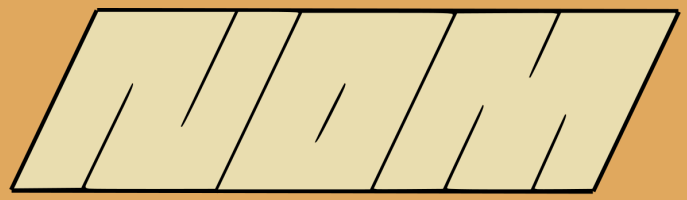
This project is funded and supported by:

THIS.Institute NIHR | Applied Research Collaboration South West Peninsula UNIVERSITY OF EXETER



Network-based Operational Modelling

Advanced network analysis session 1
Graph Metrics

Dr Sean Manzi - University of Exeter, UK

Health Service Modelling Associates Programme 2020

Session structure

- 0930 – 1000 Graph theory and why graphs are useful representations of data
- 1000 – 1030 Types of graph
- 1030 – 1045 Graph metrics
- 1045 – 1130 Comfort break
- 1130 – 1230 Implementing and interpreting graph metrics

Learning objectives

- Understand how graphs are used to represent data
- Know how to conduct an analysis of a network
- Be able to interpret common graph metrics

Graph theory

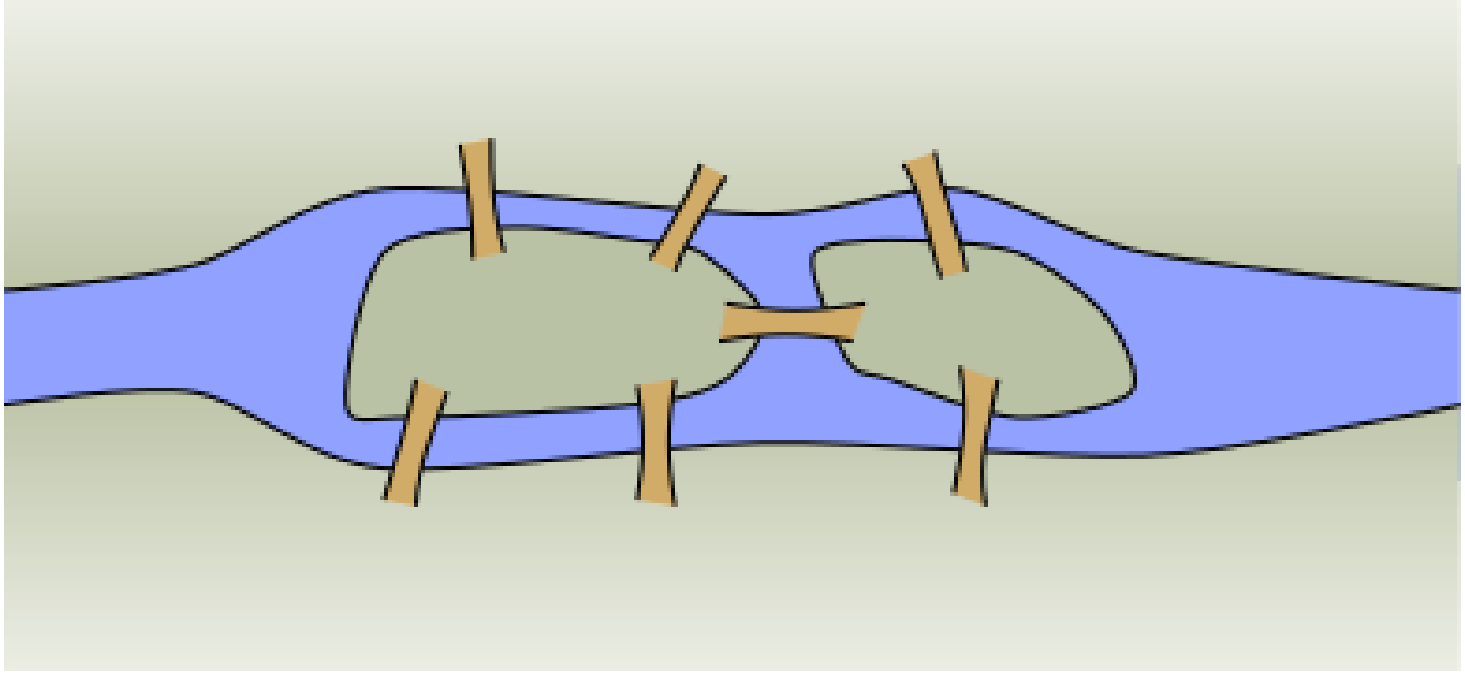
What is graph theory?

- A branch of discrete mathematics
- First studied by Leonhard Euler in 1735
- Used as a way to abstract complex information

Let's look at the problem that Euler first studied

Graph theory

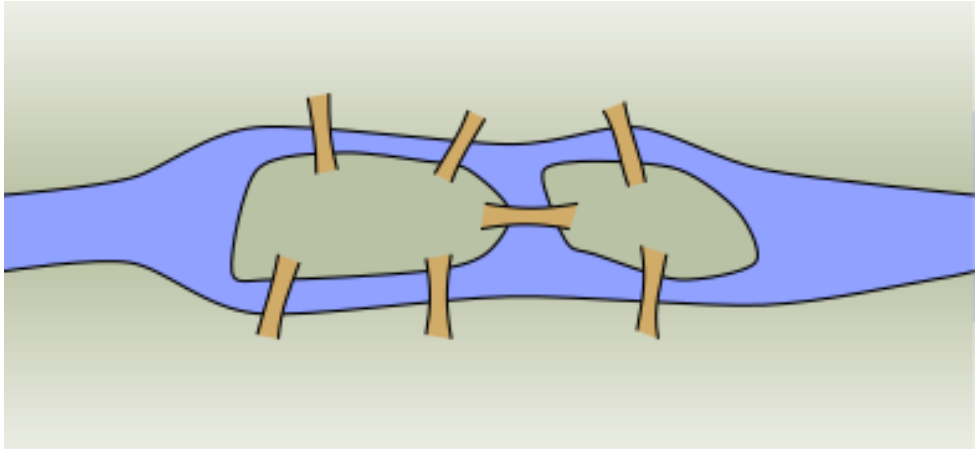
The Seven bridges of Konigsburg



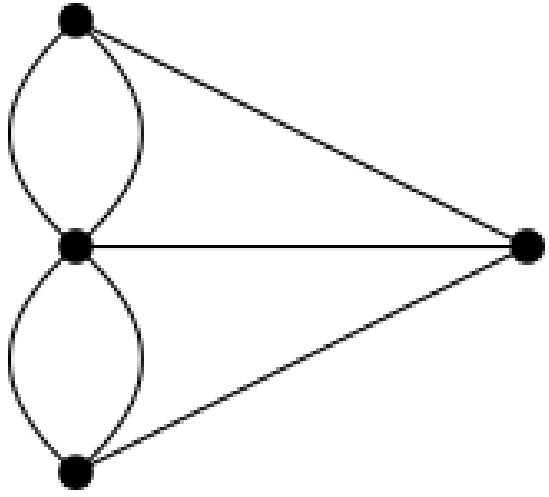
Can you plan a walk where you cross each bridge only once?

Graph theory

Literal rendering



Euler's graph representation



Graph theory

A graph is described by a pair of sets (V,E)

Vertices (nodes)

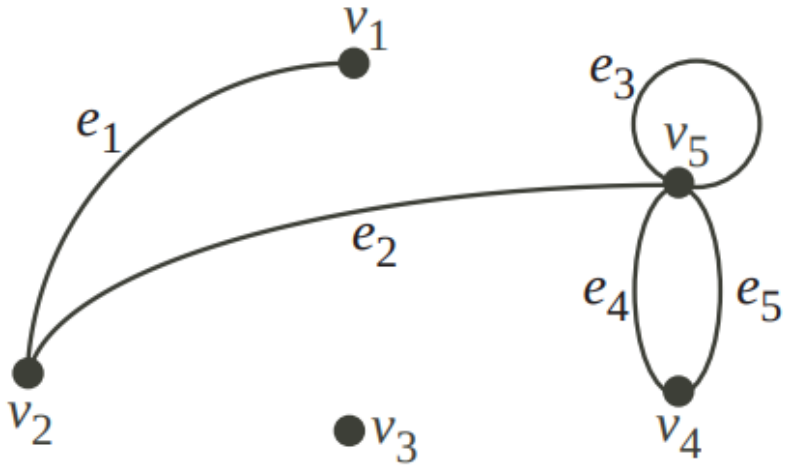
$V = \{v_1, v_2, ...v_n\}$

edges

$E = \{(v_1, v_3), (v_2, v_3), ...\}$

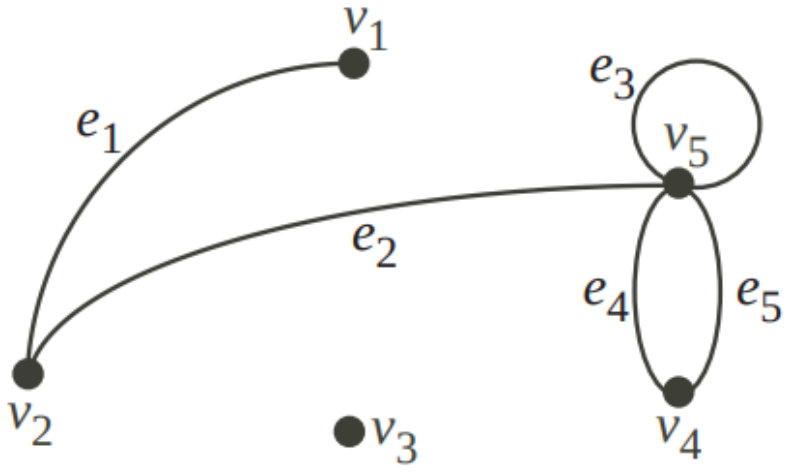
or

$E = \{e_1, e_2, ...e_n\}$



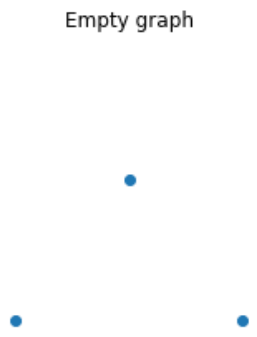
Graph theory

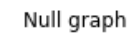
- Edges that have the same end vertices are parallel
- An edge (v,v) is a loop or self-loop
- Edges are adjacent is they share a common vertex
- Two vertices are adjacent if they are connected by an edge

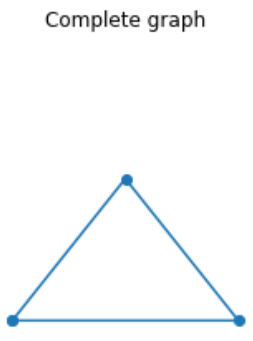


Simple graph types

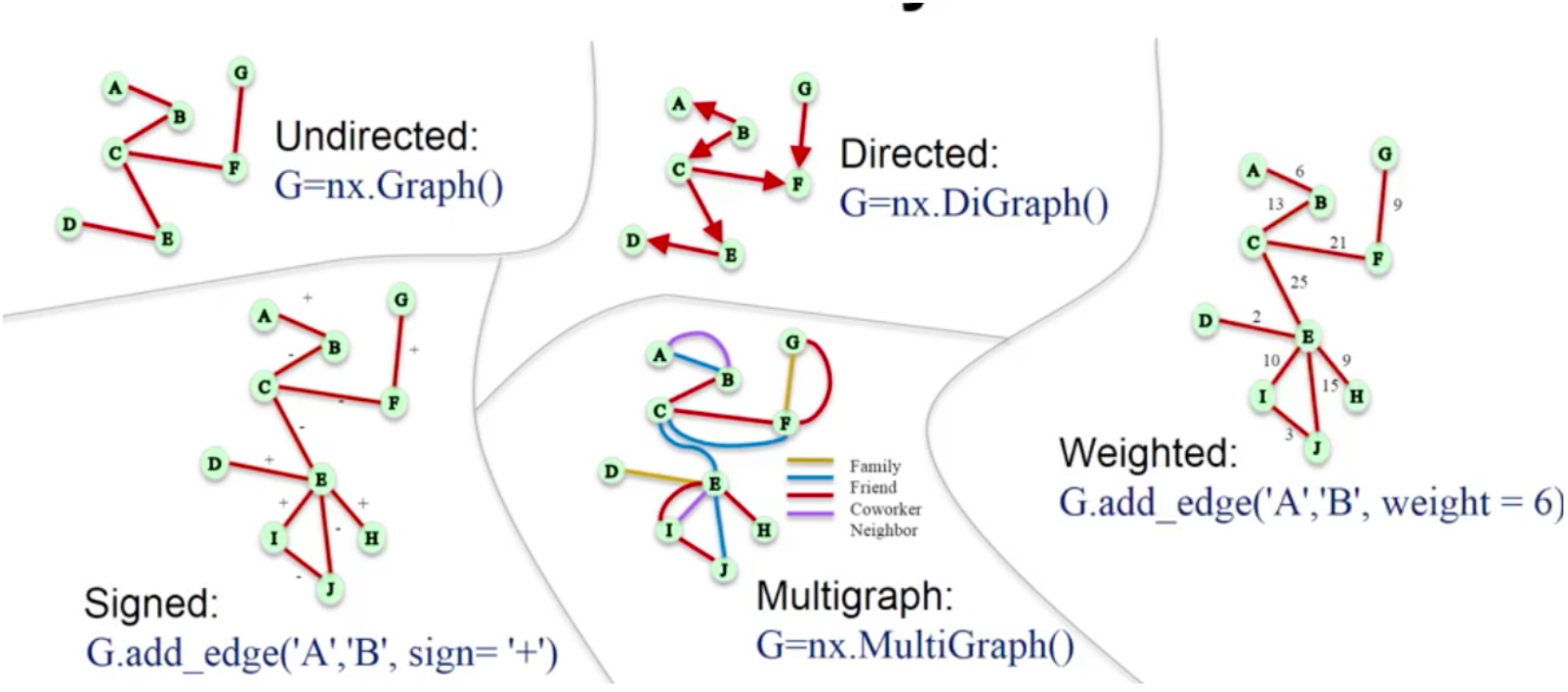
- Empty Graph: no edges
- Null graph: no vertices
- Trivial graph: one vertex
- Complete graph: every pair of vertices is adjacent







Graph types


Undirected:
G=nx.Graph()
Directed:
G=nx.DiGraph()
Weighted:
G.add_edge('A','B', weight = 6)
Signed:
G.add_edge('A','B', sign= '+')
Multigraph:
G=nx.MultiGraph()
Family
Friend
Coworker
Neighbor

Common graph metrics

Network level

- Number of nodes and edges: Base description of network size and complexity
- Density: Number of links in relation to the number of possible links
- Average degree: Average number of edges intercepting a node
- Network modularity: Connectedness of the graph

Node level

- Degree (indegree/outdegree): Number of edges intercepting a node
- Modularity: How nodes group together based connectedness
- Eigenvector centrality: Connectedness of nodes in relation to the whole network

Useful graph metrics

- Approximations and heuristics
  - Connectivity
  - Clique
  - Clustering
- Centrality
  - Degree centrality
  - Eigenvector centrality
  - Katz centrality
- Cliques
- Clustering
- Communities
  - k-clique communities
  - greedy modularity communities
- Isolates
- Link analysis
- Shortest paths
- Similarity measures

NetworkX documentation

The documentation for NetworkX and all of the various graph metrics can be found here:
https://networkx.org/documentation/stable//reference/algorithms/index.html

Degree

Description

Degree refers to the number of edges intersecting a vertex. In an undirected graph each vertex 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.

Application

The degree is an indicator of system complexity at the nearest neighbour level. In operational modelling we can understand this as the number of interfaces a service has with other services.

Function

```
nx.degree(G, nbunch=None, weight=None)
```

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

Modularity

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

Application

Can be used to determine whether connections between services are equally distributed throughout the system or irregularly. Useful indicator of whether clustering and community detection is appropriate

Function

```
nx.modularity_matrix(G, nodelist=None, weight=None)
nx.directed_modularity_matrix(G, nodelist=None, weight=None)
```

Returns the modularity matrix of G.

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 / potiential 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

Cliques

Description

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

Application

In small to medium size graphs this can be a useful technique to identify clusters of highly linked services where clustering and community detection might not be appropriate.

Function

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

Clustering coefficient

Description

For each node in a graph the number of complete triangle subgraphs is calculated as a fraction of the possible complete triangle subgraphs. This is less computationally expensive than clique finding and community detection so is often used with large graphs ($>$ 200 nodes)

Application

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

Function

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

Returns the clustering coefficent for the nodes in G

Centrality

Description

Centrality extends the concept of connectivity introduced with degree. Where degree only accounts for connectivity to a node's direct 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 eignvector centrality are those that faciliate the provision of care within a system.

Function

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

Returns the eigenvector centrality for all nodes in the graph

Isolates

Description

When a node in a graph is not connected to any other node by an edge it is called an isolate.

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

Link analysis

Description

PageRank was developed to rank webpages based on the structure of incoming links to a vertex. 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 is also a probability of an edge being traversed within the network. This can be used to build a probabilistic network model.

Function

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

Returns the PageRank of the nodes in the graph.

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)
```

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

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

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

Returns a distance value

Returns all minimum-cost edit paths transforming G1 to G2

Returns all node edits, edge edits and cost for transforming G1 to G2

Analysing a network with NetworkX

Task time

- Open the file analysis_task.py in Spyder
- Code: data import, graph builder function, analysis code
- There are node and edge data describing character associations for Game of Thrones seasons 1 and 2
- 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

Advanced network analysis

In the next session we will cover:

- How graph visualisations work
- Graph algorithms and custom layouts
- Interactive graphs with Plotly and Holoviews

Thank you

Thank you for paying attention
Hope you enjoyed the session
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