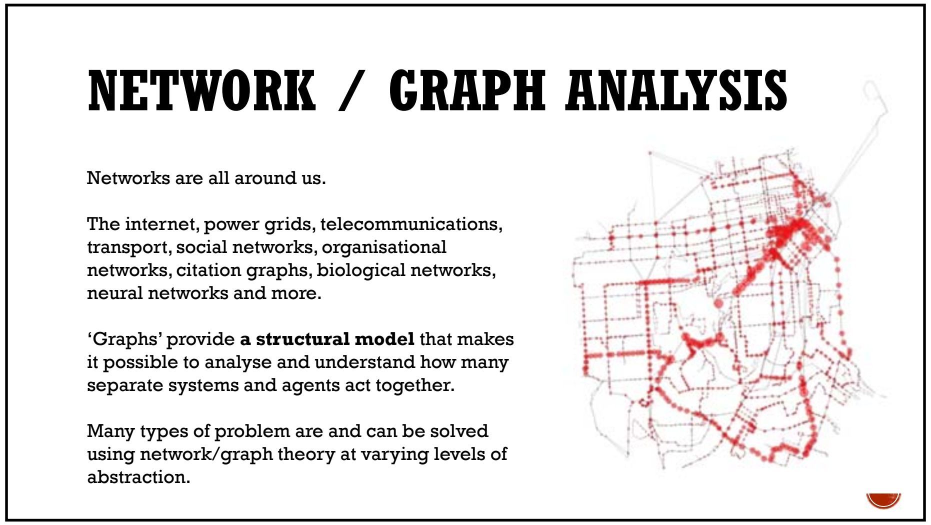
Network Analysis & NetworkX

Oliver Lock | Geocomputation | 2018



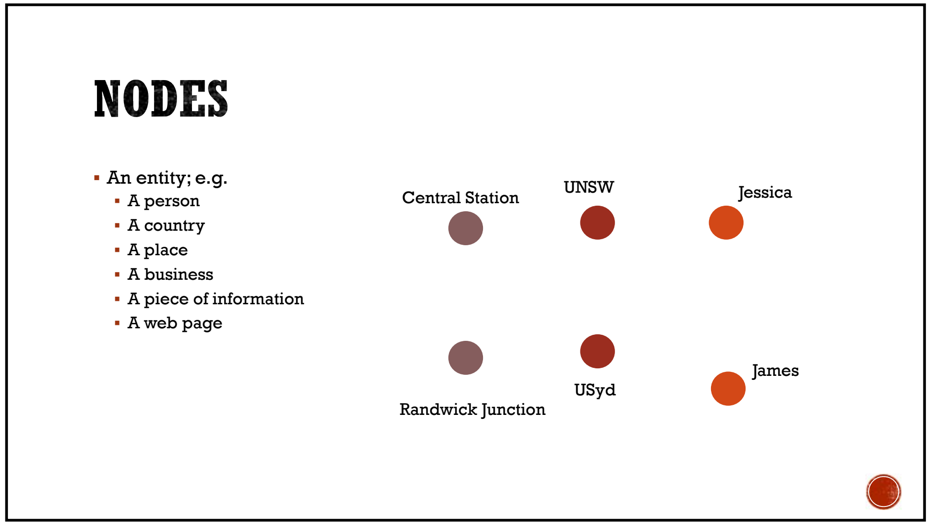
NETWORK / GRAPH ANALYSIS

Networks are all around us.

The internet, power grids, telecommunications, transport, social networks, organisational networks, citation graphs, biological networks, neural networks and more.

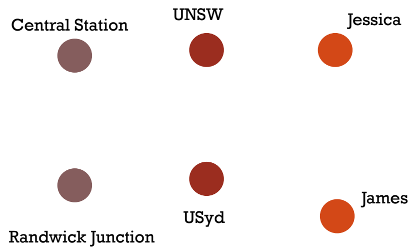
'Graphs' provide **a structural model** that makes it possible to analyse and understand how many separate systems and agents act together.

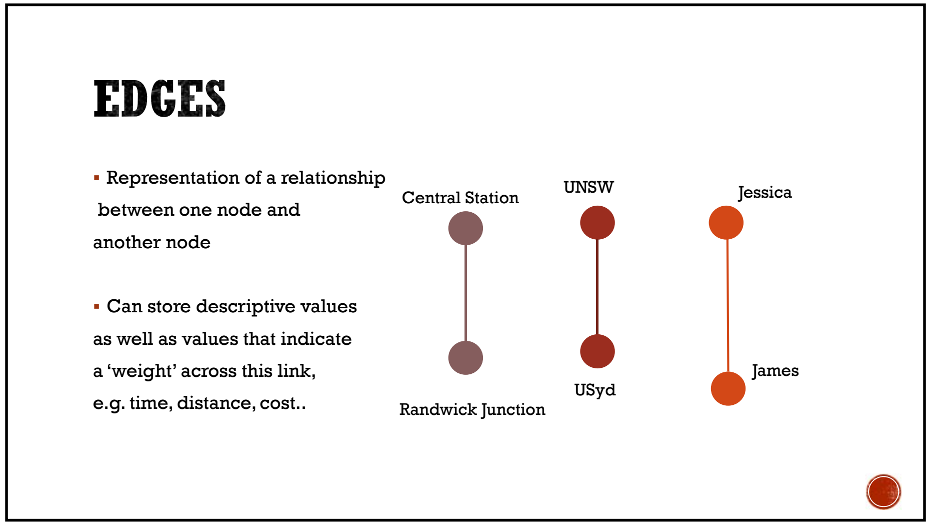
Many types of problem are and can be solved using network/graph theory at varying levels of abstraction.



NODES

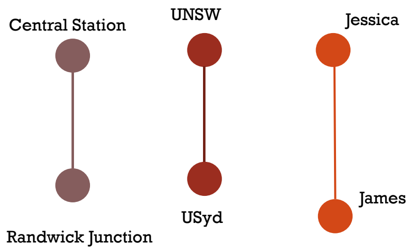
- An entity; e.g.
  - A person
  - A country
  - A place
  - A business
  - A piece of information
  - A web page





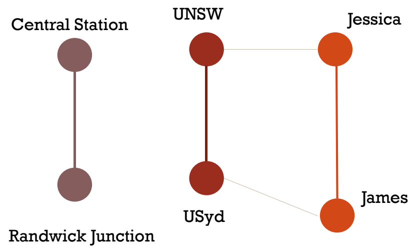
EDGES

- Representation of a relationship between one node and another node
- Can store descriptive values as well as values that indicate a 'weight' across this link, e.g. time, distance, cost..

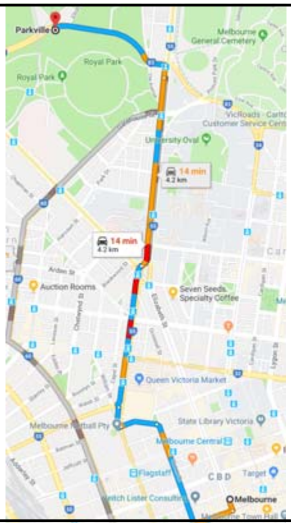


EDGES

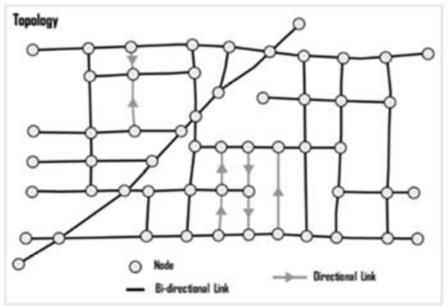
- Representation of a relationship between one node and another node
- Can store descriptive values as well as values that indicate a 'weight' across this link, e.g. time, distance, cost..



IDENTIFY NODES AND EDGES:



Typical features of a transport network for modelling



Network Data Models - Dr. Jean-Paul Rodrigue
https://transportgeography.org/?page_id=7585

Transport networks separated into systems of node: and links which may contain properties such as:

Nodes:
Identifier
Demographic / employment information (usually called a 'Centroid' in this case)
Transportation service information (e.g. whether it is a stop)
Penalties on travel time (such as intersections)

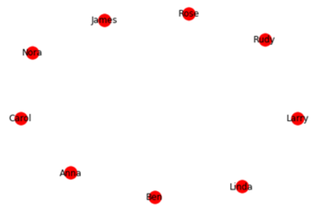
Links:
Identifier
Direction
Length
Capacity
Travel time
Type (road, pedestrian, rail route, bicycle lane)
Incline
Frequency of PT services attached to it

NETWORKX

CREATING A NODE

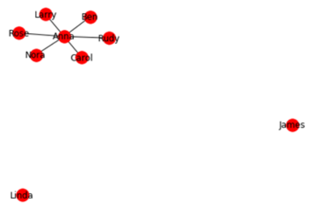
```
import networkx as nx

# Create a networkX graph under variable 'G'
G = nx.Graph()


import matplotlib.pyplot as plt
%matplotlib inline


 G.add_node('Anna')
 nx.draw(G,with_labels=True)
```

CREATING MANY NODES



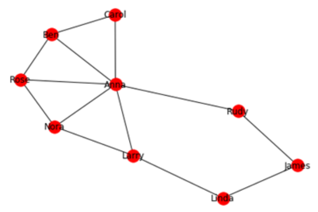
```
Friends =
['Ben','Carol','Rose','Nora','Larry','Rudy
','James','Linda']

G.add_nodes_from(Friends)
nx.draw(G,with_labels=True)
```

BUILDING FRIENDSHIPS (EDGES)



```
G.add_edge('Anna','Carol')
G.add_edge('Anna','Ben')
G.add_edge('Anna','Rose')
G.add_edge('Anna','Nora')
G.add_edge('Anna','Larry')
G.add_edge('Anna','Rudy')
nx.draw(G,with_labels=True)
```

BUILDING FRIENDSHIPS (EDGES)



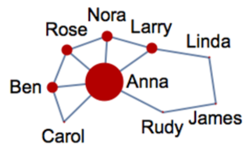
```
G.add_edge('Larry','Linda')
G.add_edge('Linda','James')
G.add_edge('James','Rudy')
G.add_edge('Larry','Nora')
G.add_edge('Nora','Rose')
G.add_edge('Rose','Ben')
G.add_edge('Ben','Carol')
nx.draw(G,with_labels=True)
```

ANALYSING NETWORKS 101 - CENTRALITY

- Once our networks are too big, messy, complex to understand mathematical measures have been developed.
- Measure of the importance of node (or edge) in a network
- Applications can vary from
  - Social – the most influential people in a network
  - Transport / Infrastructure – key infrastructure points, places that need to be more resilient or monitored closely
  - Internet – ranking content
  - Health – epidemiology and the outbreak of disease
  - Crime – to find criminal networks, to keep track of the spread of information
  - Remember to focus on the relative values

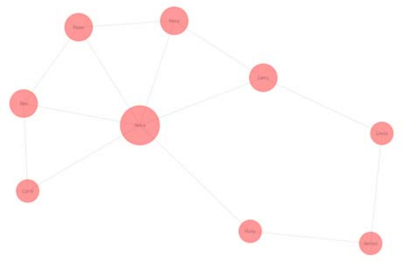
DEGREE CENTRALITY

influence on other nodes in their immediate neighbourhood



The node degree is the number of edges adjacent to that node

DEGREE CENTRALITY IN NETWORKX

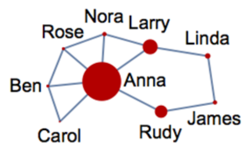


```
degree_dict =
dict(G.degree(G.nodes()))
degree_dict
```

```
{'Anna': 6,
 'Ben': 3,
 'Carol': 2,
 'James': 2,
 'Larry': 3,
 'Linda': 2,
 'Nora': 3,
 'Rose': 3,
 'Rudy': 2}
```

BETWEENNESS CENTRALITY

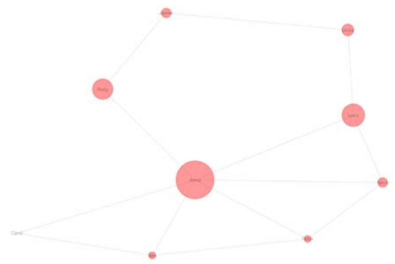
nodes that are crucial components structurally for information flow



The extent to which a node lies on the shortest paths between other nodes.

BETWEENESS CENTRALITY IN NETWORKX

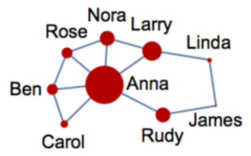
betweenness_dict =
nx.betweenness_centrality(G)



```
{'Anna': 0.5535714285714285,
 'Ben': 0.017857142857142856,
 'Carol': 0.0,
 'James': 0.03571428571428571,
 'Larry': 0.19642857142857142,
 'Linda': 0.05357142857142857,
 'Nora': 0.03571428571428571,
 'Rose': 0.017857142857142856,
 'Rudy': 0.1607142857142857}
```

CLOSENESS CENTRALITY

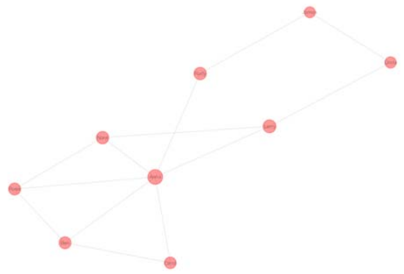
nodes in the network that are crucial for the quick spread of information



Normalised reciprocal of the sum of the shortest path distances to all other nodes. The higher this value is – the closer 'on average' to all other nodes.

CLOSENESS CENTRALITY IN NETWORKX

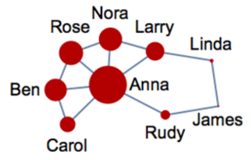
closeness_dict =
nx.closeness_centrality(G)



```
{'Anna': 0.8,
 'Ben': 0.5333333333333333,
 'Carol': 0.5,
 'James': 0.4444444444444444,
 'Larry': 0.6153846153846154,
 'Linda': 0.47058823529411764,
 'Nora': 0.5714285714285714,
 'Rose': 0.5333333333333333,
 'Rudy': 0.5714285714285714}
```

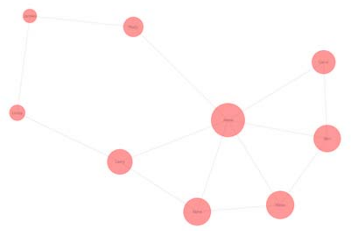
EIGENVECTOR CENTRALITY

nodes in the network that are connected to many other well-connected nodes



Upgraded version of degree centrality – where you have a higher 'value' if your neighbour has a higher 'value'. Similar to PageRank.

EIGENVECTOR CENTRALITY IN NETWORKX



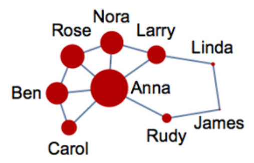
```
eigenvector_dict =
nx.eigenvector_centrality(G)
```

```
{'Anna': 0.5719492867084267, 'Ben':
0.3675125293405861, 'Carol':
0.27825878129984094, 'James':
0.09468281347772223, 'Larry':
0.3179402349249892, 'Linda':
0.1222157674286938, 'Nora':
0.3792663995194539, 'Rose':
0.3905937370646623, 'Rudy':
0.19745028144306703}
```

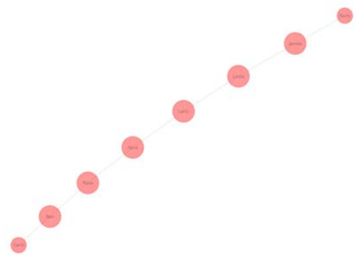
WHAT HAPPENS IF ANNA MOVES COUNTRY?

```
G.remove_node('Anna')
```

Sketch what would happen…



WHAT HAPPENS IF ANNA MOVES COUNTRY?



```
G.remove_node('Anna')
```

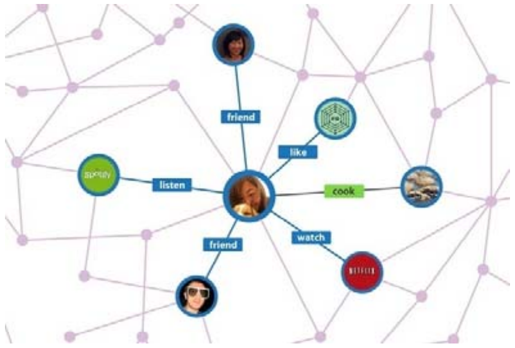
What if *Anna* was a different type of node?

- **Train station**
- **Airport**
- **Communication network**
- **Media mogul**
- **Business**
- **Trade agreement**

SOME EXAMPLES…

SOCIAL MEDIA

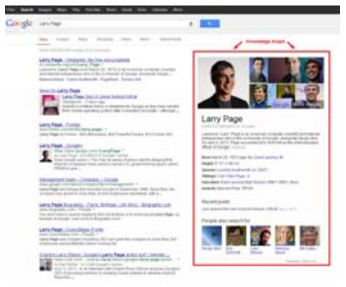
GraphQL

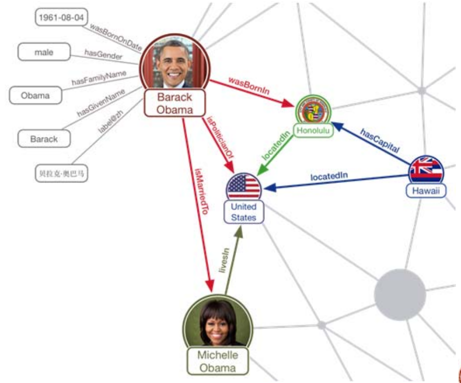


https://github.com/facebook/graphql

KNOWLEDGE / INFORMATION / WWW

E.g. Google Knowledge Graph



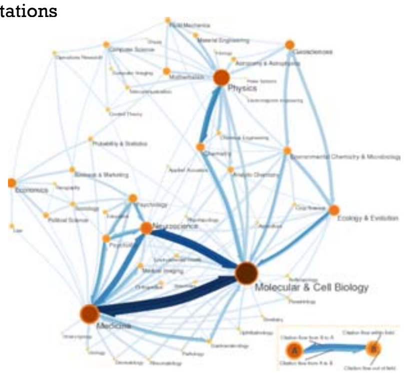


KNOWLEDGE / INFORMATION / ORGANISATIONAL

E.g. Citations

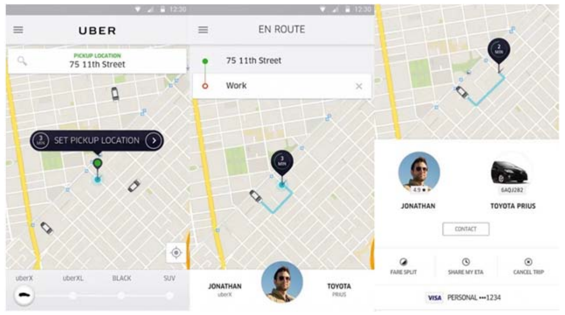
E.g. Expertise

URBAN FUTURES
AUSTRALIAN URBAN
Urban Transport
sustainable urban
urban design
urban
urban planning
URBAN DEVELOPMENT
Managing urban

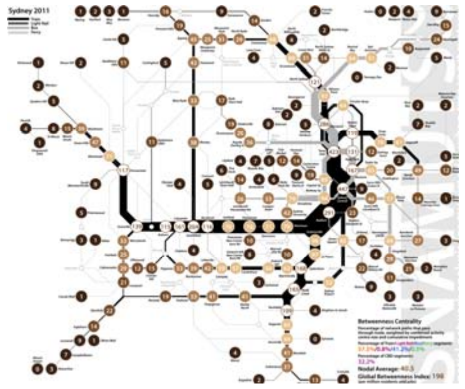


TRANSPORT

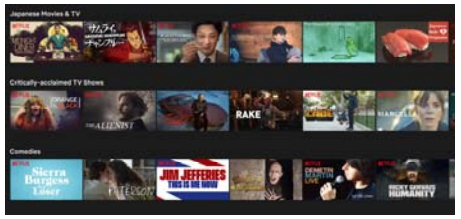
TRANSPORT (UBER)



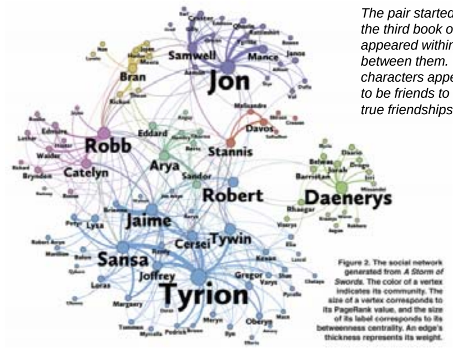
TRANSPORT (SNAMUTS)



ALMOST EVERY RECOMMENDATION SYSTEM..



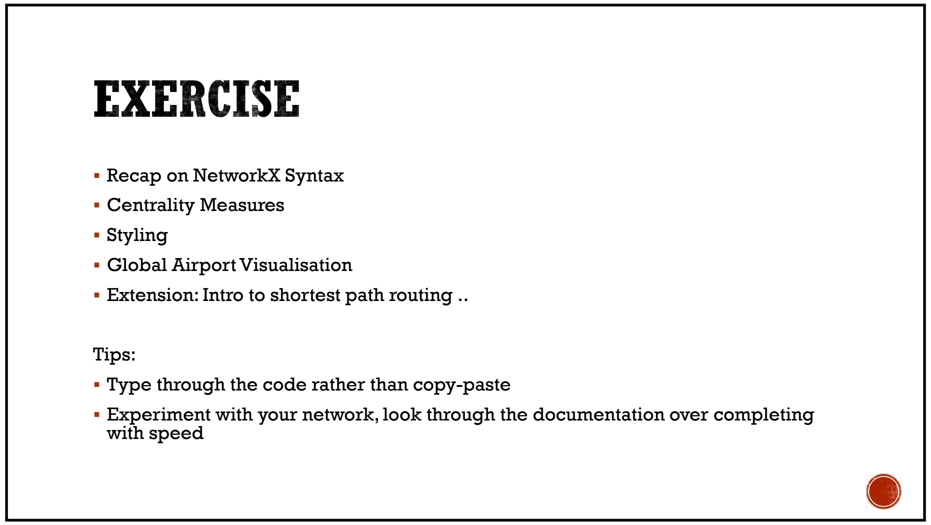
Customers Who Bought This Item Also Bought
Predictive Analytics For Dummies
Predictive Analytics: The Power to Predict Who...
Quantifying the User Experience: Practical...
Marketing Analytics: Strategic Models and...
Data Driven Marketing For Dummies



The pair started by connecting characters every time they "interacted" in the third book of the series, A Storm of Swords. Whenever two characters appeared within 15 words of one another, a link (or "edge") was added between them. The links are weighted based on how often the two characters appeared in close proximity. Characters don't necessarily have to be friends to be linked—which is a good thing because there are few true friendships in the series.

Figure 2. The social network generated from *A Storm of Swords*. The color of a vertex indicates its community. The size of a vertex corresponds to its PageRank value, and the size of its label corresponds to its betweenness centrality. An edge's thickness represents its weight.

https://qz.com/650796/mathematicians-mapped-out-every-game-of-thrones-relationship-to-find-the-main-character/



EXERCISE

- Recap on NetworkX Syntax
- Centrality Measures
- Styling
- Global Airport Visualisation
- Extension: Intro to shortest path routing ..

Tips:

- Type through the code rather than copy-paste
- Experiment with your network, look through the documentation over completing with speed

Visualising Network Concepts

Master of City Analytics | Geocomputation | Questions? Oliver Lock (o.lock@unsw.edu.au | city-informatics.com)

The tutorial should equip with skills to apply basic network analysis concepts and visualise them (creating the above image of the world's airport network and connections). You will use open data, Python and the NetworkX library.

What is network analysis?

Network analysis concerns itself with the formulation and solution of problems that have a network structure; such structure is usually captured in a 'graph'. Graph theory provides a set of abstract concepts and methods for the analysis of graphs. Graph theory approaches can be applied to a diverse amount of fields of knowledge and practice.

What is NetworkX?

NetworkX is a Python Package that provides tools for the study of the structure and dynamics of social, biological, and infrastructure networks; a standard programming interface and graph implementation that is suitable for many applications; a rapid development environment for collaborative, multidisciplinary projects; an interface to existing numerical algorithms and code written in C, C++, and FORTRAN; and the ability to painlessly work with large nonstandard data sets.

With NetworkX you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more.

Why are we using NetworkX:

- Free
- Pythonic, integrated with libraries you should be familiar with such as pandas, NumPy, Matplotlib
- Many years of development (born in 2002), continuous development
- The way the data is structured can support in excess of 10 million nodes and 100 million edges
- Highly analytically focused, visualisation medium performance
- Can import AND export many different data formats - you can do your analysis in NetworkX and ship it somewhere else

Some other platforms are:

- Gephi - https://gephi.org/
- GraphViz - http://graphviz.org

The documentation

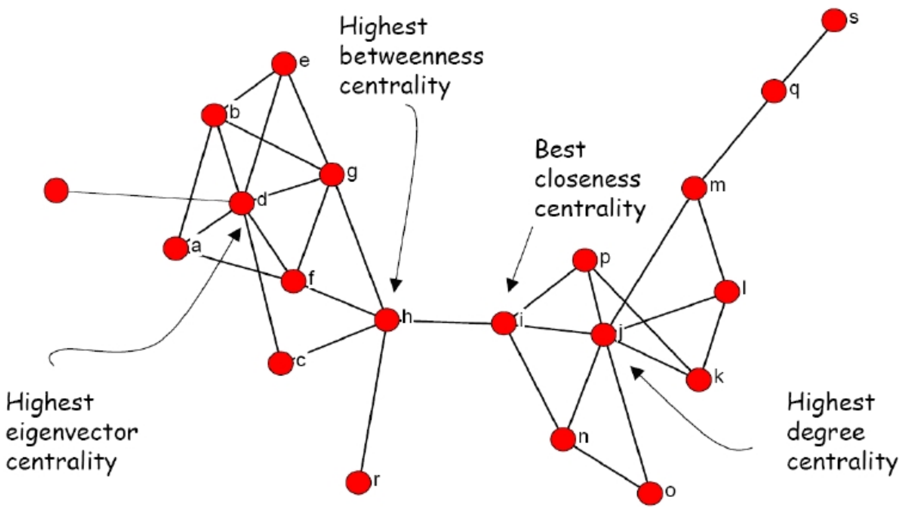
https://networkx.github.io/documentation/stable/reference/introduction.html (https://networkx.github.io/documentation/stable/reference/introduction.html)

This tutorial - Centrality measures

NetworkX doco: https://networkx.github.io/documentation/stable/reference/algorithms/centrality.html (https://networkx.github.io/documentation/stable/reference/algorithms/centrality.html)

One of the most important things that you'll want to do when you're performing a network analysis is determining the centrality of a node within a social network. Centrality description answers the question: "Which node is the most important one in the network?".

Centrality measures assign a numerical value to each vertex of a network according to its influence on the others. Depending on the type of the network, you can describe what importance means and what centrality measures make logical sense to use. It could be identified as an effective person in a social network or key infrastructure nodes in the urban networks. Four of the most well-known measures are degree centrality, betweenness centrality, closeness centrality and eigenvector centrality.


Highest betweenness centrality
Best closeness centrality
Highest eigenvector centrality
Highest degree centrality
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s

Node Degree

The node degree is the number of edges adjacent to that node. A node can have an 'in' degree and 'out' degree which is different depending on whether edges specify a direction to that node. If there is no direction, these numbers are identical. This is the most basic measure of connectedness. It is useful for assessing which nodes are 'central' with respect to spreading information and influencing others immediately adjacent.

E.g. in a network of music collaborations - how many people has this person collaborated with?

NetworkX doco:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.degree_centrality.html#networkx.
(https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.degree_centrality.html#network

Betweeness Centrality

Calculates the number of shortest paths that between other nodes that pass through this node, as a proportion of all shortest paths. Shows which nodes are more likely to be in communication paths between other nodes. Also useful in determining points where the network would break apart (weak points).

In a network of spies: who is the spy through whom most of the confidential information is likely to flow?

NetworkX doco:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.betweenness_centrality.html#net
(https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.betweenness_centrality.html#ne

Closeness centrality

Calculate the mean length of all shortest paths from a node to all other nodes in the network (i.e. how many hops on average does it take to reach every other node). Take the reciprocal of the above so that higher values are 'better'. It is a measure of reach, i.e. the spread which information can reach other nodes from a given starting node.

In network of people - how fast will a disease spread from this person to the rest of the network?

NetworkX doco:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.closeness_centrality.html#networ
(https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.closeness_centrality.html#netwo

Eigenvector Centrality

A node's eigenvector centrality is proportional to the sum of the eigenvector centralities of all nodes directly connected to it. In other words, a node with a high eigenvector centrality is connected to other nodes with high eigenvector centrality. This is similar to how Google ranks web pages: links from highly linked pages count more. This is useful in determining who is connected to the most connected nodes.

In a network of paper citations - who is the author that is most cited by other well-cited authors?

NetworkX doco:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.eigenvector_centrality.html#netw
(https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.eigenvector_centrality.html#netw

Some examples of networks & network analysis

http://www.snamuts.com/ (http://www.snamuts.com/)
http://atrf.info/papers/2012/2012_Curtis_Scheurer.pdf (http://atrf.info/papers/2012/2012_Curtis_Scheurer.pdf)
http://visjs.org/examples/network/exampleApplications/lesMiserables.html
(http://visjs.org/examples/network/exampleApplications/lesMiserables.html)
https://www.fastcompany.com/3068474/the-real-difference-between-google-and-apple (https://www.fastcompany.com/3068474/the-real-difference-between-google-and-apple)
http://www.martingrandjean.ch/connected-world-air-traffic-network/ (http://www.martingrandjean.ch/connected-world-air-traffic-network/)
http://vax.herokuapp.com/ (http://vax.herokuapp.com/)
http://ekisto.sq.ro/ (http://ekisto.sq.ro/)

Exercise 1 - Fundamental components

Getting set up

In [ ]:
```
## Access networkx libraries
import networkx as nx
import numpy as np

## Visualisation library
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
# Create a networkX graph under variable 'G'
G = nx.Graph()
```

In [ ]:
```
##Adding nodes
G.add_node('UNSW')

#Draw the network!
nx.draw(G,with_labels=True)
```

Adding nodes & edges

In [ ]:
```
#You can add nodes en-masse using something like a list

# For example, this is the 'Group of 8' universities in Australia

Go_Eight = ['UNSW','ANU','Monash','U Adelaide','U Queensland','U Sydney','U Melb','UWA']

G.add_nodes_from(Go_Eight)

## Or you can generate from a range, as below. Or even a Pandas dataframe.
###G.add_nodes_from(range(1,10))

## You can remove nodes / edges
##e.g.  G.remove_node('UNSW')
##e.g.  G.remove_edges_from(..) etc

# Draw the new nodes
nx.draw(G,with_labels=True)
```

In [ ]:
```
# Adding edges

#G.add_edge(Edge1,Edge2)
## Note that this relationship can be one-way, or two-way depending on the set up
## This won't be covered for this exercise, we will assume if a connection exists it exists in both w
ays
## https://networkx.github.io/documentation/networkx-1.10/tutorial/tutorial.html#directed-graphs

#We can use a loop in order to draw an edge between every node and UNSW as below:

for university in Go_Eight:
    G.add_edge(university,'UNSW')

nx.draw(G,with_labels=True)
```

Getting connected

In [ ]:
```
#We can provide a second loop in order to connect everything to everything

for university in Go_Eight:
    for university_2 in Go_Eight:
        G.add_edge(university,university_2,label='Go8',weight='1')

nx.draw(G,with_labels=True)
```

In [ ]:
```
# Now let's think of some other 'networks' that exist in university sphere - for example the Russel G
roup in the UK
# the PLuS alliance UNSW shares with UK and US uni and and an arbitrary group in the US

Russell_Group = ['University of Birmingham',
'University of Bristol',
'University of Cambridge',
'Cardiff University',
'Durham University',
'University of Edinburgh',
'University of Exeter',
'University of Glasgow',
'Imperial College London',
'Kings College London',
'University of Leeds',
'University of Liverpool',
'London School of Economics & Political Science',
'University of Manchester',
'Newcastle University',
'University of Nottingham',
'University of Oxford',
'Queen Mary, University of London'
'Queens University Belfast',
'University of Sheffield',
'University of Southampton',
'University College London',
'University of Warwick',
'University of York']

PLUS_Alliance = ['UNSW','Kings College London','Arizona State University']

US_Group = ['Arizona State University','Ohio State University','Penn State']


## Use some loops to join evertyhing to each other

for university in Russell_Group:
    for university_2 in Russell_Group:
        G.add_edge(university,university_2,label='Russell Group',weight='1')


for university in PLUS_Alliance:
    for university_2 in PLUS_Alliance:
        G.add_edge(university,university_2,label='PLUS',weight='1')

for university in US_Group:
    for university_2 in US_Group:
        G.add_edge(university,university_2,label='US_Group',weight='1')


nx.draw(G,with_labels=True)

## Now we have a good example of a network with some characteristics useful for displaying our centra
lity metrics
```

In [ ]:
```
##By default NetworkX draws a 'spring layout' ; there are other ways we can draw them too.

# For example we can use below to assign each node a 'position' in circular layout
pos=nx.circular_layout(G)

# Nodes and edges can be drawn separately, and set with this 'pos'.
nx.draw_networkx_nodes(G, pos, node_shape='o', alpha=0.4,node_size=100)

nx.draw_networkx_edges(G, pos, style='solid', alpha=0.1)

#Note that this is now using 'plt' to draw instead of networkx, so we are now using 'matplotlib' to d
raw.
plt.show()

## Have a play with the different shapes, colours and styles

##NetworkX drawing documentation
#https://networkx.github.io/documentation/networkx-2.1/reference/drawing.html
#https://networkx.github.io/documentation/networkx-2.1/reference/generated/networkx.drawing.nx_pylab.
draw_networkx_edges.html
#https://networkx.github.io/documentation/networkx-2.1/reference/generated/networkx.drawing.nx_pylab.
draw_networkx_nodes.html
```

In [ ]:
```
# Here's an example where the layout is randomised
pos=nx.random_layout(G)
nx.draw_networkx_nodes(G, pos, node_shape='o', alpha=0.4,node_size=100)
nx.draw_networkx_edges(G, pos, style='solid', alpha=0.1)
plt.show()
```

In [ ]:
```
# Here is the spring layout again, but visualised through matplotlib
# You can remove the axis for a nice, 'clean' look

#It can be useful to change the figure size, when you save the image they can become very high qualit
y
## This is particularly useful for very large, complex-looking networks

plt.figure(figsize=(5,5))
pos=nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, alpha=0.4,node_size=100)
nx.draw_networkx_edges(G, pos, style='solid', alpha=0.1)
plt.axis('off')
plt.tight_layout()
plt.show()
```

In [ ]:
```
## Printing this can show us some quick info on the network's stats
print(nx.info(G))
```

Exercise 2 - Calculating Centrality measures

Node degree

Before doing this, have look at the plots above and try and figure out which Universities have the highest degree, and the lowest degree centrality. You can try drawing them by hand as well as this may help.

In [ ]:
```
# Calculating Centrality measures is quite simple once your network has been properly set up
# To calculate the node degree is below
#G.degree()
```

In [ ]:
```
# Who has the highest degree? Who has the lowest?
# We can sort the degrees using a lambda expression (advanced)
degree_dict = dict(G.degree(G.nodes()))
sorted_names = sorted(degree_dict, key=lambda x: degree_dict[x])
#for k in sorted_names:
    #print("{} : {}".format(k, degree_dict[k]))
```

In [ ]:
```
# Plotting the degree is a bit more complex, and the degree for each node must be stored in a 'dictio
nary' structure
degree_dict = dict(G.degree(G.nodes()))

# The degree must then be stored in each node
nx.set_node_attributes(G, degree_dict, 'degree')

# Now the node has the attribute 'degree', let's check UNSW
print(G.node['UNSW'])
```

In [ ]:
```
##Node Degree / Centrality
##This is exactly the same as before exce

plt.figure(figsize=(12,8))

pos=nx.spring_layout(G)

nx.draw_networkx_nodes(G, pos,nodelist=degree_dict.keys(), node_size=[v * 20 for v in degree_dict.val
ues()], alpha=0.4)

nx.draw_networkx_edges(G, pos, style='solid', alpha=0.1)

nx.draw_networkx_labels(G, pos, font_size=10, font_color='k', font_family='arial', font_weight='norma
l', alpha=0.3, ax=None)

plt.axis('off')

plt.tight_layout()

plt.show()
```

EXERCISE - other centrality types - once these are done, have a go at:

- Betweenness Centrality

  Remember: Shows which nodes are more likely to be in communication paths between other nodes.

  Before you look - what do you think the highest and lowest universities will be?

  nx.betweenness_centrality(G)

- Closeness Centrality

  Remember: Shows how many hops on average it takes to reach every other node

  Before you look - what do you think the highest and lowest universities will be?

  nx.closeness_centrality(G)

- Eigenvector Centrality

  Remember: Shows who is connected to the most connected nodes

  Before you look - what do you think the highest and lowest universities will be?

  nx.eigenvector_centrality(G)

EXERCISE: - modifying the network - once these are done, have a go at:

- A) new relationship - a group of universities from another continent
- B) removing a node
- C) Replotting the graphs to see the difference
- D) Producing a 'summary' file or dataframe of all the centrality measures for each node
- E) Extension : try using a 'weighted' metric, and add weights on either the node or relationship.

Exercise 3 - Analysis of global airport network

The following exercise uses OpenFlights data to calculate our centrality measures in a real network

The schema of the data can be found here: https://openflights.org/data.html (https://openflights.org/data.html)

We will be using:

- Three letter code of all airports, the 'IATA'
- Latitude / Longitude of all airports
- The number of routes which operate between these airports

In [ ]:
```
# import libaries
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.lines as mlines

## The following code is used to download and clean up the data for the exercise.
###

## I have kept it here for your interest and referencing,
## we will not be covering this - querying, joining etc was covered in Prog Cities content ##

## Define dataframe columns for airports
airport_col = ['ID', 'Name', 'City', 'Country','IATA', 'ICAO', 'Lat', 'Long', 'Alt',
                   'Timezone', 'DST', 'Tz database time zone', 'type', 'source']

## Download the data for airports , convert to dataframe
airport_df = pd.read_csv("https://raw.githubusercontent.com/jpatokal/openflights/master/data/airport
s.dat",
                            names = airport_col, index_col = 0)

## Define dataframe columns for routes
route_cols = ['Airline', 'Airline ID', 'Source Airport', 'Source Airport ID','Dest Airport', 'Dest Ai
rport ID', 'Codeshare', 'Stops', 'equipment']

## Download the data for routes, convert to dataframe
routes_df = pd.read_csv("https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes.da
t", names = route_cols)

#clean up data, change 'object' type to numeric and drops NaNs
routes_df['Source Airport ID'] = pd.to_numeric(routes_df['Source Airport ID'].astype(str), 'coerce')
routes_df['Dest Airport ID'] = pd.to_numeric(routes_df['Dest Airport ID'].astype(str), 'coerce')
routes_df = routes_df.dropna(subset=["Source Airport ID", "Dest Airport ID"])

## Aggregate so each origin-destination pair has a 'sum' of all routes between them
routes_sum =  pd.DataFrame(routes_df.groupby(['Source Airport', 'Dest Airport']).size().reset_index(n
ame='counts'))

## Only consider routes with valid IATA pair between airports
routes_sum = pd.merge(routes_sum,airport_df,left_on="Source Airport",right_on="IATA")
routes_sum = pd.merge(routes_sum,airport_df,left_on="Dest Airport",right_on="IATA")

# Create a variable flights, make sure it is an integer in all instances
routes_sum['flights'] = routes_sum['counts'].astype(int)

###

##Make the graph!
graph = nx.from_pandas_edgelist(routes_sum,'Source Airport','Dest Airport','flights')
graph
```

In [ ]:
```
## We can use the following to look at the data behind the graph
## As you can see it contains an origin, destination and value for 'flights' in the edges
## As you can also see there is no data stored in the nodes

#Run these separately :
#graph.nodes(data=True)
#graph.edges(data=True)
#print(nx.info(graph))
```

In [ ]:
```
## We can plot the graph just like before
## What do you notice about this graph?

plt.figure(figsize=(20,20))
pos=nx.spring_layout(graph)
nx.draw_networkx_nodes(graph, pos, alpha=0.2,node_size=100)
nx.draw_networkx_edges(graph, pos, style='solid', alpha=0.1)
plt.axis('off')
plt.tight_layout()
plt.show()
```

In [ ]:
```
# Let's add the location!
# This is very similar to when we used spring, or circular, or random layout before - but we have to
 make the
# 'pos' list ourselves

## Creating a pos_list based on longitude and latitude
routes_lats = airport_df[['IATA','Long','Lat']]
xy = routes_lats.drop_duplicates()
xy['pos'] = list(zip(xy.Long, xy.Lat))
routes_lats = xy[['IATA','pos']]
routes_lats = routes_lats.set_index('IATA')
routes_lats = routes_lats.to_dict('index')
pos_dict={}

for key, value in routes_lats.items():
    for key2,value2 in value.items():
        pos_dict[key] = np.asarray(value2)


plt.figure(figsize=(18,10))
nx.draw_networkx_nodes(graph, pos_dict, alpha=0.2,node_size=100)
nx.draw_networkx_edges(graph, pos_dict, style='solid', alpha=0.1)
plt.axis('off')
plt.tight_layout()
plt.show()
```

In [ ]:
```
## Adding in node degree, making lines thicker for multiple flights between pairs
degree_dict = dict(graph.degree(graph.nodes()))
nx.set_node_attributes(graph, degree_dict, 'degree')

routes_lats = airport_df[['IATA','Long','Lat']]
xy = routes_lats.drop_duplicates()
xy['pos'] = list(zip(xy.Long, xy.Lat))
routes_lats = xy[['IATA','pos']]
routes_lats = routes_lats.set_index('IATA')
routes_lats = routes_lats.to_dict('index')
pos_dict={}

for key, value in routes_lats.items():
    for key2,value2 in value.items():
        pos_dict[key] = np.asarray(value2)

plt.figure(figsize=(36,20))
flight_weight = nx.get_edge_attributes(graph,'flights')
nx.draw_networkx_nodes(graph, pos_dict, nodelist=degree_dict.keys(), node_size=[v * 10 for v in degre
e_dict.values()], node_shape='o', alpha=0.1)
nx.draw_networkx_edges(graph, pos_dict, style='solid',color='k', alpha=0.025,width=flight_weight.valu
es())
plt.axis('off')
plt.tight_layout()
plt.show()
```

In [ ]:
```
# Let's use the formula from earlier to explore the node degree between these
# What conclusions can we make from this?

#sorted_names = sorted(degree_dict, key=lambda x: degree_dict[x])
#for k in sorted_names:
    #print("{} : {}".format(k, degree_dict[k]))
```

Extension questions

- What are the top five airports for the other measures of centrality?
- Visualise one of the other centrality measures
- Filter the data so it shows and/or calculates metrics only for flight routes coming to or leaving Australia

Extension - Other Network Applications - Shortest Path

There are a large number of different types of analysis performed on networks, beyond centrality. One of the most common in transport is 'shortest path'. This traverses the network between nodes to find the path with least 'weight' on edges to get between A and B. This extension is based on an existing tutorial (http://avinashu.com/tutorial/pythontutorialnew/NetworkXBasics.html (http://avinashu.com/tutorial/pythontutorialnew/NetworkXBasics.html)) which highlights some good ways to do your own shortest path and visualise in NetworkX.

In [ ]:
```
import networkx as nx

# The following line initializes two empty directed graph objects
G1=nx.DiGraph()
G2=nx.DiGraph()

# An empty undirected graph object can be initialized using the command
# G=nx.Graph()
```

In [ ]:
```
G1.add_node(1)
G1.add_node(2)
G1.add_node(3)
G1.add_node(4)
G1.add_node(5)
G1.add_node(6)
G1.nodes()
```

In [ ]:
```
list_nodes = [1, 2, 3, 4, 5, 6]
G2.add_nodes_from(list_nodes)
G2.nodes()
```

In [ ]:
```
G1.add_edge(1, 2, weight = 2.0)
G1.add_edge(1,3, weight = 4.0)
G1.add_edge(2, 3, weight = 1.0)
G1.add_edge(2, 4, weight = 4.0)
G1.add_edge(2, 5, weight = 2.0)
G1.add_edge(3, 5, weight = 3.0)
G1.add_edge(4, 6, weight = 2.0)
G1.add_edge(5, 4, weight = 3.0)
G1.add_edge(5, 6, weight = 2.0)
G1.edges()
```

In [ ]:
```
list_arcs = [(1,2,2.0) , (1,3,4.0) , (2,3,1.0) , (2,4,4.0) , (2,5,2.0) , (3,5,3.0) , (4,6,2.0) , (5,4
,3.0) , (5,6,2.0)]
G2.add_weighted_edges_from(list_arcs)
G2.edges()
```

In [ ]:
```
sp = nx.dijkstra_path(G1,source = 1, target = 6)
print(sp)
```

In [ ]:
```
print(nx.shortest_path(G1,source = 1, target = 6))
```

In [ ]:
```
# First we import the matplotlib python plotting package
import matplotlib.pyplot as plt
# We then set the coordinates of each node
G1.node[1]['pos'] = (0,0)
G1.node[2]['pos'] = (2,2)
G1.node[3]['pos'] = (2,-2)
G1.node[4]['pos'] = (5,2)
G1.node[5]['pos'] = (5,-2)
G1.node[6]['pos'] = (7,0)
# The positions of each node are stored in a dictionary
node_pos=nx.get_node_attributes(G1,'pos')
# The edge weights of each arcs are stored in a dictionary
arc_weight=nx.get_edge_attributes(G1,'weight')
# Create a list of arcs in the shortest path using the zip command and store it in red edges
red_edges = list(zip(sp,sp[1:]))
# If the node is in the shortest path, set it to red, else set it to white color
node_col = ['white' if not node in sp else 'red' for node in G1.nodes()]
# If the edge is in the shortest path set it to red, else set it to white color
edge_col = ['black' if not edge in red_edges else 'red' for edge in G1.edges()]
# Draw the nodes
nx.draw_networkx(G1, node_pos,node_color= node_col, node_size=450)
# Draw the node labels
# nx.draw_networkx_labels(G1, node_pos,node_color= node_col)
# Draw the edges
nx.draw_networkx_edges(G1, node_pos,edge_color= edge_col)
# Draw the edge labels
nx.draw_networkx_edge_labels(G1, node_pos,edge_color= edge_col, edge_labels=arc_weight)
# Remove the axis
plt.axis('off')
# Show the plot
plt.show()
```