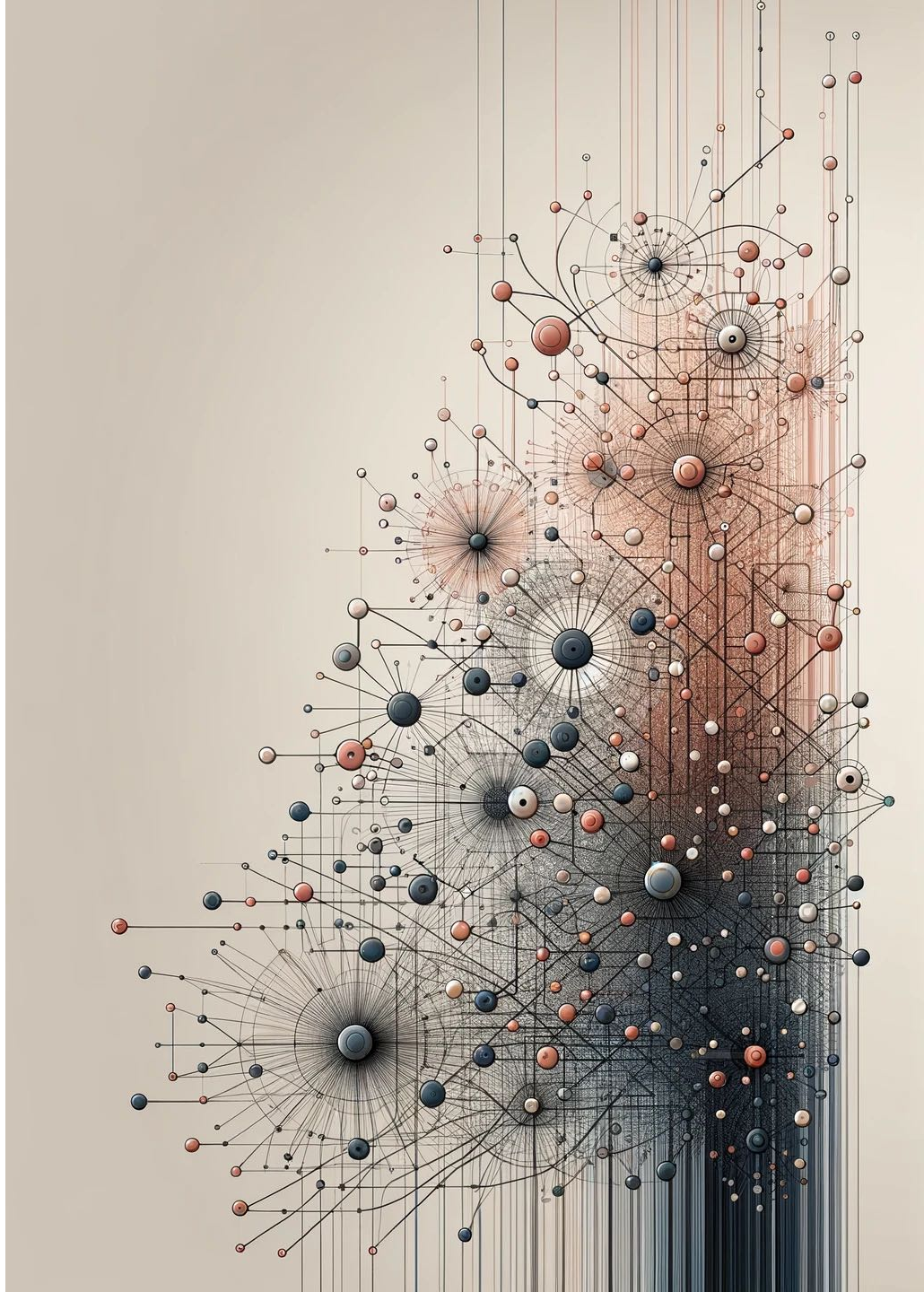
UNIVERSITÀ DI TORINO

Analisi e Visualizzazione delle Reti Complesse

NS02 - Recap on Graphs

Prof. Rossano Schifanella



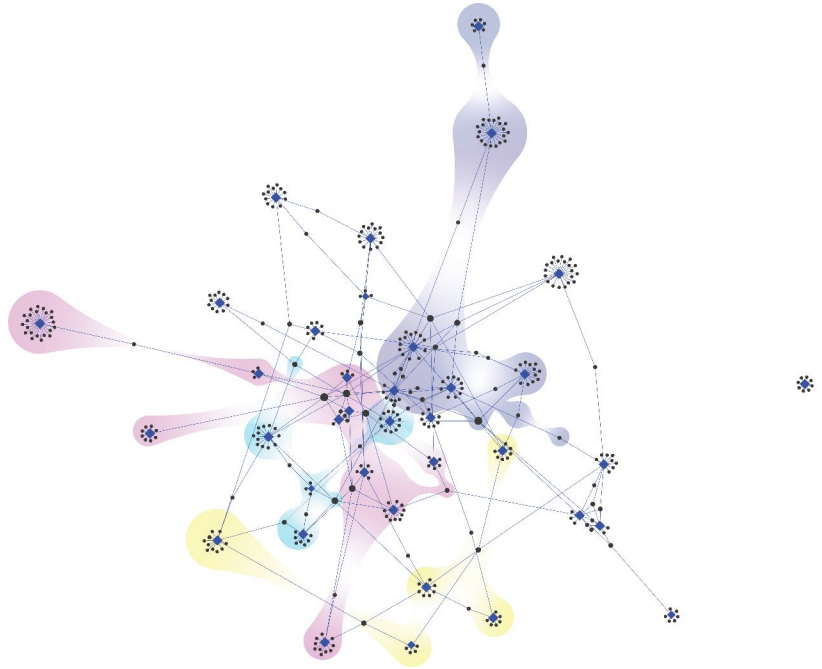
We  graphs

Agenda

- Basic definitions of graphs
- Paths and Connectivity
- Distance and BFSearch
- Small world phenomenon

Why graphs?

- Graphs are a **mathematical model** that helps to represent complex systems in terms of **nodes** and **links**
- We need a **language** to understand the essential elements of a network
- With a language, we will be able to talk appropriately about:
  - **properties** that characterize the structure and behavior of networks
  - **roles** of networks in affecting **processes** occurring on network structures



Basic definitions

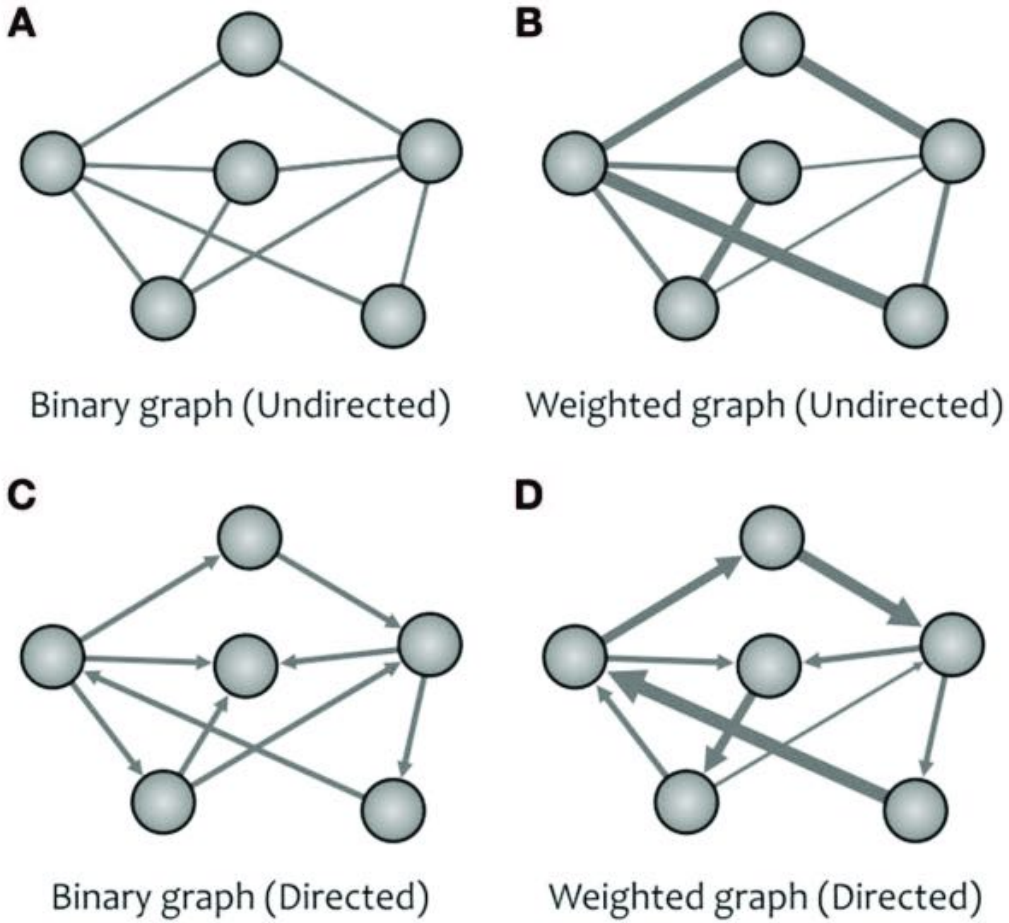
A **graph** is composed of nodes and links $G = (N, L)$

nodes (or vertices) $N = \{n_1, n_2, \ldots, n_l\} = \{1, 2, \ldots, l\}$

links (or edges or arcs) $L = \{(i, j) : i, j \in N\}$

Graphs can be **directed** or **undirected** i, j 

Graphs can be **weighted** or **unweighted** $w_{ij}\ (i, j, w)$


A
B
Binary graph (Undirected)
Weighted graph (Undirected)
C
D
Binary graph (Directed)
Weighted graph (Directed)

Introduction to NetworkX

- **NetworkX** is a Python library for the creation, manipulation, and study of complex networks of nodes and edges.
- It was created in 2005 by Aric Hagberg, Dan Schult, and Pieter Swart.
- NetworkX is designed to handle the analysis of large-scale complex networks.

Key Features

- Supports standard and nonstandard network algorithms.
- Provides tools to study the structure and dynamics of complex networks.
- Easily integrates with other scientific libraries like NumPy, SciPy, and Matplotlib.

Useful Links

- NetworkX Documentation
- NetworkX GitHub Repository
- NetworkX Tutorials

Alternative Frameworks to NetworkX

1. igraph

- **Pros:**
  - Highly efficient for large graphs.
  - Rich set of algorithms.
  - Interfaces available for R, Python, and C/C++.
- **Cons:**
  - Less intuitive API compared to NetworkX.
  - Limited support for dynamic graphs.

2. Graph-tool

- **Pros:**
  - Extremely fast due to C++ backend.
  - Supports parallel computation.
  - Extensive visualization capabilities.
- **Cons:**
  - Steeper learning curve.
  - Requires additional dependencies (Boost C++ libraries).

3. SNAP (Stanford Network Analysis Project)

- **Pros:**
  - Optimized for performance and scalability.
  - Supports large-scale network analysis.
  - Available for C++ and Python.
- **Cons:**
  - Limited documentation and community support.
  - Less flexible API.

Summary

NetworkX	Easy to use, well-documented	Slower for very large graphs
igraph	Efficient, rich algorithms	Less intuitive API
Graph-tool	Fast, parallel computation	Steeper learning curve
SNAP	Performance, scalability	Limited documentation

Creating graphs in NetworkX

NetworkX offers several ways to create different types of graphs

```
import networkx as nx
```

Undirected Graph

```
G = nx.Graph()  # Create empty graph
G.add_node(1)   # Add single node
G.add_nodes_from([2, 3])  # Add multiple nodes
G.add_edge(1, 2)  # Add edge between nodes
G.add_edges_from([(2, 3), (1, 3)])  # Add multiple edges
```

Add direction to the edges

```
D = nx.DiGraph()
D.add_edges_from([('B', 'C'), ('C', 'A')])
```

Add weight to the edges

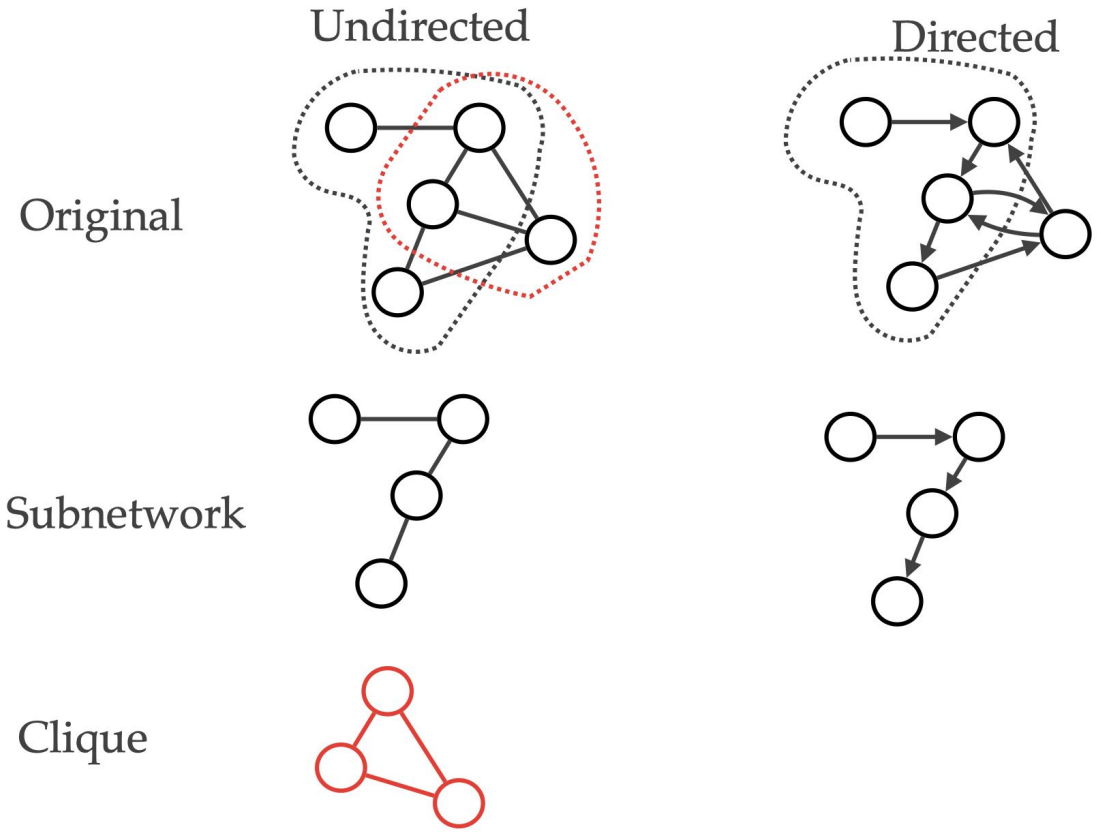
```
W = nx.Graph()
W.add_edge('X', 'Y', weight=5.0)
W.add_edge('Y', 'Z', weight=3.5)
```

Documentation

- Tutorial: Creating and Manipulating Graphs
- Graph Class Overview

Subnetworks

- A **subnetwork** is a network obtained by selecting a subset of the nodes and all of the links among these nodes
- A **clique** is a complete subnetwork

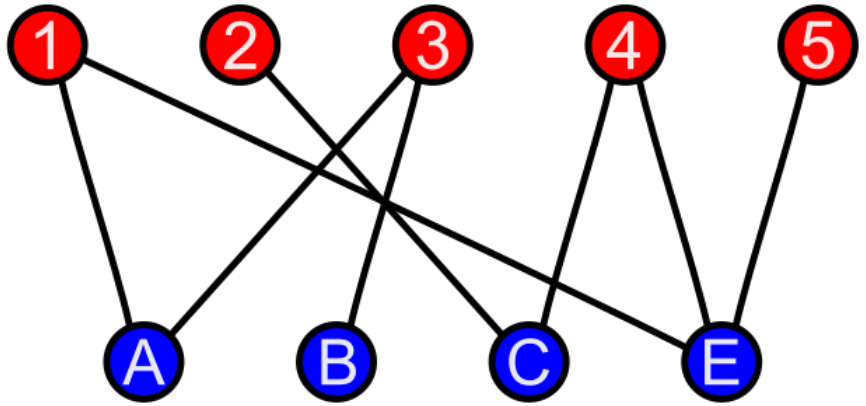


Documentation

- `networkx.find_cliques()`
  - Returns all maximal cliques in an undirected graph
  - A maximal clique is a clique that cannot be extended by including one more adjacent vertex
- `networkx.subgraph()`
  - Returns a subgraph of the graph containing the specified nodes
  - Preserves all edges between nodes that are in the subset
- `networkx.ego_graph()`
  - Returns the induced subgraph of neighbors centered at a specific node
  - Can specify a radius to include nodes up to n-steps away
- `networkx.clique_number()`
  - Returns the size of the largest maximal clique in the graph

Bipartite graph

- **Two types of nodes**
- Connections only happen between nodes of different type
- Example: actor, movie dataset



NetworkX: Bipartite Graphs

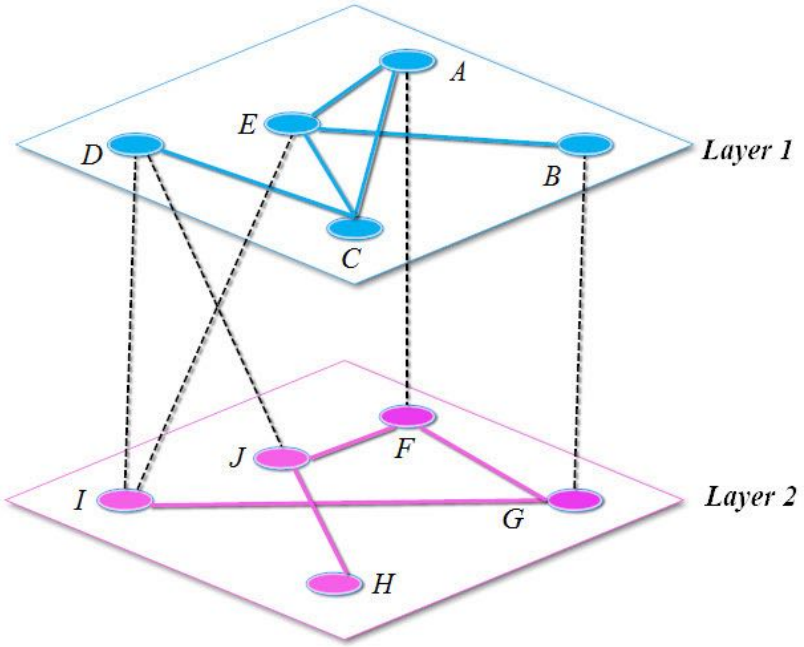
```
# Create bipartite graph
B = nx.Graph()

# Add nodes with their bipartite set
B.add_nodes_from(['A1', 'A2', 'A3'], bipartite=0)  # First set
B.add_nodes_from(['B1', 'B2'], bipartite=1)        # Second set

# Add edges (only between different sets)
B.add_edges_from([('A1', 'B1'), ('A2', 'B1'), ('A3', 'B2')])
```

Documentation

- Bipartite module

Multilayer networks

- A network can have **multiple layers**, each with its own nodes and edges
    - Example: air transportation networks of distinct airlines, with some but not complete overlap of airport nodes
- **Intralayer links** among nodes in the same layer, **interlayer links** across layers
- If the sets of nodes in the different layers are identical, we call the network a multiplex; interlayer links are **couplings** linking the same node across layers
    - Example: layers to represent different types of relationships in a social network, such as friendship, family ties, coworkers, etc.



Networks of networks

- Generally, each layer in a multilayer network can have its nodes and edges. We call this a **network of networks**.
    - Examples: the electrical power grid, the Internet.
- The **interlayer links** capture network interactions in the different layers.

  For example, power stations communicate via the Internet, and the power grid powers Internet routers.
- **Cascading failures** are one type of unpredictable vulnerabilities that arise in networks of networks.
    - **Example:** If a power station fails, it can disrupt the Internet routers it powers, which in turn can affect other power stations that rely on those routers for communication, leading to a cascading failure across both networks.

- **Interdependencies** between different networks can lead to complex behaviors and vulnerabilities that are not present in isolated networks.
    - **Example:** In a transportation network, delays in one airline's network can propagate to other airlines' networks due to shared airports and resources.
- **Modeling and analyzing** these networks of networks can help in understanding and mitigating risks associated with their interdependencies.
    - **Example:** Researchers use multilayer network models to study the resilience of infrastructure networks to natural disasters, ensuring that critical services remain operational even when parts of the network are disrupted.

NetworkX and Multilayer Networks

- NetworkX does not have built-in support for multilayer networks, but you can use additional packages like **Pymnet** or **MultinetX** to handle multilayer networks.

Pymnet

- **Pymnet** is a Python library specifically designed for multilayer networks.
- It provides tools for creating, manipulating, and visualizing multilayer networks.
  - **Documentation:** Pymnet Documentation

MultinetX

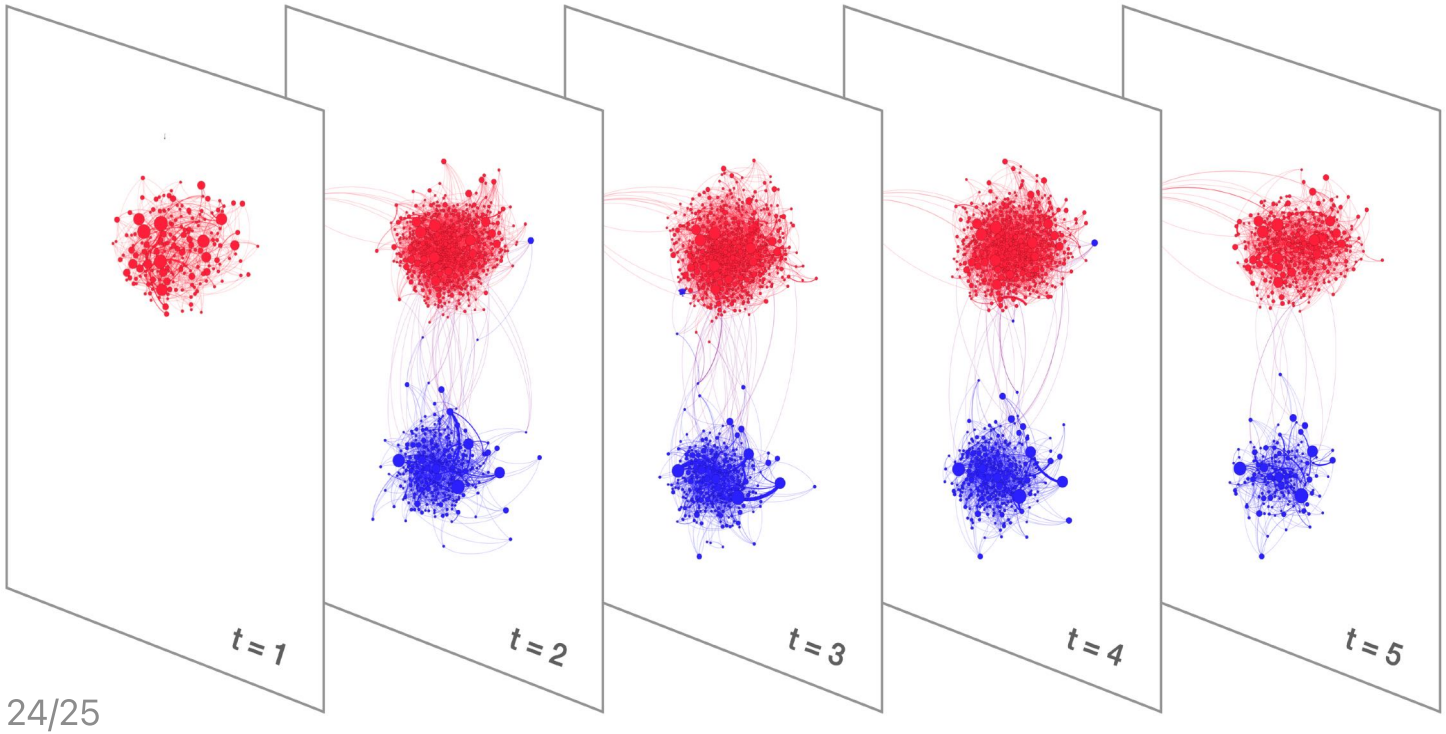
- **MultinetX** is another Python package for multilayer networks.
- It extends NetworkX to support multilayer network structures.
  - **Documentation:** MultinetX Documentation

Example: Creating a Multilayer Network with MultinetX

```
import multinetx as mx

# Create a multilayer network
G = mx.MultilayerGraph()

# Add nodes and edges for different layers
G.add_layer('Layer 1')
G.add_layer('Layer 2')

G.add_node(1, layer='Layer 1')
G.add_node(2, layer='Layer 1')
G.add_edge(1, 2, layer='Layer 1')

G.add_node(3, layer='Layer 2')
G.add_node(4, layer='Layer 2')
G.add_edge(3, 4, layer='Layer 2')

# Add interlayer edges
G.add_edge(1, 3, layer='Interlayer')
G.add_edge(2, 4, layer='Interlayer')
```

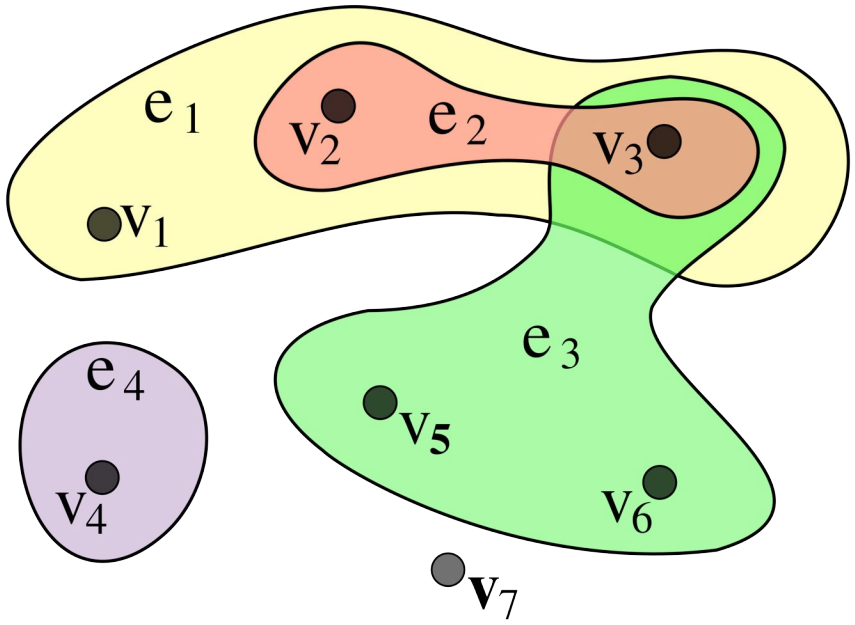
Temporal networks

- A temporal network is a multilayer network in which the layers represent links at different times (temporal snapshots)
  - Example: a Twitter retweet network



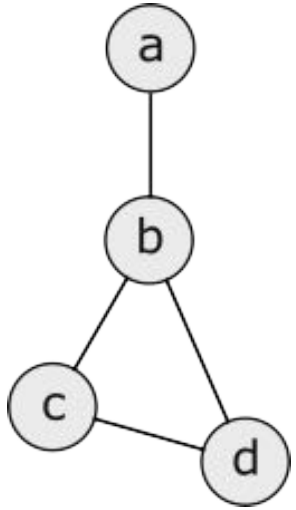
Hypergraphs

- **Interaction may occur between multiple nodes simultaneously**
- The hypergraph is represented by a sequence of **hyperedges** of size greater or equal to 2
  - For example: $E = (v_2, v_3), (v_1, v_2, v_3), (v_4), (v_3, v_5, v_6)$
  - co-authorship network



Neighbors

$$N_a = \{b\}$$
$$N_b = \{a, c, d\}$$
$$N_c = \{b, d\}$$
$$N_d = \{b, c\}$$

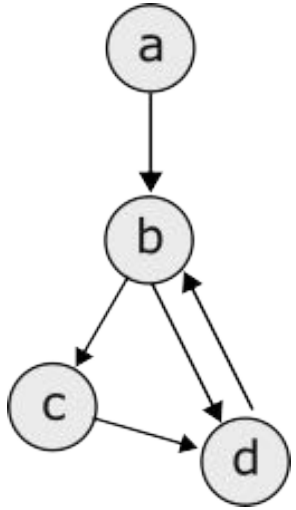


Successors

$S_a = \{b\}$
$S_b = \{c, d\}$
$S_c = \{d\}$
$S_d = \{b\}$

Predecessors

$P_a = \{\}$
$P_b = \{a, d\}$
$P_c = \{b\}$
$P_d = \{b, c\}$



Degree

Number of links (or neighbors)

$i \rightarrow N_i \qquad k_i = |N_i|$ **degree**

Singleton: a node whose degree is zero

$N_i = \{\}, k_i = 0$

In directed networks:

$k_i^{in} = |P_i|$ **in-degree**

$k_i^{out} = |S_i|$ **out-degree**

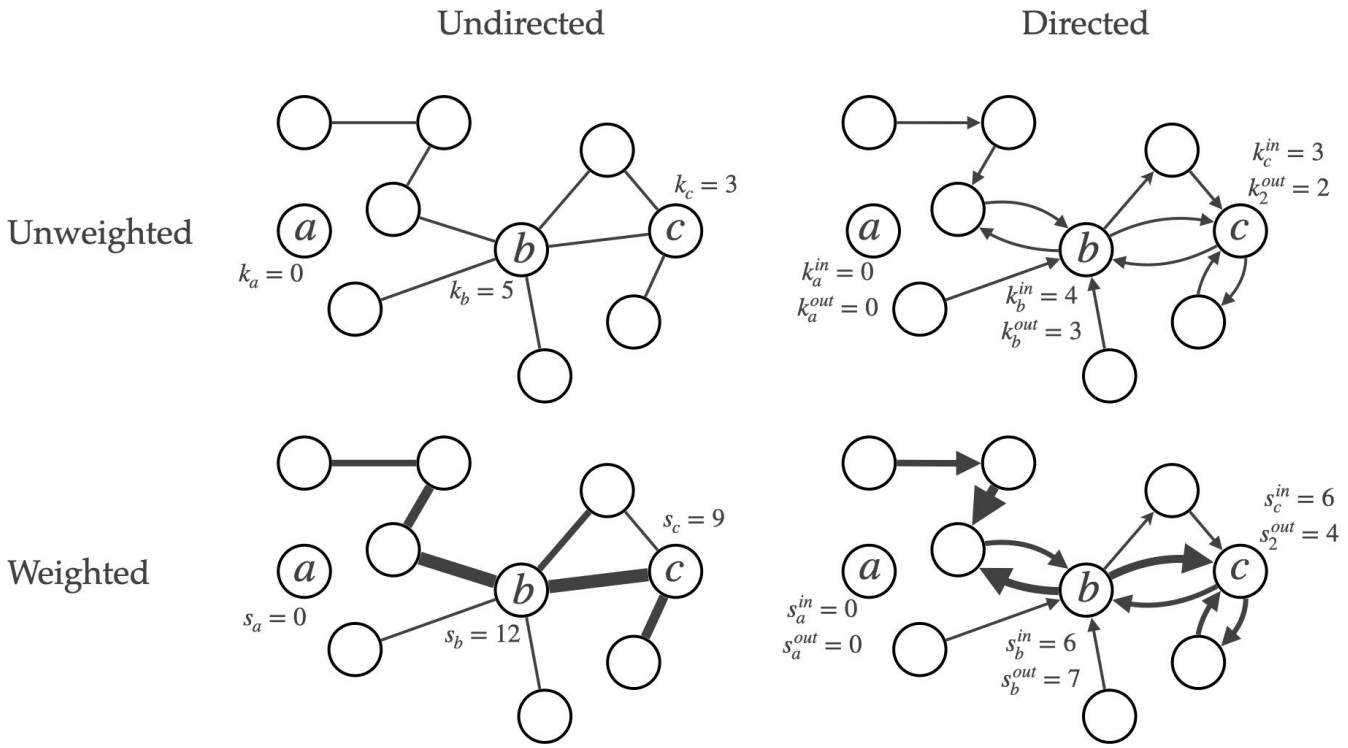
$k_i = k_i^{in} + k_i^{out}$

Strength

strength or **weighted-degree** $s_i = \sum_{j \in N_i} w_{ij}$

in-strength $s_i^{in} = \sum_{j \in P_i} w_{ji}$

out-strength $s_i^{out} = \sum_{j \in S_i} w_{ij}$



UNIVERSITÀ DI TORINO

NetworkX: Node Adjacency

```
# Get neighbors (undirected)
G.neighbors(node)       # Returns iterator of neighbors
list(G.neighbors(node)) # Returns list of neighbors

# Get predecessors (directed)
G.predecessors(node)    # Returns iterator of predecessors

# Get successors (directed)
G.successors(node)     # Returns iterator of successors

# Get degree
G.degree(node)          # Returns degree of node
G.in_degree(node)       # Returns in-degree (directed)
G.out_degree(node)      # Returns out-degree (directed)

# Get strength
G.degree(node, weight="weight")          # Returns strength of node
```

Density and sparsity

Network size = number of nodes	$\lvert N \rvert$ or, simply N
Number of links	$\lvert L \rvert$ or, simply L

In an undirected network:

Maximum possible number of links	$L_{max} = \binom{N}{2} = \frac{N(N-1)}{2}$
Density	$d = \frac{L}{L_{max}} = \frac{2L}{N(N-1)}$
Sparsity if $d \ll 1 \Rightarrow sparse$	$\begin{cases} \text{sparse:} & L \approx N \\ \text{dense:} & L \approx N^2 \end{cases}$

In a directed network:

Maximum possible number of links	$L_{max} = N(N-1)$
Density	$d = \frac{L}{L_{max}} = \frac{L}{N(N-1)}$

UNIVERSITÀ DI TORINO

Example: Facebook

- Rough orders-of-magnitude approximations:
- $N \approx 10^9$
- $L \approx 10^3 * N$
- $d = \frac{2L}{N(N-1)} \approx \frac{L}{N^2} = \frac{10^3 N}{N^2} = \frac{10^3}{10^9} = 10^{-6} \ll 1$

Table 1.1 Basic statistics of network examples. Network types can be (D)irected and/or (W)eighted. When there is no label the network is undirected and unweighted. For directed networks, we provide the average in-degree (which coincides with the average out-degree).

Network	Type	Nodes (N)	Links (L)	Density (d)	Average degree ($\langle k \rangle$)
Facebook Northwestern Univ.		10,567	488,337	0.009	92.4
IMDB movies and stars		563,443	921,160	0.000006	3.3
IMDB co-stars	W	252,999	1,015,187	0.00003	8.0
Twitter US politics	DW	18,470	48,365	0.0001	2.6
Enron Email	DW	87,273	321,918	0.00004	3.7
Wikipedia math	D	15,220	194,103	0.0008	12.8
Internet routers		190,914	607,610	0.00003	6.4
US air transportation		546	2,781	0.02	10.2
World air transportation		3,179	18,617	0.004	11.7
Yeast protein interactions		1,870	2,277	0.001	2.4
C. elegans brain	DW	297	2,345	0.03	7.9
Everglades ecological food web	DW	69	916	0.2	13.3

Low-Density Networks

- **World Wide Web (WWW):**
  - Nodes: Web pages
  - Links: Hyperlinks between pages
  - **Density:** Very low, typically around 10^{-6} to 10^{-5}, as most web pages link to only a few other pages out of billions.
- **Social Networks:**
  - Nodes: Individuals
  - Links: Friendships or connections
  - **Density:** Generally low, around 10^{-4} to 10^{-3}, as individuals typically have a limited number of connections compared to the total number of users.
- **Protein-Protein Interaction Networks:**
  - Nodes: Proteins
  - Links: Interactions between proteins
  - **Density:** Low, around 10^{-3} to 10^{-2}, as each protein interacts with a small subset of possible proteins.

High-Density Networks

- **Fully Connected Networks (Cliques):**
  - Nodes: Any set of entities
  - Links: Every possible pair of nodes is connected
  - **Density:** Maximum density, $d = 1$, as every node is connected to every other node.
- **Small Social Groups:**
  - Nodes: Members of a small group (e.g., a family or close-knit team)
  - Links: Relationships or interactions
  - **Density:** High, often close to $d = 1$, as most members are connected to each other.
- **Local Area Networks (LANs):**
  - Nodes: Computers or devices within a local network
  - Links: Direct connections or communication links
  - **Density:** High, especially in small networks where most devices are interconnected, typically 0.5 to 1.

Average degree

$$\langle k \rangle = \frac{\sum_{i \in N} k_i}{N}$$

In an undirected network:

$$\langle k \rangle = \frac{2L}{N}$$

Given

$$d = \frac{2L}{N(N-1)} \Rightarrow L = \frac{dN(N-1)}{2}$$

$$\langle k \rangle = \frac{2L}{N} = \frac{dN(N-1)}{N} = d(N-1)$$

The average degree is also connected to the density d

$$d = \frac{\langle k \rangle}{N-1} = \frac{\langle k \rangle}{k_{max}}$$

NetworkX: Density and Average Degree

```python
# Compute density
density = nx.density(G)
print("Network density:", density)

# Compute average degree
degrees = [d for n, d in G.degree()]
avg_degree = sum(degrees) / nx.number_of_nodes(G)
print("Average degree:", avg_degree)

# or simply
avg_degree = sum(dict(G.degree()).values()) / nx.number_of_nodes(G)
print("Average degree:", avg_degree)
```

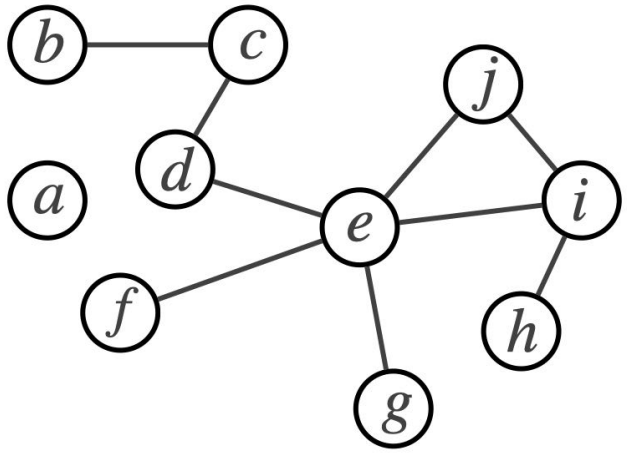
Network representations

Adjacency Matrix

$N \times N$ matrix

$$a_{ij} = \begin{cases} 0 & \text{no edge} \\ 1 & (i,j) \in L \end{cases}$$

In an undirected network: $a_{ij} = a_{ji}$



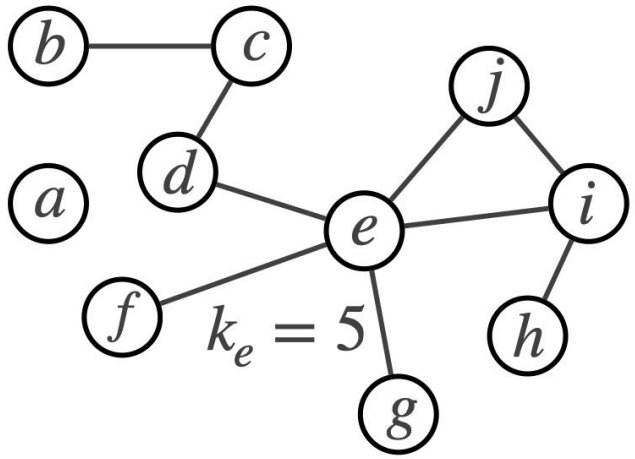
	a	b	c	d	e	f	g	h	i	j
a	0	0	0	0	0	0	0	0	0	0
b	0	0	1	0	0	0	0	0	0	0
c	0	1	0	1	0	0	0	0	0	0
d	0	0	1	0	1	0	0	0	0	0
e	0	0	0	1	0	1	1	0	1	1
f	0	0	0	0	1	0	0	0	0	0
g	0	0	0	0	1	0	0	0	0	0
h	0	0	0	0	0	0	0	0	1	0
i	0	0	0	0	1	0	0	1	0	1
j	0	0	0	0	1	0	0	0	1	0

Network representations

Adjacency Matrix

Degree

$$k_i = \sum_j a_{ij} = \sum_j a_{ji}$$



	a	b	c	d	e	f	g	h	i	j
a	0	0	0	0	0	0	0	0	0	0
b	0	0	1	0	0	0	0	0	0	0
c	0	1	0	1	0	0	0	0	0	0
d	0	0	1	0	1	0	0	0	0	0
e	0	0	0	1	0	1	1	0	1	1
f	0	0	0	0	1	0	0	0	0	0
g	0	0	0	0	1	0	0	0	0	0
h	0	0	0	0	0	0	0	0	1	0
i	0	0	0	0	1	0	0	1	0	1
j	0	0	0	0	1	0	0	0	1	0

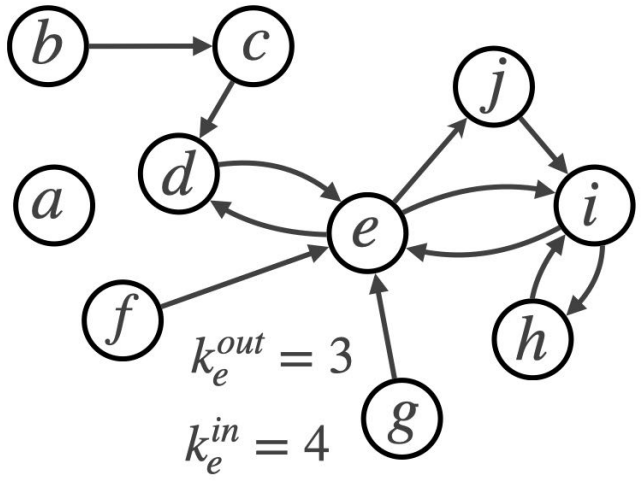
Network representations

Adjacency Matrix

In a **directed** graph the matrix is **not symmetric**

$$k_i^{out} = \sum_j a_{ij}$$

$$k_i^{in} = \sum_j a_{ji}$$



	a	b	c	d	e	f	g	h	i	j
a	0	0	0	0	0	0	0	0	0	0
b	0	0	1	0	0	0	0	0	0	0
c	0	0	0	1	0	0	0	0	0	0
d	0	0	0	0	1	0	0	0	0	0
e	0	0	0	1	0	0	0	0	1	1
f	0	0	0	0	1	0	0	0	0	0
g	0	0	0	0	1	0	0	0	0	0
h	0	0	0	0	0	0	0	0	1	0
i	0	0	0	0	1	0	0	1	0	0
j	0	0	0	0	0	0	0	0	1	0

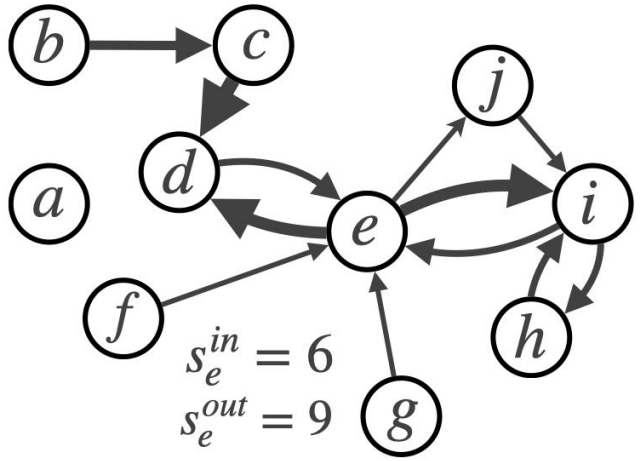
Network representations

Adjacency Matrix

In a **weighted** graph

$$a_{ij} = \begin{cases} 0 & \text{no edge} \\ w_{ij} & (i,j) \in L \end{cases}$$

$$s_i^{out} = \sum_j w_{ij}$$
$$s_i^{in} = \sum_j w_{ji}$$



	a	b	c	d	e	f	g	h	i	j
a	0	0	0	0	0	0	0	0	0	0
b	0	0	3	0	0	0	0	0	0	0
c	0	0	0	4	0	0	0	0	0	0
d	0	0	0	0	2	0	0	0	0	0
e	0	0	0	4	0	0	0	0	4	1
f	0	0	0	0	1	0	0	0	0	0
g	0	0	0	0	1	0	0	0	0	0
h	0	0	0	0	0	0	0	0	2	0
i	0	0	0	0	2	0	0	2	0	0
j	0	0	0	0	0	0	0	0	2	0

Sparse network representations

- The memory/disk storage needed by an adjacency matrix is proportional to N^2
- In sparse networks (most real-world networks), this is inefficient: most of the space is wasted storing zeros (non-links); for very large networks, adjacency matrices are unfeasible
- It is much more efficient, often necessary, to store only the actual links and assume that if a link is not listed, it means it is not present
- There are two commonly used sparse network representations:
  - **Adjacency list**
  - **Edge list**

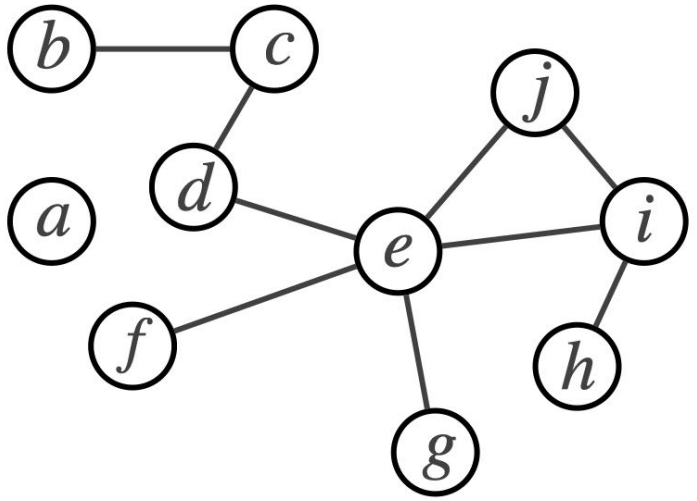
Adjacency List

Undirected network:

list each link twice

Directed network:

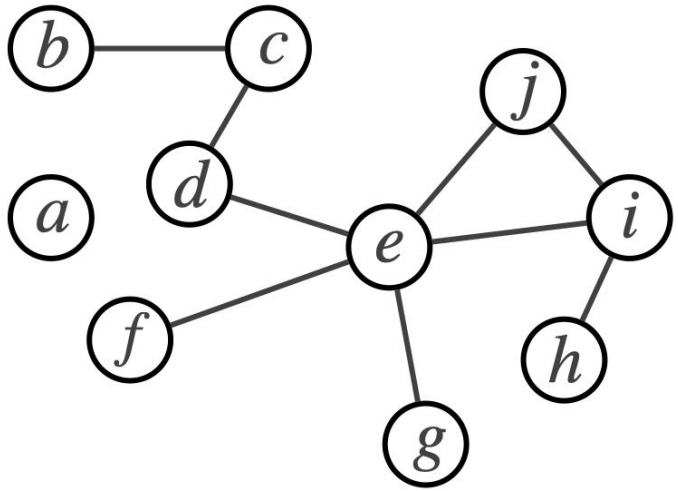
list only existing links




```
a
b    c
c    b d
d    c e
e    d f i j g
f    e
g    e
h    i
i    h j
j    e i
```

Edge list

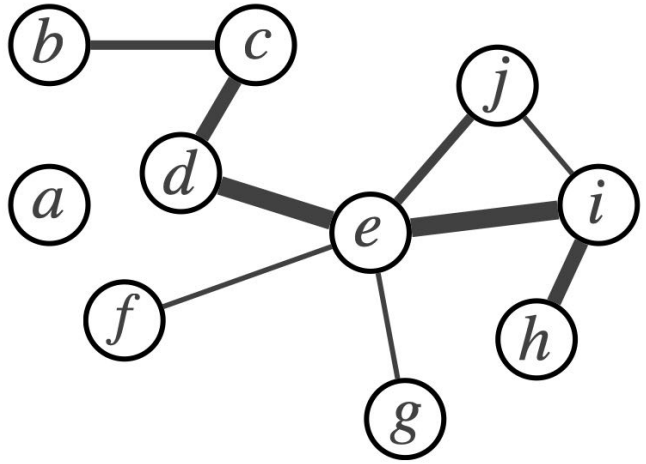
unweigthed



b	c
c	d
d	e
e	f
e	g
e	i
e	j
h	i
i	j

L

weighted



b	c	2
c	d	3
d	e	4
e	f	4
e	g	1
e	i	1
e	j	2
h	i	3
i	j	1

L

Paths and connectivity

Paths and cycles

path: $\{n_1, n_2, \ldots, n_k\}$ where $\forall i : (n_i, n_{i+1}) \in L$

A path has a **length** l

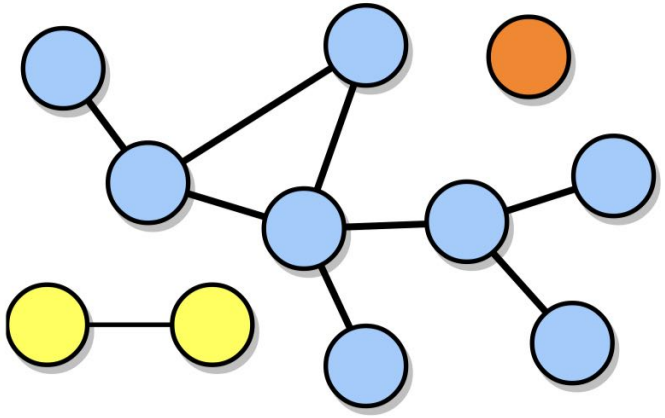
- number of edges in the path
- $l = k - 1$

if we have repeating nodes $\Rightarrow$ **cycles**

no repeating nodes $\Rightarrow$ **simple path**

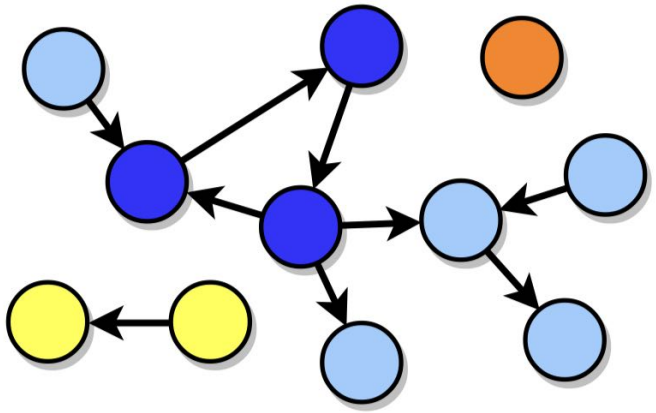
Connectedness and components

- A network is **connected** if there is a path between any two nodes
- If a network is not connected, it is **disconnected** and has multiple connected components
- A connected component is a connected subnetwork
    - The largest one is called **giant component**; it often includes a substantial portion of the network
    - A **singleton** is the smallest-possible connected component



Connectedness and components

- A directed network can be **strongly connected** or **weakly connected** if there is a path between any two nodes, respecting or disregarding the link directions, respectively
- The **in-component** of a strongly connected component S is the set of nodes from which one can reach S, but that cannot be reached from S
- The **out-component** of a strongly connected component S is the set of nodes that can be reached from S, but from which one cannot reach S

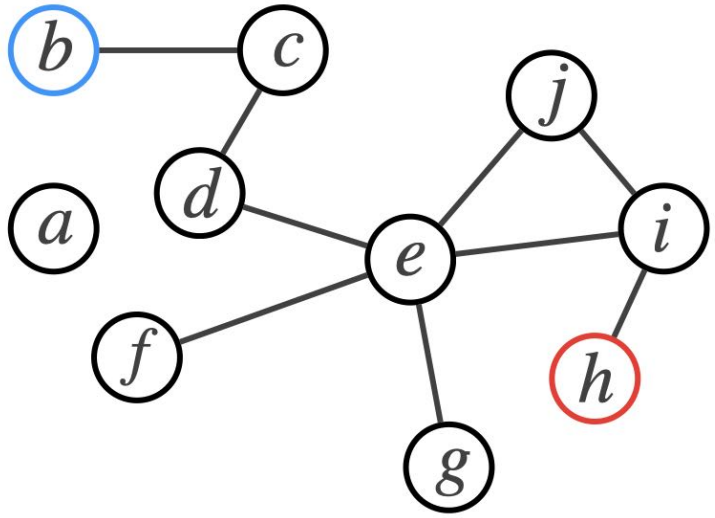


NetworkX: Connectivity Methods

```
# Check if graph is connected
nx.is_connected(G)          # Returns True if graph is connected
nx.is_strongly_connected(G) # For directed graphs
nx.is_weakly_connected(G)  # For directed graphs

# Get components
nx.connected_components(G)          # Iterator of connected components
nx.strongly_connected_components(G) # For directed graphs
nx.weakly_connected_components(G)  # For directed graphs

# Get largest component
max(nx.connected_components(G), key=len)  # Returns nodes in largest component
nx.connected_components(G).__next__()     # Alternative way

# Count components
nx.number_connected_components(G)          # Returns number of components
nx.number_strongly_connected_components(G) # For directed graphs
nx.number_weakly_connected_components(G)  # For directed graphs
```

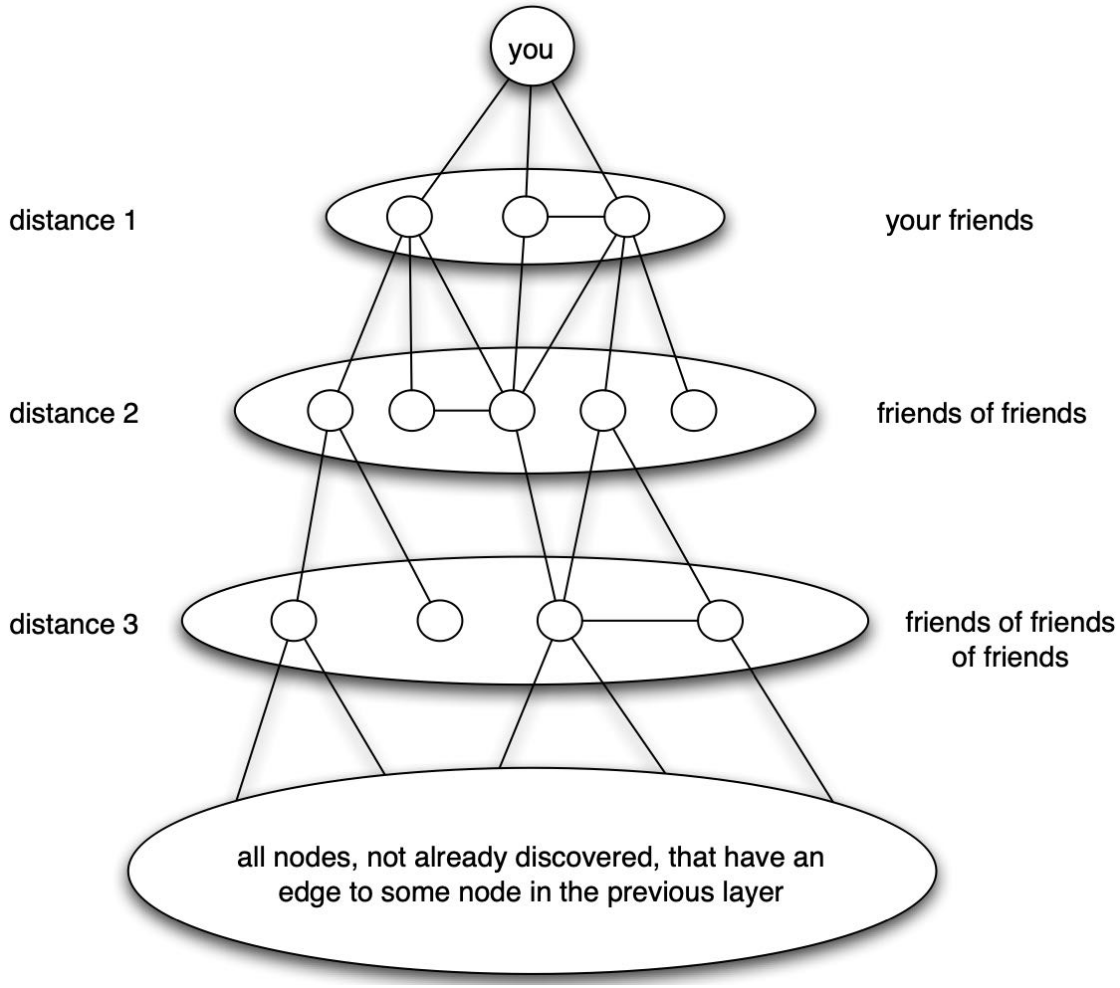
Shortest paths

- Two examples of paths from source b to target h
  - $\{b, c, d, e, i, h\}$
  - $\{b, c, d, e, j, i, h\}$
- **shortest path**: **minimal** path between two nodes
  - in weighted networks: weights can represent distances from two adjacent nodes
- **distance** between two nodes: shortest path length
- a **singleton** is considered at distance ∞ from any other nodes

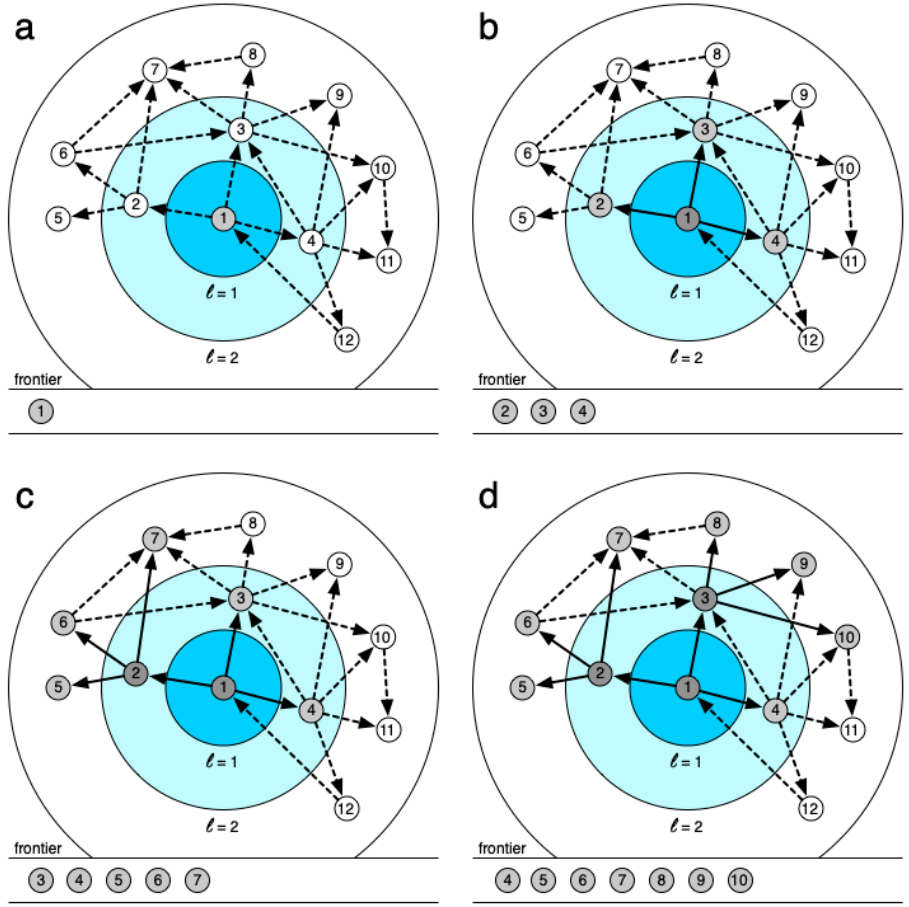


BFS (Breadth First Search)

- One of the most efficient algorithm to find distance
- Start from a **source node** (root)
- Visit the **entire breadth of the network**, within some distance from the source, before moving to a greater depth



- Each node has an attribute storing its distance l from the source
- initially $l(node) = -1$ except for $l(source) = 0$
- A queue (FIFO) holds the frontier, initially contains the source
- A directed shortest path tree, initially contains all the nodes and no links
- Iterate until the frontier is empty:
  - Remove next node i in frontier
  - For each neighbor/successor j of i with $l(j) = -1$:
  - Queue j into frontier



Average Path Length and diameter

- **Average Path Length (APL)**
  - undirected network $\langle l \rangle = \frac{2 \sum_{ij} l_{ij}}{N(N-1)}$
  - directed netwrok $\langle l \rangle = \frac{\sum_{ij} l_{ij}}{N(N-1)}$
- **Diameter** $l_{max} = max_{ij}(l_{ij})$

With disconnected components:

$\langle l \rangle = \frac{\sum_{ij} l_{ij}}{N(N-1)} = \infty$ you can rewrite it for undirected network $\langle l \rangle = \left(\frac{\sum_{ij} \frac{1}{l_{ij}}}{\binom{N}{2}} \right)^{-1}$

NetworkX: Shortest Paths Methods

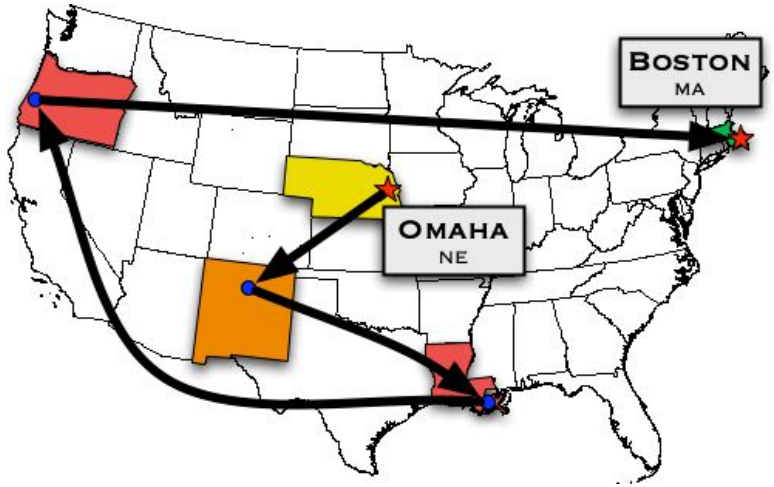
```
# Get shortest path length between two nodes
nx.shortest_path_length(G, source, target)

# Get all shortest paths between two nodes
nx.all_shortest_paths(G, source, target)

# Get average shortest path length
nx.average_shortest_path_length(G)

# Get shortest path lengths from source to all nodes
# Returns dictionary with {node: length}
nx.single_source_shortest_path_length(G, source)

# Get diameter of graph
nx.diameter(G)
```

Small world phenomenon

Milgram's Small-World Experiment

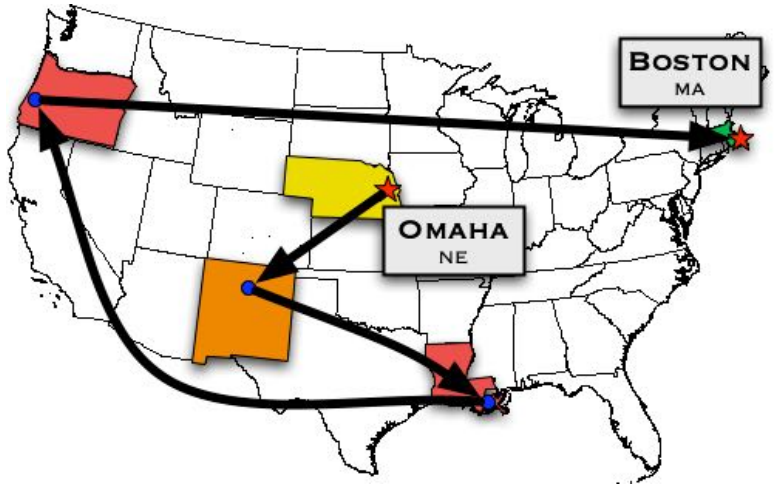
Overview

- Conducted by **Stanley Milgram** in the 1960s to study **social connections**.
- Investigated how many steps it takes for a letter to reach a specific person via acquaintances.
- The foundation for the **"six degrees of separation"** concept.



Experimental Setup

- **Participants:** 160 people from Omaha, Nebraska, and Wichita, Kansas.
- **Targets:** Two individuals in Massachusetts:
  - A **stockbroker** in Boston.
  - The **wife of a student** in Sharon.
- **Instructions:**
  - Participants were asked to **send the letter to a personal acquaintance** who they believed was **more likely to know the target**.



Results & Insights

- **42 letters reached the target** (26% success rate).
- **Average path length:** 6.5 steps (range: **3 to 12 steps**).
- The number of **steps was much lower than expected**, demonstrating the **small-world phenomenon**.
- Provided the first empirical evidence for the idea that **people are more connected than they assume**.

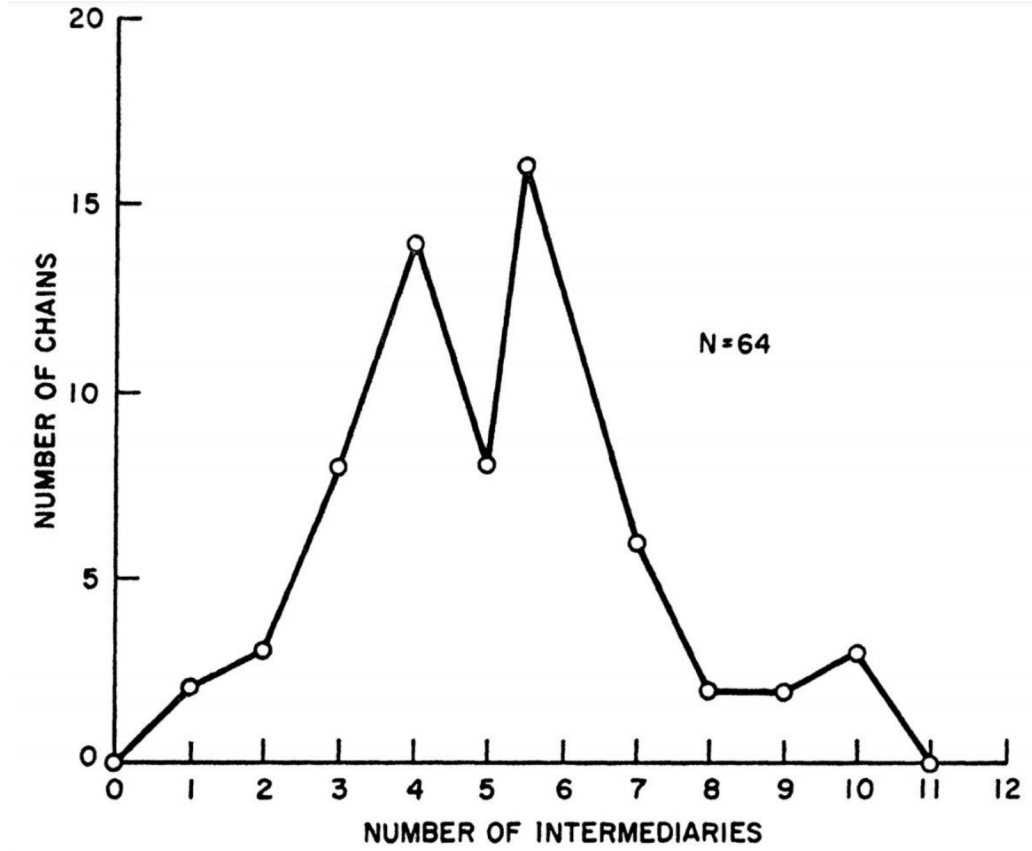


FIGURE 1

Lengths of Completed Chains

Impact on Network Science

- Inspired **research in social networks**, graph theory, and modern connectivity studies.
- Directly influenced today's **social media algorithms** and network analysis.
- Helped lay the groundwork for understanding **how information, trends, and diseases spread** through societies.

Key Takeaways

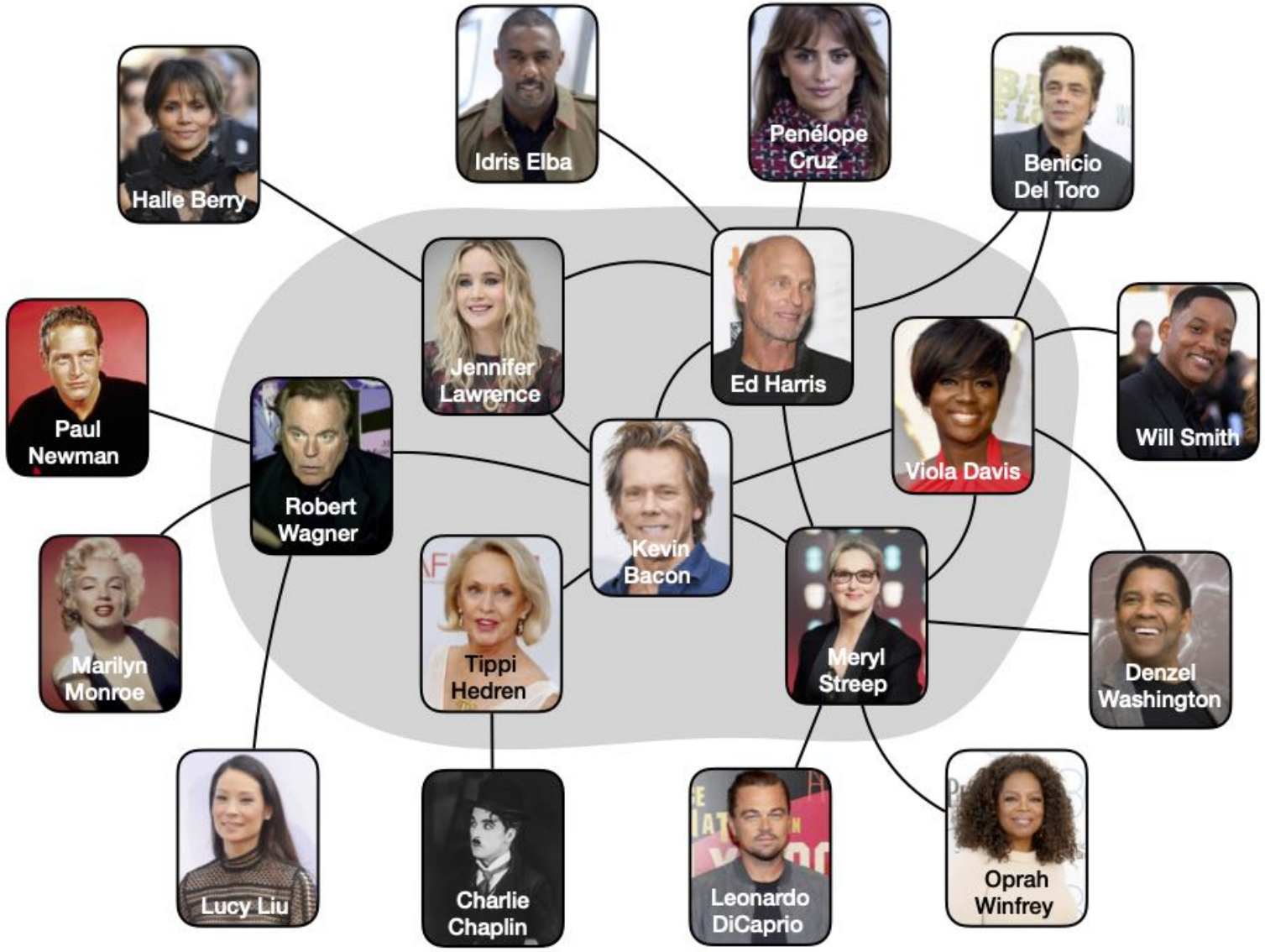
✅ **Human society is highly interconnected**.
✅ **People are just a few steps away from any other person** in the network.
✅ **The small-world phenomenon continues to be a fundamental principle in modern network science.**

Other small world experiments

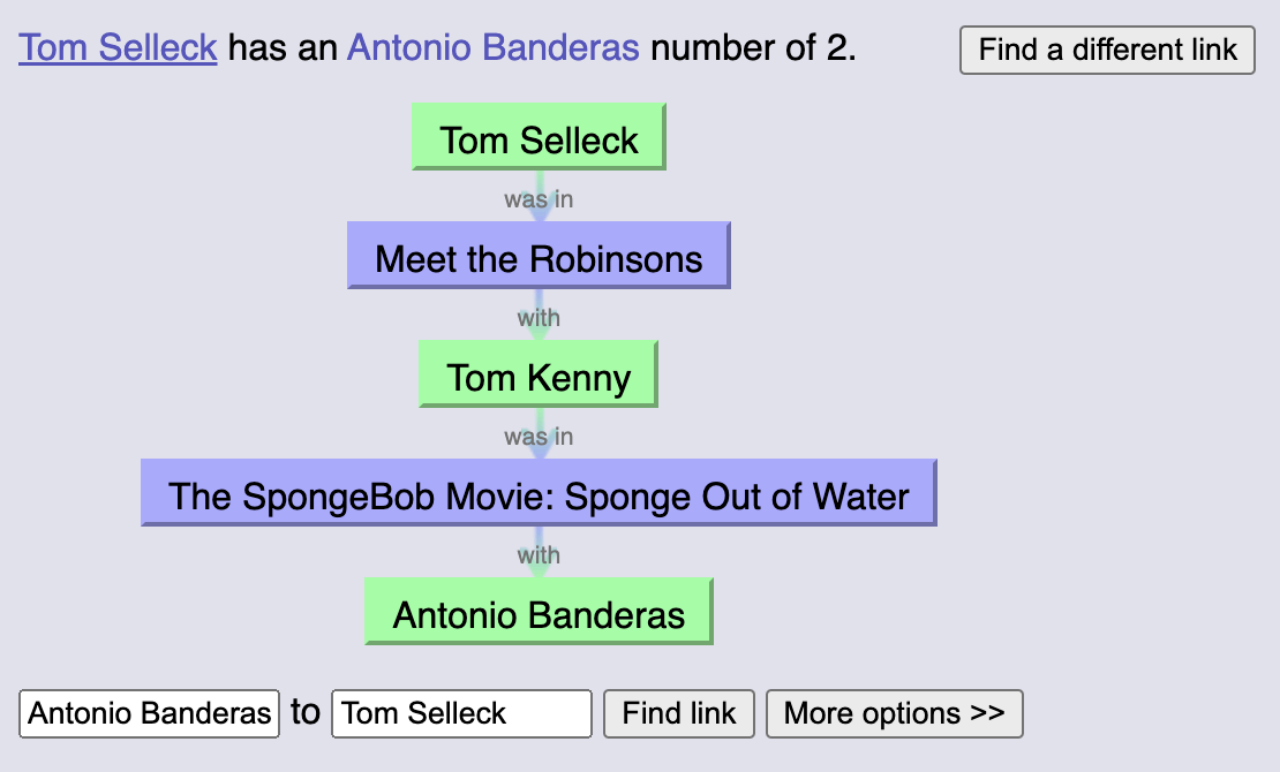
- Erdös number
- Oracle of Bacon
- Yahoo! email (Kleinberg, 2003)
- Instant Messaging (Leskovec and Horvitz, 2007)
- Facebook (Boldi, Vigna and others, 2011)

Bacon's oracle

- Movie co-star network
- oracleofbacon.org



- Not only Kevin Bacon
- Can you find two stars separated by more than four links?
- Play the game and try!



Instant messaging

- 240 millions of nodes
- median: 7
- average: 6.6

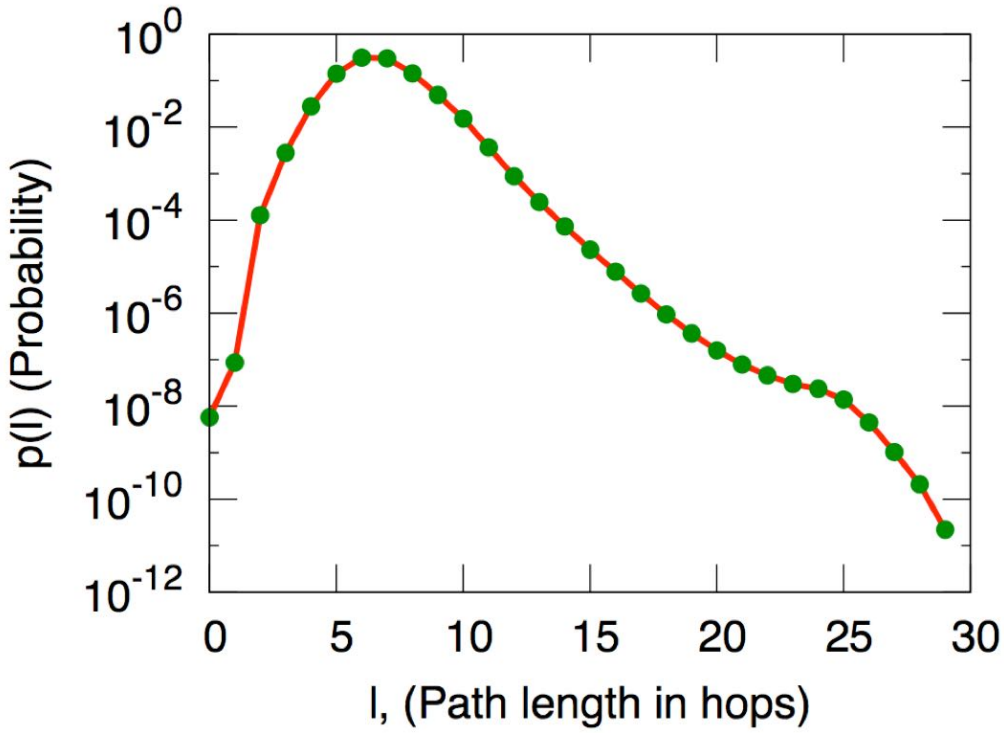


Figure 2.11: The distribution of distances in the graph of all active Microsoft Instant Messenger user accounts, with an edge joining two users if they communicated at least once during a month-long observation period [273].

The New York Times

Separating You and Me? 4.74 Degrees

By John Markoff and Somini Sengupta

Nov. 21, 2011

The world is even smaller than you thought.

Adding a new chapter to the research that cemented the phrase "six degrees of separation" into the language, scientists at Facebook and the University of Milan reported on Monday that the average number of acquaintances separating any two people in the world was not six but 4.74.

The original "six degrees" finding, published in 1967 by the psychologist Stanley Milgram, was drawn from 296 volunteers who were asked to send a message by postcard, through friends and then friends of friends, to a specific person in a Boston suburb.

When a short path is really short?

- It depends on the size of the network!
- Observe the relationship between APL and network size when considering networks (or subnetworks) of different sizes
- We say that the average path length is short when it grows very slowly with the size of the network, say, logarithmically:

$$\langle l \rangle \approx logN$$

UNIVERSITÀ DI TORINO

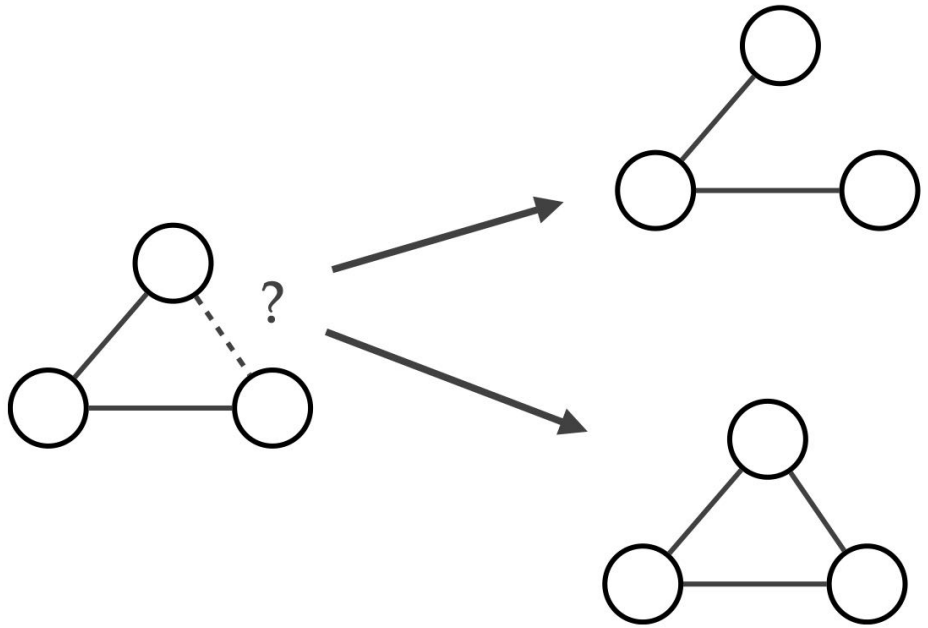
Small world networks

Table 1.1 Basic statistics of network examples. Network types can be (D)irected and/or (W)eighted. When there is no label the network is undirected and unweighted. For directed networks, we provide the average in-degree (which coincides with the average out-degree).

Network	Type	Nodes (N)	Links (L)	Density (d)	Average degree ($\langle k \rangle$)
Facebook Northwestern Univ.		10,567	488,337	0.009	92.4
IMDB movies and stars		563,443	921,160	0.000006	3.3
IMDB co-stars	W	252,999	1,015,187	0.00003	8.0
Twitter US politics	DW	18,470	48,365	0.0001	2.6
Enron Email	DW	87,273	321,918	0.00004	3.7
Wikipedia math	D	15,220	194,103	0.0008	12.8
Internet routers		190,914	607,610	0.00003	6.4
US air transportation		546	2,781	0.02	10.2
World air transportation		3,179	18,617	0.004	11.7
Yeast protein interactions		1,870	2,277	0.001	2.4
C. elegans brain	DW	297	2,345	0.03	7.9
Everglades ecological food web	DW	69	916	0.2	13.3

A friend of a friend

- In social networks, we have **triangles**
- **Very common** phenomenon



Clustering coefficient

The clustering coefficient of a node is the **fraction of pairs of the node's neighbors that are connected to each other**

In alternative: the **ratio between the number of triangles that include the node, and the maximum number of triangles in which the node could participate**

If:

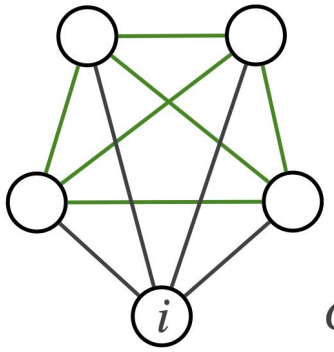
- N_i is the set of the neighbors of i while $k_i = |N_i|$ is the degree of i
- $\tau(i)$ is the number of triangles involving i:

$$C(i) = \frac{\tau(i)}{\tau_{max}(i)} = \frac{\tau(i)}{\binom{k_i}{2}} = \frac{2\tau(i)}{k_i(k_i-1)}$$

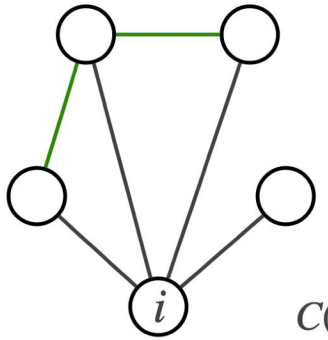
Network clustering coefficient

The clustering coefficient of the network is the **average of the clustering coefficients of the nodes**:

$$C = \frac{\sum_{i:k_i>1} C(i)}{N_{k>1}}$$



$$C(i) = \frac{2 \cdot 6}{4 \cdot 3} = 1$$



$$C(i) = \frac{2 \cdot 2}{4 \cdot 3} = \frac{1}{3}$$

Network clustering coefficient

Some networks, e.g., **social networks**, tend to have **high clustering coefficients because of triadic closure**: we meet through common friends

Other networks, e.g., bipartite and tree-like networks, have low clustering coefficient

Table 1.1 Basic statistics of network examples. Network types can be (D)irected and/or (W)eighted. When there is no label the network is undirected and unweighted. For directed networks, we provide the average in-degree (which coincides with the average out-degree).

Network	Type	Nodes (N)	Links (L)	Density (d)	Average degree ($\langle k \rangle$)
Facebook Northwestern Univ.		10,567	488,337	0.009	92.4
IMDB movies and stars		563,443	921,160	0.000006	3.3
IMDB co-stars	W	252,999	1,015,187	0.00003	8.0
Twitter US politics	DW	18,470	48,365	0.0001	2.6
Enron Email	DW	87,273	321,918	0.00004	3.7
Wikipedia math	D	15,220	194,103	0.0008	12.8
Internet routers		190,914	607,610	0.00003	6.4
US air transportation		546	2,781	0.02	10.2
World air transportation		3,179	18,617	0.004	11.7
Yeast protein interactions		1,870	2,277	0.001	2.4
C. elegans brain	DW	297	2,345	0.03	7.9
Everglades ecological food web	DW	69	916	0.2	13.3

NetworkX: Clustering Coefficients

```
# Get local clustering coefficient for a node
nx.clustering(G, node)

# Get local clustering coefficients for all nodes
# Returns dictionary {node: coefficient}
nx.clustering(G)

# Get average clustering coefficient for whole network
nx.average_clustering(G)
```

What is Transitivity?

- **Transitivity** is a measure of how interconnected a network is.
- It quantifies the **global clustering** of a graph by analyzing the ratio of **triangles** to **triplets**.

$$T = \frac{3 \times \text{number of triangles}}{\text{number of connected triplets}}$$

- **Triangles**: A set of three fully connected nodes.
- **Connected triplets**: A node with edges to two others, forming an open or closed triangle.

Transitivity vs. Average Clustering

1. Transitivity (`nx.transitivity(G)`)

- Measures the **global** clustering.
- Considers the **ratio of all triangles to all triplets** in the graph.
- **More influenced by high-degree nodes**.

2. Average Clustering (`nx.average_clustering(G)`)

- Measures the **local** clustering.
- Computes **clustering coefficients for each node** and takes the average.
- **Each node contributes equally**.

Step 1: Count Triangles

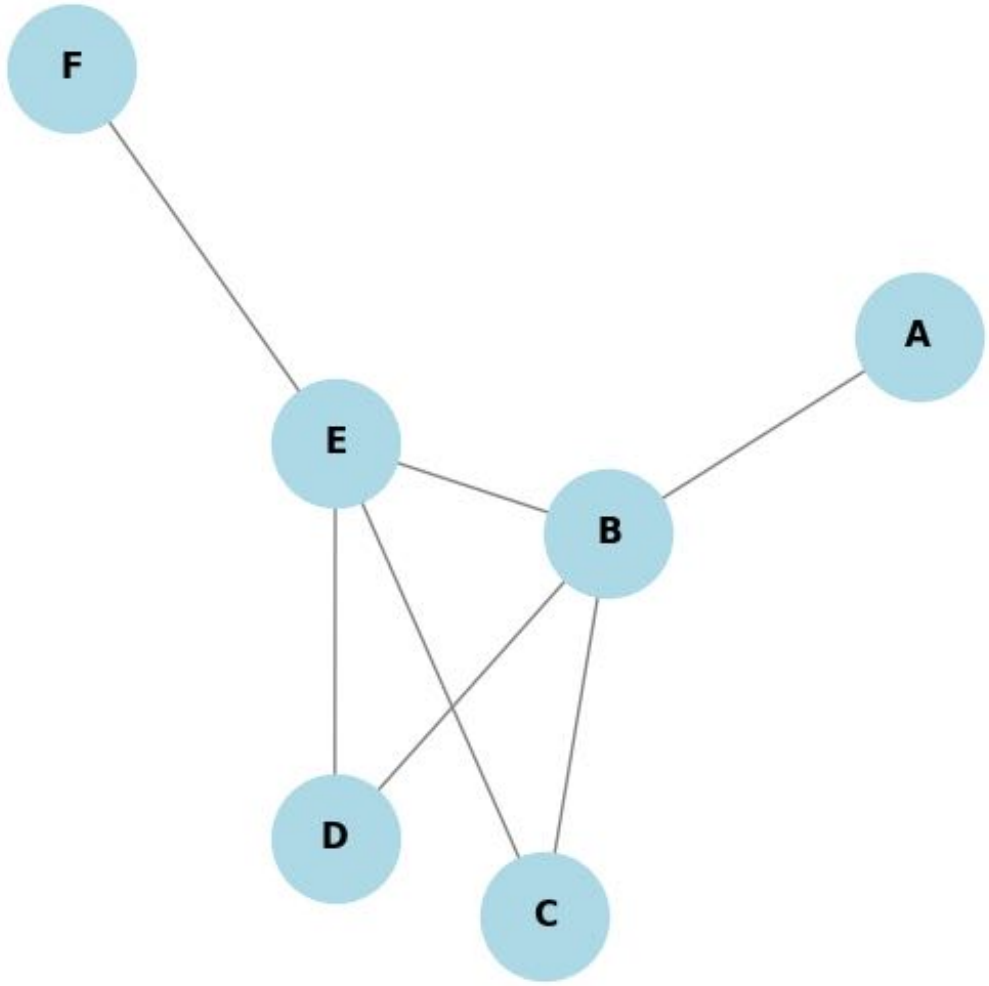
- (B, D, E), (B, C, E) → **2 triangles**

Step 2: Count Connected Triplets

- Triplets: Any node with 2 connections (open or closed).
- Assume **14 connected triplets**.

Step 3: Compute Transitivity

$$T = \frac{3 \times 2}{14} = \frac{6}{14} \approx 0.429$$



Reading material

[ns1] **Chapter 1** and **Chapter 2** (no homophily/assortativity for now)

[ns2] **Chapter 2**

UNIVERSITÀ DI TORINO

Q&A