Network Theory and Dynamic Systems

02. Network Elements

SOSE 2025

Dr. -Ing. Stefania Zourlidou

Institute for Web Science and Technologies
Universität Koblenz

universität koblenz
weiter:denken

WeST
Web Science and Technologies

Recap from Previous Lecture

- Clarify the Course Objectives
- Software and Libraries
- An Introduction to Networks and Network Science
- Example of Networks

Objectives of this Lecture

- Basic Components:
  - Nodes (entities, e.g., individuals, computers)
  - Links (connections or interactions, e.g., friendships, data transfers)
- Network Types and Representations:
  - Undirected vs. Directed (e.g., social friendships vs. web hyperlinks)
  - Weighted vs. Unweighted (e.g., traffic volume vs. simple connections)
- Properties that Characterize Structure & Behavior of Networks

1. Basic Definitions

Definitions: Network or Graph

- A **network** or **graph** G has two parts:
  - a set of N elements, called **nodes** or **vertices**, and
  - a set of L pairs of nodes, called **links** or **edges**
- The link (i, j) joins the nodes i and j
- Two nodes are **adjacent** or **connected** or **neighbors** if there is a link between them

Definitions: Undirected/Directed Networks

universität koblenz
weiter:denken

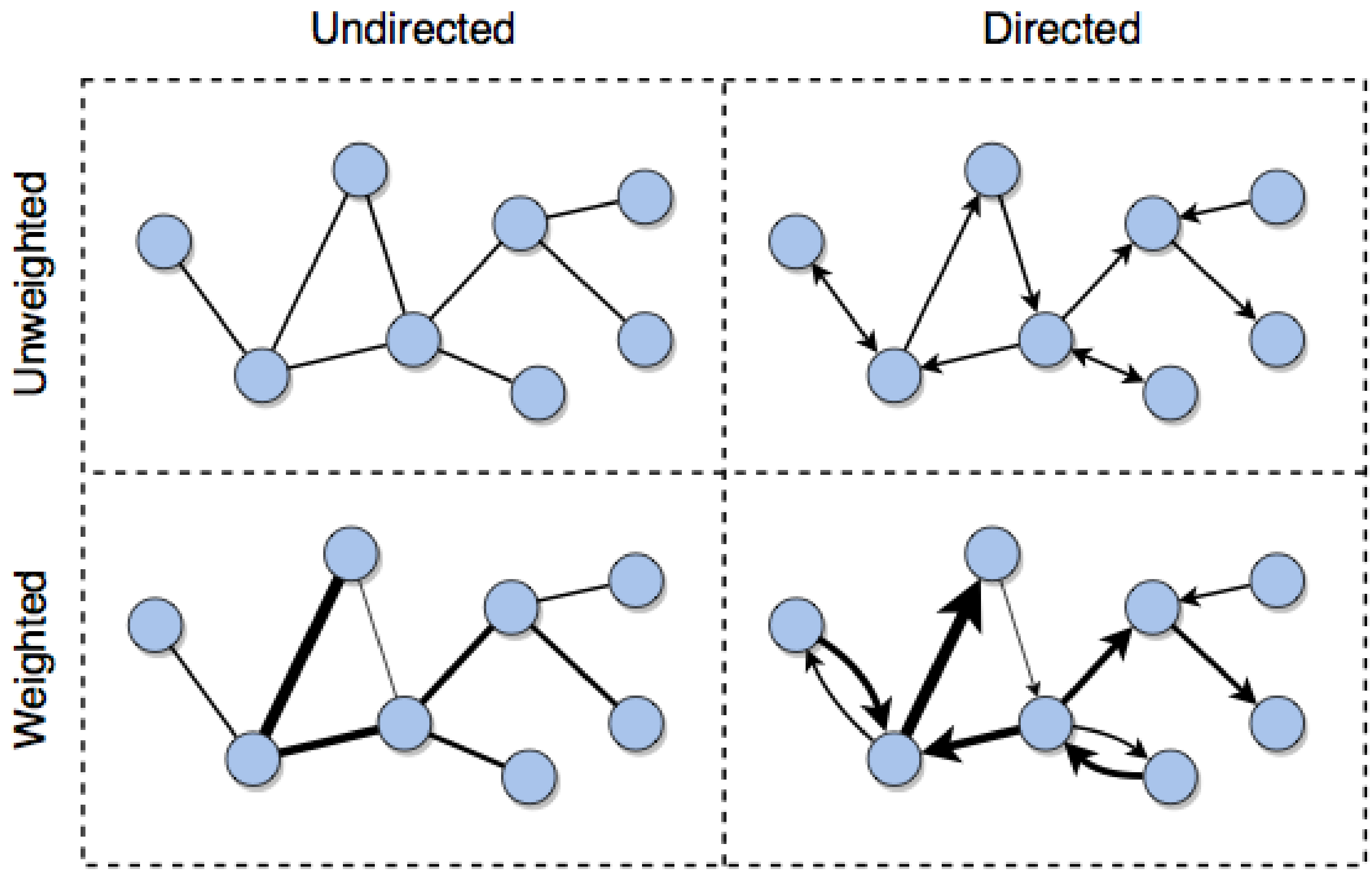
- A network can be **undirected** or **directed**
- **Directed Networks (Digraph)**
  - Links (directed edges) indicate directionality from a **source** node to a **target** node. The link (i, j) points from node i (source) to node j (target).
  - *Example*: Web hyperlink networks, where (i, j) means webpage i links to webpage j.
- **Undirected Networks:**
  - Links represent **bidirectional** relationships; the order of nodes is irrelevant
  - *Example*: Friendship networks, where connection (i, j) indicates mutual friendship

Definitions: Unweighted/Weighted Networks

universität koblenz
weiter:denken

- A network can be **unweighted** or **weighted**
  - **Unweighted Network**:
    - Links have no assigned numerical value; they only represent the presence of a connection
    - *Example*: Social networks indicating simple friendships between individuals
  - **Weighted Network**:
    - Links carry numerical values ("weights"), denoted as (i,j,w), indicating the strength, capacity, or frequency of interactions between nodes i and j
    - *Example*: Traffic networks, where link weights represent traffic volumes between cities
  - Networks can simultaneously be **directed and weighted**, consisting of directed weighted links.
  - *Example*: Airline flight networks, where the directed weighted link (i,j,w) shows the number of flights from airport i (source) to airport j (target)

Graphical Representations of Undirected/Directed and Weighted Networks



- **Can you think of a few examples in each of these categories?**

2. Handling Networks in Code

Tools for Managing, Analyzing, and Visualizing Networks (1/3)

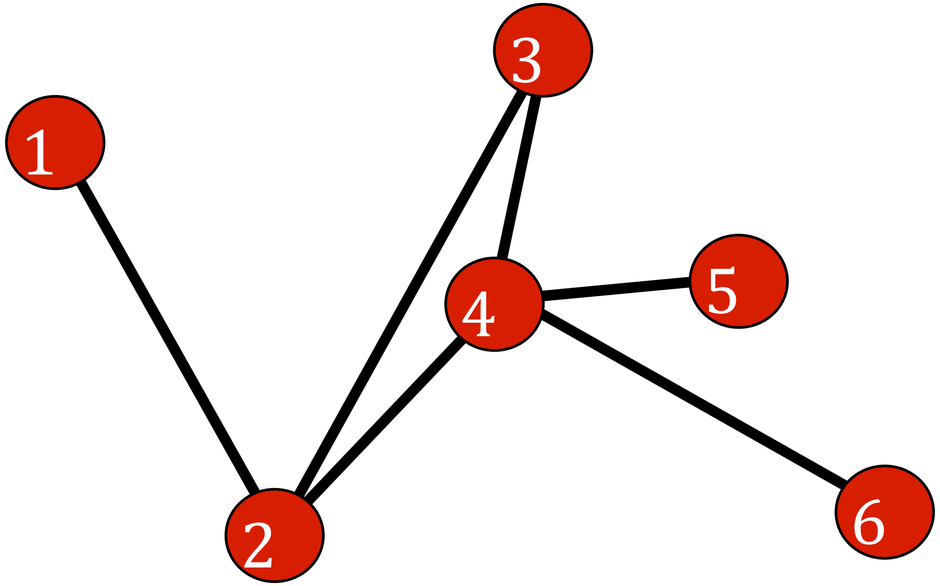
- Challenge: Handling large networks with numerous nodes and links can be complex
- Solution: Specialized software and programming libraries simplify network management, analysis, and visualization

Tools for Managing, Analyzing, and Visualizing Networks

- Visualization and Analysis Tools:
  - Gephi:
    - Interactive platform for network visualization
    - Ideal for exploratory analysis and intuitive graphical representation
  - Programming Libraries (e.g., NetworkX[1] in Python):
    - Robust toolkit for creating, manipulating, and analyzing networks
    - Includes built-in data structures, algorithms (e.g., shortest-path algorithms), network metrics (centrality, clustering), and generators (random network models)

1: See documentation and tutorial https://networkx.org/documentation/stable/tutorial.html

Python and NetworkX

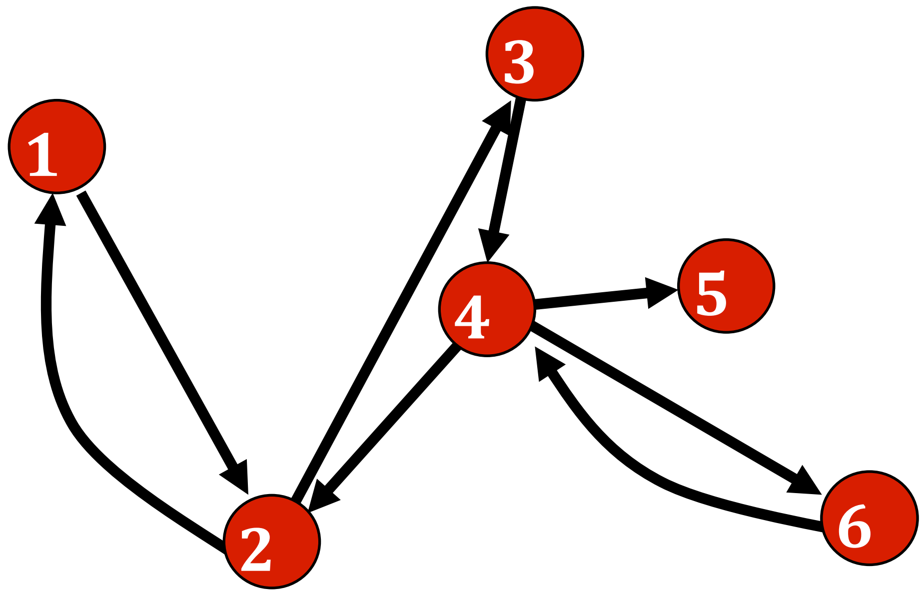



```
import networkx as nx # always!

G = nx.Graph()
G.add_node(1)
G.add_nodes_from([2,3,…])
…
G.add_edge(1,2)
G.add_edges_from([(2,3),(2,4),…])
…
G.nodes()
G.edges()
G.neighbors(4)

for n in G.nodes:
    print(n, G.neighbors(n))
for u,v in G.edges:
    print(u, v)
```

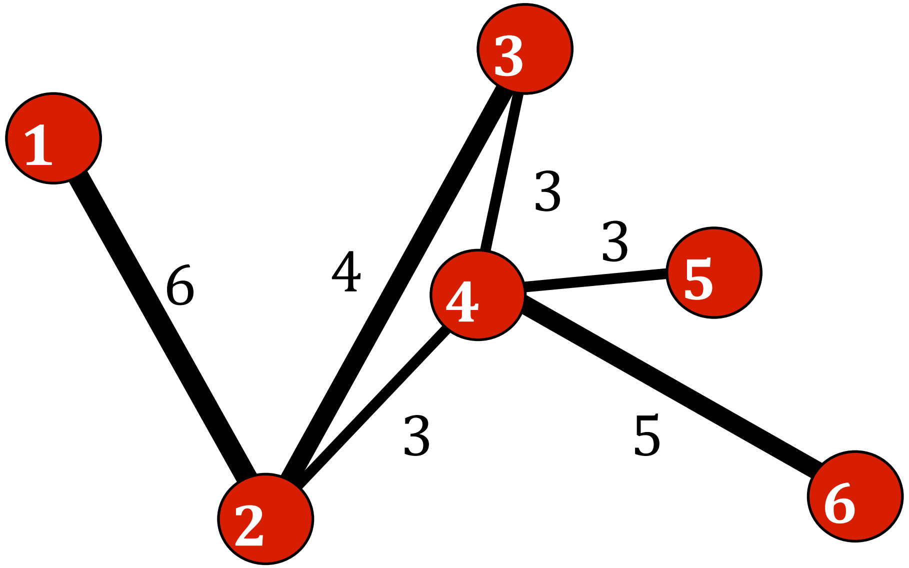
Directed Networks




```
import networkx as nx # don’t forget!

D = nx.DiGraph()
D.add_edge(1,2)
D.add_edge(2,1)
D.add_edges_from([(2,3),(3,4),…])
…
D.number_of_nodes()
D.number_of_edges()
D.edges()
D.successors(2)
D.predecessors(2)
D.neighbors(2)
```

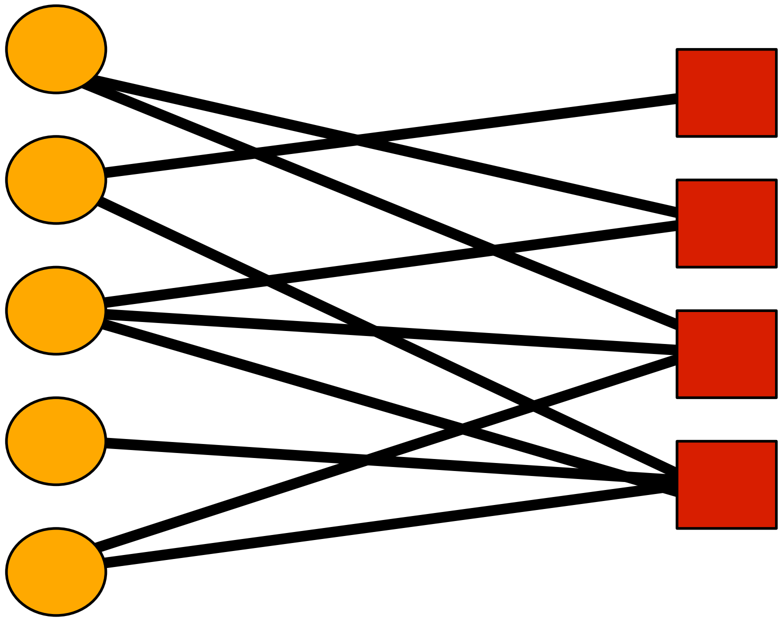
Weighted Networks



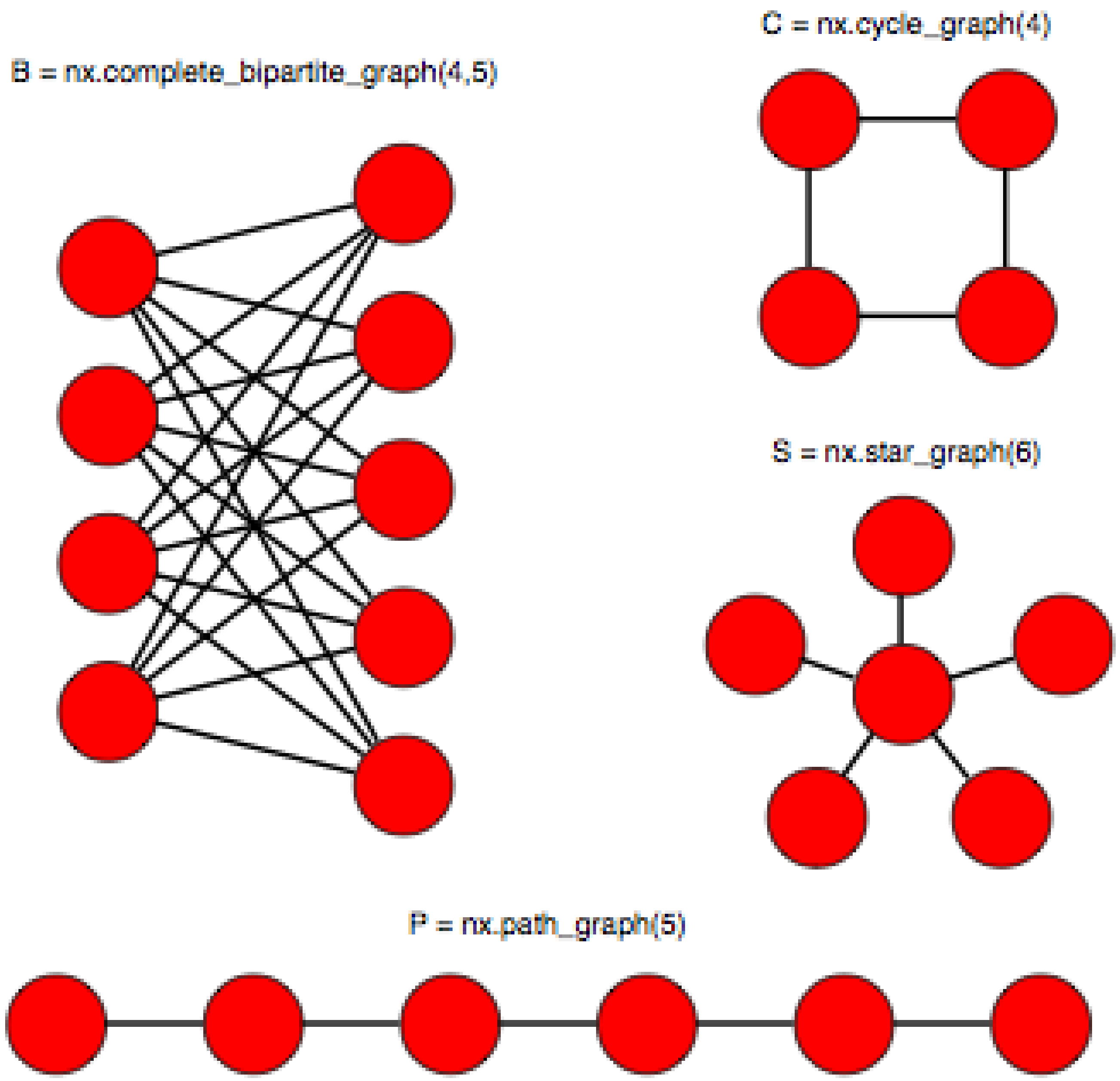

```
W = nx.Graph()
W.add_edge(1,2,weight=6)
…
W.add_weighted_edges_from([(4,5,3),(4,6,5),…])
…
W.edges()
W.edges(data='weight')
for (u,v,d) in W.edges(data='weight'):
    if d>3:
        print('(%d, %d, %d)’%(u,v,d))
```

Bipartite Networks

- Definition:
  - A network consisting of two distinct groups of nodes, where connections (links) occur only between nodes from different groups. Nodes within the same group are never directly connected
- *Examples*:
  - Movies and Actors: Actors linked to films they've starred in
  - Songs and Artists: Musicians connected to the songs they perform
  - Students and Classes: Students enrolled in specific courses
  - Products and Customers: Customers connected to products they've purchased



Many Networks Generators



3. Density and Sparsity

Density and Sparsity (1/2)

- Network size N = number of nodes
- L = number of links
- Maximum possible number of links: $$L_{max} = \binom{N}{2} = \frac{N(N-1)}{2}$$
- Density: $$d = \frac{L}{L_{max}} = \frac{2L}{N(N-1)}$$
- The network is **sparse** if $d << 1$

Density and Sparsity (2/2)

- In a **directed** network things are a bit different
  - Maximum possible number of links: $L_{max} = N(N-1)$
  - Density: $d = \dfrac{L}{L_{max}} = \dfrac{L}{N(N-1)}$
- In a **complete** network, all pairs of nodes are connected and $d = 1$

```
G.number_of_nodes()
G.number_of_edges()
nx.density(G)
nx.density(D)

CG = nx.complete_graph(8471)
print(nx.density(CG)) # what does this print?
```

Example: Facebook

- Rough orders-of-magnitude approximations:
  - $N \approx 10^9$
  - $L \approx 10^3 \times N$
  - $d \approx L / N^2 \approx 10^3 N / N^2 \approx 10^3 / 10^9 = 10^{-6}$
- Most (but not all) real-world networks are similarly **sparse** because the number of links scales proportionally to N, whereas the maximum scales with N^2

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

universität koblenz
weiter:denken

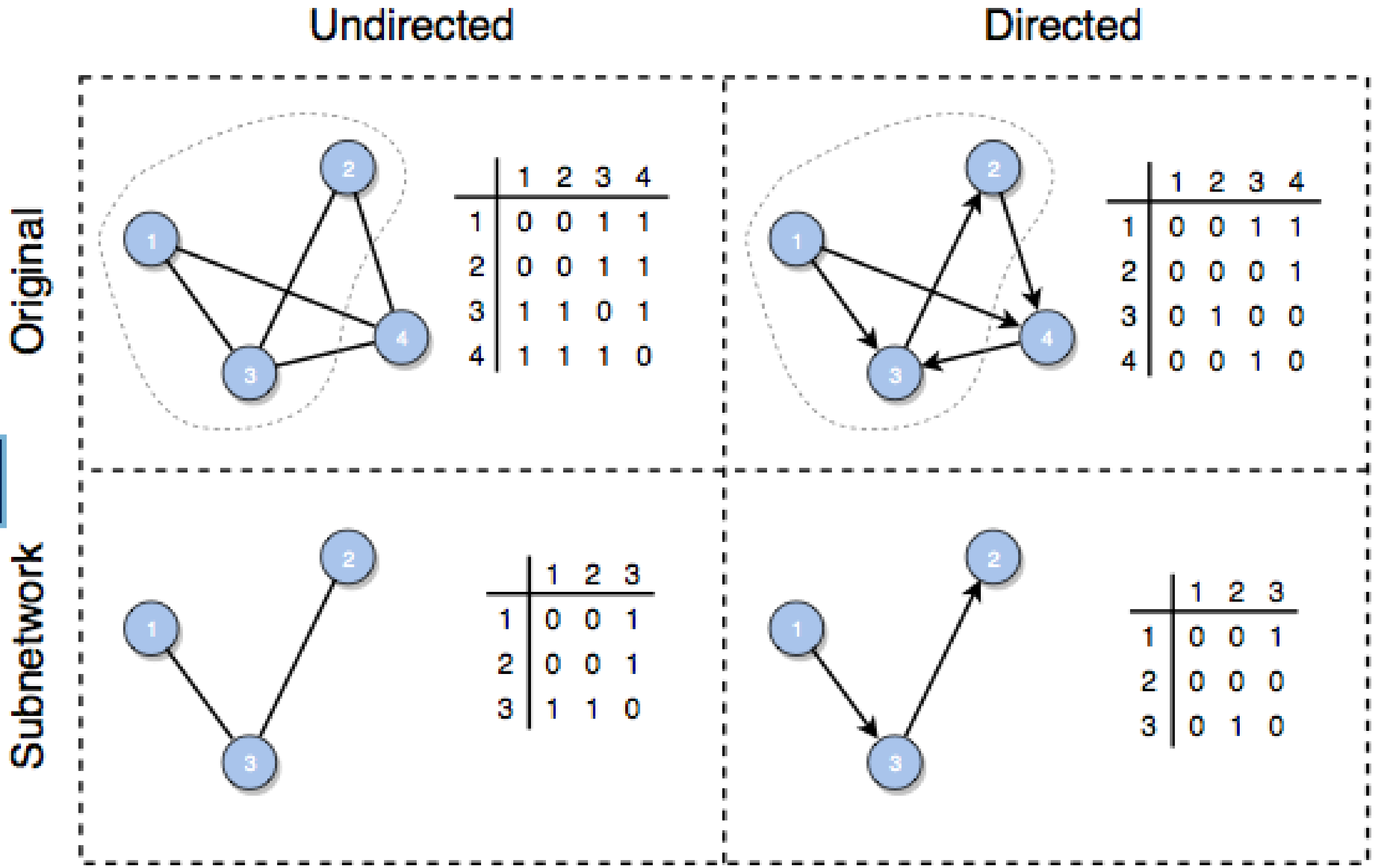
4. Subnetworks

Subnetworks

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

```
S = nx.subgraph(G, node_list)
```

- A **clique** is a complete subnetwork



5. Degree

Degree

- The **degree** of a node is its number of links, or neighbors
- We typically use k_i to denote the degree of node i
- A node without neighbors is called a **singleton** (k=0)

```
G.degree(2) # returns the degree of node 2
G.degree()  # dict with the degree of all nodes of G
```

Degree: Directed Networks

- In a directed network we have
  - **in-degree** of a node = number of incoming links k^{in}_i
  - **out-degree** of a node = number of outgoing links k^{out}_i

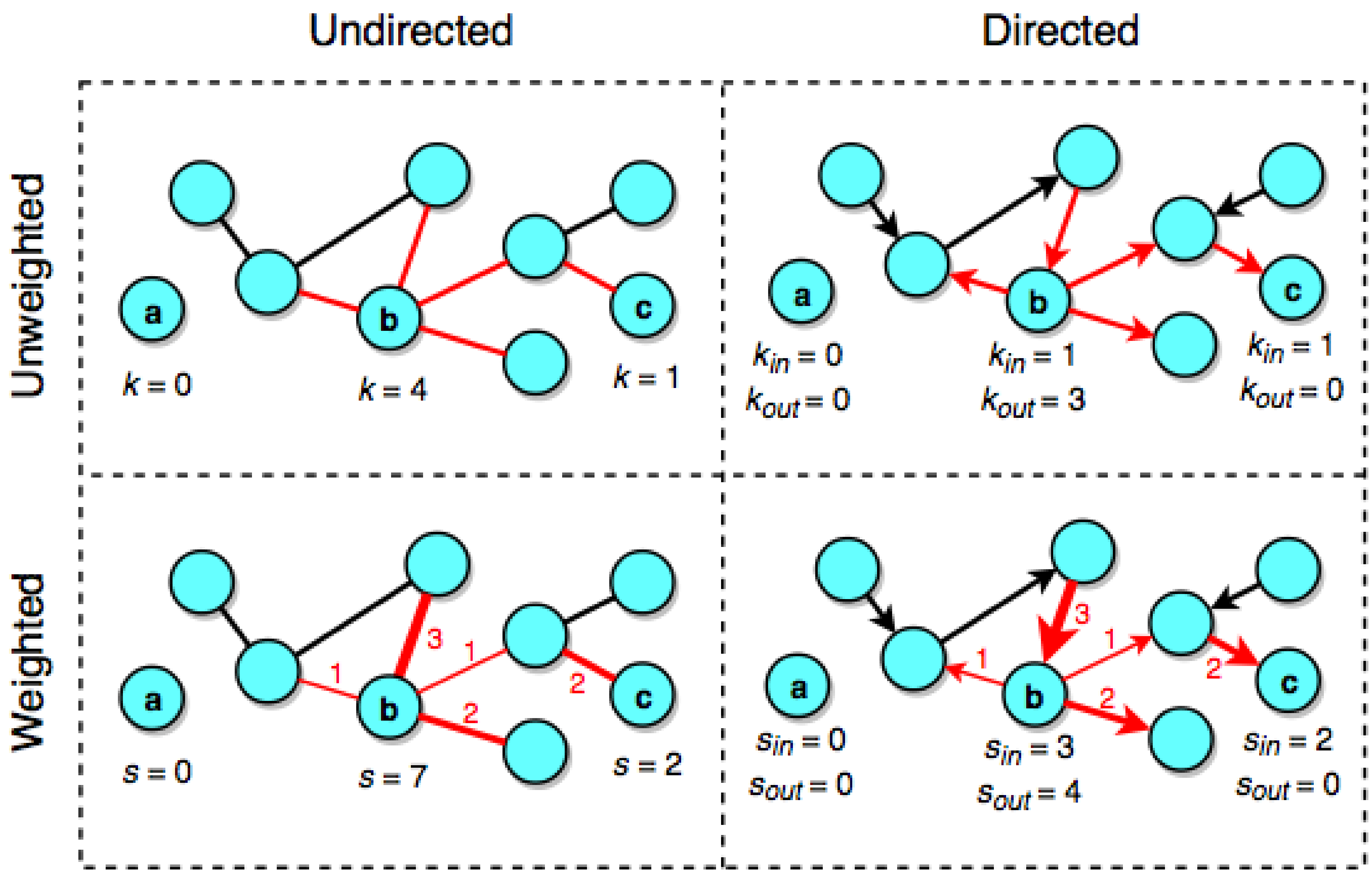
```
D.in_degree(4)
D.out_degree(4)
D.degree(4)
```

Strength or Weighted Degree

- In a weighted network we have **strength** $s_i = \sum_j w_{ij}$ (a.k.a. **weighted degree**)

- In a weighted directed network we have
  - **in-strength** $s_i^{in} = \sum_j w_{ji}$
  - **out-strength** $s_i^{out} = \sum_j w_{ij}$

```
W.degree(4) # degree
W.degree(4, weight='weight') # strength
```


Undirected
Directed
Unweighted
Weighted
a
b
c
$k = 0$
$k = 4$
$k = 1$
$k_{in} = 0$
$k_{out} = 0$
$k_{in} = 1$
$k_{out} = 3$
$k_{in} = 1$
$k_{out} = 0$
$s = 0$
$s = 7$
$s = 2$
$s_{in} = 0$
$s_{out} = 0$
$s_{in} = 3$
$s_{out} = 4$
$s_{in} = 2$
$s_{out} = 0$

Average Degree

- The **average degree** of a network is $\langle k \rangle = \frac{\sum_i k_i}{N}$
- We can connect network size, number of links, density, and average degree
- In undirected networks:

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

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

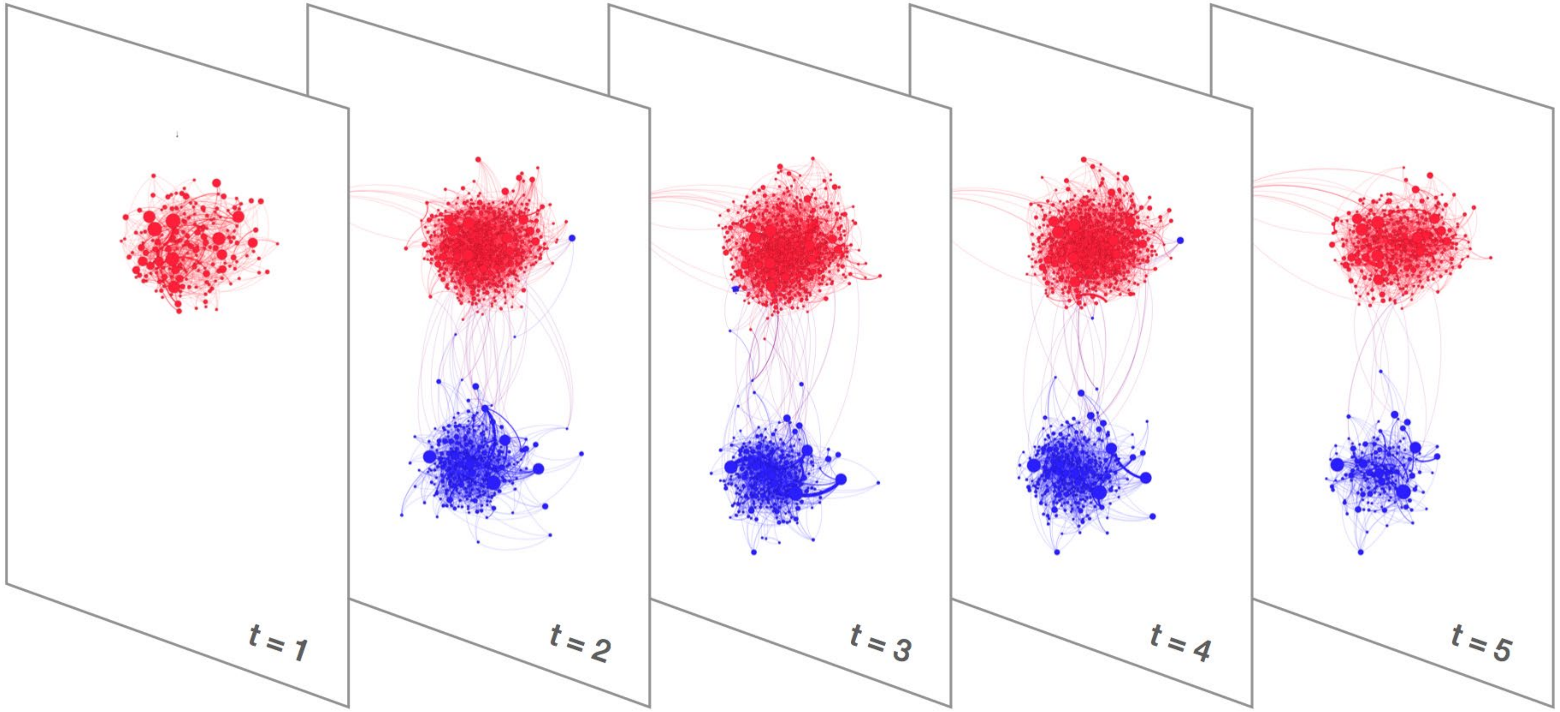
6. Multilayer and Temporal Networks

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 relationship in a social network, such as friendship, family ties, coworkers, etc.

Temporal Networks

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



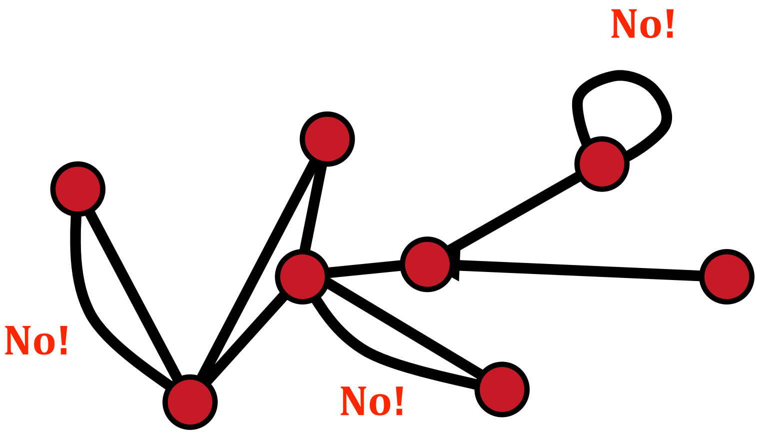
Multilayer Networks (Networks of Networks)

universität koblenz
weiter:denken

- In general, each layer in a multilayer network can have its own nodes and edges. We call this a **network of networks**
  - Examples: the electrical power grid and Internet
- **Interlayer Links:**
  - Capture interactions between different layers
  - These connections allow dependencies and influences to propagate across layers
  - Example: power stations communicate via the Internet, Internet routers are powered by the power grid
  - **Cascading failures:**
    - A failure in one layer (e.g., a power outage) can trigger failures in another (e.g., loss of Internet connectivity)
    - Systemic risk in interdependent networks

Simplifying Assumptions

- We will assume:
  - single-layer networks with a single type of nodes and a single type of link
  - no self-loops
  - at most a single link between two nodes (possibly two links with opposite directions in directed networks)

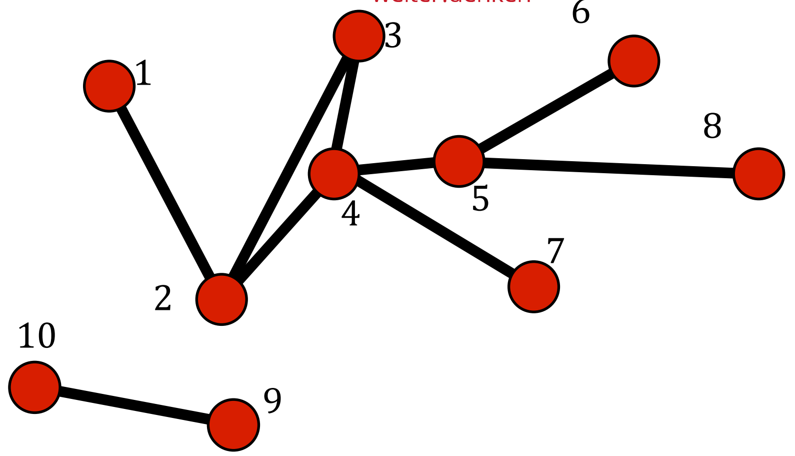


7. Network Representations

Network Representations

- **Adjacency Matrix**: N x N matrix where each element $a_{ij} = 1$ if i and j are adjacent, zero otherwise
- The diagonal elements are zero because we have no self-loops
- In undirected networks, the matrix is symmetric: $a_{ij} = a_{ji}$

```
nx.adjacency_matrix(G)
print(nx.adjacency_matrix(G))
G.edge[3][4]
G.edge[3][4]['color']='blue'
G.edge[3][4]
G.edge[4]
```

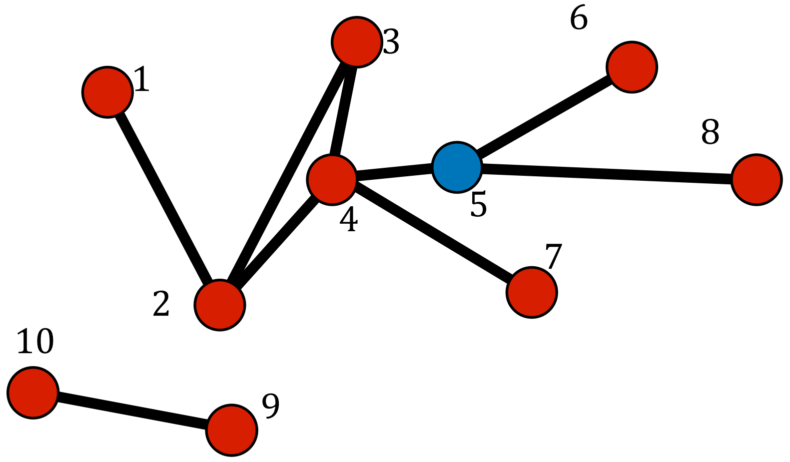


	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0
4	0	1	1	0	1	0	1	0	0	0
5	0	0	0	1	0	1	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	1	0

Network Representations: Undirected Nets

- In undirected networks, the degree is obtained by summing adjacency matrix elements across rows or columns:

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

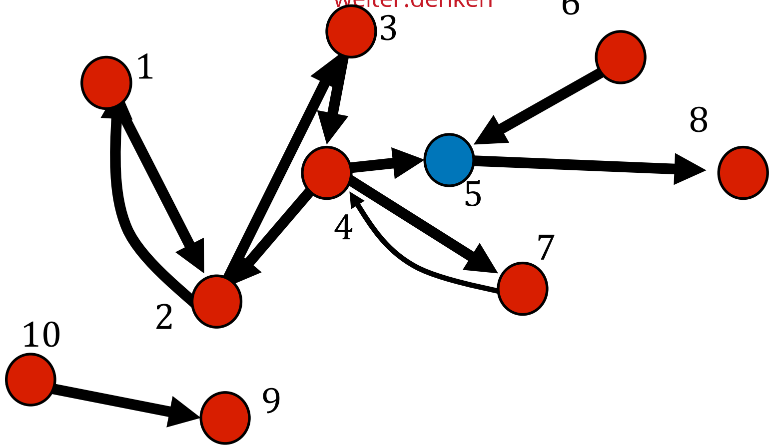


	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0
4	0	1	1	0	1	0	1	0	0	0
5	0	0	0	1	0	1	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	1	0

Network Representations: Directed Nets

- In directed networks, the adjacency matrix is **not** symmetric
- The **out-degree** is obtained by summing adjacency matrix elements across **rows**: $k_i^{out} = \sum_j a_{ij}$
- The **in-degree** is obtained by summing adjacency matrix elements across **columns**: $k_i^{in} = \sum_j a_{ji}$

```
print(nx.adjacency_matrix(D))
D.edge[3][4]
D.edge[4][3]
D.edge[4]
```

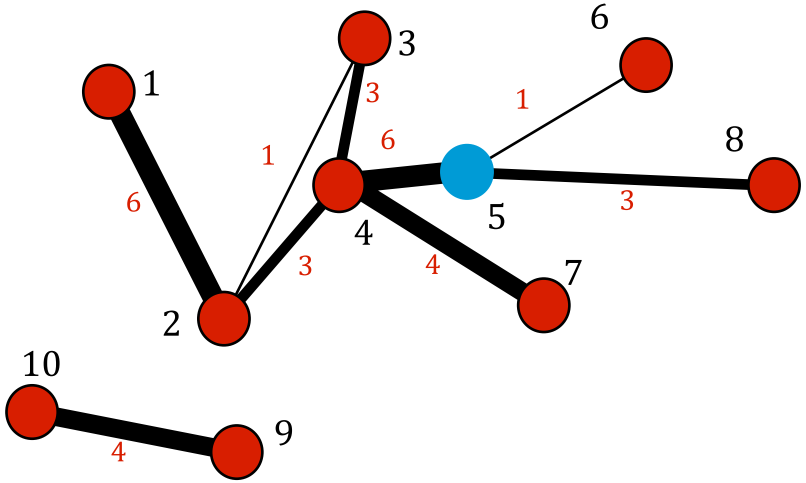


	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	1	0	0	0
5	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0

Network Representations: Weighted Nets

- In weighted networks, each element w_{ij} represents the weight of the link between i and j, zero if there is no link
- If undirected, the strength is obtained by summing adjacency matrix elements across rows or columns
- If directed, the in/out-strength is obtained by summing adjacency matrix elements across columns/rows

```
print(nx.adjacency_matrix(W))
W.edge[2][3]
W.edge[2]
W.edge[2][3]['weight'] = 2
W.edge[2][3]
W.edge[2]
```



	1	2	3	4	5	6	7	8	9	10
1	0	6	0	0	0	0	0	0	0	0
2	6	0	1	3	0	0	0	0	0	0
3	0	1	0	3	0	0	0	0	0	0
4	0	3	3	0	6	0	4	0	0	0
5	0	0	0	6	0	1	0	3	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	4	0	0	0	0	0	0
8	0	0	0	0	3	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	4
10	0	0	0	0	0	0	0	0	4	0

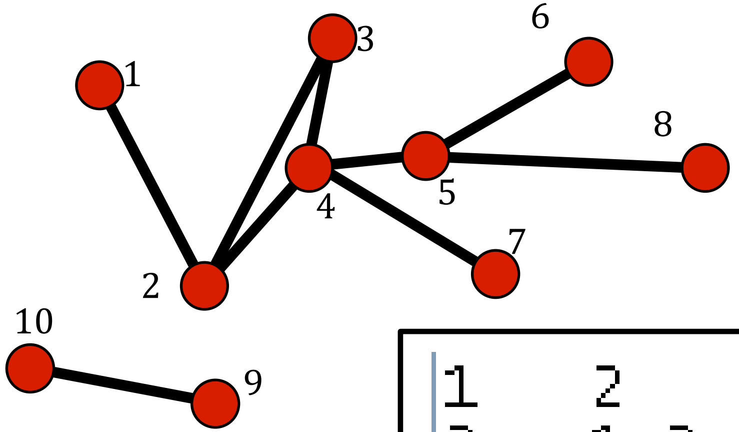
Sparse Network Representations

universität koblenz weiter:denken

- Adjacency matrix storage scales with N^2, where N is the number of nodes
- In sparse networks (common in real-world systems), this is highly *inefficient*—most entries represent absent links (zeros)
  - Solution: Store only existing links, assuming missing entries imply no connection
- Two commonly used representations for sparse networks:
  - **Adjacency list:** Each node lists its neighbors
  - **Edge list:** Stores each link as a pair (or triplet, if weighted) of nodes

Adjacency List

- List of neighbors for each node
- In undirected networks, each link is listed twice
- In weighted networks, each neighbor is replaced by a pair (neighbor, weight)



1	2			
2	1	3	4	
3	2	4		
4	2	3	5	7
5	4	6	8	
6	5			
7	4			
8	5			
9	10			
10	9			

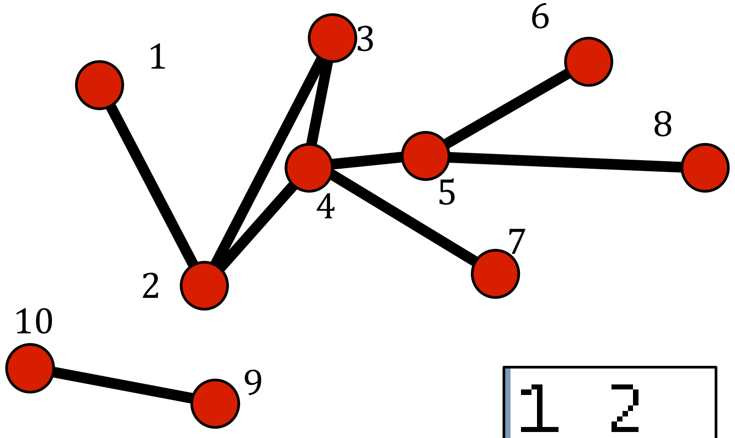
```
G.neighbors(2)

for n,neighbors in G.adjacency():
    for neighbor,link_attributes in neighbors.items():
        print('(%d, %d)' % (n,neighbor))

nx.write_adjlist(G, "netfile.adjlist")
G2 = nx.read_adjlist("netfile.adjlist") # G and G2 are
isomorphic
```

Edge List

- List of node pairs that are connected
- In weighted networks, each pair is replaced by a triplet (i, j, weight)




```
1 2
2 3
2 4
3 4
4 5
4 7
5 6
5 8
9 10
```

```
for i,j in G.edges:
    print('%d %d' %(i,j))

nx.write_edgelist(G, "netfile.edgelist")
G3 = nx.read_edgelist("netfile.edgelist") # G and G3 are isomorphic

nx.write_weighted_edgelist(W, "wf.edges") # store  weights
W2 = nx.read_weighted_edgelist("wf.edges") # W and W2 are isomorphic
```

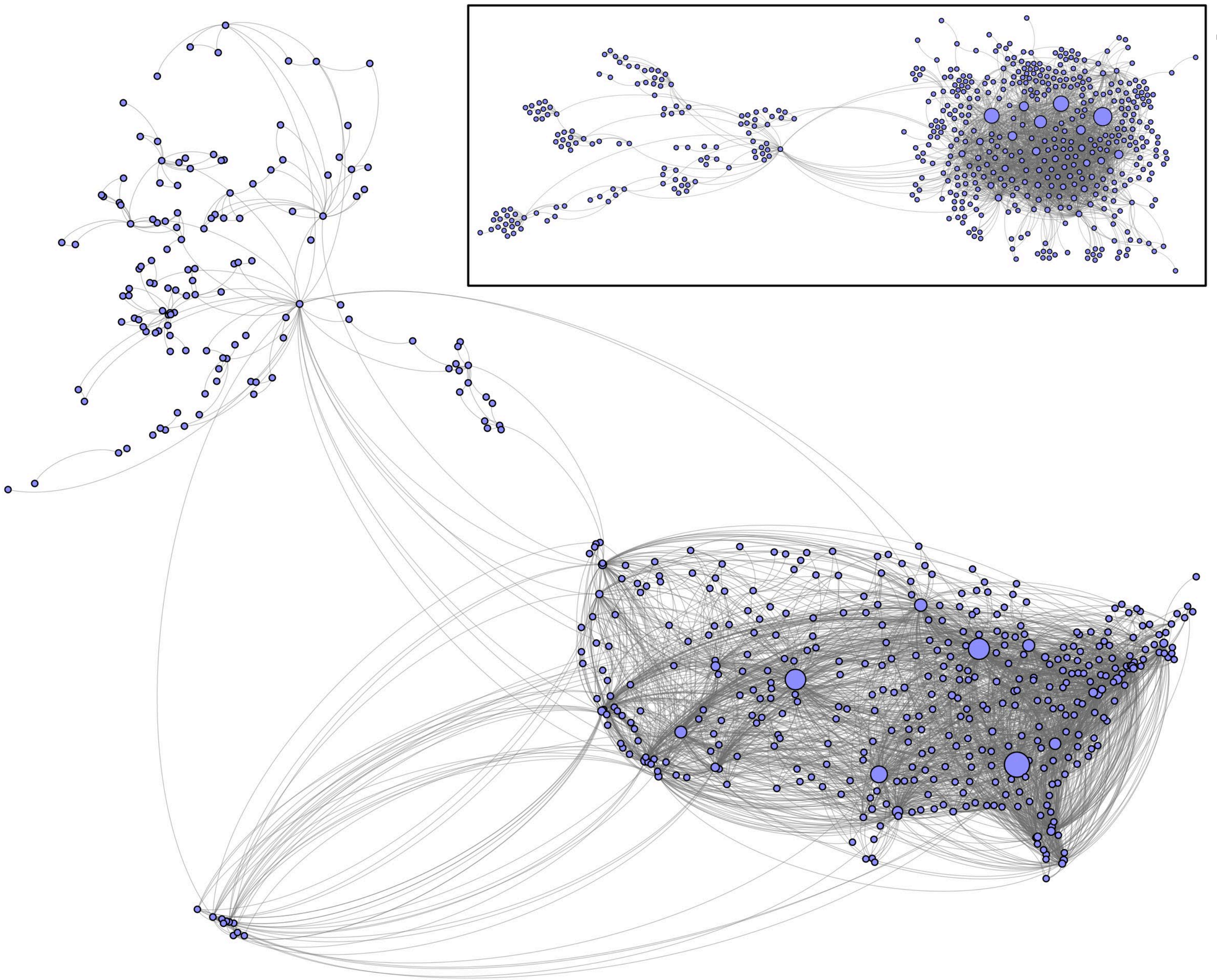
8. Drawing Networks

Why Use Network Layout Algorithms?

- Network visualization reveals structural patterns, relationships, and anomalies that are not obvious from raw data
- Layout algorithms assign positions to nodes in 2D (or 3D) space, enabling intuitive graphical representations
- Common Types of Layouts:
  - **Geographic Layouts:**
    - Preserve real-world spatial coordinates
    - *Example*: Air transportation networks plotted using airport locations
  - **Concentric Circles / Layered Layouts:**
    - Highlight hierarchy or flow in small networks
    - *Example*: Organizational charts, citation trees
  - **Force-Directed Layouts:**
    - Most widely used; simulate physical forces to spread nodes evenly and minimize edge crossings
    - *Application*: Used in most network visualizations in Chapter 0 to enhance clarity and interpretability

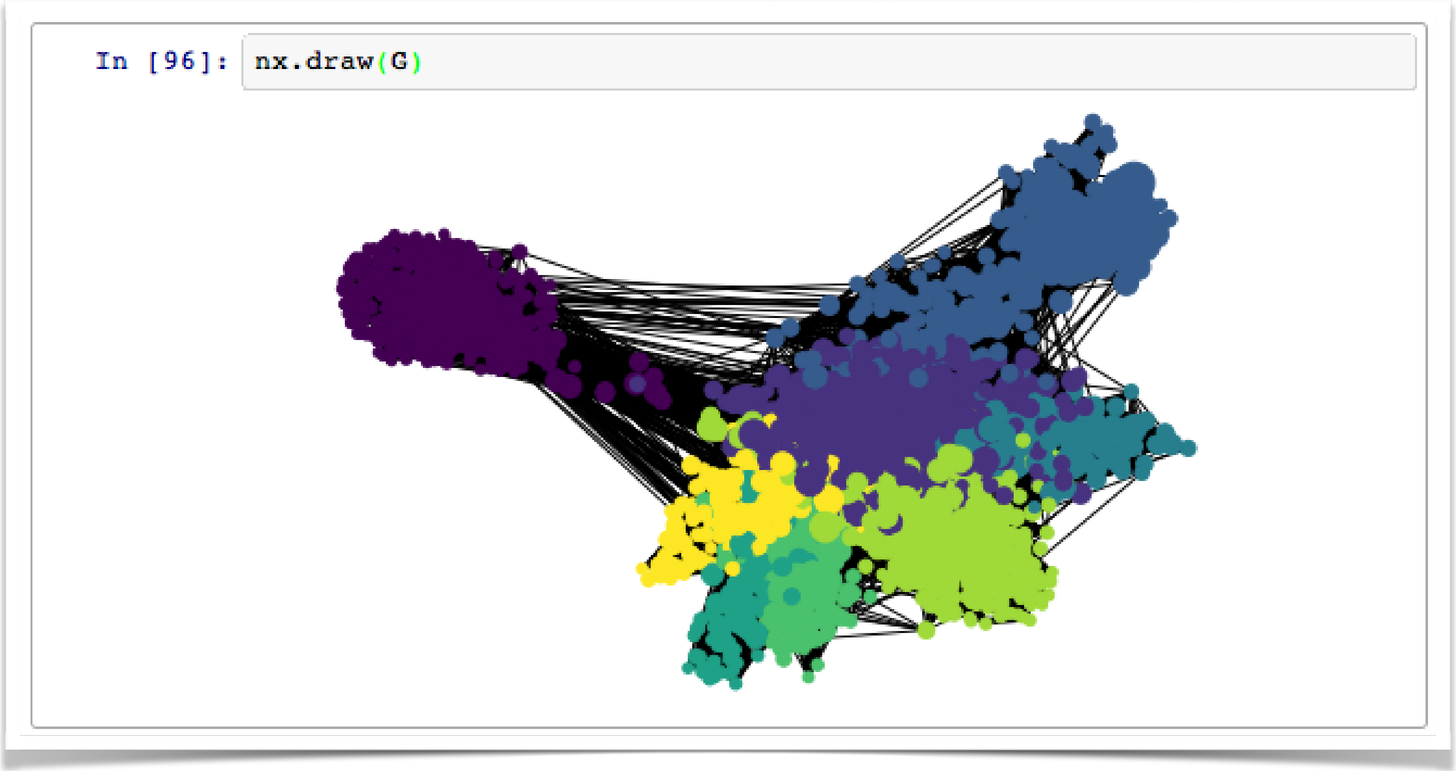
Geographic Layout

- Illustration of a Geographic Layout used in network visualization



Force-Directed Layouts (Spring Layouts)

```
import matplotlib.pyplot
nx.draw(G, node_color=colors, node_size=sizes)
```



Force-directed layout (a.k.a. **spring layout**) algorithms:

- Simulates a physical system where:
  - Connected nodes attract each other (like springs),
  - All nodes repel each other (like charged particles)
- Place connected nodes close together
- Ensure uniform link lengths
- Minimize edge crossings for visual clarity
- These layouts often reveal community structures in networks—clusters of densely connected nodes—especially when the network is not too dense or too large

universität koblenz
weiter:denken

9. Summary

Summary (1/2)

1. **Network Components**
   - A network consists of nodes (individual elements) and links (connections between the nodes)
2. **Subnetwork**
   - A part of the network that includes some nodes and all links connecting these nodes
3. **Directed vs. Undirected Networks**
   - In directed networks, links have a direction indicating an one-way relationship
   - In undirected networks, links show a two-way, reciprocal connection
4. **Weighted vs. Unweighted Networks**
   - Weighted networks assign values to links that can represent various attributes such as importance or distance
   - Unweighted networks treat all links equally
5. **Multilayer Networks**
   - These networks have multiple layers with different types or sets of nodes and links
   - A multiplex is a type of multilayer network where the same nodes are repeated across different interconnected layers

Summary (2/2)

6. **Network Density**
   - Measures the proportion of potential node connections that are actual connections
   - A complete network has a density of one, with all possible node pairs connected
7. **Node Degree and Strength**
   - Degree refers to the number of connections a node has
   - In directed networks, nodes have "in-degrees" and "out-degrees" for incoming and outgoing links, respectively
   - If the network is weighted, "strength" measures the total weight of a node’s connections, differentiated into "in-strength" and "out-strength" in directed scenarios
8. **Network Representations**
   - Networks can be efficiently represented using adjacency lists or edge lists, particularly useful for sparse networks
9. **NetworkX Library**
   - A powerful Python library used for creating, manipulating, and studying the structure and dynamics of complex networks

References

[1] Menczer, F., Fortunato, S., & Davis, C. A. (2020). A First Course in Network Science Cambridge: Cambridge University Press.

- Chapter 1 Network Elements

[2] OLAT course page: https://olat.vcrp.de/auth/RepositoryEntry/4669112833

Further Readings

- NetworkX documentation https://networkx.org/documentation/stable/tutorial.html