The Internet[work]

DSTA

1 Summary of Trade Networks

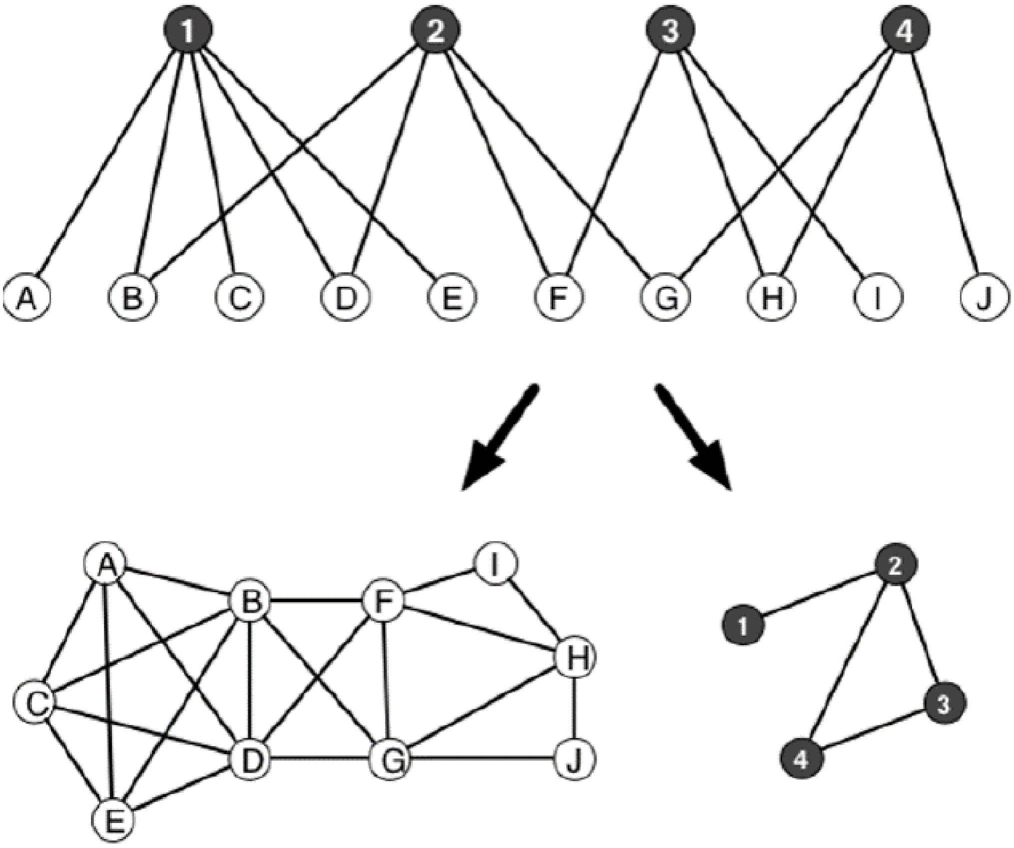
1.1 The directed network model

Theme: discover non-trivial relationships among countries

look at how they trade and what they trade

1.2 Bipartite networks

The country-to-product network induces country-to-country and product-to-product relationships.



1.3 Reconstruction

$$C = M_{cp} \cdot M_{cp}^T$$

$$P = M_{cp}^T \cdot M_{cp}$$

1.4 Analysis of neighbours

For a node i, let k_i be its degree.

For directed networks: $k_i = k_i^{in} + k_i^{out}$.

The distribution of degree $P(k)$ provides a signature of the network.

The average degree is denoted $\langle k \rangle$.

1.5 Reciprocity

For a given directed network, reciprocity is the probability that of having links in both directions between two vertices.

R measures how the economies of two countries become interconnected (or interdependent).

$$r = \frac{L^{\leftrightarrow}}{L}$$

$L^{\leftrightarrow}$: number of reciprocal links

L: total number of links.

1.6 Assortativity

Do vertices tend to connect with those with similar/dissimilar degree?

Compute the avg. degree of i's neighbors:

$$K_{nn}(i) = \frac{\sum_{\langle ij \rangle} k_j}{k_i}$$

Find the avg. K_{nn} of nodes which have degree d:

$$K_{nn}(d) = \frac{\sum_{i:k_i=d} K_{nn}(i)}{n_d}$$

where n_d is the number of nodes having degree d.

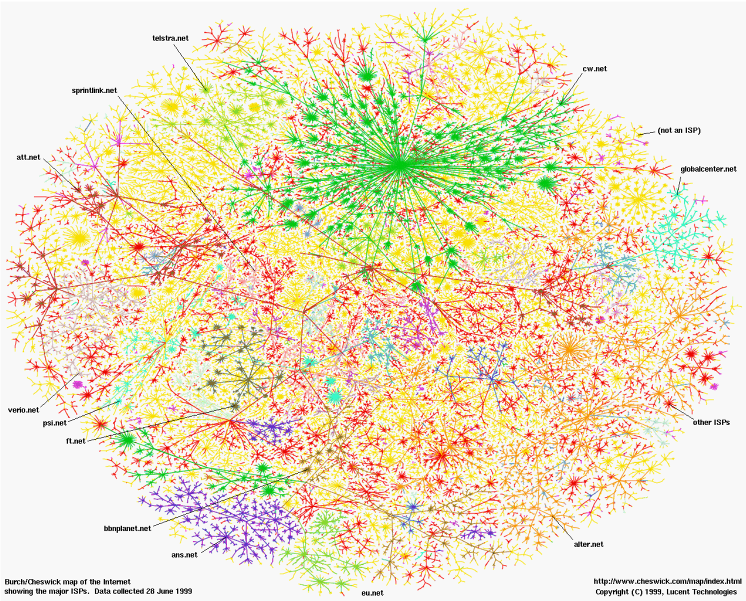
. . .

- Are d and $K_{nn}(d)$ close?
- does assortativity grow over time?

2 The Internet Network

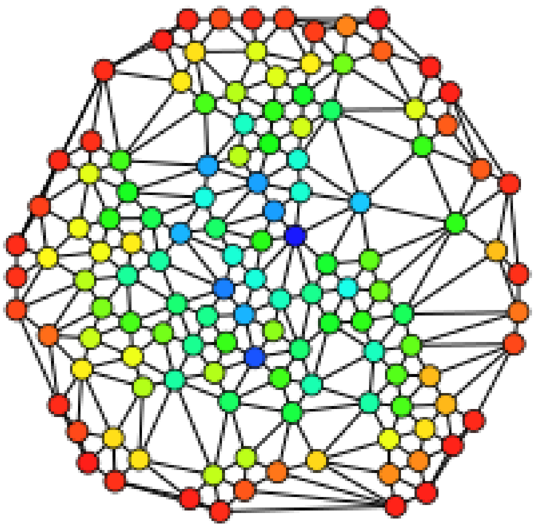
2.1 The need for resolution

The Internet Service Provider network:



2.2 From visualisation back to data

Thanks to the Beautifulsoup project, images of networks in `.svg` format can be imported into a Networkx structure.



```
from bs4 import BeautifulSoup

FILE = 'data/svg-example.svg'

op = open(FILE, 'r')

xml = op.read()

soup = BeautifulSoup(xml)
```

```
G = nx.Graph()

attrs = { "line" : ["x1", "y1", "x2", "y2"] }

# what lines are there?
for attr in attrs.keys():
    tmps =  soup.findAll(attr)
```

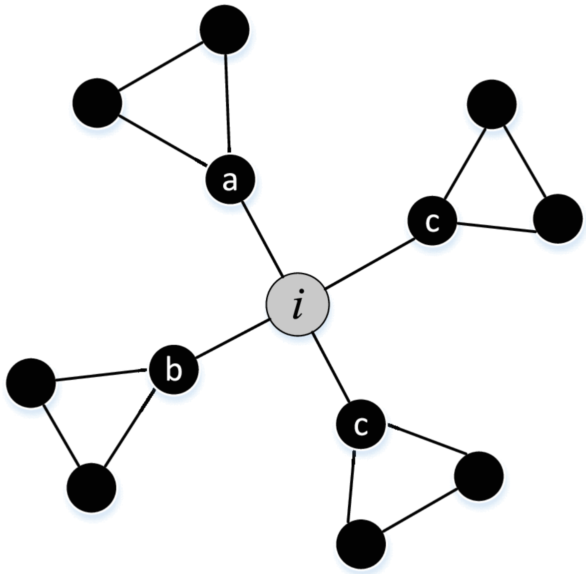
Details in Ch. 3 of the textbook.

3 Node Centrality

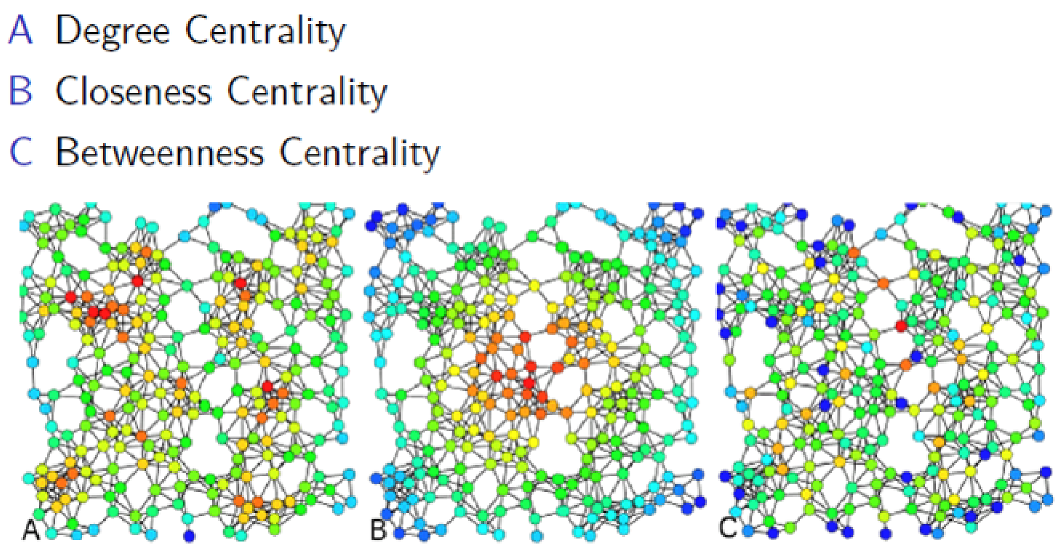
3.1 Find important nodes

Centrality is about importance, of a vertex or edge, within the whole network.

The topology of the network should reflect such importance, so we do not need to **inspect** the entities.



3.2 Comparing centralities



3.3 Degree centrality

High degree leads to higher centrality

3.4 Closeness centrality

Being in close reach to anywhere.

Let d_{ij} be the distance between i and j on the graph.

. . .

$$c_j = \frac{1}{\sum_{j \neq i} d_{ij}}$$

3.5 Harmonic centrality

Immunised against isolated vertices/disconnection

$$c_j^h = \sum_{j \neq i} \frac{1}{d_{ij}} = \sum_{d_{ij} < \infty, j \neq i} \frac{1}{d_{ij}}$$

3.6 Betweenness centrality

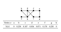
Being in the middle/facilitating all contacts/conversations

Let D_{jl} be the number of distinct paths that exist between node j and node l.

Let also $D_{jl}(i)$ be the number of those paths that go via i

. . .

$$b(i) = \sum_{\substack{j,l=1..n, \\ i \neq j \neq l}} \frac{D_{jl}(i)}{D_{jl}}$$

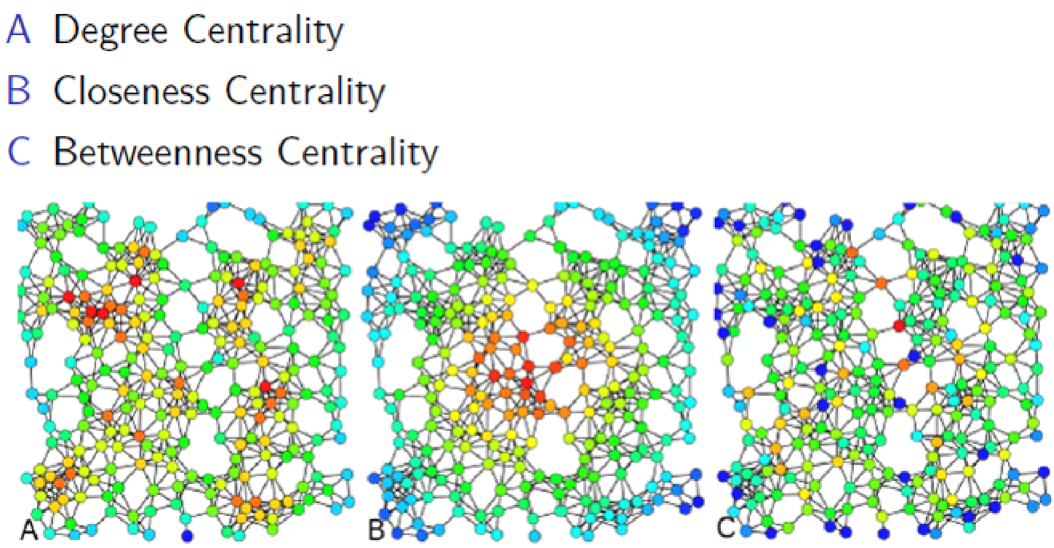


1 shortest path in 4 (or 25%) goes through b, the same with g.

[Brandes 2001] computes $b(i)$ in $O(|V| \cdot |E|)$: too slow to be practical even on small networks.

Estimates based on sampling are used instead.

Good estimates are valuable when the network evolves in a fully-dynamic way: edges and vertices are arbitrarily inserted/removed over time.



4 Eigenvector centrality

4.1 A reflective definition

my c. is the average of my neighbors c.'s,

which in turn depends on my own centrality.

. . .

The dominant e-vector $\mathbf{v_1}$ describes the direction of maximum shape-preserving expansion

$$A\mathbf{v_1} = \lambda_1 \mathbf{v_1}$$

$$A\mathbf{v_1} = \lambda_1 \mathbf{v_1}$$

. . .

$$\mathbf{v_1} = \frac{1}{\lambda_1} A\mathbf{v_1}$$

. . .

For each vertex i:

$$v_{1_i} = \frac{1}{\lambda_1} \sum_j a_{ij} \cdot v_{1_j}$$

which is the needed centrality measure.

4.2 Computing Eigenvector centrality

1. Compute the dominant Eigenpair $(\lambda_1, \mathbf{v_1})$ of A;
2. sort vertices according to the v_{1_i} value they *"scored."*