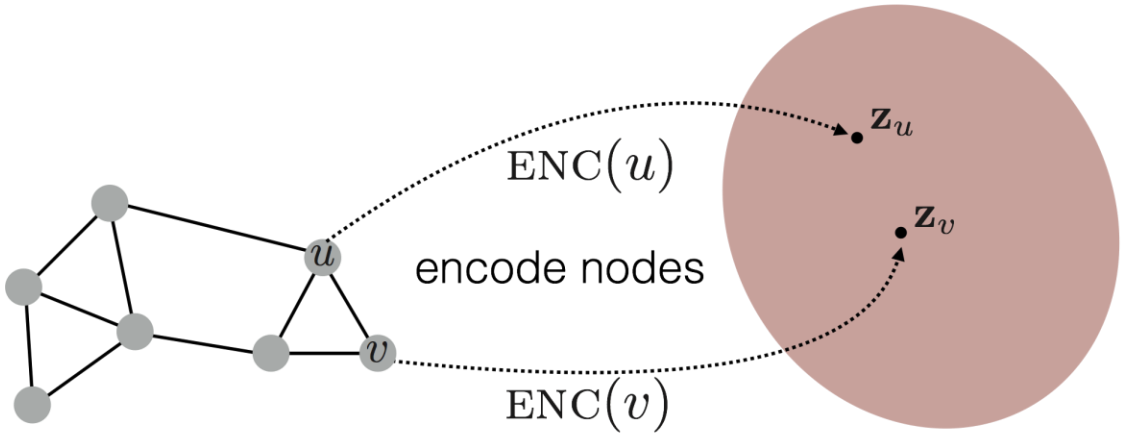


Disease similarity network

2-dimensional node embeddings

All in one slide

Goal: Map nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the network



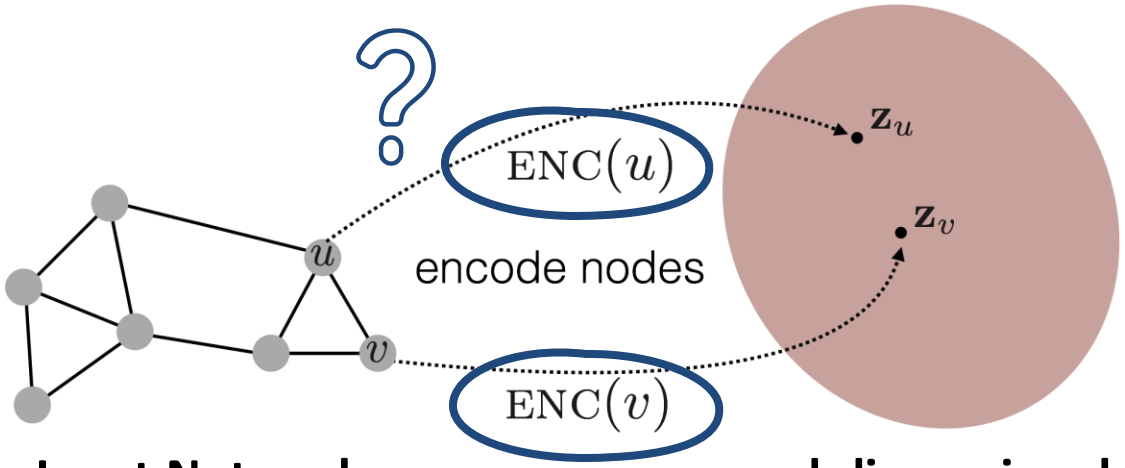
Input Network

d-dimensional embedding space

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

All in one slide

Goal: Map nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the network



Input Network

d-dimensional embedding space

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

Learning node embeddings

1. **Define an encoder** (a function ENC that maps node u to embedding $\mathbf{z}_u$): $\text{ENC}(v) = \mathbf{z}_v$
2. **Define a node similarity function** (a measure of similarity in the input network)
3. **Optimize parameters of the encoder** so that: $\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$

Node embeddings methods

There are several methods, mostly sharing the same **encoder.**

What changes is the concept of similarity

Two nodes have similar embeddings if…

…they are connected?

…they share many neighbors?

….they have similar local network structure?

etc.

Two quite different examples:

- **Adjacency-based similarity**
- **Random Walk-based similarity**

Adjacency-based similarity to embed graph nodes

- **Similarity function** is the edge weight between *u* and *v* in the network
- **Intuition:** Dot products between node embeddings approximate edge existence

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

- $\mathcal{L}$: loss (what we want to minimize)
- $\sum_{(u,v) \in V \times V}$: sum over all node pairs
- $\mathbf{z}_u^\top \mathbf{z}_v$: embedding similarity
- $\mathbf{A}_{u,v}$: (weighted) adjacency matrix for the graph

Find embedding matrix $\mathbf{Z} \in \mathbb{R}^{d \times |V|}$ that minimizes the loss $\mathcal{L}$:

- Option 1: Stochastic gradient descent (SGD)
  - Highly scalable, general approach
- Option 2: Solve matrix decomposition solvers
  - e.g., SVD or QR decompositions
  - Need to derive specialized solvers

Adjacency-based similarity to embed graph nodes

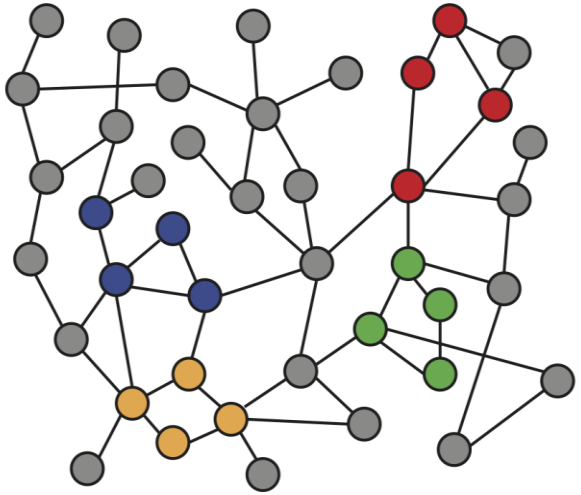
$O(|V|^2)$ runtime

- Must consider all node pairs
- $O([E|)$ if summing over non-zero edges

$O(|V|)$ parameters

- One learned embedding per node

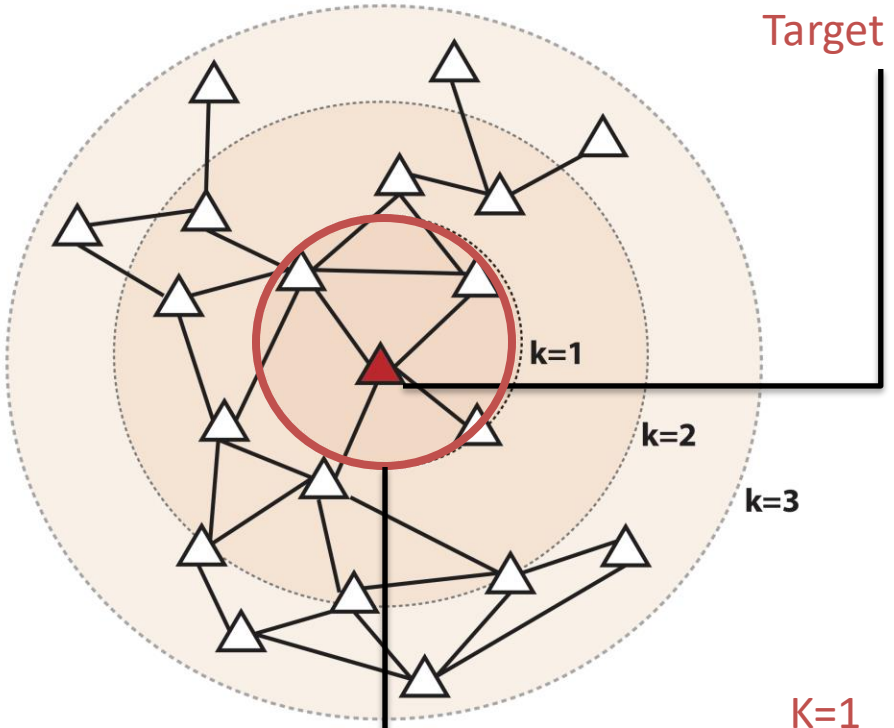
Main limitation: *it only considers direct connections*



Red nodes are obviously more similar to Green nodes compared to Orange nodes, despite none being directly connected

Higher-order neighborhoods

The main idea is to define similarity function on the basis of **higher-order neighborhoods.**



How can we stochastically define these higher order neighborhoods?

Neighborhood-informed similarities

Idea: Learn node embedding such that nearby nodes are close together, considering **indirect connections** as well.

Given a node u, nearby nodes are nodes belonging to $N_R(u)$

$N_R(u)$ = neighbourhood of u obtained by some strategy R

Neighborhood-informed similarities and embeddings

Idea: Learn node embedding such that nearby nodes are close together, considering **indirect connections** as well.

Given a node u, nearby nodes are nodes belonging to $N_R(u)$

$N_R(u)$ = neighbourhood of u obtained by some strategy R

Given $G = (V, E)$

Goal is to learn $f: u \rightarrow \mathbb{R}^d$

→ Given node u, we want to learn feature representation $f(u)$ that is predictive of nodes in u's neighborhood $N_R(u)$

$$\max_f \sum_{u \in V} \log \Pr(N_R(u) | \boldsymbol{z}_u)$$

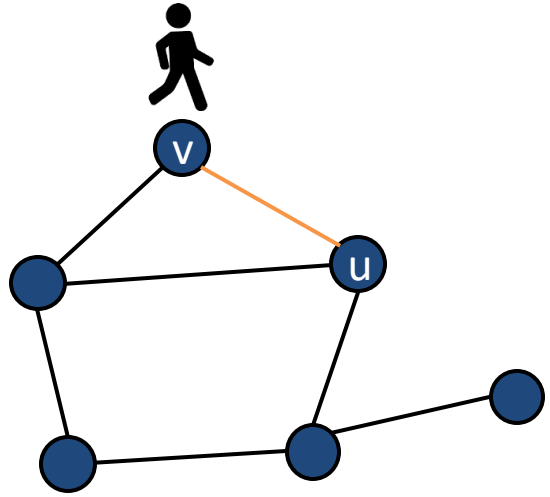
Assuming that the conditional likelihood factorizes as follows: $P(N_R(u)|\mathbf{z}_u) = \prod_{n_i \in N_R(u)} P(n_i|\mathbf{z}_u)$

Note: Random walks

A random walk on a graph is a process that begins at some vertex, and at each time step moves to another vertex.
When the graph is **unweighted**, the vertex the walk moves to is chosen **uniformly at random among the neighbors** of the present vertex.

Given a graph G, consider the walker is placed at vertex v.
At each stage, the walker has to move to another vertex u.
→ The probability of moving to u is

$$m_{uv} = \begin{cases} \dfrac{1}{\deg(v)} & if\ (v,u) \text{ is an edge in the graph } G \\ 0 & otherwise \end{cases}$$



The matrix $M = (m_{vu})$ is a Markov matrix, or **transition matrix**.
As each stage occurs, a sequence of adjacent vertices is produced: a **random walk**.
The entry u,v of a matrix M^k represents the probability that a random walk of length k starting at v ends at u.

When the graph is **weighted**, moving to a neighbor has **probability proportional to the weight** of the corresponding edge.

Random Walk-based embeddings

1. Simulate many short random walks starting from each node using a strategy R
2. For each node u, get $\mathrm{N}_R(u)$ as a sequence of nodes visited by random walks starting at u
3. For each node u, learn its embedding by predicting which nodes are in $\mathrm{N}_R(u)$:

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

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and $\mathbf{v}$ co-occuring on random walk, i.e., use softmax to parameterize $P(v|\mathbf{z}_u)$

Random Walk-based embeddings

1. Simulate many short random walks starting from each node using a strategy R
2. For each node u, get $\mathrm{N}_R(u)$ as a sequence of nodes visited by random walks starting at u
3. For each node u, learn its embedding by predicting which nodes are in $\mathrm{N}_R(u)$:

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

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

sum over all nodes u

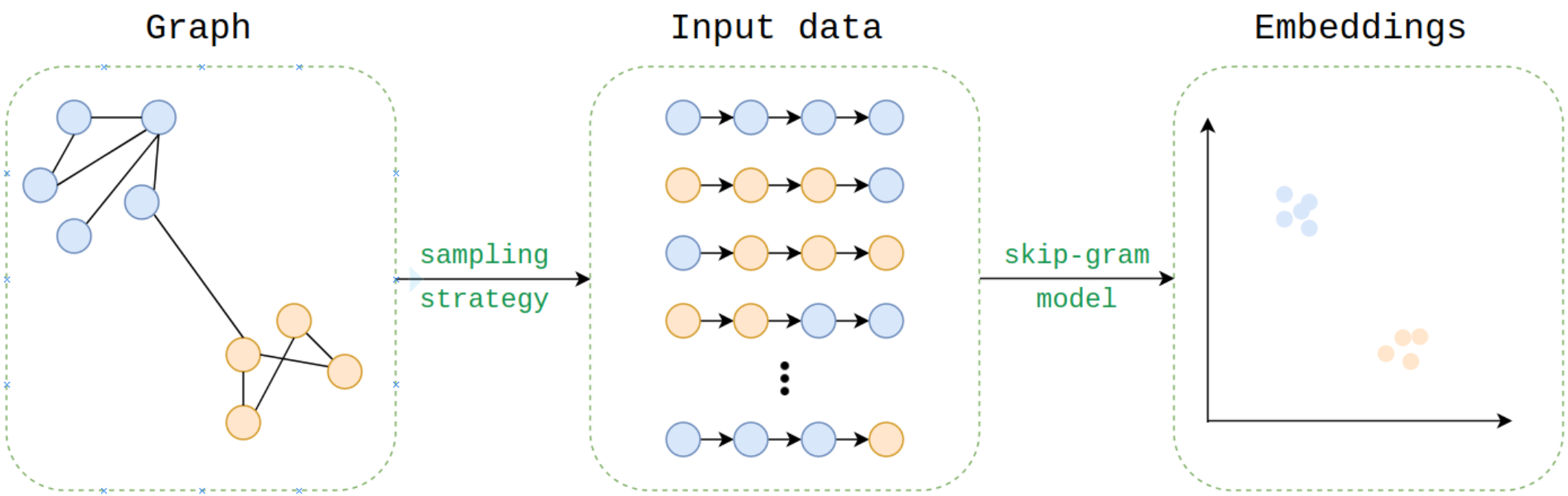
sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk, i.e., use softmax to parameterize $P(v|\mathbf{z}_u)$

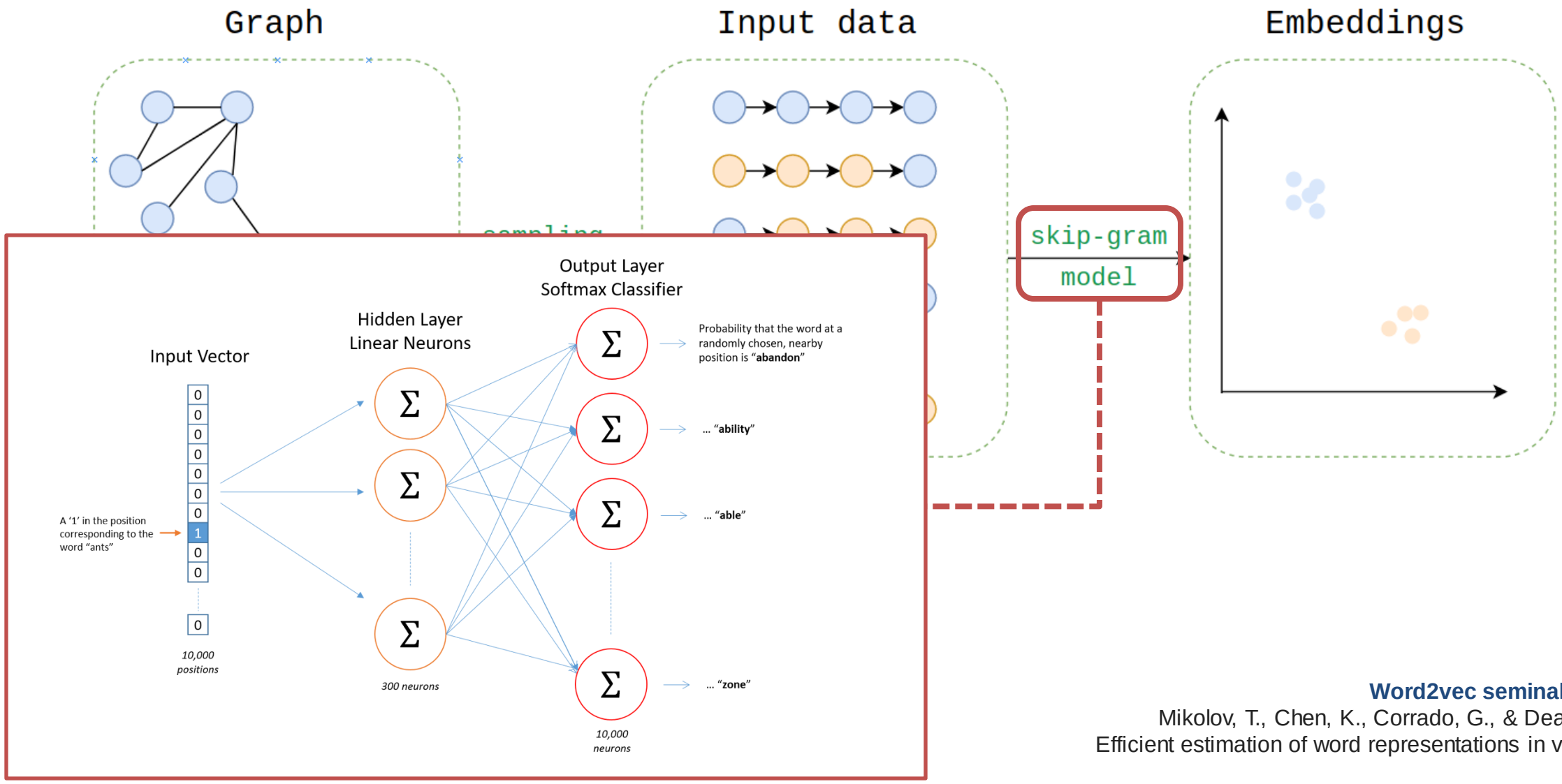
VERY EXPENSIVE!

i.e., instead of normalizing w.r.t. all nodes, just normalize against k random **negative samples**

How does it work in practice?

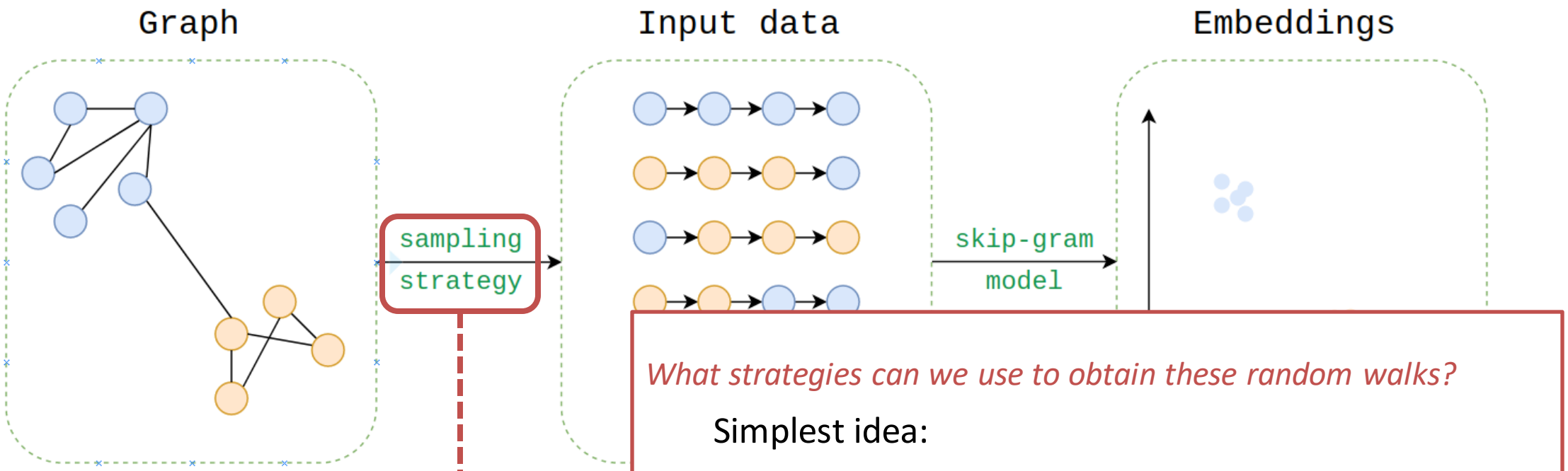


How does it work in practice?



Word2vec seminal paper [16]
Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013)
Efficient estimation of word representations in vector space

How does it work in practice?



What strategies can we use to obtain these random walks?

Simplest idea:

Fixed-length, unbiased random walks starting from each node (i.e., DeepWalk from Perozzi et al., 2013)

Can we do better?

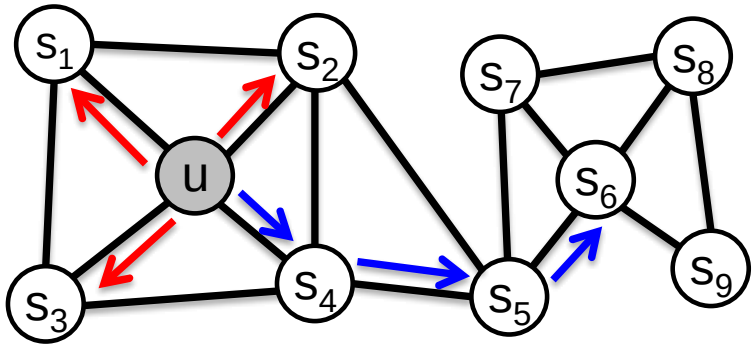
Grover et al., 2016; Ribeiro et al., 2017; Abu-El-Haija et al., 2017 and many others

node2vec

One famous example: node2vec with **biased random walks**

→ **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 neighborhood $N_R(u)$ of a given node u:



→ BFS

→ DFS

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

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

Biased Random Walks interpolate between these two strategies to obtain the most informative Neighborhood.

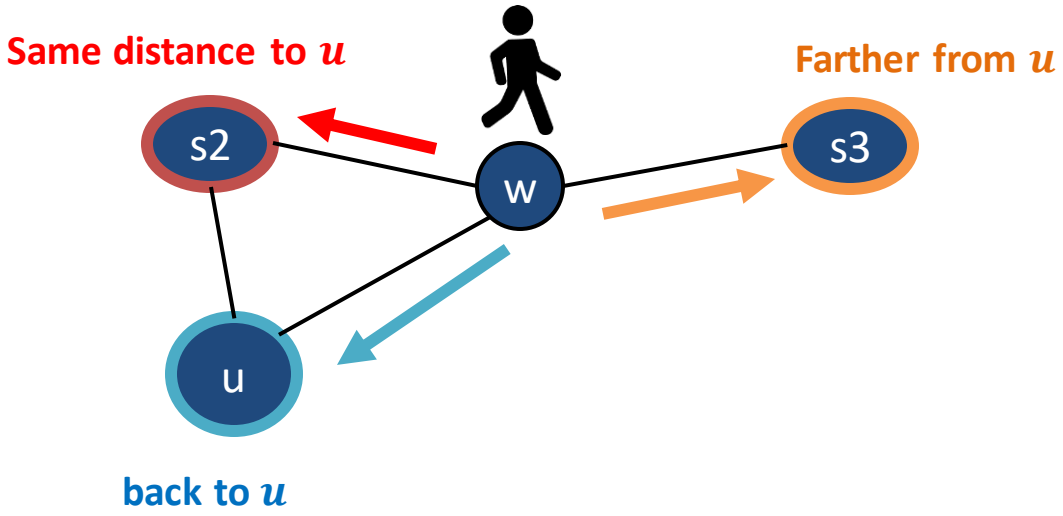
node2vec: *biased* random walks

Biased random walk R that given a node u generates neighborhood $N_R(u)$

Two parameters:

Return parameter p: Return back to the previous node

In-out parameter q: Moving outwards (DFS) vs. inwards (BFS)



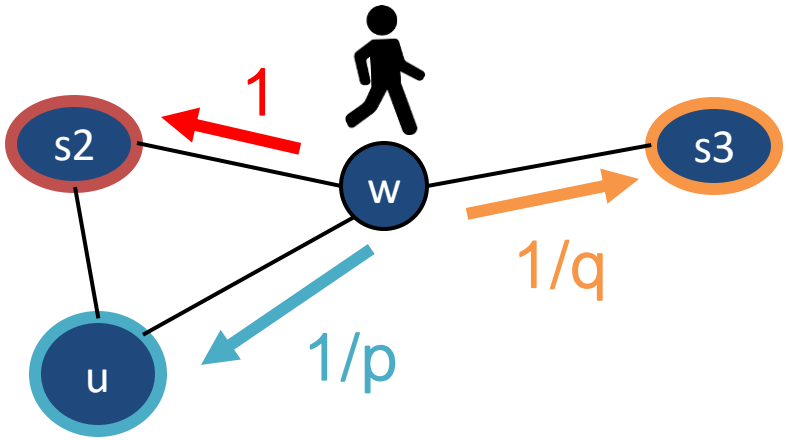
node2vec: *biased* random walks

Biased random walk R that given a node u generates neighborhood $N_R(u)$

Two parameters:

Return parameter p: Return back to the previous node

In-out parameter q: Moving outwards (DFS) vs. inwards (BFS)



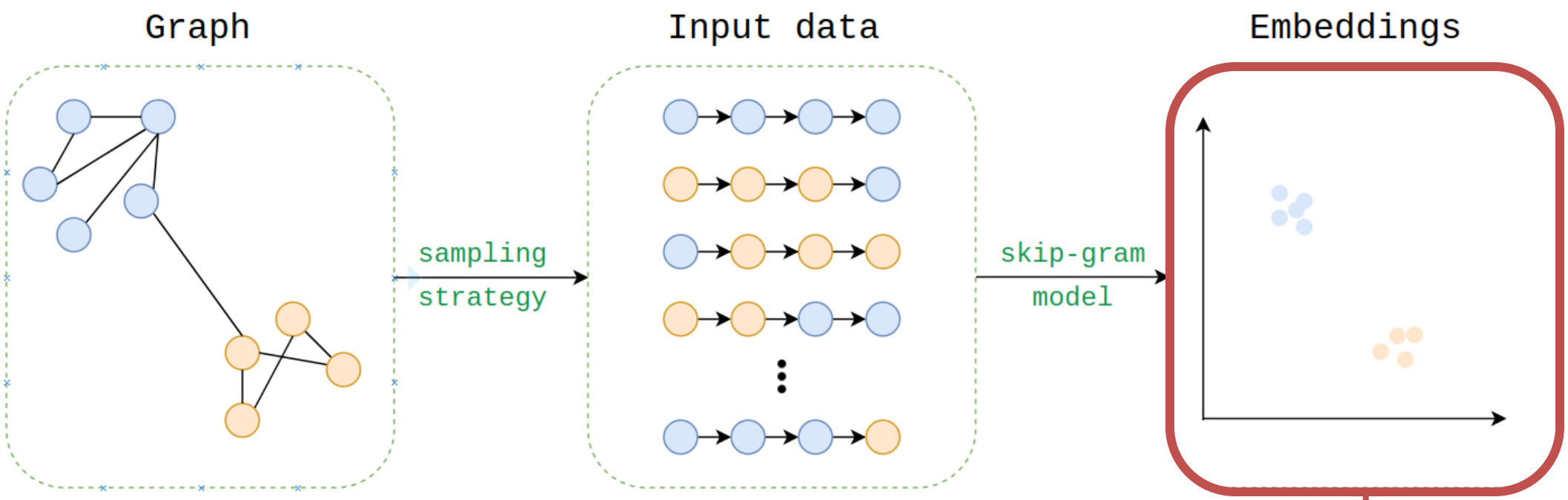
Edge weights

The transition probability $\pi_{us_i} = \alpha_{pq}(u, s_i) \cdot w_{us_i}$

$$\alpha_{pq}(u, s_i) = \begin{cases} \frac{1}{p} & \text{if } d(u, s_i) = 0 \\ 1 & \text{if } d(u, s_i) = 1 \\ \frac{1}{q} & \text{if } d(u, s_i) = 2 \end{cases}$$

Where $d(u, s_i)$ denotes the shortest path distance between u and s_i.

node2vec



Node labeling/classification

Link prediction (nearby nodes in the embedding space may be linked)

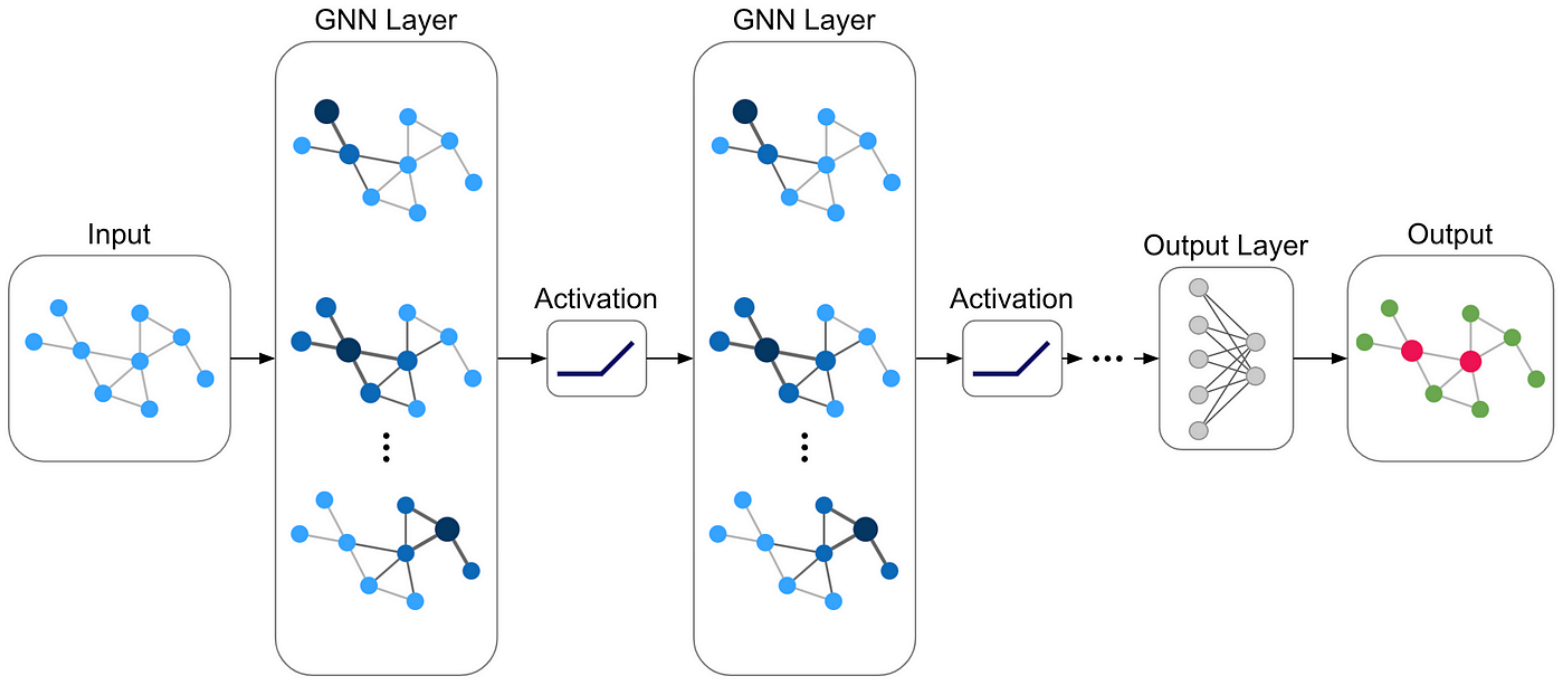
Node clustering, ...

What about supervised tasks on graphs?

Some of the tasks we mentioned so far can also be achieved in a **supervised** fashion, **optimizing the extraction of information from Graphs for the specific task at hand**.

This is the (very large) class of **Graph Neural Networks (GNNs)**.

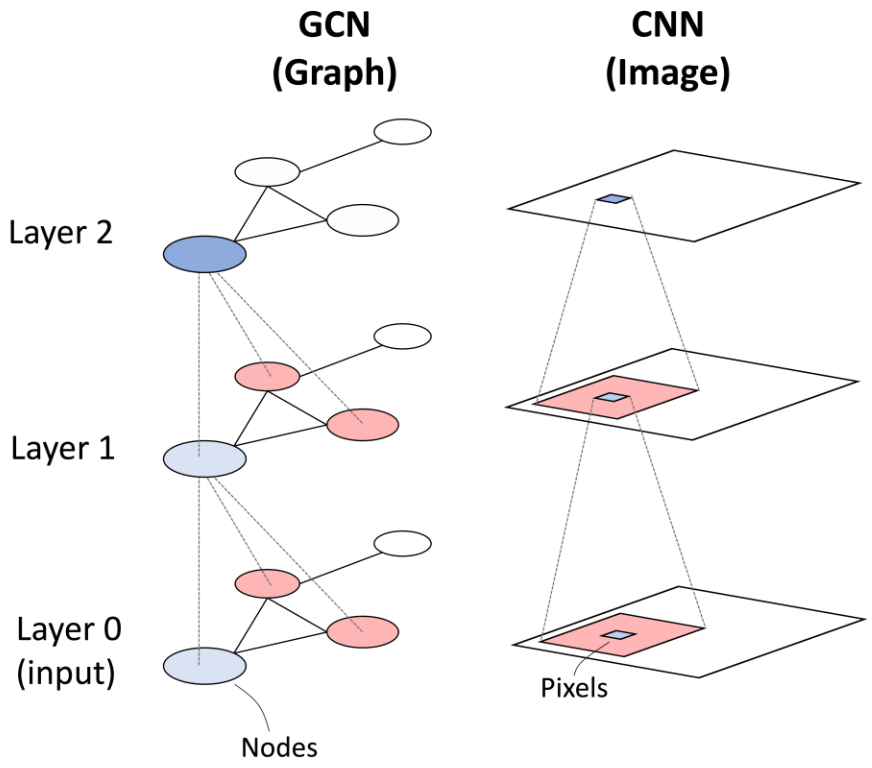
In practice, we still generate intermediate graph embeddings, but unlike classic algorithms and unsupervised embeddings (which pass results as features to downstream ML models), these methods are **fully end-to-end graph native solutions.**

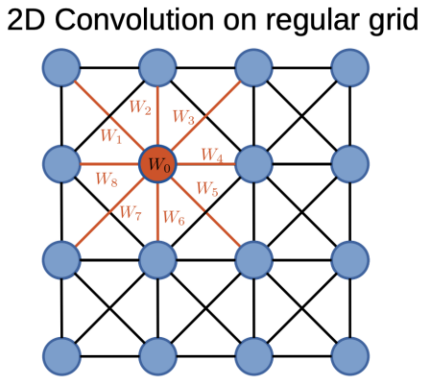


The simplest (and most famous) GNN model is the **Graph Convolutional Network**

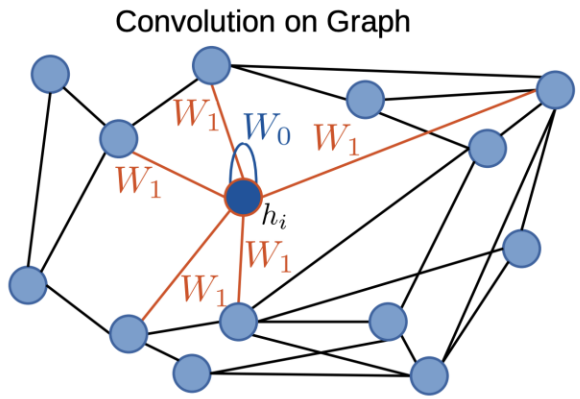
Graph Convolutional Networks

Graph convolutional networks (GCNs) as presented by Kipf and Welling (2017) is a class of **neural network** designed to operate on graphs.
They perform a **convolution** in the same way that traditional convolutional neural networks (CNNs) perform a convolution-like operation when operating on images.





- Channel-wise weight-sharing!

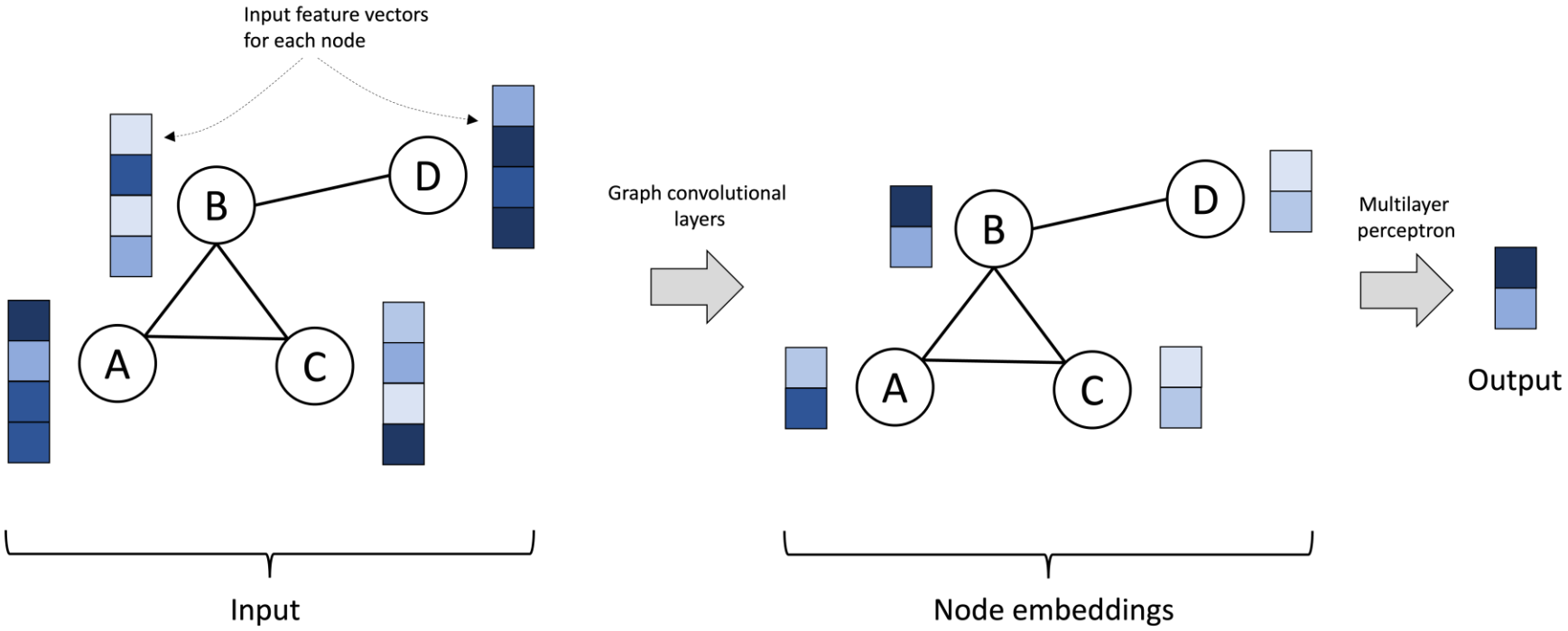


- Node-wise weight-sharing!

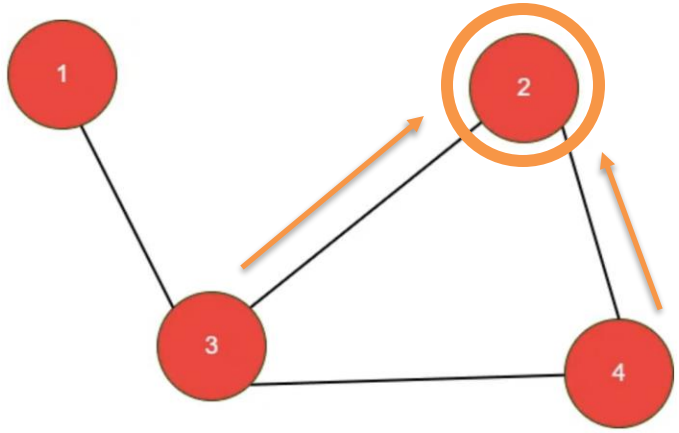
Graph Convolutional Networks

Fundamentally, a GCN takes as input a graph together with a set of feature vectors, where each node is associated with its own feature vector.
The GCN is then composed of a series of **graph convolutional layers** that iteratively transform the feature vectors at each node.

The output is then the graph associated with output vectors associated with each node. These output vectors can be (and often are) of different dimension than the input vectors.



How do these Graph Convolutional Layers work?



What if I want to aggregate for each node the information on its neighboring nodes?

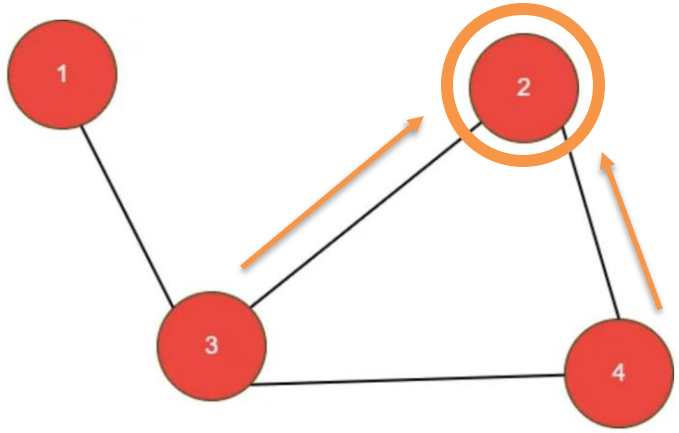
$A =$

	1	2	3	4
1	0	0	1	0
2	0	0	1	1
3	1	1	0	1
4	0	1	1	0

$X =$

1	-1.3	2.2
2	0.6	-1.1
3	2.0	1.4
4	5.1	1.2

How do these Graph Convolutional Layers work?



What if I want to aggregate for each node the information on its neighboring nodes?

$A =$

	1	2	3	4
1	0	0	1	0
2	0	0	1	1
3	1	1	0	1
4	0	1	1	0

$X =$

1	-1.3	2.2
2	0.6	-1.1
3	2.0	1.4
4	5.1	1.2

$A \times X =$

	1	2	3	4
1	0	0	1	0
2	0	0	1	1
3	1	1	0	1
4	0	1	1	0

$\times$

1	-1.3	2.2
2	0.6	-1.1
3	2.0	1.4
4	5.1	1.2

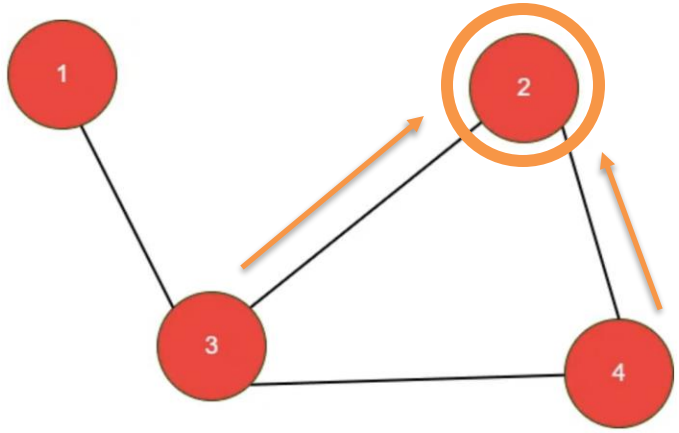
$=$

1	2.0	1.4
2	7.1	2.6
3	5.8	2.3
4	2.6	0.3

Are we happy?

Do you see any problem here…?

How do these Graph Convolutional Layers work?



What if I want to aggregate for each node the information on its neighboring nodes?

$$A = \begin{array}{|c|cccc|}\hline & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 \\ 4 & 0 & 1 & 1 & 0 \\ \hline \end{array} \quad X = \begin{array}{|c|cc|}\hline 1 & -1.3 & 2.2 \\ 2 & 0.6 & -1.1 \\ 3 & 2.0 & 1.4 \\ 4 & 5.1 & 1.2 \\ \hline \end{array}$$

$$A \times X = \begin{array}{|c|cccc|}\hline & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 \\ 4 & 0 & 1 & 1 & 0 \\ \hline \end{array} \times \begin{array}{|c|cc|}\hline 1 & -1.3 & 2.2 \\ 2 & 0.6 & -1.1 \\ 3 & 2.0 & 1.4 \\ 4 & 5.1 & 1.2 \\ \hline \end{array} = \begin{array}{|c|cc|}\hline 1 & 2.0 & 1.4 \\ 2 & 7.1 & 2.6 \\ 3 & 5.8 & 2.3 \\ 4 & 2.6 & 0.3 \\ \hline \end{array}$$

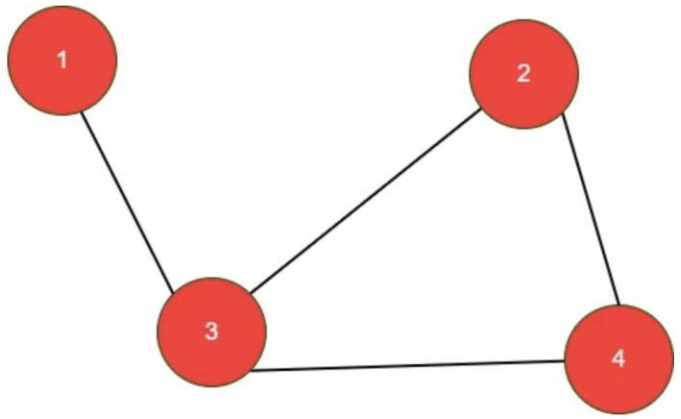
Problem 1

When we taking the multiplication between A and X, **we have ignored the feature of the node itself**. That means the first row of the resultant matrix should also contain features of node 1.

Problem 2

A is typically not normalized and therefore the multiplication with A will completely change the scale of the feature vectors. **Nodes with large degrees will have large values in their feature representation** while nodes with small degrees will have small values.

How do these Graph Convolutional Layers work?



$A =$

	1	2	3	4
1	0	0	1	0
2	0	0	1	1
3	1	1	0	1
4	0	1	1	0

$X =$

1	-1.3	2.2
2	0.6	-1.1
3	2.0	1.4
4	5.1	1.2

Solving **problem 1**:

$\tilde{A} =$

	1	2	3	4
1	0	0	1	0
2	0	0	1	1
3	1	1	0	1
4	0	1	1	0

$+$

	1	2	3	4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

$=$

	1	2	3	4
1	1	0	1	0
2	0	1	1	1
3	1	1	1	1
4	0	1	1	1

A I

Solving **problem 2**:

$D^{-1}\tilde{A} =$

	1	2	3	4
1	1/2	0	0	0
2	0	1/3	0	0
3	0	0	1/4	0
4	0	0	0	1/3

$\times$

	1	2	3	4
1	1	0	1	0
2	0	1	1	1
3	1	1	1	1
4	0	1	1	1

$=$

	1	2	3	4
1	0.5	0	0.5	0
2	0	0.33	0.33	0.33
3	0.25	0.25	0.25	0.25
4	0	0.33	0.33	0.33

D^{-1} $\tilde{A}$

$D^{-1}\tilde{A}X =$

	1	2	3	4
1	0.5	0	0.5	0
2	0	0.33	0.33	0.33
3	0.25	0.25	0.25	0.25
4	0	0.33	0.33	0.33

$\times$

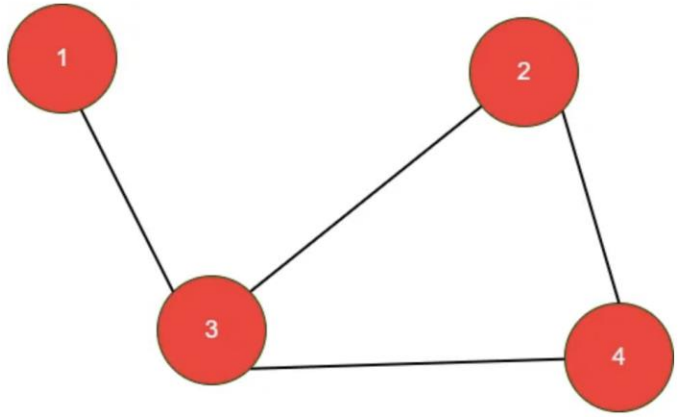
1	2.0	1.4
2	7.1	2.6
3	5.8	2.3
4	2.6	0.3

$=$

1	3.9	1.85
2	5.17	1.73
3	4.38	1.65
4	5.17	1.73

$D^{-1}\tilde{A}$ X

How do these Graph Convolutional Layers work?



$A =$

	1	2	3	4
1	0	0	1	0
2	0	0	1	1
3	1	1	0	1
4	0	1	1	0

$X =$

1	-1.3	2.2
2	0.6	-1.1
3	2.0	1.4
4	5.1	1.2

Finally, we apply simple weights…

$W =$

1
-1

$D^{-1}\tilde{A}XW =$

1	3.9	1.85
2	5.17	1.73
3	4.38	1.65
4	5.17	1.73

$D^{-1}\tilde{A}X$ ×

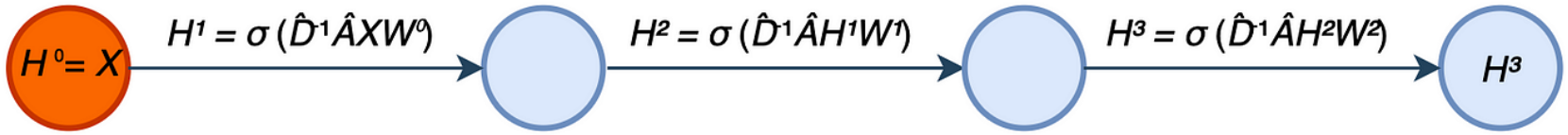
1
-1

W =

1	2.05
2	3.44
3	2.73
4	3.44

…and an activation function (σ).

A complete (multi-layer) Graph Convolutional Neural Network



So, to sum up…

1.Data preparation: Represent the graph in a format suitable for GCNs. This involves transforming the graph into a node adjacency matrix (A) and a node feature matrix (X).

2.Convolution operations: GCNs perform convolution operations on the node feature matrix using learnable weight matrices.

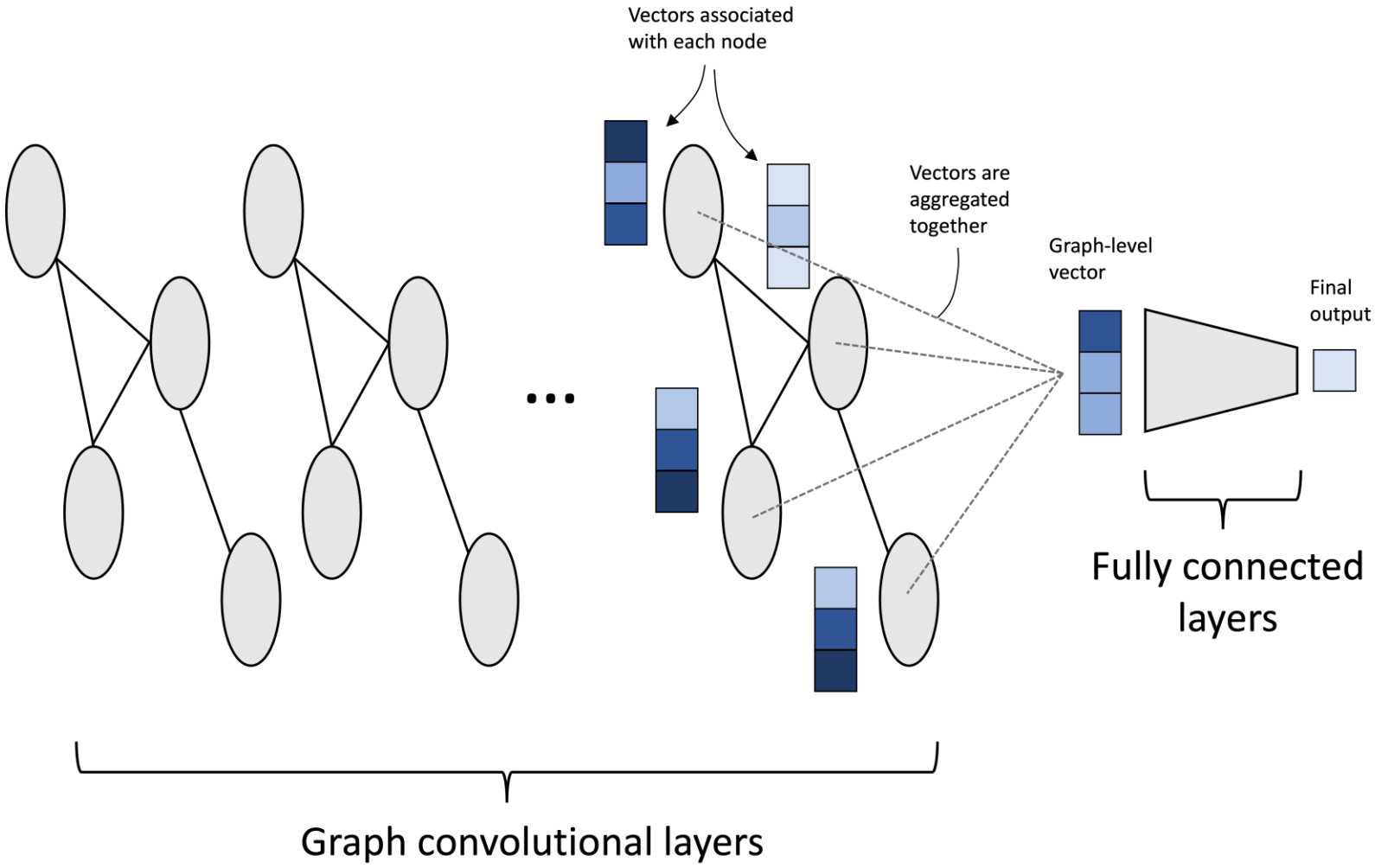
3.Non-linear activation: GCNs apply a non-linear activation function to the convolved node features, allowing them to learn more complex representations.

4.Training and prediction: GCNs are trained using backpropagation during supervised learning tasks, such as node classification or link prediction. Once trained, GCNs can be used to make predictions on new graph-structured data by following the same steps.

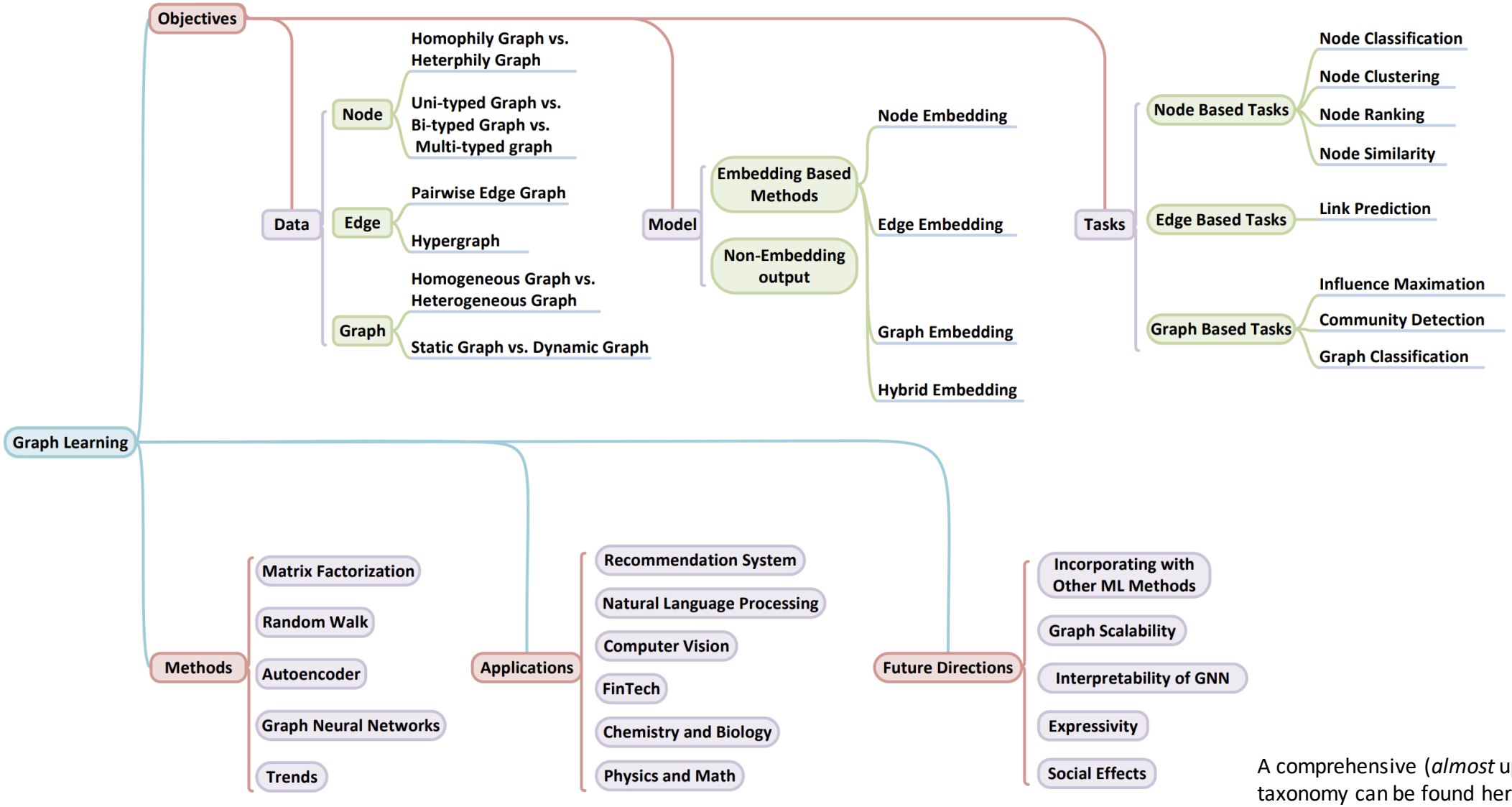
What about Graph-level tasks?

To perform a graph-level task, such as classifying graphs, we need to aggregate information across nodes.

A simple way to do this is to perform a simple **aggregation step where we aggregate all of the vectors associated with each node into a single vector associated with the entire graph**.



A taxonomy of Graph Learning



A comprehensive (*almost* up-to-date) taxonomy can be found here: [15]

Case Study

An application to COVID-19 host genetics analysis

An application to COVID-19 host genetics analysis

Objective: identifying high-order interactions of Genetic Variants associated to organs affected by negative side-effects of COVID-19 during acute phase of the disease of severe patients.

Study sample:

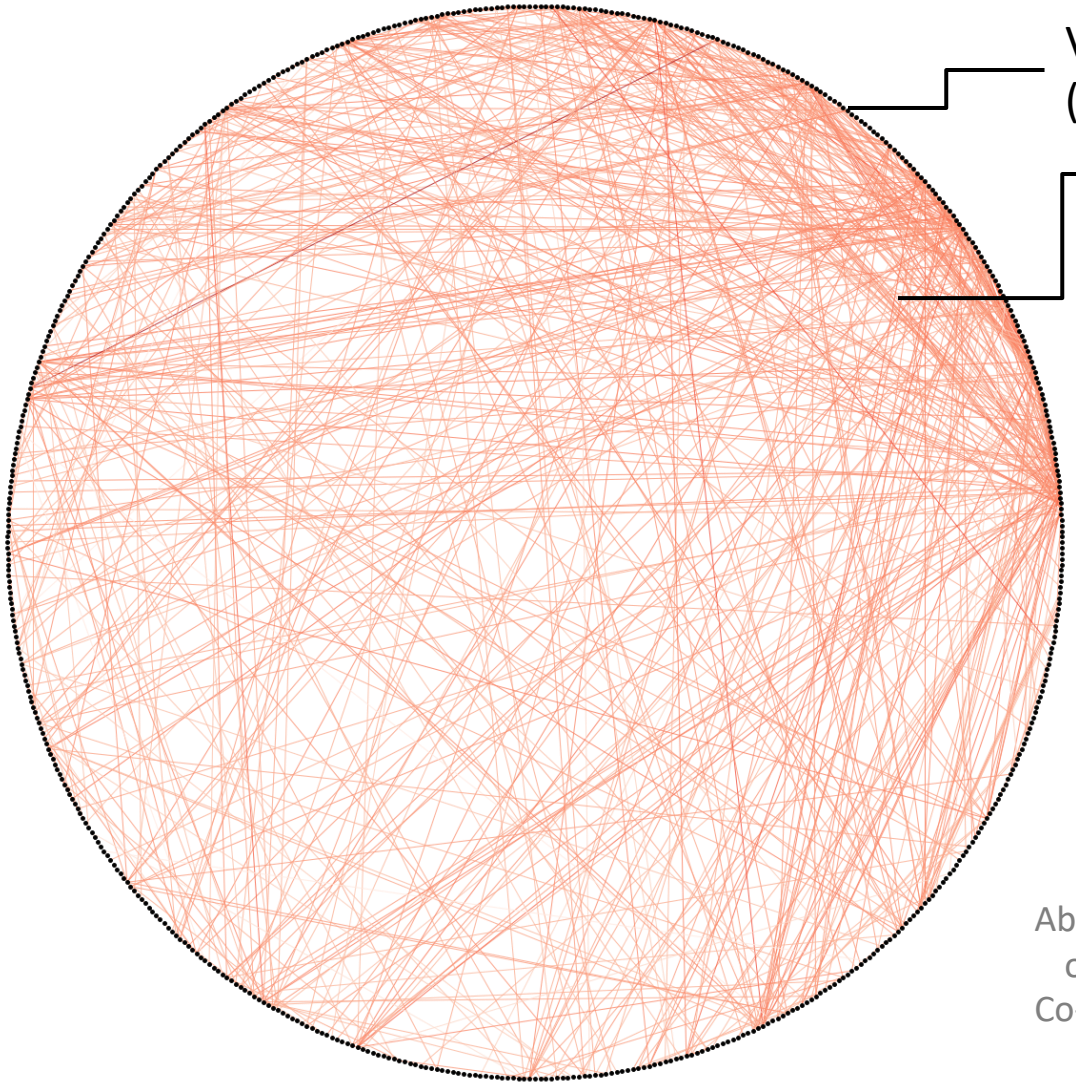
- **1,118** Female hospitalized for COVID-19. Of which, **431 severe inpatients.**
- **2,787 rare and low-frequency variants (Boolean features)**
  i.e. Minor Allele Frequency (MAF) between 0.1% and 5% ← **Extremely sparse data**
- **10 organs involved by acute phase**

Organ Involved	Count (severe)
hearth_A	246.0
hearth_B	45.0
hearth_T	41.0
kidney	85.0
liver1	41.0
liver2	94.0
lymphoid_C	17.0
lymphoid_N	382.0
taste_Hypogeusia	37.0
taste_Hypousia	34.0

Overview of the methodology:

1. Construct the network of differentially associated variants in the severe population
2. Construct an *heterogeneous* network of differentially associated variants and organs in the severe population
3. Identify groups of interacting variants associated to each organ

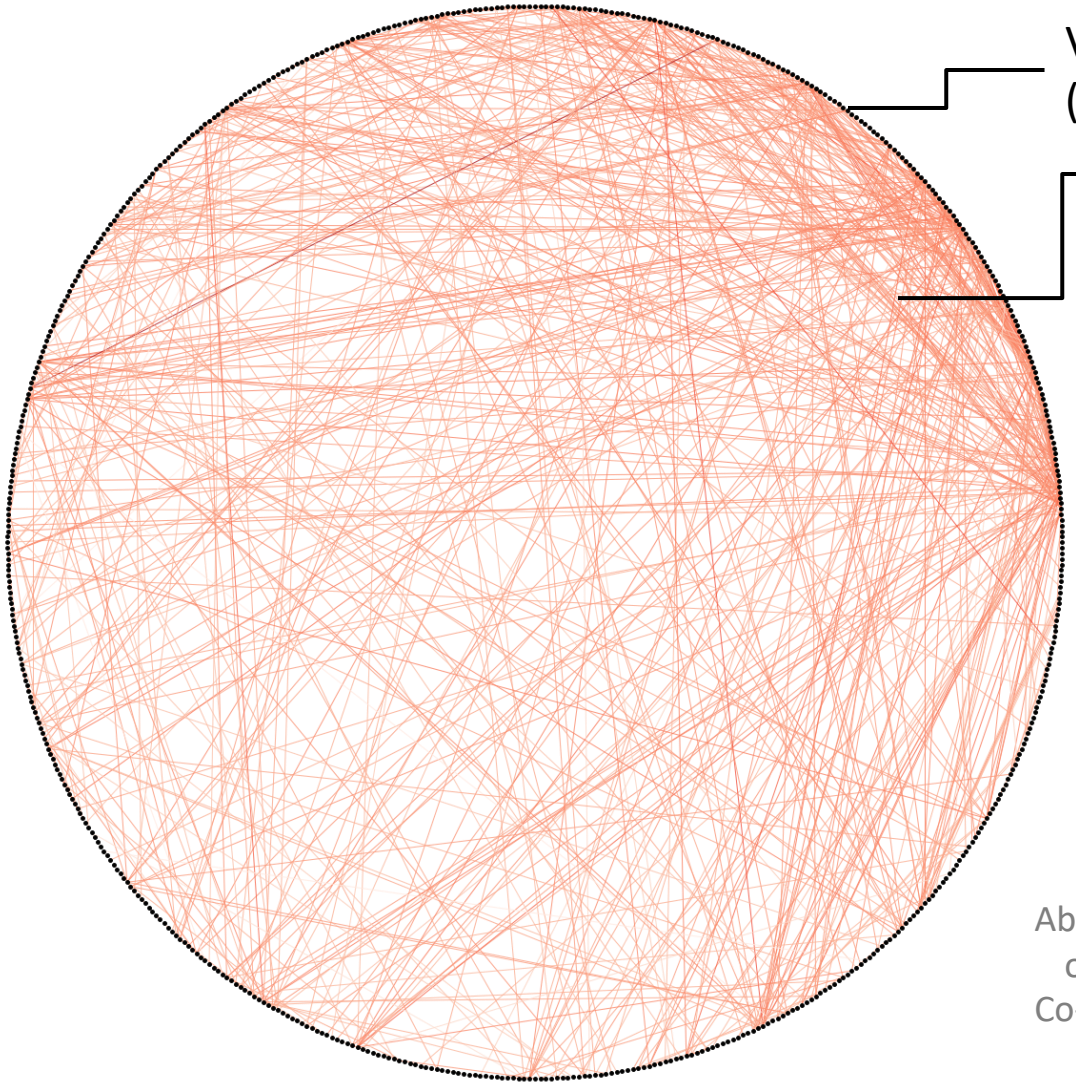
1. Construct the network of differentially associated variants in the severe population



DIFFERENTIAL NETWORK IN A NUTSHELL

Links in the network are those with a **co-occurrence** value that is **significantly higher in the severe population** w.r.t. all other patients.

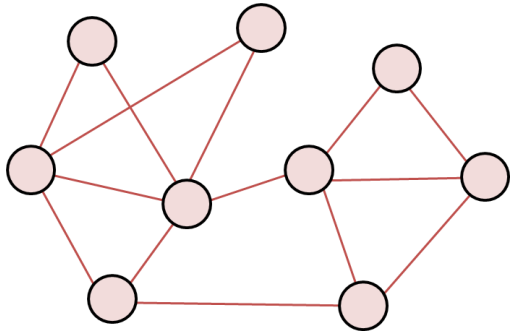
1. Construct the network of differentially associated variants in the severe population



DIFFERENTIAL NETWORK IN A NUTSHELL

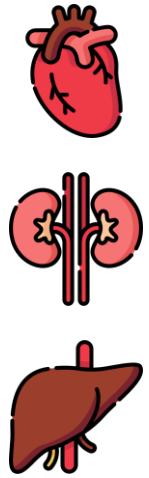
Links in the network are those with a **co-occurrence** value that is **significantly higher in the severe population** w.r.t. all other patients.

...So now we have:



Gene Variants (Boolean) and their interactions

Organs involved (Boolean)



2. Construct an *heterogeneous* network of variants and organs

An heterogeneous network is a network with different **types** of nodes.

Network construction:

Links between organs and variants (and pairs of organs) in the network are estimated on the basis of the **(Pointwise Mutual Information)^k** association measure (with minimum quantile threshold)

Co-occurrence frequency

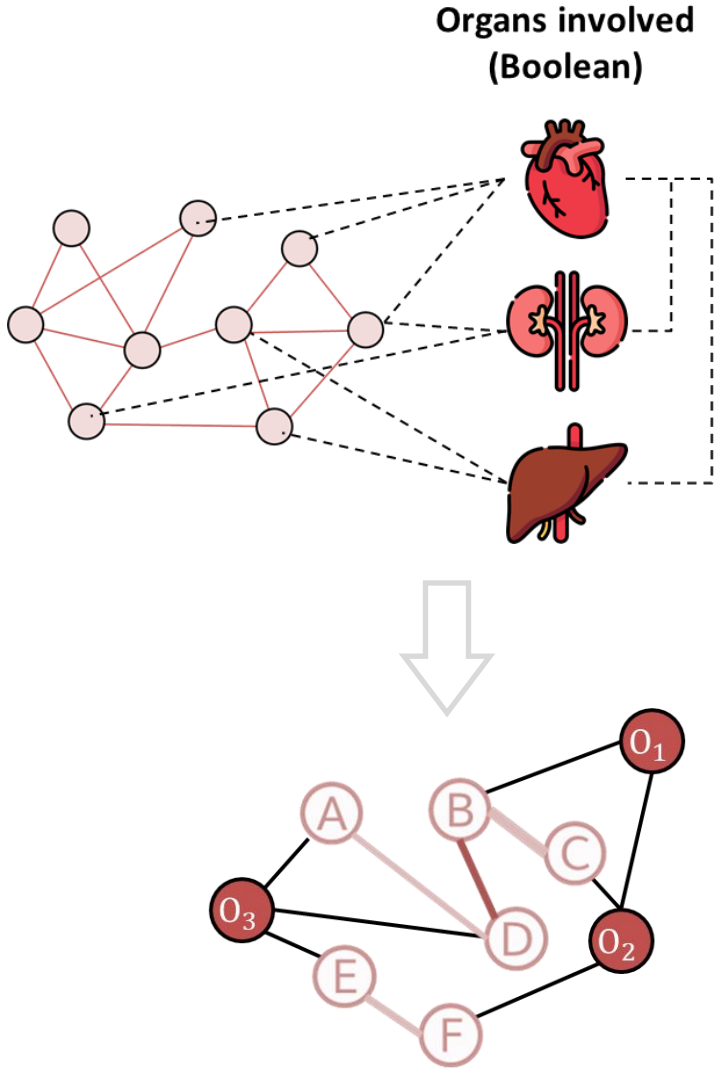
Correction term

$$\mathrm{PMI}^2(a,b) = \log \frac{p(a,b)^2}{p(a)p(b)}$$

Variant relative frequency

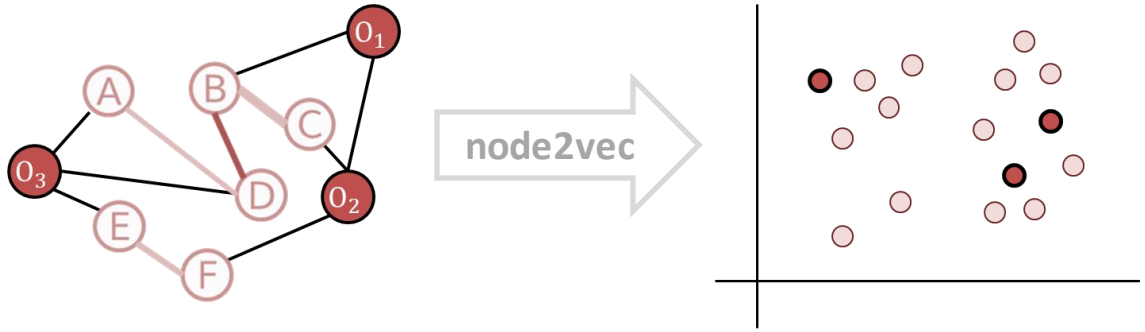
$$\mathrm{PMI}^3(a,b) = \log \frac{p(a,b)^3}{p(a)p(b)}.$$

$$\mathrm{PMI}^k(a,b) = \mathrm{PMI}(a,b) - (-(k-1)\log p(a,b)).$$



This family of association metrics comes from the text mining literature.
It highlights relevant associations between potentially very rare terms (variants).

3. Identify groups of interacting variants associated to each organ



In our Case Study:

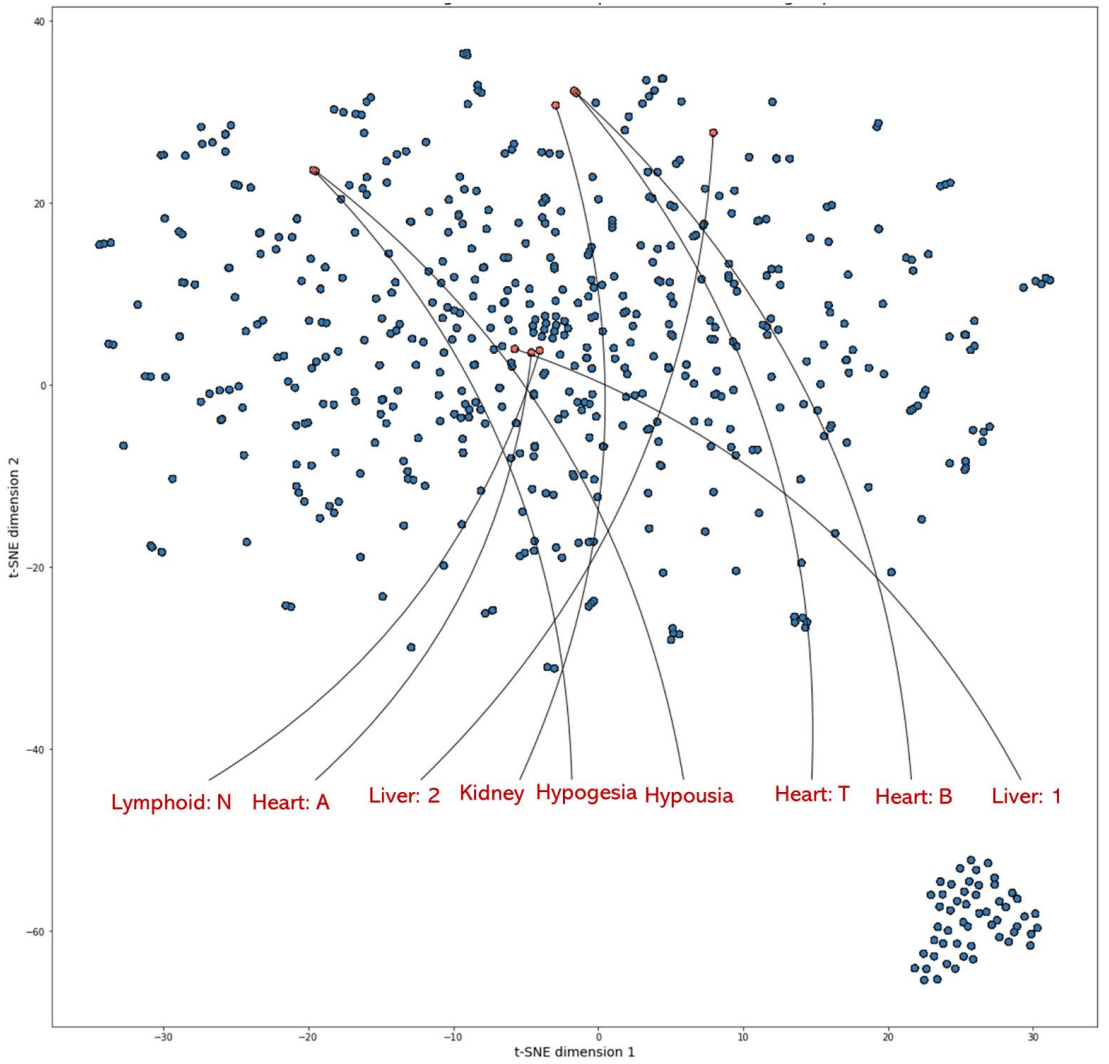
Heterogeneous Network construction

- PMI^3 metric (t = 0.99)

→ 613 nodes and 1726 edges

Once constructed the heterogeneous network, we map it to the embedding space using node2vec.

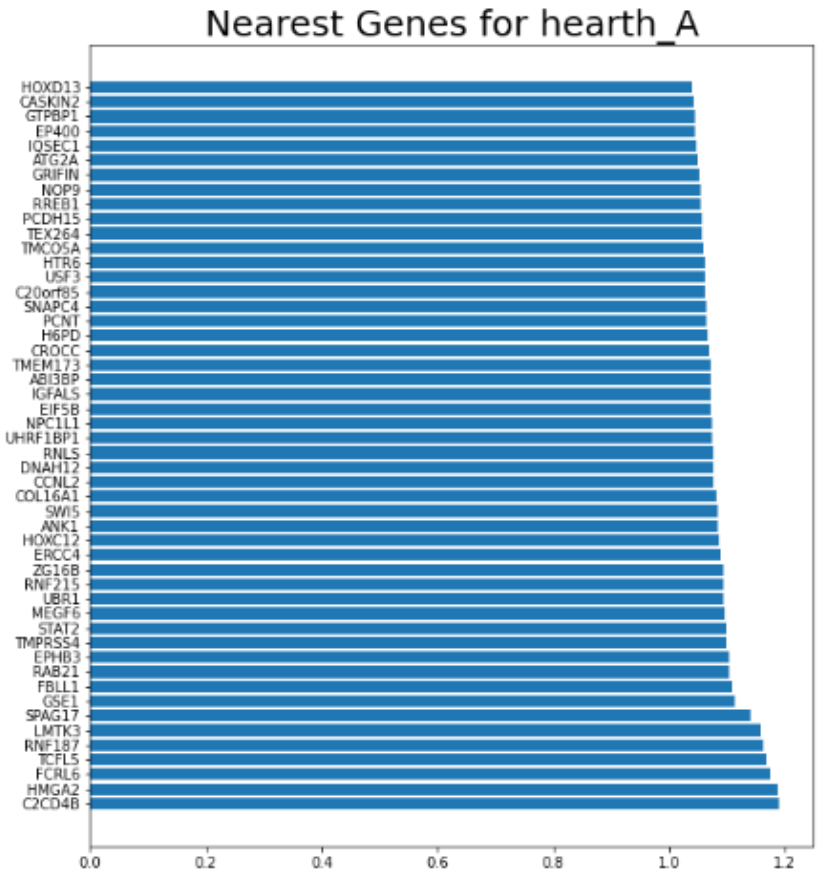
- 100 Random Walks per node
- RW length: 50 nodes
- p = 2; q = 0.5 (**global view** of this very large network)

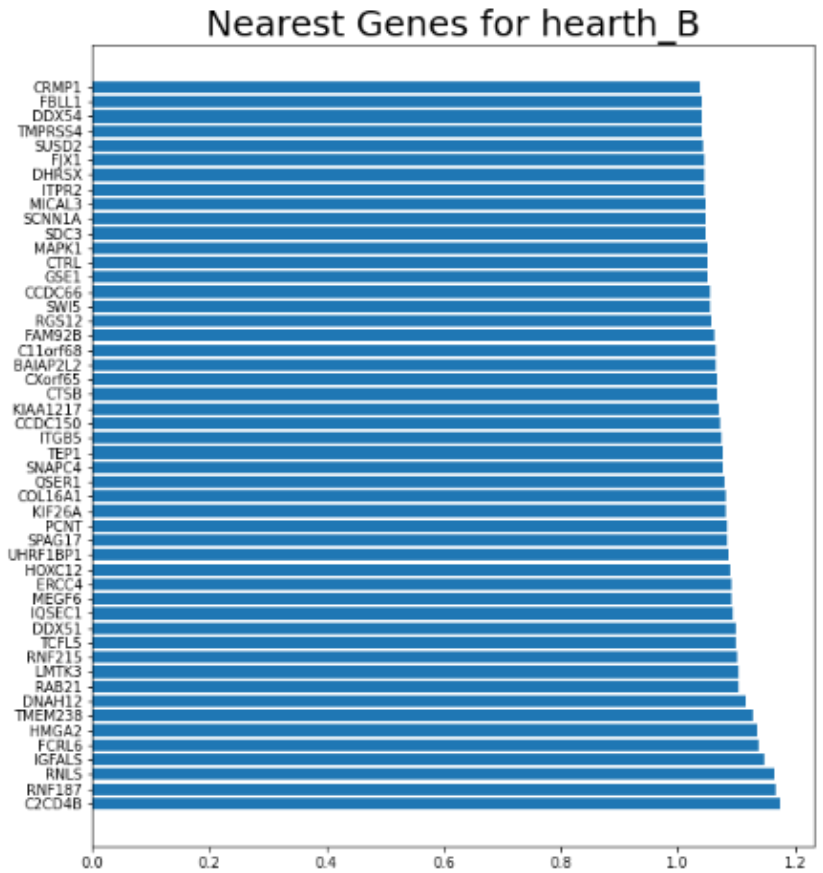


3. Identify groups of interacting variants associated to each organ

Finally…

- We compute the organ-to-variant distance matrices
- We group the k-closest variants to each organ.





To sum up today's lecture…

"Data Is In A Relationship, It's Complicated."

A key discovery of network science is that the architecture of networks emerging in various domains of science, nature, and technology are similar to each other, a consequence of being governed by the same organizing principles.

Consequently **we can use a common set of mathematical tools to explore these systems**. [Barabàsi]

Irrespectively of your domain of application, this lecture should have provided you with the basic tools to

- **Construct a network/graph from data**, identifying entities and connections
- **Analyze the characteristics of** entities (**nodes**), identifying the most relevant ones (with relevance depending on your context)
- **Analyze the topological structure** of the network, and identify structural groups of entities (**communities**)
- **Classify nodes**, or predict their connections, exploiting **representation learning** methods from Machine Learning literature.

Thank you

Michela Carlotta Massi
michelacarlotta.massi@polimi.it
michela.massi@fht.org

References

General Useful Material (books and websites)

- Network Science, Barabasi (book). Read online: http://networksciencebook.com/
- Trudeau, Richard J. *Introduction to graph theory*. Courier Corporation, 2013. Van Steen, M. (2010).
- Graph theory and complex networks. *An introduction*, *144*.
- Michael M. Bronstein, Joan Bruna et al., Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges (2021). Read online: https://geometricdeeplearning.com/

Software:

Networkx: https://networkx.org/ **[Python]**

Stellargraph: https://stellargraph.readthedocs.io/en/stable/ **[Python]**

Igraph: https://igraph.org/ **[R, Python]**

References

[1] Qiao, Lishan, et al. "Data-driven graph construction and graph learning: A review." *Neurocomputing* 312 (2018): 336-351

[2] Mixing patterns in networks, M. E. J. Newman, arXiv:cond-mat/0209450

[3] *Application of Graph Theory for Identifying Connectivity Patterns in Human Brain Networks: A Systematic Review.*

[4] Girvan M. and Newman M. E. J., Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99, 7821–7826 (2002)

[4] Fast algorithm for detecting community structure in networks, Phys. Rev. E, 69 (6) (2004), p. 066133

[6] S. Boettcher, A.G. Percus, Extremal optimization for graph partitioning, Phys. Rev. E, 64 (2) (2001), p. 026114

[7] M. Chen, K. Kuzmin, B.K. Szymanski, Community detection via maximization of modularity and its variants, IEEE Trans. Comput. Soc. Syst., 1 (1) (2014), pp. 46-65

[8] J. Liu, T. Liu, Detecting community structure in complex networks using simulated annealing with k-means algorithms, Phys. Stat. Mech. Appl., 389 (11) (2010), pp. 2300-2309

[9] M. Tasgin, A. Herdagdelen, H. Bingol, Community Detection in Complex Networks Using Genetic Algorithms, arXiv preprint arXiv:0711.0491, (2007)

[10] G. Palla, I. Derényi, I. Farkas, T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society, Nature, 435 (7043) (2005), pp. 814-818

References

[11] Muhammad Aqib Javed et al., Community detection in networks: A multidisciplinary review, Journal of Network and Computer Applications, 2018

[12] Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proc. of the 20th ACM SIGKDD*. ACM, 2014.

[14] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proc. of the 22nd ACM SIGKDD*. ACM, 2016.

[15] Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., & Murphy, K. (2022). Machine learning on graphs: A model and comprehensive taxonomy. Journal of Machine Learning Research, 23(89), 1-64.

[16] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781