NETWORK SCIENCE OF ONLINE INTERACTIONS

Chapter 1: Network Elements

Joao Neto

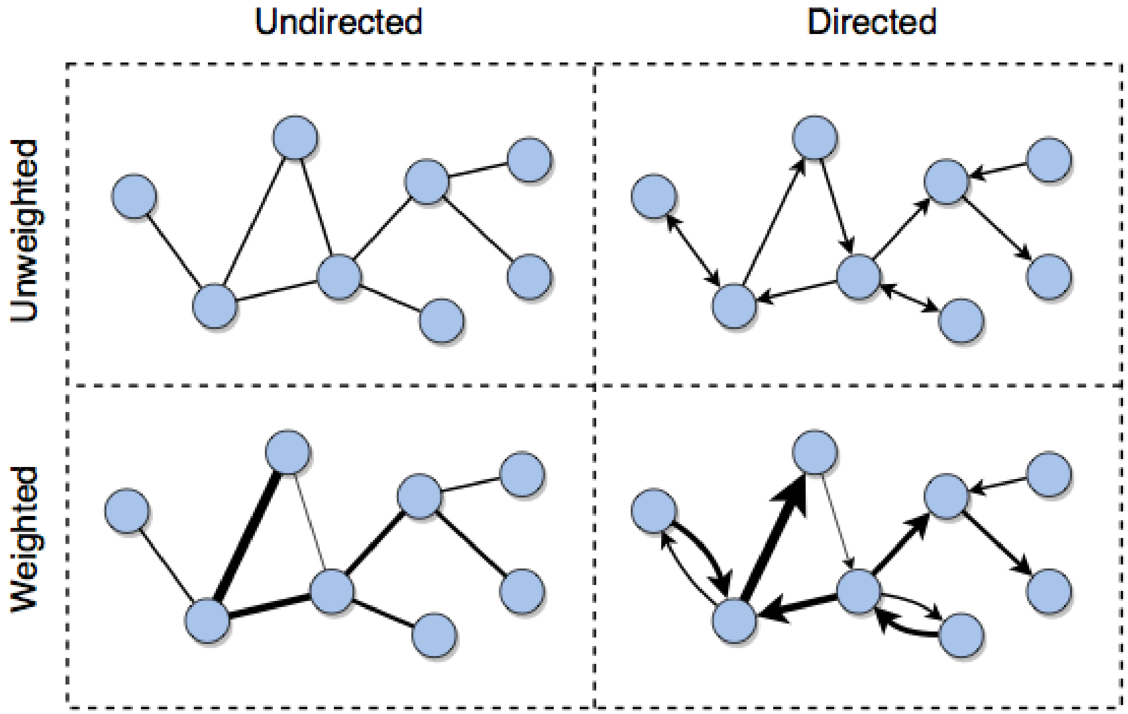
21/Apr/2023

LAST SUMMARY

- Using networks to understand complex phenomena is a type of modelling, with many (implicit or explicit) design choices
- Applying it as a black box is unlikely to tell you much
- At a large scale, it becomes increasingly useful
- In this course you will
  - Learn the basic tools of networks
  - Use them to understand some slice of Reddit

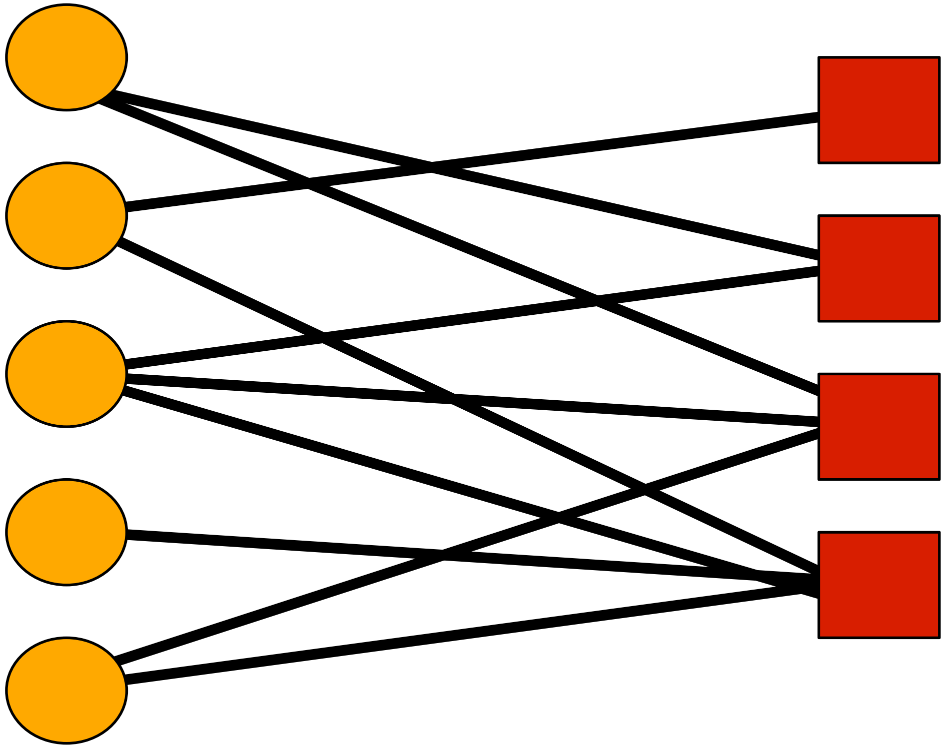
1.1 BASIC DEFINITIONS

- 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.
- A network can be **undirected** or **directed**. A directed network is also called a **digraph**. In directed networks, links are called **directed links** and the order of the nodes in a link reflects the direction: **the link (i, j) goes from the source node i to the target node j.**
  - Always check, can be the opposite
- A network can be **unweighted** or **weighted**. In a weighted network, links have associated weights: **the weighted link (i,j,w) between nodes i and j has weight w.** A network can be both directed and weighted, in which case it has directed weighted links.



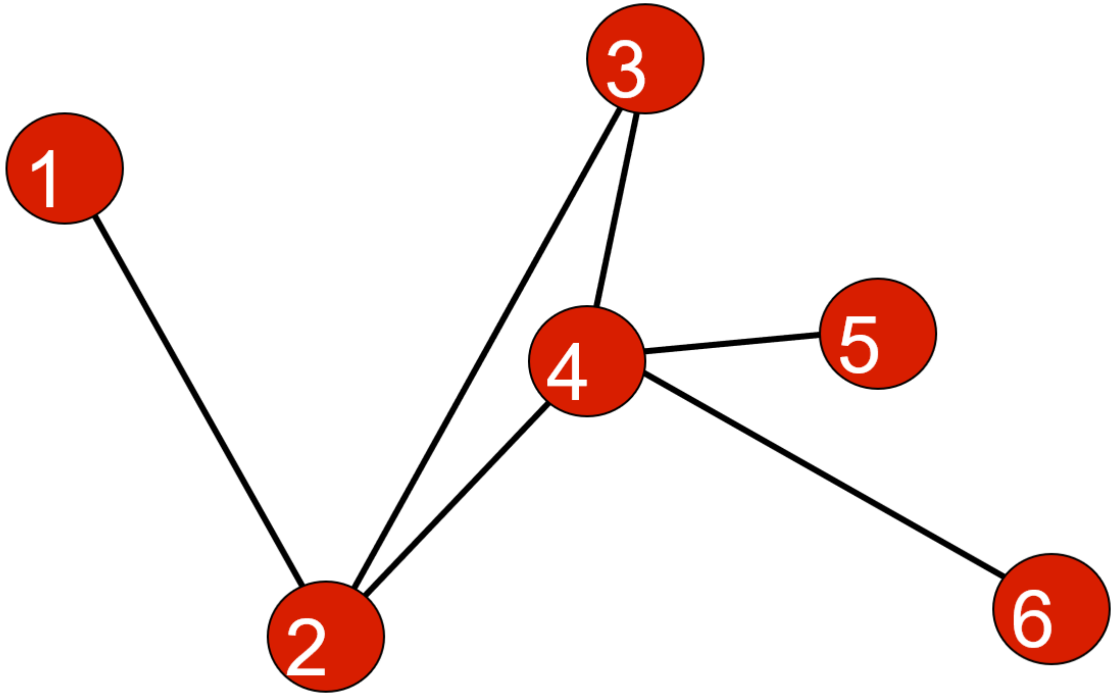
1.1 BASIC DEFINITIONS

- Bipartite networks
  - Two groups of nodes such that links only connect nodes from different groups and not nodes from the same group
- Examples
  - Movies and between
  - Songs and artists
  - Classes and students
  - Products and customers



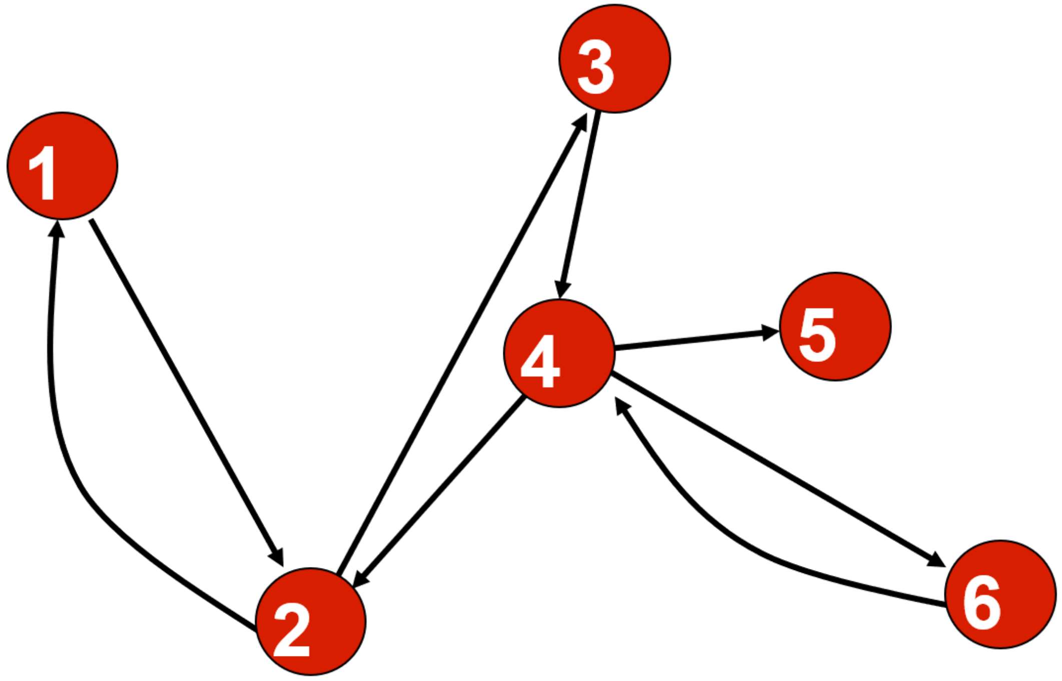
1.2 HANDLING NETWORKS IN CODE

```
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),…])
…
list(G.nodes())
list(G.edges())
list(G.neighbors(4))

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



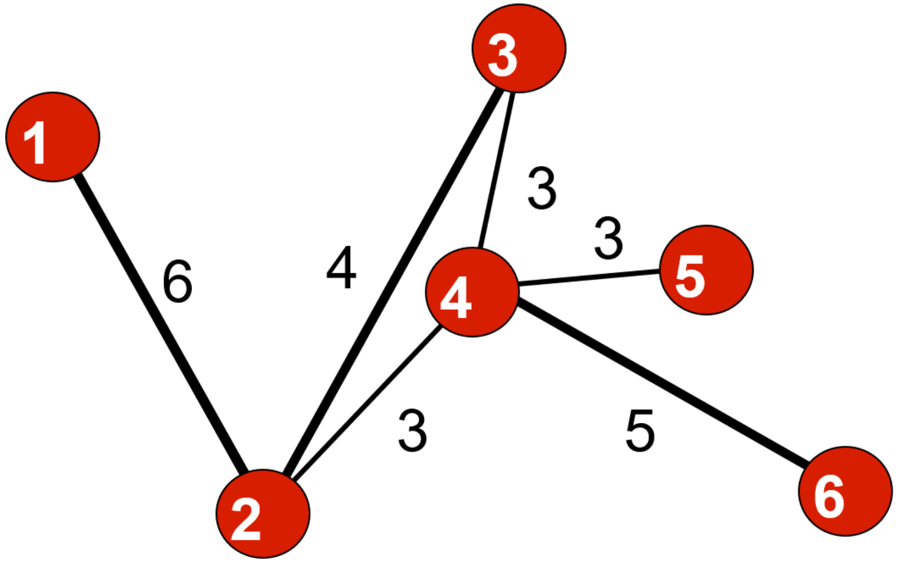
1.2 HANDLING NETWORKS IN CODE

```
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)
```



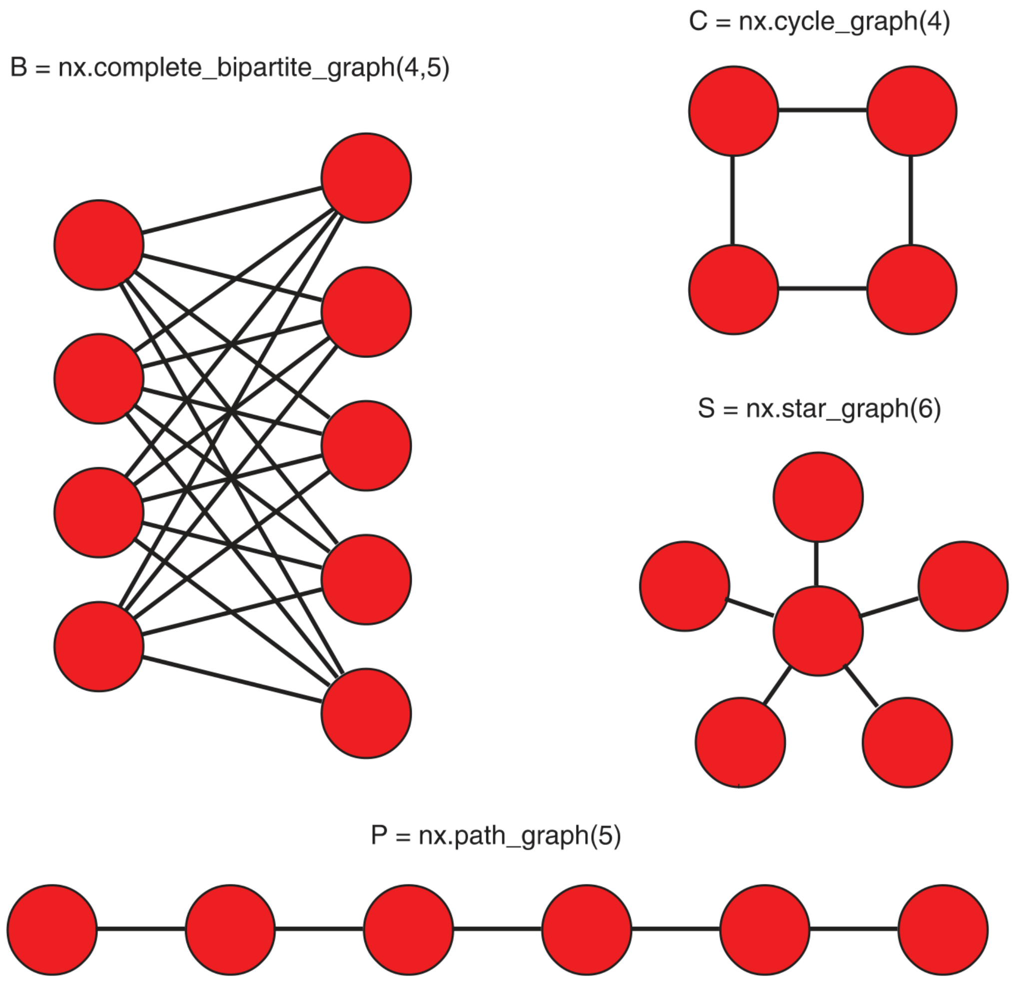
1.2 HANDLING NETWORKS IN CODE

```
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))
```



1.2 HANDLING NETWORKS IN CODE

- Network generators
  - **A lot** of them
  - Includes all the models we'll talk in Chapter 5
    - Watts-Strogatz
    - Erdos-Renyi
    - Barabasi-Albert



1.3 DENSITY AND SPARSITY

- 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
- Large networks are usually very sparse
  - $L \sim N$
  - $L_{max} \sim N^2$
- Facebook: $N \approx 10^9,\ L \approx 10^3 N \therefore d \approx 10^{-6}$

1.3 DENSITY AND SPARSITY

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

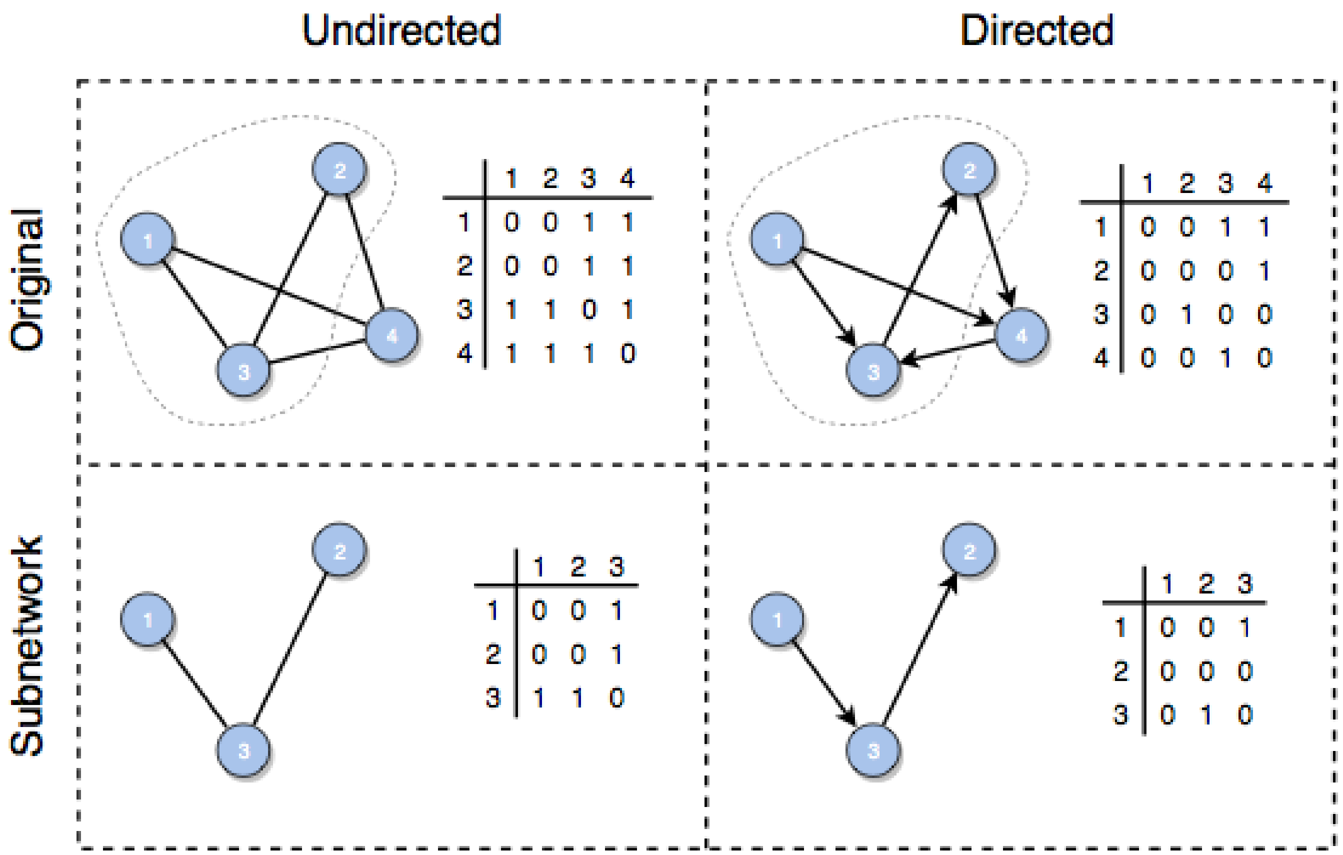
Table 1.1 Basic statistics of network examples. Network types can be (D)irected and/or (W)eighted. When there is no label the network is undirected and unweighted. For directed networks, we provide the average in-degree (which coincides with the average out-degree).

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

1.4 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

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



1.5-1.7 DEGREE

- The **degree** k_i of a node is its number of links, or neighbors
  - Most basic metric of a node, many other build on it

```
G.degree()
```

- Directed network
  - **In-degree**: number of incoming links k_i^{in}
  - **Out-degree**: number of outgoing links k_i^{out}

```
D.in_degree()
D.out_degree()
D.degree() #sum of both
```

- Weighted network
  - **Strength** (or weighted degree): $s_i = \sum_j w_{ij}$

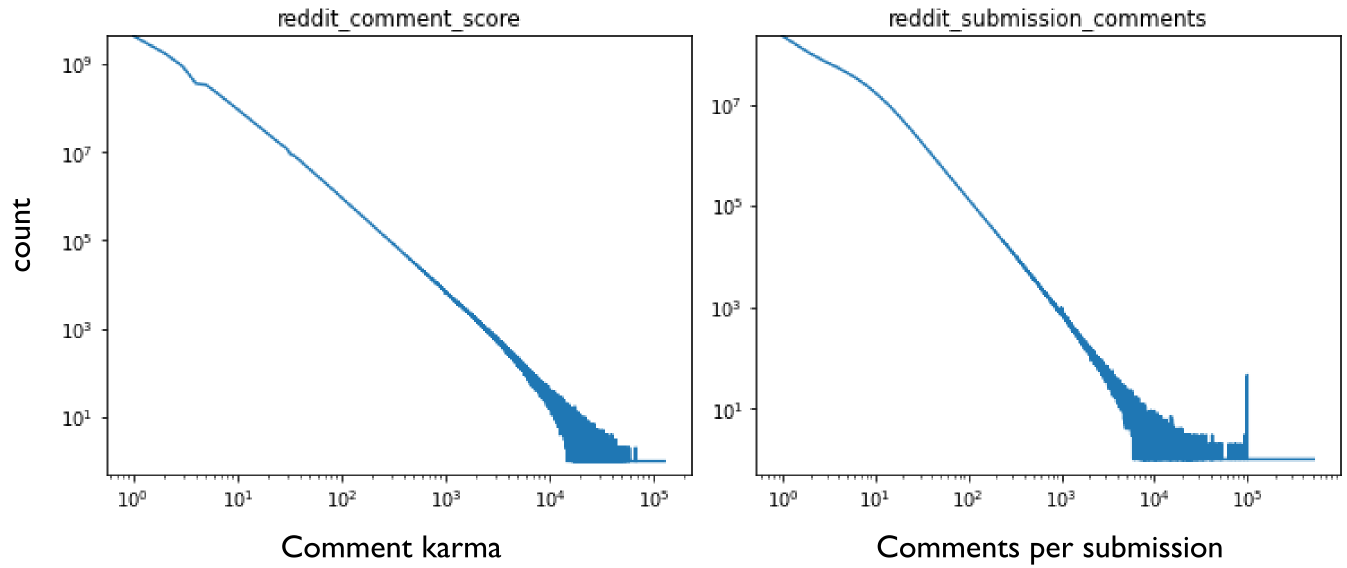
```
G.degree(weight='weight')
```

- Weighted, directed network
  - In-strength: $s_i^{in} = \sum_j w_{ji}$
  - Out-strength: $s_i^{out} = \sum_j w_{ij}$

```
D.in_degree(weight='weight')
D.out_degree(weight='weight')
D.degree(weight='weight')
```

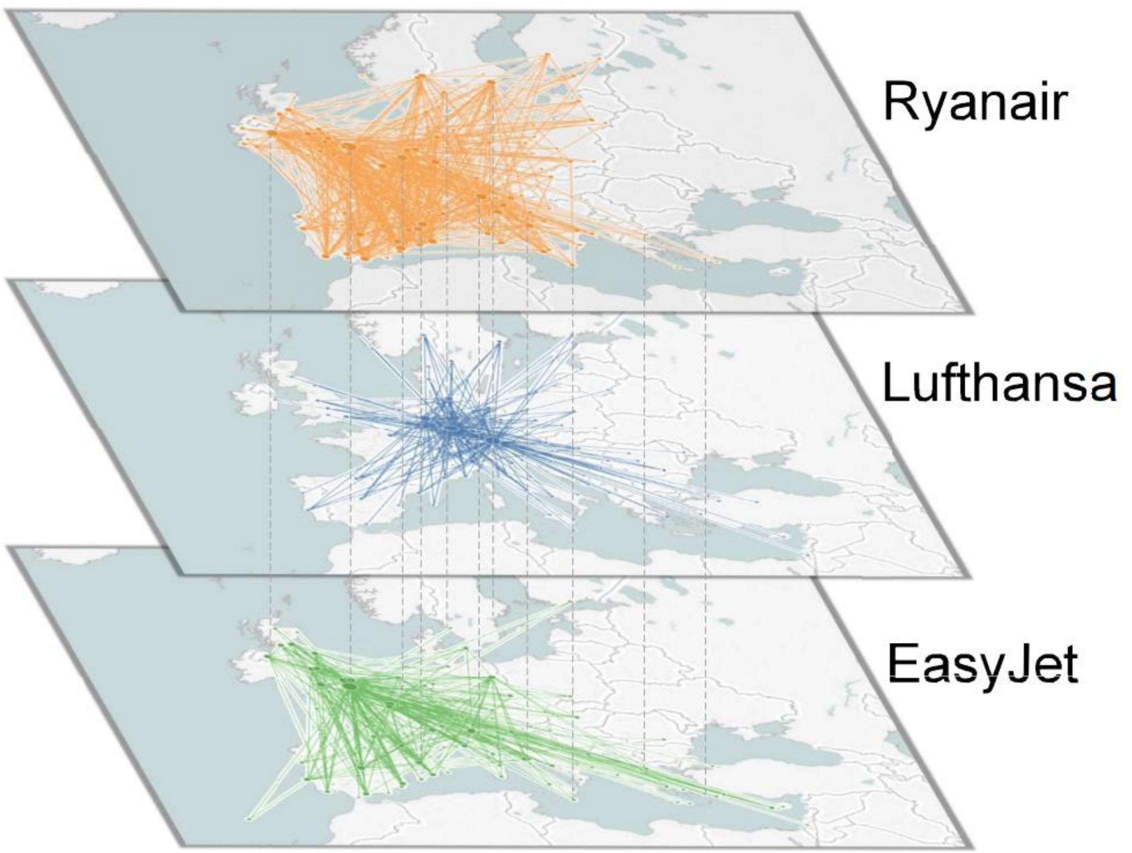
1.5-1.7 DEGREE

- **Average degree**: $\langle k \rangle = \frac{\sum_i k_i}{N}$
  - $\langle k \rangle = d(N-1)$
  - $d = \frac{\langle k \rangle}{N-1} = \frac{\langle k \rangle}{k_{max}}$
- Warning: most large networks we study are **heavy-tailed**, and $\langle k \rangle$ is not informative



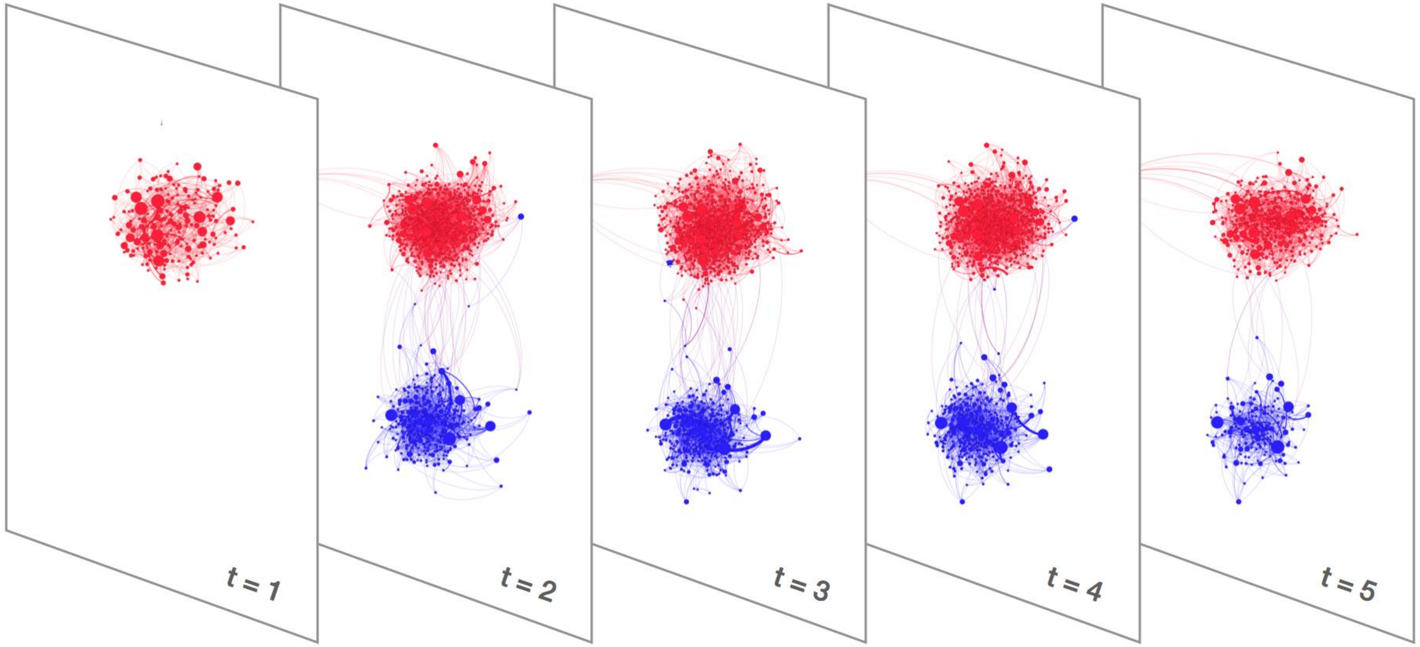
1.8 MULTILAYER AND TEMPORAL NETWORKS

- **Multilayer networks**
  - Each layer has its own nodes and links, connected in some form
  - **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: air transportation network with multiple companies



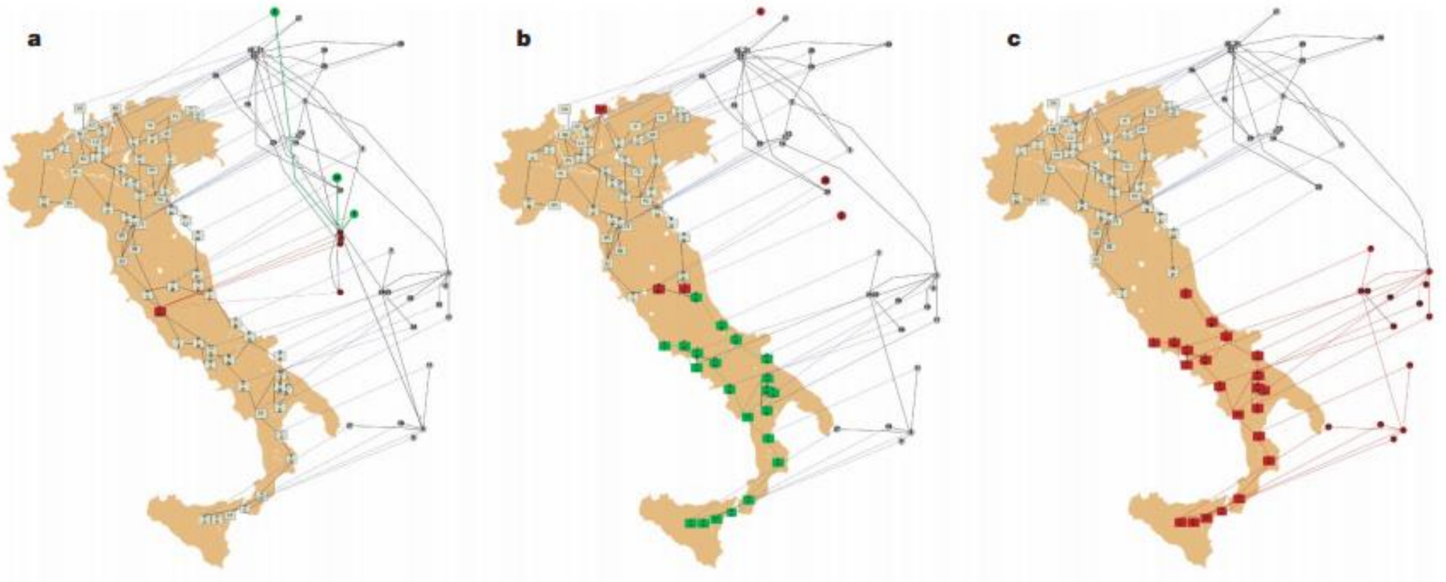
1.8 MULTILAYER AND TEMPORAL NETWORKS

- **Temporal networks**
  - A multiplex in which the layers represent links at different times (temporal **snapshots**)
  - Example: any evolving network (e.g. social media)
  - Harder to analyze, hard to find data



1.8 MULTILAYER AND TEMPORAL NETWORKS

- Networks of networks
  - Each layer in a multilayer network can be a network
  - Interactions between layers can be very complex, leading to unexpected behavior
  - Example: cascade failure in power grids
  - Object of intense research



(2003 Italy blackout)

- In this course:
  - Single layer networks
  - One link type
  - No self-loops

1.9 NETWORK REPRESENTATIONS

- **Adjacency matrix**
  - *N* x *N* matrix where each element $a_{ij} = 1$ if *i* and *j* are adjacent, zero otherwise
  - Comes with useful mathematical properties
  - Symmetric if undirected
  - Real numbers if weighted
  - Degree/strength is line sum
  - If directed:
    - In-degree: column sum
    - Out-degree: row sum

	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

	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

```
print(nx.adjacency_matrix(W))
```

1.9 NETWORK REPRESENTATIONS

- Adjacency matrix too memory-heavy for large networks ($\sim N^2$)
- **Adjacency list**: list of neighbors of each node
- **Edge list**: list of node pairs (links) in the network

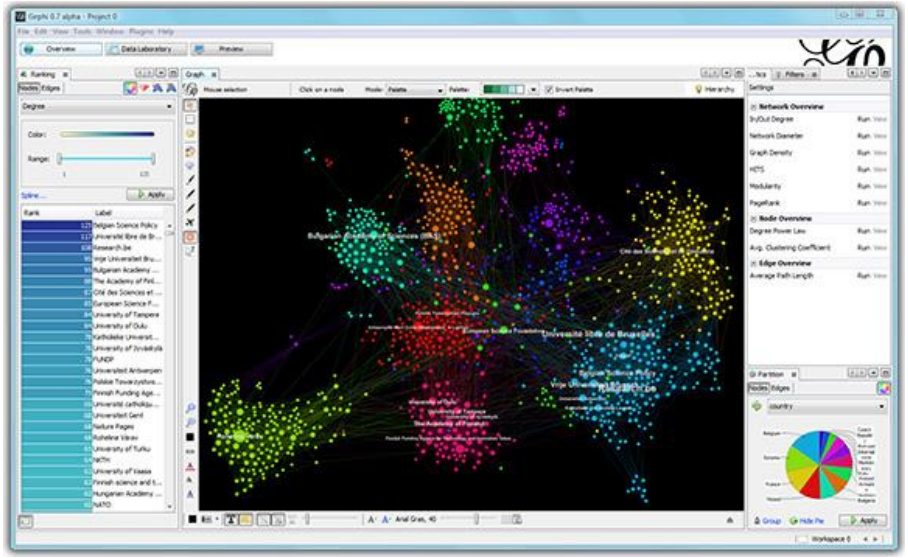
```
nx.write_adjlist(G, "netfile.adjlist")
G2 = nx.read_adjlist("netfile.adjlist")
```

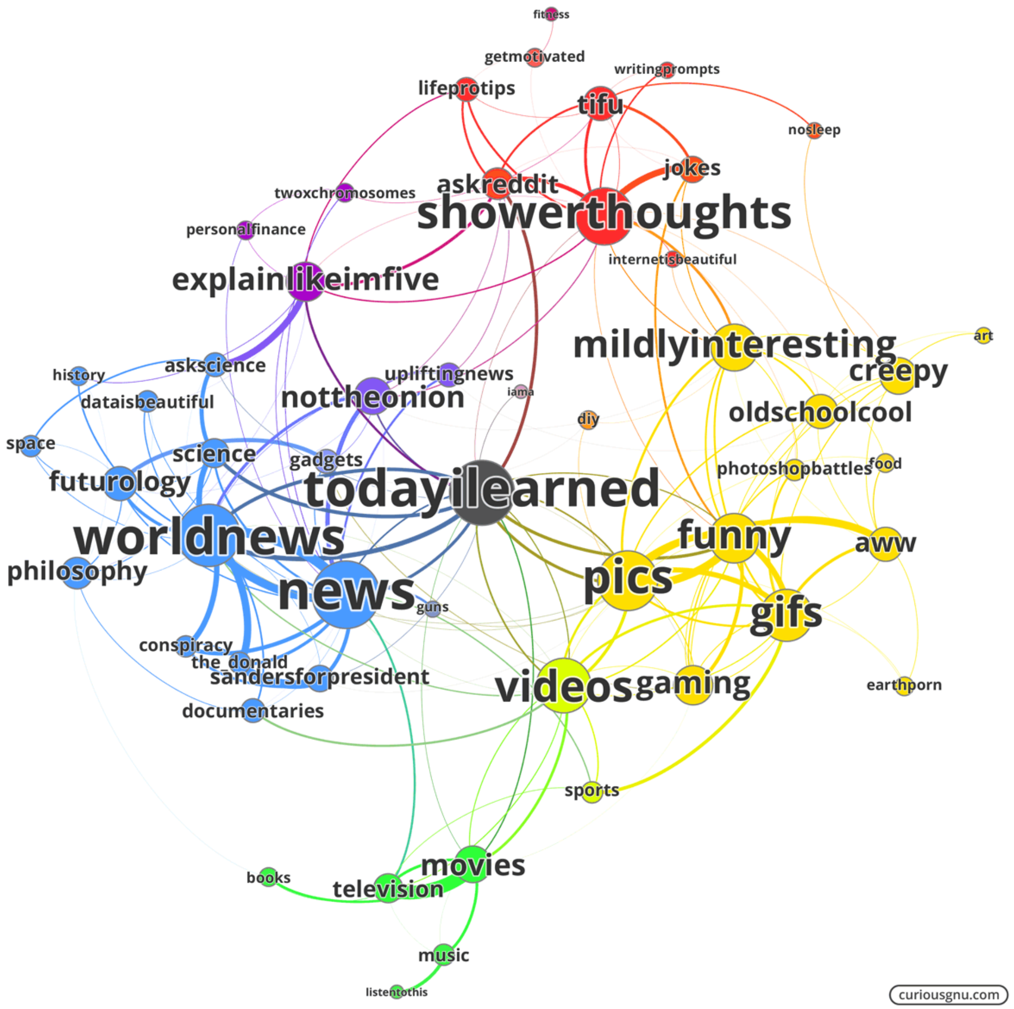
```
nx.write_edgelist(G, "netfile.edgelist")
G3 = nx.read_edgelist("netfile.edgelist")
```

```
nx.write_weighted_edgelist(W, "wf.edges")
W2 = nx.read_weighted_edgelist("wf.edges")
```

1.10 DRAWING NETWORKS

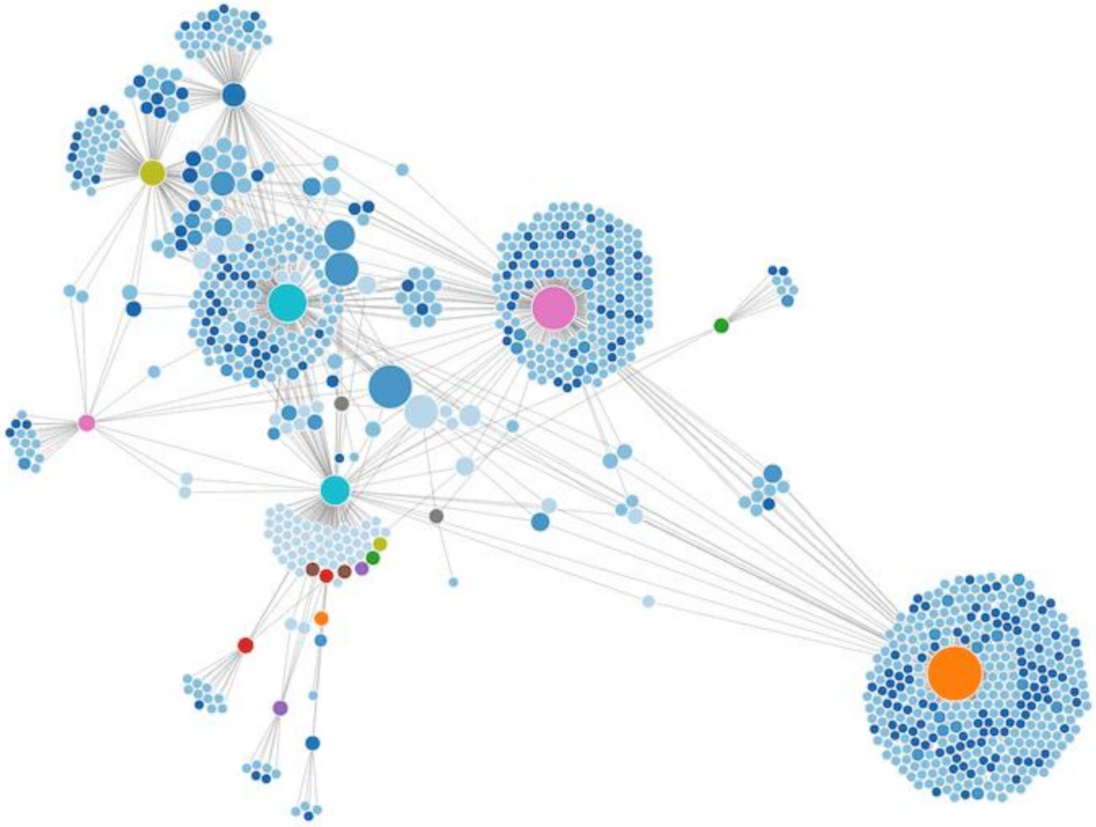
- Network visualization is an artform
- NetworkX Viz is based on matplotlib
  - Completely customizable
  - Lots of boilerplate code
  - Github Copilot/chatGPT helps a lot
- Many alternatives, like gephi





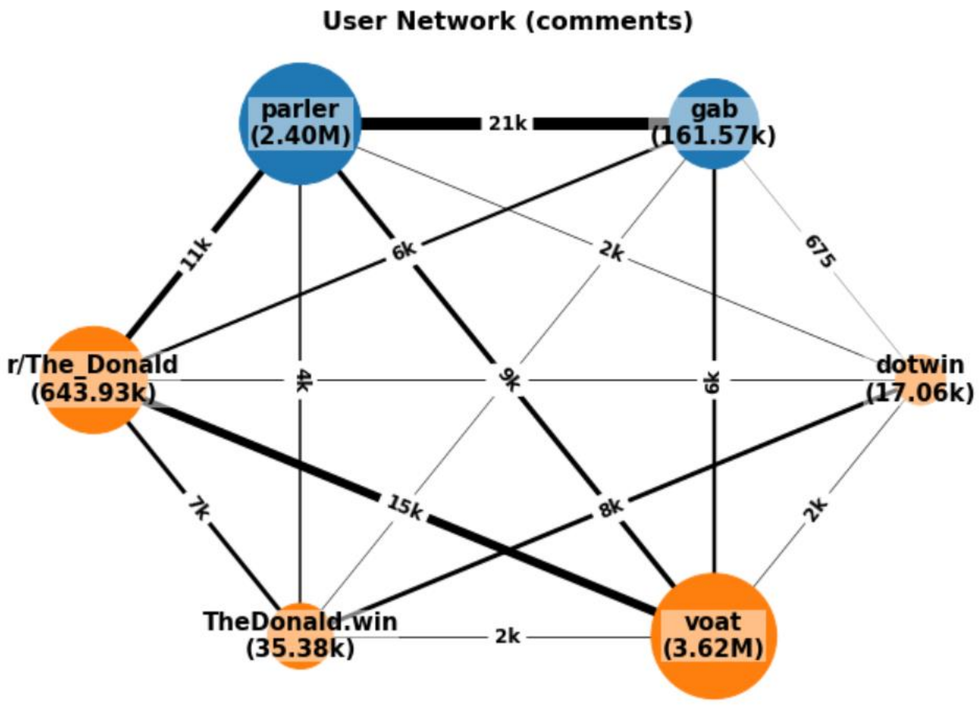
1.10 DRAWING NETWORKS

- Define a layout
  - By hand (e.g. embedded networks)
  - An algorithm
- Force-directed (spring) layout
  - Connected nodes are placed near each other
  - Links have similar length
  - Link crossings are minimized
  - Easier to spot community structure if it exists
- Draw the graph elements, either all at once or one by one (more control)



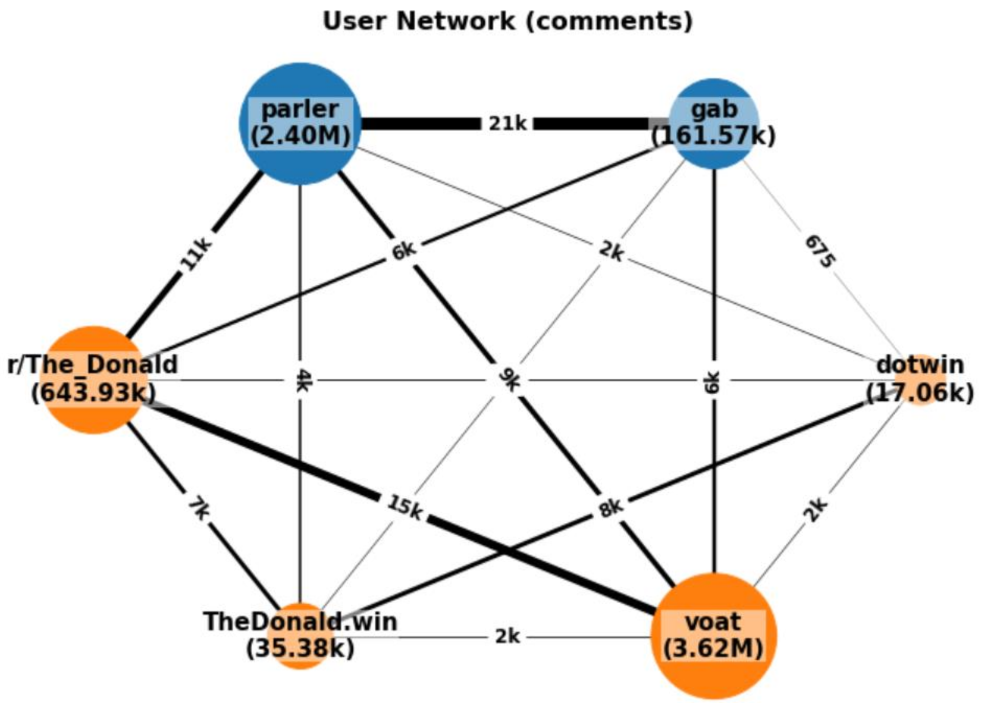
1.10 DRAWING NETWORKS

```
# Loads data
G = nx.from_pandas_edgelist(df, 'site1', 'site2', weight_attribute,
create_using=nx.DiGraph)

# Adds user count as node attribute
nx.set_node_attributes(G, name='users', values=platform_count_dict)

# node colors
node_colors_dict = {
    'dotwin': 'tab:orange',
    'gab': 'tab:blue',
    'parler': 'tab:blue',
    'r/The_Donald': 'tab:orange',
    'TheDonald.win':'tab:orange',
    'voat': 'tab:orange'}
node_colors = [node_colors_dict[x] for x in G.nodes()]

# node layout
pos = nx.circular_layout(G)

# node labels
node_labels = {}
for node in G.nodes():
    node_labels[node] = '{}\n({:.2h})'.format(node,
Float(G.nodes[node]['users']))

# node size
users_min = min([G.nodes[node]['users'] for node in G.nodes])
node_sizes = [node_size*(1+np.log(G.nodes[node]['users']/users_min)) for
node in G.nodes]

# edge width
edge_width = [edge_size*G.edges[e][weight_attribute]/100 for e in G.edges]
```



1.10 DRAWING NETWORKS

```
fig, ax = plt.subplots(figsize=figsize)
fig.patch.set_facecolor('white')
ax.set_title(title, fontsize=16, fontweight='bold')

# Draws nodes and edges
#nx.draw_networkx(G, pos, arrowstyle='-|>',arrowsize=40, width = edge_width,
node_size = node_sizes, node_color=node_colors, font_size=15,
with_labels=False, arrows=True, **{'connectionstyle':'arc3,rad=0.2’})

nx.draw_networkx_nodes(G, pos, ax=ax, node_size = node_sizes,
node_color=node_colors)
nx.draw_networkx_labels(G, pos, ax=ax,labels=node_labels, font_size=15,
font_weight='bold', bbox=dict(facecolor='white', alpha=0.5,
edgecolor='none', pad=0.5))
nx.draw_networkx_edges(G, pos, ax=ax, width = edge_width,
edge_color='black', node_size = node_sizes, arrows=True,
connectionstyle='arc3,rad=0.1')

# draws labels on edges with weight > edge_label_lim
for edge in G.edges(data=True):
    w = edge[2][weight_attribute]
    if w > edge_label_lim:
        nx.draw_networkx_edge_labels(G, pos,
edge_labels={(edge[0],edge[1]):'{:0.0f} %'.format(w)}, font_size=12,
font_weight='bold', clip_on=True, label_pos=0.3,
bbox=dict(facecolor='white', alpha=0.5, edgecolor='none', pad=0.5))

# Beautifies plot
fig.patch.set_facecolor('white')
ax.spines[:].set_visible(False)
ax.set_xlim([1.05*x for x in ax.get_xlim()])
ax.set_ylim([1.1*y for y in ax.get_ylim()])
```



SUMMARY

- Many types of networks
  - Directed/Undirected
  - Multilayer
    - Multiplex
      - Temporal
- Basic metrics are already useful
- Network visualization is cumbersome, but there are ways
  - Copilot/chatGPT
  - Gephi