Network Analysis

Greg Baker

Week 11 — 21st October 2025

Agenda

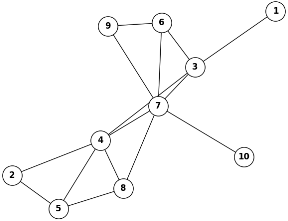
- Network Analysis Intro
- Characterising Graphs
- Important Nodes
- Visualising Graphs
- Finding Communities
- Advanced Topics

What is Network Analysis?

- Analysis of problems involving dependencies between observations
- Examples: traffic flows, social influences, protein interactions, fraud detection, social media networks
- Sometimes we do network analysis on its own (e.g. find the key influencers)
- Sometimes it is an input into a numeric model (e.g. is being central to something highly predictive?)

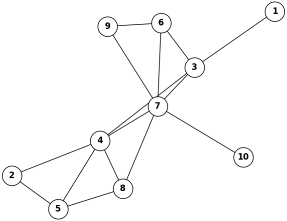
What is a Network?

- A graph consisting of nodes (vertices) connected by edges
- **Nodes:** entities such as people, accounts or objects
- **Edges:** relationships or flows linking pairs of nodes
- Directed graphs (DiGraphs) have arrows
- MultiGraphs have parallel edges



Building a Network by Hand

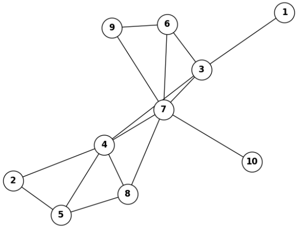
```
import networkx as nx
G = nx.Graph()
G.add_edge(1, 3)
G.add_edge(2, 4)
G.add_edge(2, 5)
G.add_edge(3, 4)
G.add_edge(3, 6)
G.add_edge(3, 7)
G.add_edge(4, 5)
G.add_edge(4, 7)
G.add_edge(4, 8)
G.add_edge(5, 8)
G.add_edge(6, 7)
G.add_edge(6, 9)
G.add_edge(7, 8)
```



Calling conventions for networkx functions

```
print(networkx.degree(G, 5))
```

3



Visualising with draw_networkx

The simplest way to look at a small graph is to use NetworkX's draw_networkx function. A useful first step is to compute a **spring layout**, which models the graph with repelling nodes and attractive edges so clusters naturally separate.

```
layout = networkx.spring_layout(Graph)
networkx.draw_networkx(Graph, pos=layout)
```

The spring layout gives an initial aesthetic position for the nodes that we can refine later with other layouts if needed.

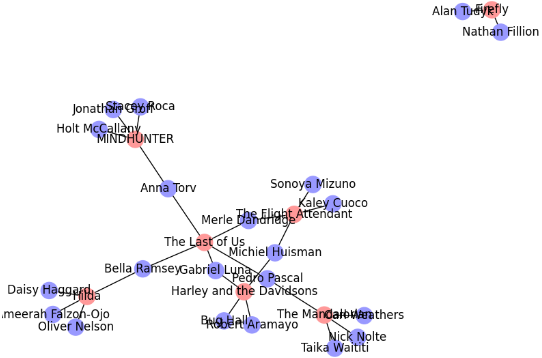
D3 Visualisation

If you know JavaScript. . . D3 is a JavaScript library for creating dynamic, interactive visualisations in the browser. The exported JSON can be loaded into a D3 template to explore the network interactively.
Samples:

- `https://pausanias.symmachus.org/network_viz`
- `http://merah.cassia.ifost.org.au/game-explorer`

How we might do an actor network

- Connect actors to shows to form a **bipartite** network (shown)
- Or, connect actors directly when they co-star in a show

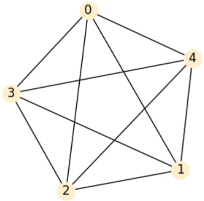


Characterising Graphs

Is it Nearly a Lattice?

Density: *Ratio of actual edges to maximum possible edges*

- A lattice has every possible edge
- Maximum possible density




```
print(networkx.density(lattice_graph))
print(networkx.density(full_actor_graph))
```

```
1.0
0.07
```

Are there Islands?

Connected Components

```
print(list(networkx.connected_components(full_actor_graph)))
```

Analysing a Component

Many kinds of network analysis assumes full connectivity — it is possible to get from all nodes to all other nodes somehow.
We can make it true by analysing each island separately.

```
all_islands =
    networkx.connected_components(full_actor_graph)
actor_graph = networkx.subgraph(full_actor_graph,
    list(all_islands)[0])
```

Shortest paths

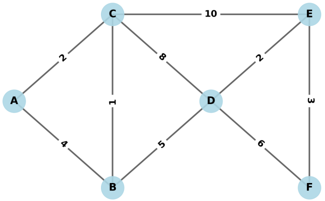
- The path with the lowest total weight (or fewest edges) between two nodes
- Edge weights typically represent distance or cost
- Algorithms such as Dijkstra or breadth-first search can find them

Dijkstra's Shortest Path Algorithm

- Start with tentative distances set to infinity except the source
- Visit the closest unvisited node and update neighbours
- Repeat until all nodes are processed
- Guarantees optimal paths for positive edge weights
- https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html
- https://www.davbyjan.com/

Dijkstra Example: Setup

Find shortest path from A to E



- Start at A, distance $= 0$
- All others: distance $= \infty$
- Visit closest unvisited node
- Update neighbors' distances

Dijkstra Example: Step by Step

Step	A	B	C	D	E	F	Action
0	0	∞	∞	∞	∞	∞	Start at A
1	0	4	2	∞	∞	∞	Visit A, update B,C
2	0	3	2	10	12	∞	Visit C, update B,D,E
3	0	3	2	8	12	∞	Visit B, update D
4	0	3	2	8	10	14	Visit D, update E,F
5	0	3	2	8	**10**	13	Done! E reached

Shortest path: A $\rightarrow$ C $\rightarrow$ B $\rightarrow$ D $\rightarrow$ E (total: 10)

Computing Shortest Paths in NetworkX

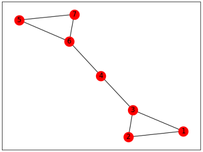
```python
import networkx as nx

# Create graph with weights
G = nx.Graph()
G.add_edge('A', 'B', weight=4)
G.add_edge('A', 'C', weight=2)
# ... add more edges ...

# Find shortest path
path = nx.shortest_path(G, source='A', target='E',
                                  weight='weight')
print(path)  # ['A', 'C', 'B', 'D', 'E']

# Get path length
```

June 2025: Dijkstra's Algorithm has been beaten

- Team from Tsinghua University (China) won Best Paper at top theory conference (STOC 2025)
- Not many people believed that it would be possible to beat Dijkstra's algorithm
- 65 years is a long time in computer science for no progress
- The big idea: don't always process nodes in sorted order

Key Metrics

- **Clustering coefficient (C)**: likelihood that two neighbours of a node are connected.
  - If three nodes are connected by at least two edges, what is the probability that they are connected by three?
  - This graph has a clustering coefficient of 0.6666



- **Average shortest path length (ℓ)**: mean distance between all pairs of nodes.
    - Find the shortest path between every pair of nodes

Calculating cluster metrics with networkx

Probability of triadic closure:

```
networkx.average_clustering(Graph)
```

Average Shortest Path:

```
nx.average_shortest_path_length(Graph)
```

Characterising Networks

		Clustering	
		Small	**Large**
Avg. Path	**Small**	Random graph (Erdös–Rényi)	Small-world (Watts–Strogatz)
	Large	Line / chain	"Unusual" / 2–D lattice

Clustering coefficient

- Small: $C < 0.05$ (few triangles)
- Large: $0.3 \lesssim C \lesssim 0.7$ (many triangles)

Average path length

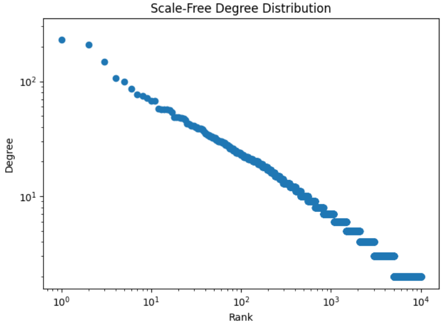
- Small: scales like $\log N$ (often $\ell < 10$ for $N \approx 10^4$)
- Large: scales linearly, e.g. $\ell \approx N/2$ for a chain

Small World Networks

- Characterised by hubs and low-degree nodes
- More robust against single-node failures
- Information or disease can traverse the network rapidly due to short paths
- Examples: website navigation, airline routes, power grids, social networks
- Famous for the "six degrees of separation" phenomenon
- Recognising a small-world structure guides modelling of diffusion and resilience

Scale-Free Networks

- Degree distribution follows a power law
- Formed by **preferential attachment**
  - New nodes prefer to attach to high-degree nodes
  - Not quite "winner-takes-all" but close
- Visualised with a log-log plot
- Average path can be ultra-small ($\approx \log \log N$) thanks to hubs
- Clustering often moderate to high



Characterisation Benchmarks

	Clustering C	**Path** ℓ	**Typical example**
Random (ER)	< 0.05	$\sim \ln N / \ln \bar{k}$	phone call logs
Small-world	0.3–0.7	$\sim \log N$	power grid, airline routes
Scale-free	≥ 0.1	$\sim \log \log N$	web, Twitter followers
Chain	0	$\sim N/2$	ring buffer, token ring LAN
Lattice	> 0.3	$\sim N^{1/d}$	grid sensor network

Centrality

Important Nodes (Centrality)

- **Degree Centrality:** Number of connections
- **Closeness Centrality:** Average shortest-path distance from a node to all others
- **Betweenness Centrality:** Shortest paths through the node. (Slow, but you can use approximations)
- **Eigenvector Centrality:** Importance of connected nodes
  - If we released an equal number of random-walking ants on each nodes and waited, where would the ants end up?
- **Page Rank:** Importance based on incoming links
- **Current Flow Betweenness:** Imagine edges are resistors. How much current flows through each node? (Very slow.)

Centrality in Code

```
networkx.degree_centrality(Graph)
networkx.closeness_centrality(Graph)
networkx.betweenness_centrality(Graph)
networkx.eigenvector_centrality(Graph)
networkx.pagerank(Graph)
networkx.current_flow_betweenness_centrality(Graph)
```

Ego Networks

- Subgraph containing a node (the ego) and its neighbours
- Useful for analysing local structure or influence
- Extract with `networkx.ego_graph(Graph, 'Pedro Pascal')`
- Visualise to inspect potential information flow around the ego

Ego Network in Code

```
ego = max(Graph, key=Graph.degree)
ego_net = networkx.ego_graph(Graph, ego)
networkx.draw(ego_net)
```

Finding Communities

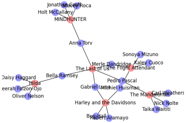
- Maximise intra-community edges, minimise inter-community edges
- Modularity optimisation
- Algorithms aim for dense intra-community edges

Louvain's Algorithm

- **Modularity** measures how much more densely connected nodes are within a community compared to random wiring
- Start with every node in its own community
- Move nodes to neighbouring communities to greedily increase this modularity score
- Collapse each community into a single node and repeat the search
- Iteration stops when no moves improve modularity, yielding a hierarchy
- Available via `networkx.algorithms.community.louvain_communities`

Community Detection in Code

```
from networkx.algorithms import community
communities =
    community.greedy_modularity_communities(actor_
len(communities)
```




```
communities=[frozenset({'Daisy Haggard', 'Bella Ramsey', 'Hilda',
  'Ameerah Falzon-Ojo', 'Oliver Nelson'}),
 frozenset({'Bug Hall', 'Harley and the Davidsons', 'Michiel Huisman',
   'Robert Aramayo', 'Gabriel Luna'}),
 frozenset({'Nick Nolte', 'Taika Waititi', 'The Mandalorian',
   'Carl Weathers'}),
 frozenset({'Sonoya Mizuno', 'Kaley Cuoco', 'The Flight Attendant',
    'Merle Dandridge'}),
 frozenset({'Jonathan Groff', 'Stacey Roca', 'Holt McCallany',
    'MINDHUNTER'}),
 frozenset({'Anna Torv', 'The Last of Us', 'Pedro Pascal'})]
```

Cliques as Communities

- Fully connected subgraphs (cliques) can reveal tightly knit groups
- Very slow

```
for c in networkx.clique.find_cliques(Graph):
    print(c)
networkx.clique.cliques_containing_node(Graph,
    'A')
```

Other Useful Functions

```
graph = networkx.read_edgelist(filename)
networkx.write_edgelist(graph, filename)
df = networkx.to_pandas_dataframe(graph)
graph = networkx.from_pandas_dataframe(df)
networkx.readwrite.json_graph.node_link_data(graph)
```

More Algorithms

NetworkX includes a large collection of algorithms beyond those covered here. `http://networkx.readthedocs.io/en/stable/reference/algorithms.html`

Summary

- Networks model relationships between entities
- Networks can be characterised by a few simple-to-calculate measures
- Centrality helps identify important nodes
- Community detection groups related nodes; modularity measures its quality
- D3 enables interactive exploration of graph data