NIHR | Applied Research Collaboration South West Peninsula

@penCHORD_UoE
@peninsula_ARC

Module 4 : Network Analysis
Session 4C : Advanced Network Analysis
Elliott Coyne

"Together, Further"

HSMA 5

#hsma5isalive

Graph Metrics

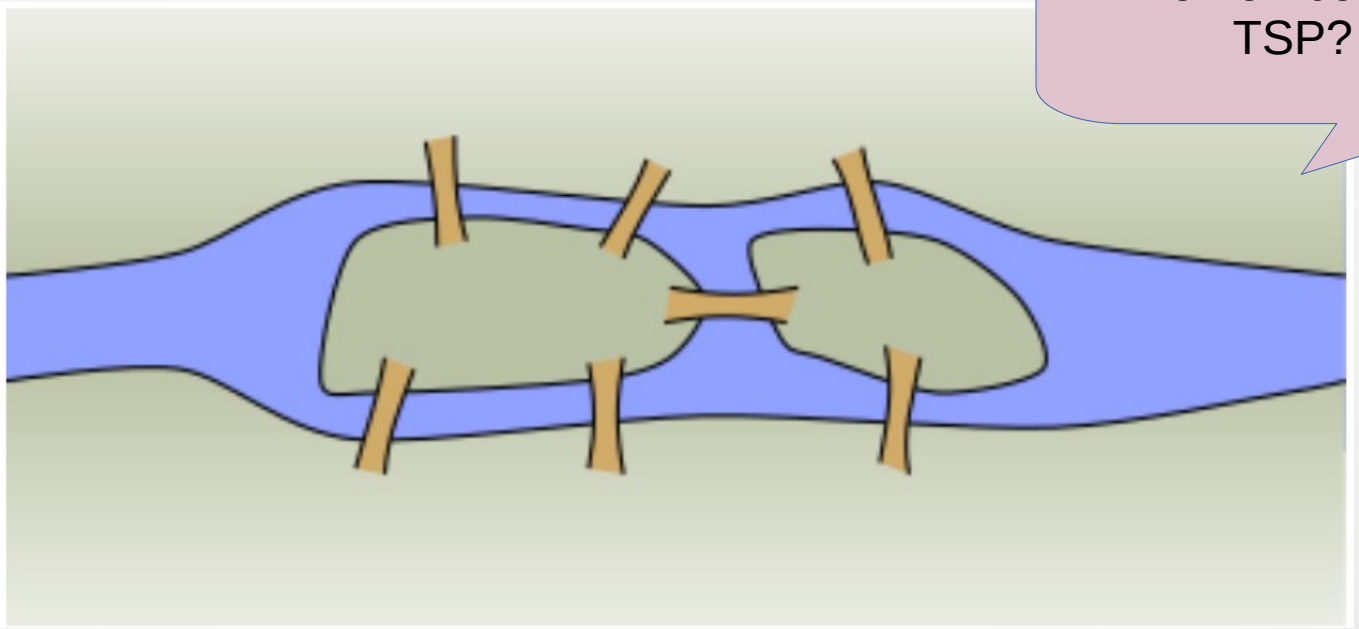
- Graph metrics inform us about the topological structure of the network and the relationships between the various nodes
- It’s a core component of network analysis – and important to be able to interpret such metrics, so as to understand what is actually happening within the network
- This understanding can then be used convey insights to others

History

Let's look at the problem that Leonhard Euler first studied in 1735

The Seven bridges of Konigsburg – 7 bridges and 2 islands...

Remember the TSP?



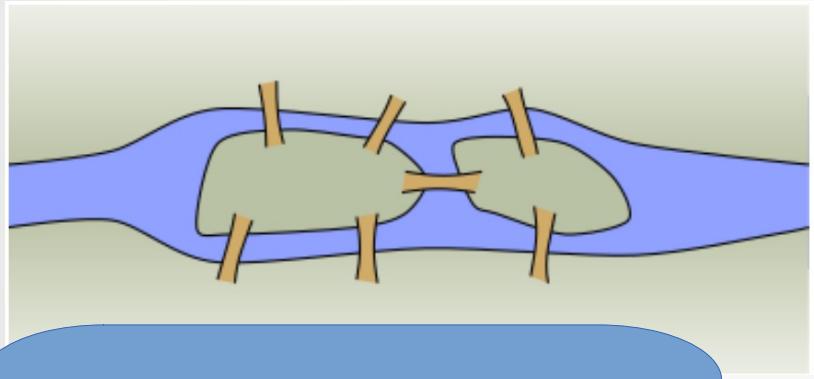
Individual Task (5 mins)
Can you plan a walk where you cross each bridge only once?
Also, represent the above as a graph – with edges and vertices.

Mini Task

5m 00s

History

Literal rendering...



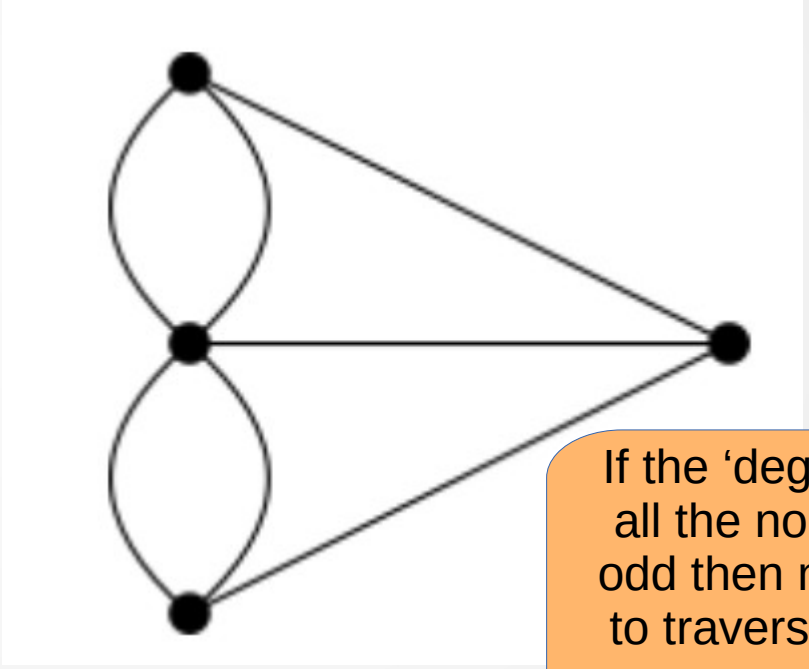
Abstracting real world problems to solve them mathematically… And graph theory was born!

Each node has 3 edges i.e. Degree of 3

Euler's graph representation…

Paths across bridges are **edges**

Banks are **vertices** (nodes)



If the 'degree' of all the nodes is odd then no way to traverse only once

Even number of nodes/ vertices
Odd number of edges
*… Unique walk **NOT** possible*

Terminology 1

A graph is composed of two key elements: Vertices and Edges

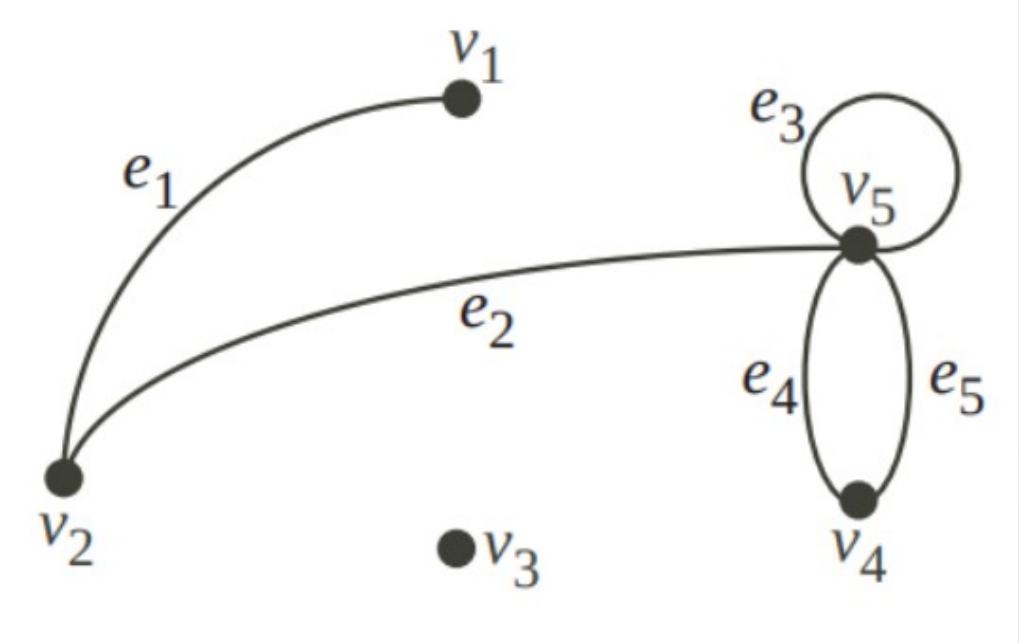
Vertices (nodes) – dots, entities

V = {v1 , v2 , ...vn }

Edges – lines, links

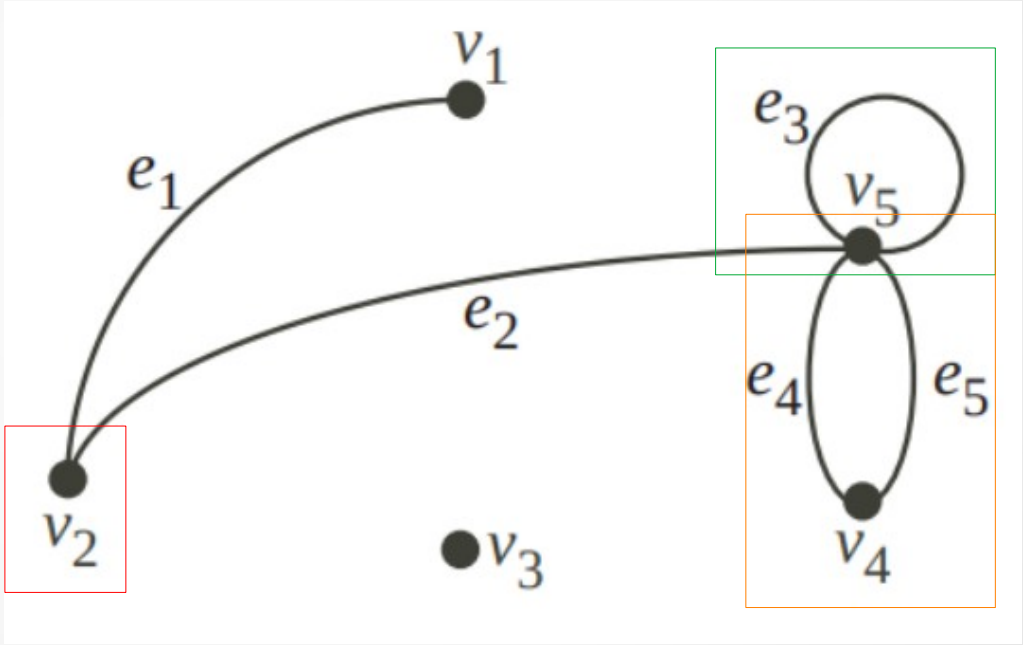
E = {(v1 , v2), (v2 , v5), …} or

E = {e1 , e2 , ...en }



Terminology 2

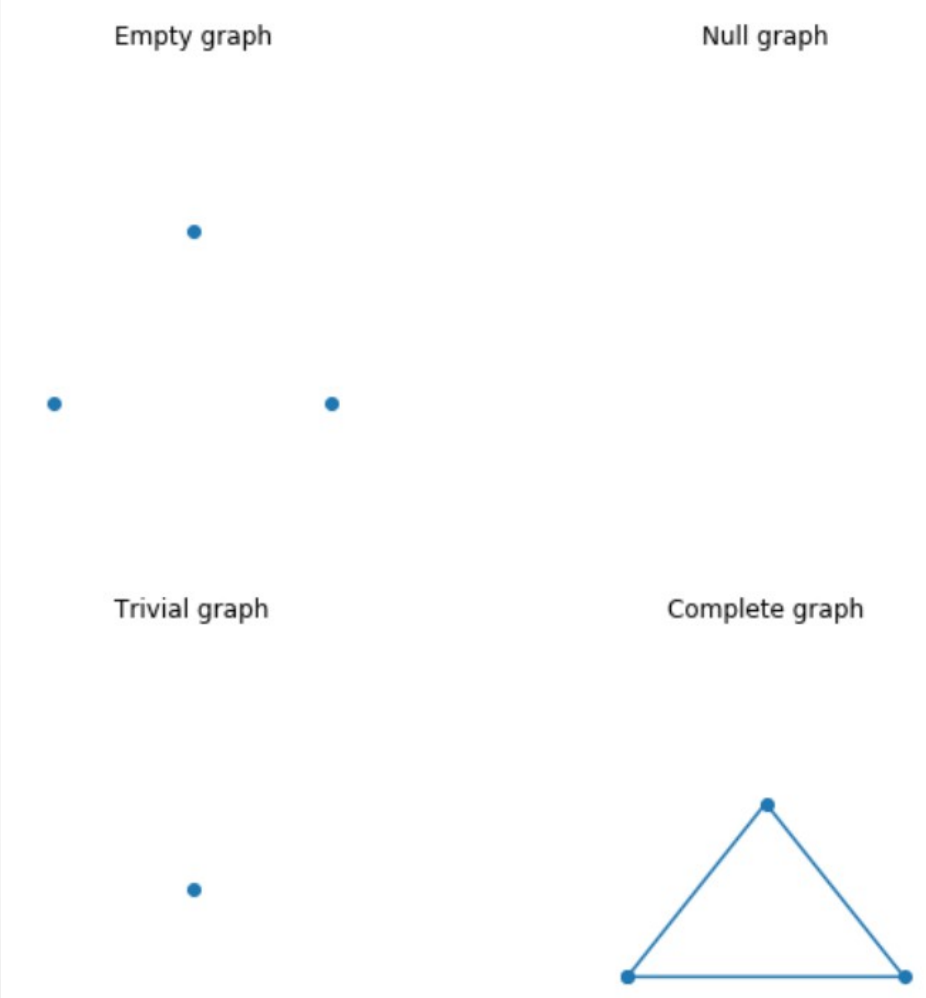
- Edges that have the same end vertices are parallel i.e., e4, e5
- An edge (v,v) is a loop or self-loop
- Edges are adjacent is they share a common vertex *i.e., e1 & e2 (v2)*
- Two vertices are adjacent if they are connected by an edge *i.e., v1 & v2 and v2 & v5.*



Simple graph types

- Empty Graph: no edges (only vertices/ nodes)
- Null graph: no vertices
- Trivial graph: one vertex
- Complete graph: every pair of vertices is adjacent

Each node is connected to every other node

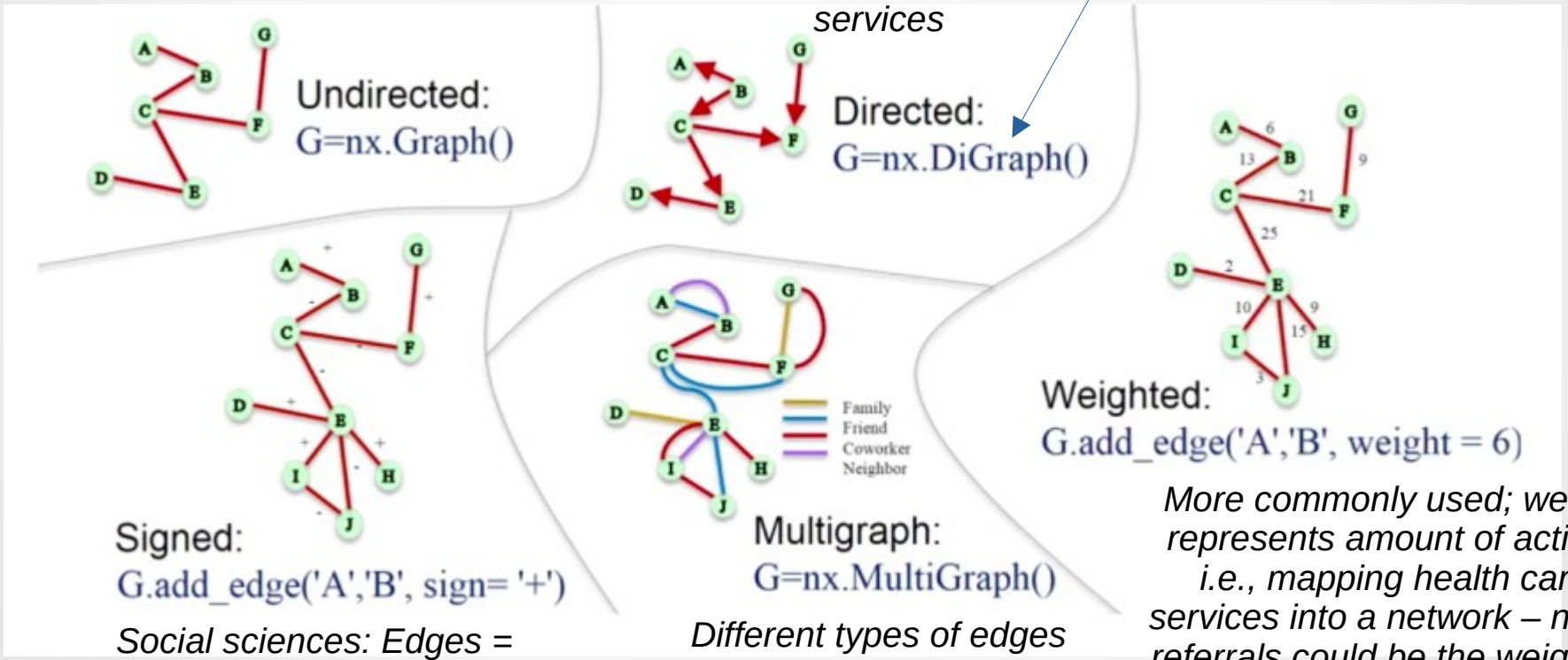


Graph types

Just a link – direction of travel doesn't matter i.e., on social media if Smith is friends with Jones, Jones is friends with Smith

Health care data… People or services

Network X commands



Social sciences: Edges = relationships which can be friendships (+) or hostility (-)

https://www.hindawi.com/journals/cin/2017/1235715/

Different types of edges (representing categories) i.e., relationships

More commonly used; weight represents amount of activity i.e., mapping health care services into a network – no. of referrals could be the weight of an edge

Common graph metrics

Network level – All of Graph

•*Number of nodes and edges*: Base description of network size and complexity
i.e. 5 nodes and 3 edges < 200 nodes and 60 edges

•*Density*: Number of links in relation to the number of possible links i.e. between 0 (‘empty’) and 1 (‘complete’)

•*Average degree*: Average number of edges intercepting a node i.e. calc. of average number of edges for all nodes across the graph- provides an idea of overall connectivity within the graph

•*Network modularity*: Connectedness of the graph i.e. how many nodes are linked together within close proximity; ‘groupings’ → Calculated and mapped to graph – *Example to follow*

Node level – Individual Nodes

•*Degree* (indegree/ outdegree): Number of edges intercepting a node
i.e., Directed Graphs have 2 types : in + out = overall ‘degree’
Indirect = ‘degree’
Good way to understand how well nodes are connected. e.g. those with higher degree are connected to more people/ services

•*Modularity*: How well connected a particular nodes are in relation to others i.e., close together

•*Eigenvector centrality*: Connectedness of nodes in relation to the whole network i.e., connectedness of node across the entire network/ graph
Can be used to identify nodes that facilitate connections across a network

More complex graph metrics

More complex measures of a graph…..

Approximation and heuristic techniques

- Connectivity
- Clique (how well nodes group together and if they form specific clusters more distant from the rest of the graph)
- Clustering (in health services clustering occurs by geography)

Centrality

- Degree centrality (i.e., which nodes are most highly connected by numbers of connections)
- Eigenvector centrality (aka ‘eigencentrality’ or ‘prestige score’, measure of influence of node in a network. i.e., high eigenvector score means that a node is connected to many nodes who themselves have high scores e.g., influential)
- Katz centrality (i.e., alternative variant measure to eigenvector centrality – calculated differently)

- **Communities** (i.e., how nodes group together) *c.f. Cliques, Clustering)*
  - k-clique communities
  - greedy modularity communities

Random walks through graph

- **Isolates** (i.e., nodes not connected to the rest of the graph, not interacting with anything else. Operationally poses the Q “should this be happening?”)
- **Link analysis** (i.e., possible connections that could form between nodes – predictive element i.e., what connections *could* form given closeness of nodes to each other and nearby connections i.e., care pathways)
- **Shortest paths** (i.e.,similar to Euler’s bridge problem – which is shortest path and which nodes and edges should be traversed.)
- **Similarity measures** (i.e., changes overtime to look at overall structure and size of network changes from one time period to another – looking at whole graphs).

NetworkX Documentation

Further reading form previous slides (especially if you're into the maths!)….

The documentation for NetworkX and all of the various graph metrics can be found here:

https://networkx.org/documentation/stable//reference/algorithms/index.html

… And take a break – 10 mins

Metric Highlight: Degree

Description
Degree refers to the number of edges intersecting a vertex. In an undirected graph each vertex (node) has a single degree value. In a directed graph each vertex has two values; an in-degree and an out-degree.
Average (in/out) degree is used to describe the degree of a graph.
May look at other descriptive stats – min, max, SD, etc, or plot a histogram.

Application
The *degree is an indicator of system complexity* at the nearest neighbour level (i.e., those nodes that are the closest). In operational modelling we can understand this as the number of interfaces a service has with other services. May want to increase or decrease number of interfaces that different health services teams have. Can be a lot for teams to have to remember how to deal with different teams, or services (if they have different processes)

NetworkX Function

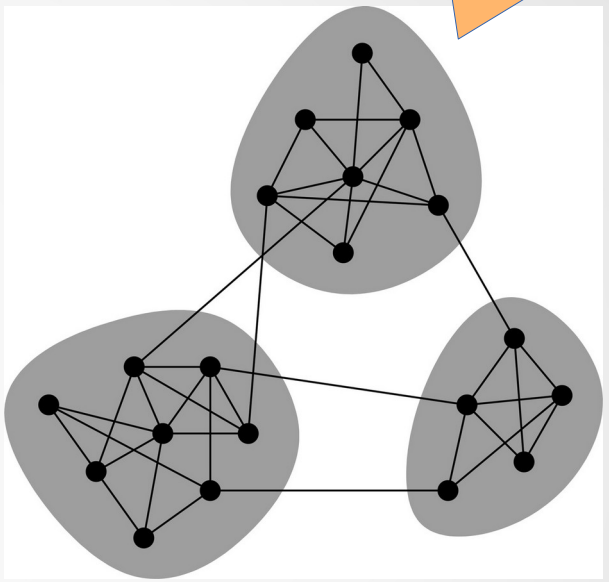
```
nx.degree(G, nbunch=None, weight=None)
# nbunch for a subset of nodes
```

Returns a degree view of single node or of n-bunch of nodes. If n-bunch is omitted, then return degrees of all nodes.

Metric Highlight: Modularity

May help split by geographic region

Description
A *measure of the structure of a network*. It measures how well the network divides into modules with many connections within a module and fewer connections between modules *c.f. grouping*



Application
Can be used to determine whether connections between services are *equally* distributed throughout the system or *irregularly*. ***Useful preliminary indicator of whether clustering and community detection is appropriate*** (in large networks clustering can be 'NP hard problems' – modularity is a less computationally expensive and useful pre-cursor indicator)

Function

```
# undirected graph
nx.modularity_matrix(G, nodelist=None, weight=None)
```

OR

```
# directed graph
nx.directed_modularity_matrix(G, nodelist=None, weight=None)
```

Returns the modularity matrix of G.

Metric Highlight: Density

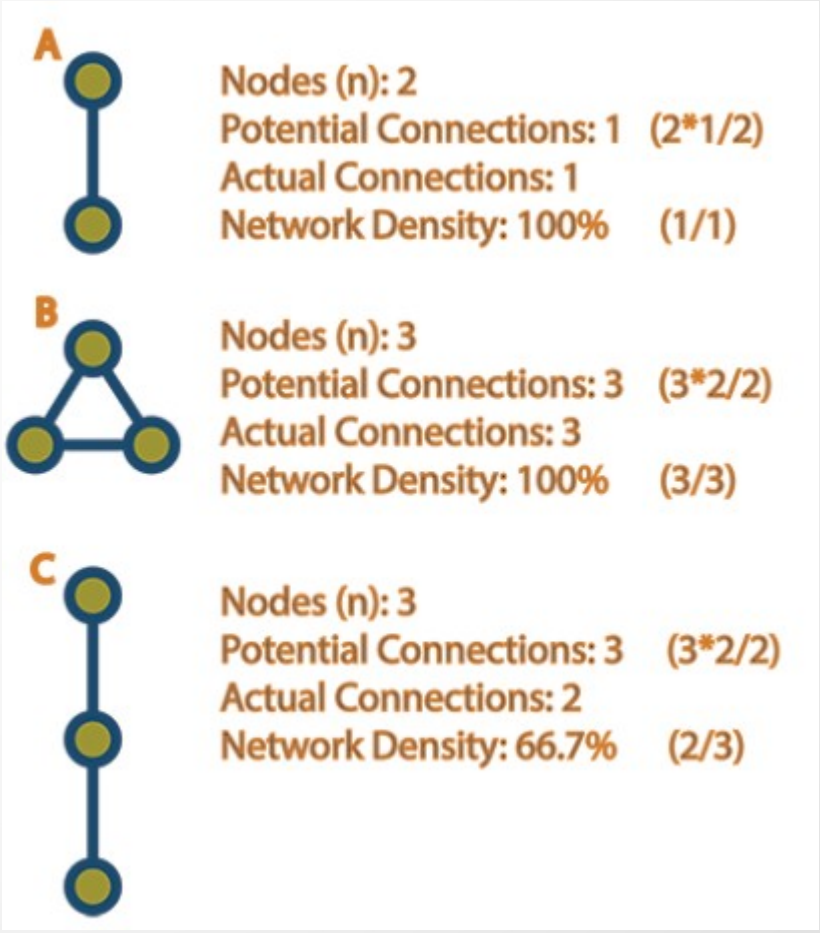
Description
This describes the completeness of the graph in terms of the number of possible connections and the number of actual connections given as network density = actual connections / potential connections. This gives a number between 0 (no edges) and 1 (complete network)

Application
Density provides a measure of how highly connected or sparsely connected your system is. *If the aim is to have every service connected to every other service you would want to see a high density value but if you want distinct pathways with minimal crossover between pathways you would want to see a lower density value*

Function

```
nx.density(G)
```

Returns the density value of a graph



Metric Highlight: Cliques

Community detection strategies – looking for sub-graphs within graphs

Description
Cliques are formed by removing edges around a node in a graph until the subgraph becomes complete. It is one strategy for community identification and clustering

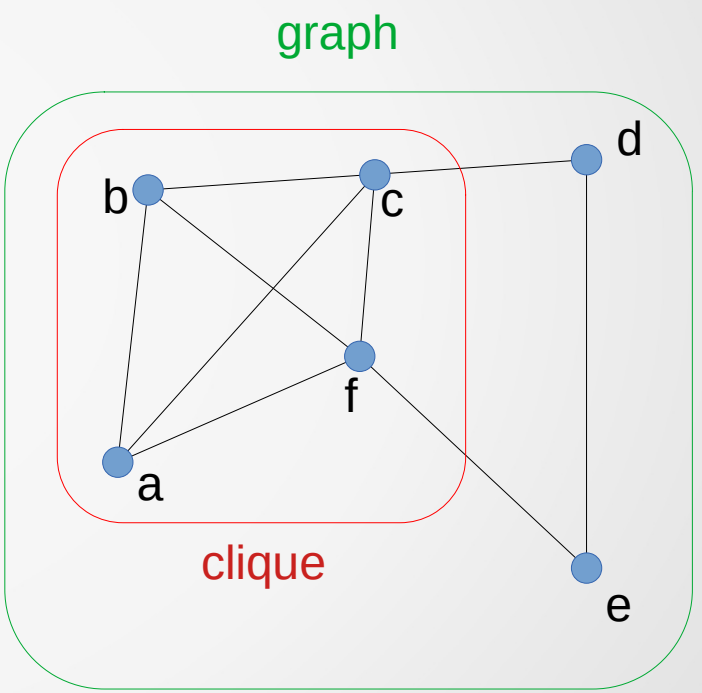
Application
In small to medium size graphs (< 200 nodes) this can be a useful technique to identify clusters of highly linked services - where clustering and community detection might not be appropriate. Computationally expensive as has to consider every node within network.

Function

```
list(nx.find_cliques(G))

# slightly faster
nx.number_of_cliques(G, nodes=None, cliques=None)
```

Returns all maximal cliques in an undirected graph.
Returns the number of maximal cliques for each node (i.e. is it worth looking for cliques in the graph? Perhaps use clustering and modularity instead.)



Where every node is connected together

Metric Highlight: Clustering Coefficient

Description
For each node in a graph the number of actual complete triangle subgraphs is calculated as a fraction of the possible complete triangle subgraphs. i.e., you take a node and then look to find triangles off of a particular node. So that means that you're looking for where a node is connected to two other nodes and those two nodes are also connected to create a triangle. How many of these complete triangles exist and how many could be formed.
This is less computationally expensive than clique finding and community detection so is often used with **large graphs** (>200 nodes) i.e., take a node and look for triangles off that node.

Application
This is used to identify highly connected services that are interfacing with many other services. The coefficient is useful for colouring nodes to visually explore connectivity with a graph

Large graphs tend to be more sparse (i.e. more nodes but fewer connections)

Function

```
nx.clustering(G, nodes=None, weight=None)
```

Returns the clustering coefficient for the nodes in G (ea. Node has it's own value)

Metric Highlight: (Eigenvector) Centrality

Description
Centrality **extends the concept of connectivit**y introduced with degree.
Where degree only accounts for connectivity to a node's direct (nearest) neighbours, other centrality measures take into account connectivity reaching further out into the network. Eigenvector centrality is a common and robust method that calculates the centrality of a node on the basis of its facilitation of connections within the network.

Application
Measures of centrality and particularly eigenvector centrality can be used to determine the importance of a node in facilitating connections throughout a network. Those services with a high eigenvector centrality are those that facilitate the provision of care within a system.
Useful in health services when looking at health systems as you get to see who is the facilitator of care i.e., in mental health we see liaison psychiatry, crisis resolution treatment teams – are core caregivers – and refer between services within the system.

Function

```
nx.eigenvector_centrality(G, max_iter=100...)
```

Degree of nodes beyond the nearest neighbours

Returns the eigenvector centrality for all nodes in the graph: approximate approach – hence increasing iterations increases accuracy.

Metric Highlight: Isolates

Description
When a node in a graph is not connected to any other node by an edge it is called an isolate. Do they need to be investigated, or simply removed?

Application
Where services are not interacting with the rest of the system they will be isolated in the graph. These functions provide a quick way to identify isolated services with a system

Function

```
nx.isolates(G)
nx.number_of_isolates(G)
```

Returns the isolated nodes in the graph G
Returns the number of isolated nodes in G

Metric Highlight: Link Analysis

Description
PageRank was developed to rank webpages based on the structure of incoming links to a vertex (node). It is ***best used with a weighted directed graph***.
The quality of incoming links determines the PageRank of a vertex.

Application
PageRank is an *alternative to other centrality measures* such as eigenvector centrality. It has the advantage that the PageRank value attributed to each vertex (node) is also a probability of an edge being traversed within the network. This can be used to build a probabilistic network model i.e., what is the chance that someone in Service A is going to be referred to Service B, or Service C, or Service D. *Best used with weighted, directed graphs.*

Function

```
nx.pagerank(G, alpha=0.85, personalisation=None,
   max_iter=100, tol=1e-06...)
```

Returns the PageRank of the nodes in the graph.

Tolerance and alpha values – amount of change that is tolerable between iterations before it stops i.e. how robust and persistent values are

Metric Highlight: Shortest Paths

Description
Path finding is a common task in network analysis as it enables you to know how distant one node is from another.

Application
Shortest path algorithms are useful for understanding how services are separated by other intermediary services and for determining common care pathways. This technique could be used to identify patient pathways and to design more efficient pathways.

Function

```
nx.shortest_path(G, source=None, target=None, weight=None,
    method='dijkstra')
nx.has_path(G, source, target)
```

Different methods available – i.e. random walk is computationally expensive as trying at random

Returns the shortest paths in the graph for all nodes if source and/or target are not specified
Returns True if there is a path from the source to the target

Metric Highlight: Similarity Measures

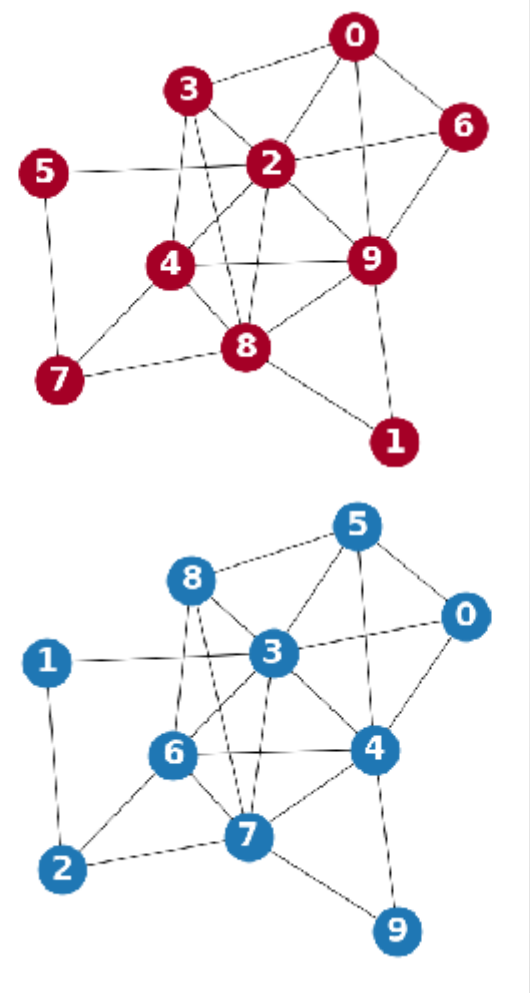
Description
If you want to directly compare two graphs, similarity measures can be used. This is technically described as the number of edge/node changes required to make two graphs isomorphic (i.e. the same). This is an NP-hard problem, so computationally expensive on larger graphs.

Application
When comparing changes to a system over time, similarity measures provide a single function approach to determining what has changed in a graph from one time step to another.

Function

```
nx.graph_edit_distance(G1, G2...)
nx.optimal_edit_paths(G1, G2...)
nx.optimize_edit_paths(G1, G2...)
```

Returns a distance value i.e. how different the 2 graphs are
Returns all minimum-cost edit paths to transform G1 to G2
Returns all node and edge edits required to transform G1 to G2



Task – Analysis Task

- Open the ***analysis_task.ipynb*** file in the ***exercise*** folder.
- You are provided with node and edge data character associations for Game of Thrones seasons 1 and 2. Also see https://networkofthrones.wordpress.com
- Adapt the code to perform the analysis on both seasons 1 and 2
- Compare the various metrics for both seasons.
- Be prepared to come back and tell us something interesting you found from the analysis.

Further Info & Acknowledgement

Slides adapted from Sean Manzi

Checkout https://www.project-nom.com for more information and training on the use of network-based operational modelling for whole system modelling in healthcare