

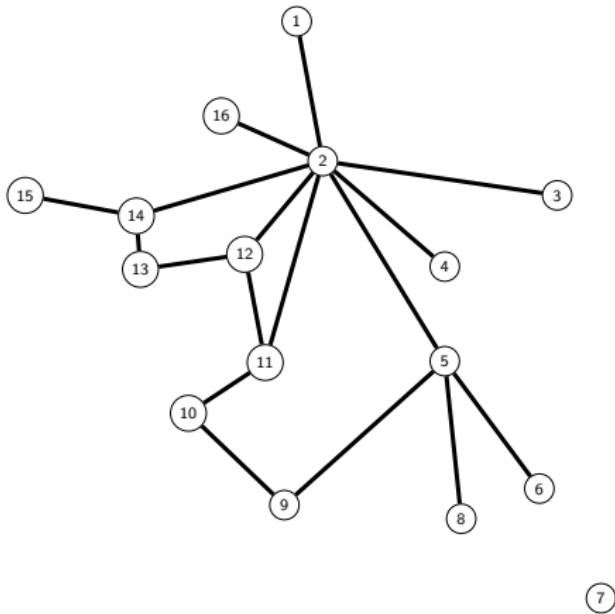
# Chapitre 3: métriques de graphes et coût de calculs

Sophie Achard

LJK, CNRS, Univ. Grenoble Alpes

M1 MIASHS

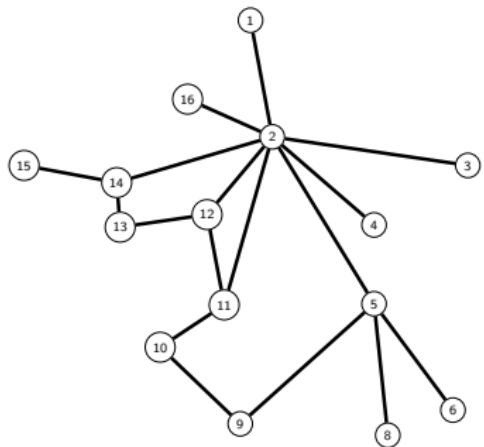
# Encodage d'un graphe : liste d'ajacence



**Figure:** Example of graph with  $N=16$  and  $M = 17$ ,  $V = \{1, \dots, 16\}$ .

$$E = \{[1, 2], [2, 3], [2, 4], [2, 5], [2, 11], [2, 12], [2, 14], [2, 16], [5, 6], [5, 8], [5, 9], [9, 10], [10, 11], [11, 12], [12, 13], [13, 14], [14, 15]\}.$$

# Encodage d'un graphe : matrice d'adjacence

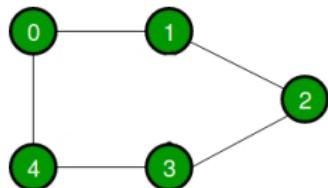


0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	1	1	0	1	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

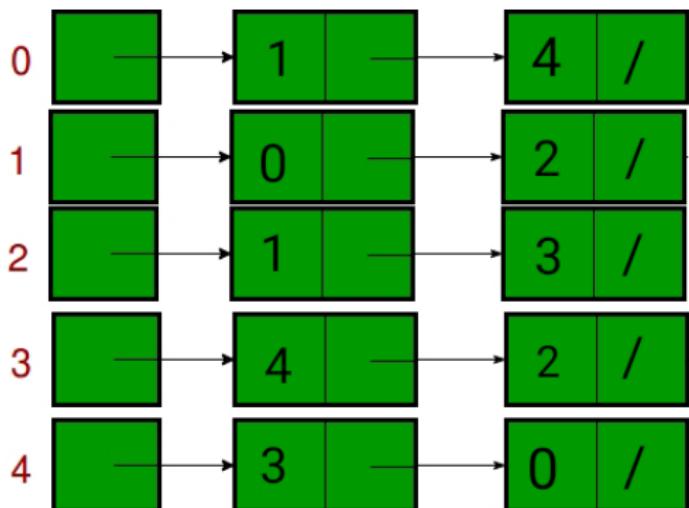
(7)

Figure: Example of graph with  $N=16$  and  $M = 17$ ,  $V = \{1, \dots, 16\}$ .

# Encodage simple d'un graphe



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	0
2	0	1	0	1	0
3	0	0	1	0	1
4	1	0	0	1	0



# Couts de calculs

Que se passe-t-il quand on veut ajouter, enlever ou trouver une arête dans un graphe ?

Operation	Adjacency matrix	Adjacency list
Storage space	$O(N^2)$	$O(N + M)$
Insert edge	$O(1)$	$O(1)$
Delete edge	$O(1)$	$O(M/N)$
Find edge	$O(1)$	$O(M/N)$
Enumerate edge	$O(N)$	$O(M/N)$

**Table:** Worst case of time complexity for four operations on a graph with  $N$  vertices and  $M$  edges. Insert means add an edge in the graph. Delete means remove an edge in the graph. Find corresponds to the test that a given pair of vertices are connected with an edge. Enumerate is the list of the neighbours of a given vertex.

# Distribution des degrés

On suppose que le degré est stocké à part.

Pour construire la distribution on doit compter le nombre de noeuds avec degré  $k$ .

On a besoin d'un tableau de  $N$  cases. Pour chaque noeud du graphe, on incrémente la case de 1 correspondant au degré du noeud.

Pour chaque  $k$ ,  $p_k = \text{nombre dans la case}/N$ .

Le coût total est donc de  $O(N)$ .

# Plus court chemin

Rappel chemin, pas unique

longueur du chemin

Rappel avec la matrice d'adjacence:

Soit  $A$  la matrice d'adjacence d'un graphe non dirigé  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ . Soit  $r$  un entier. Le coefficient  $a_{ij}^{(r)}$  de la matrice  $A^r$  est égal au nombre de chemin de longueur  $r$  qui existe entre  $i$  et  $j$ .

Ceci revient à faire des calculs de matrices et donc le coût est  $O(N^3)$ .

# Une alternative : algorithme de Breadth-first search

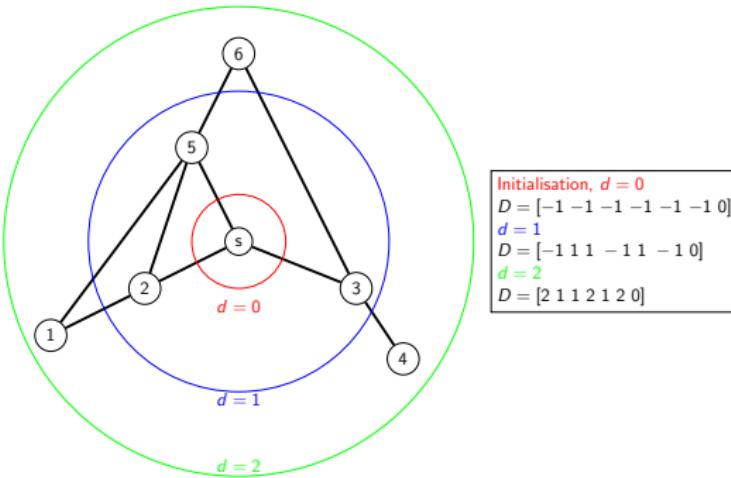


Figure: Illustration of the breadth first search algorithm.

## Une alternative : algorithme de Breadth-first search

Une première implémentation naïve:

For  $s$  in  $1, \dots, N$  Initialisation

Create an array on  $N$  elements,  $D$

Initialise it to  $D[s] = 0$ , and for all  $t$ ,  $t \neq s$ ,  $D[t] = -1$

Initialise  $d$  to 0.

While  $d < N$  do

Step 1 Use the distance array  $D$ , Find all vertices that correspond to a distand  $d$  from  $s$ .

Step 2 For each vertices found with distance  $d$ , check its neighbours and store them, if there are unknown, i.e. with a value of -1 in  $D$ .

Step 3 Test if the number of neighbours is 0, the algoritm stops. If not, set the distance of the neighbours to  $d + 1$

Step 4 Increase the value of  $d$  by 1

end while

## Justification de l'algorithme

- ① Tout noeud dont la distance à  $s$  est  $d$  a un voisinage dont la distance minimale à  $s$  est  $d - 1$ .
- ② Supposons que l'on connaisse tous les noeuds de distance  $d$  ou moins à  $s$ . Pour tous voisins des noeuds à distance  $d$  de  $s$ , il existe un chemin de longueur  $d + 1$  de ce voisin à  $s$ . Donc chacun de ses voisins est au plus à une distance  $d + 1$  de  $s$ . Mais il est possible que ce soit plus petit que  $d + 1$ .

Conclusion : On a bien trouvé tous les noeuds à distance  $d + 1$  de  $s$  et donc on connaît tous les noeuds à distance  $d + 1$  et moins. On s'arrête quand tous les noeuds du graphe sont parcourus.

On trouve aussi par cet algo les composantes connexes du graphe.

# Coût de calcul

The computation cost is decomposed in 3 steps:

- ① Update the distance matrix  $D$ :  $O(N)$ .
- ② For each iteration, we are looking for all the nodes at distance  $d$ :  $O(N)$ . If there are  $r$  iterations, the total cost is  $O(rN)$ .
- ③ We have to explore all the neighbours of one node:  $O(M/N)$ . This for all the nodes. So the cost is  $O(M)$ .

Finally the total cost is  $O(N + rN + M)$ . In the case where the worst case for iteration is  $N$ , this means the cost is  $O(M + N^2)$ .

Ce coût est très pessimiste, dans la majorité des cas on a  $O(M + N \log N)$ .

Cet algo ne trouve pas le plus court chemin, l'algo utilisé s'appelle Djikstra.

# Clustering

We recall here the definitions of clustering as introduced previously.  
The global clustering coefficient is defined as:

$$C = \frac{3 \times \#\text{triangles}}{\#\text{connected triplets}}. \quad (1)$$

Let  $\mathcal{N}_i$  denote the neighbourhood of node  $i$ . We have:  $d_i = |\mathcal{N}_i|$ . The local clustering coefficient associated to node  $i$  is defined as:

$$C_i = \frac{2 \times \#\text{edges connecting nodes in } \mathcal{N}_i}{d_i(d_i - 1)}. \quad (2)$$

## Proposition

*The number of triangles of a graph with adjacency matrix  $A$  is equal to*

$$\text{Trace}(A^3)/6$$

## Clustering : algorithme

The algorithm consists in scanning all the nodes of the graph and extracting the properties of the induced graph attached to each node.

$$C_i = \frac{2 \times \#\text{edges connecting nodes in } \mathcal{N}_i}{d_i(d_i - 1)}.$$

For one node  $i$ ,

For the numerator,

- Find the list of neighbourhood
- count the connected nodes
- the total number of pairs is equal to  $d_i(d_i - 1)/2$ , where  $d_i$  is the degree of node  $i$ .

## Cout de calcul

In terms of complexity, we need to scan all the pairs of nodes, the computational cost will be proportional to

$$N(\langle d^2 \rangle - \langle d \rangle)/2$$

where

$$\langle d \rangle = \frac{1}{N} \sum d_i$$

and

$$\langle d^2 \rangle = \frac{1}{N} \sum d_i^2$$

The computational cost will be dominated by  $\langle d^2 \rangle$ . This number is divergent for scale free networks where  $p_k \sim k^{-\alpha}$ . The crucial point is to find the connected neighbours. This can be improved!

# Centrality

The goal is to define a vector of weights, denoted  $\mathbf{x}$ , with the same dimension of nodes characterising the centrality of each node.

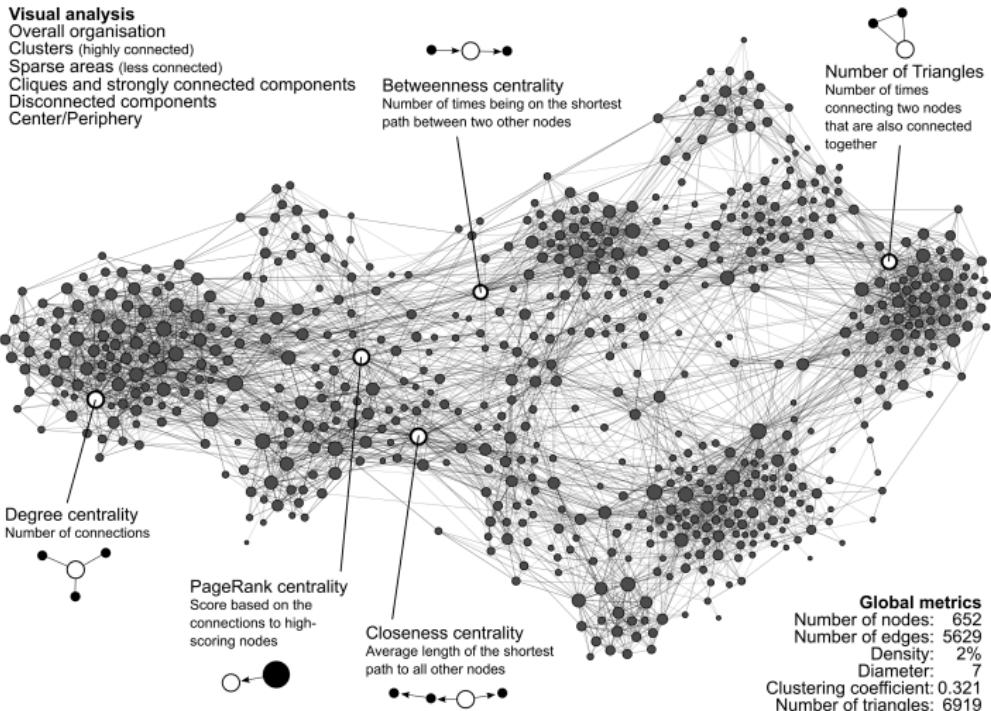
Centrality is known to indicate most influential nodes, but they are rather less informative for the vast majority of nodes which are not influential. Centrality indices are highly dependent on network topologies (see node influence metrics).

We will speak about 4 different definitions of centrality: *degree centrality*, *eigenvector centrality*, *closeness centrality* and *betweenness centrality*.

Some examples: twitter, internet infrastructures, super-spreader of disease

...

# Various centralities

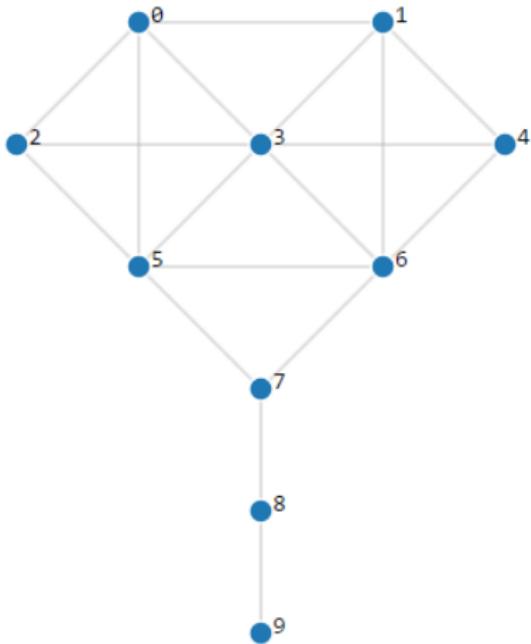


<https://dev.clariah.nl/files/dh2019/boa/0605.html>

## degree centrality

For all  $i \in V$ ,

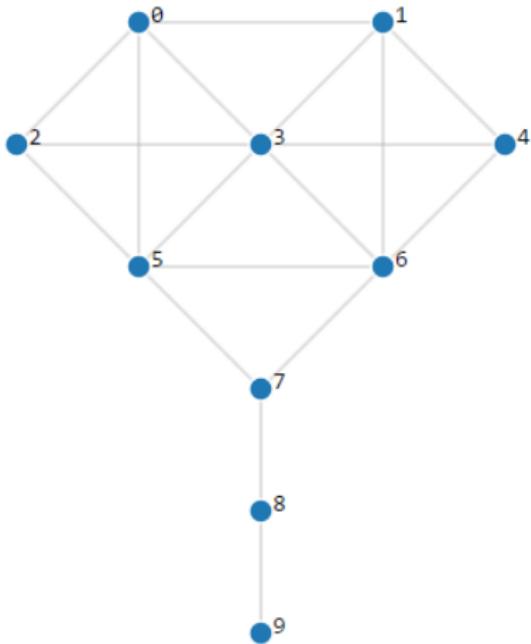
$$x_i = d_i = \#\{j \in V, (i, j) \in E\}$$



# closeness centrality

Let  $i$  and  $j$  be two nodes of a graph. Let us denote  $l_{ij}$  the length of the shortest path between  $i$  and  $j$ .

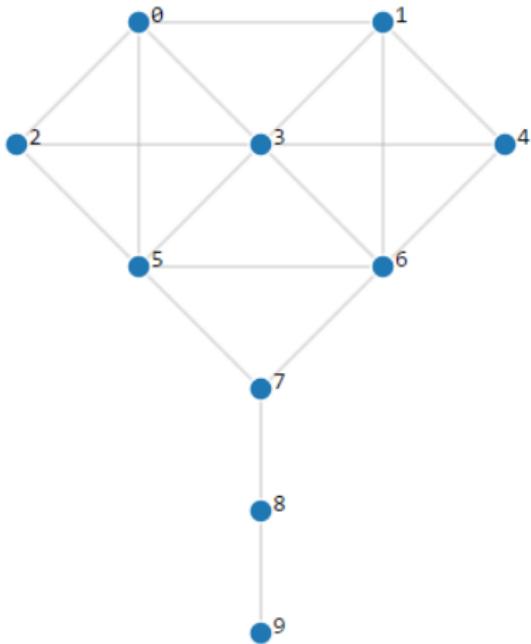
$$x_i = \frac{1}{N-1} \sum_{j \in V, j \neq i} l_{ij}.$$



# closeness centrality

Let  $i$  and  $j$  be two nodes of a graph. Let us denote  $l_{ij}$  the length of the shortest path between  $i$  and  $j$ . harmonic centrality

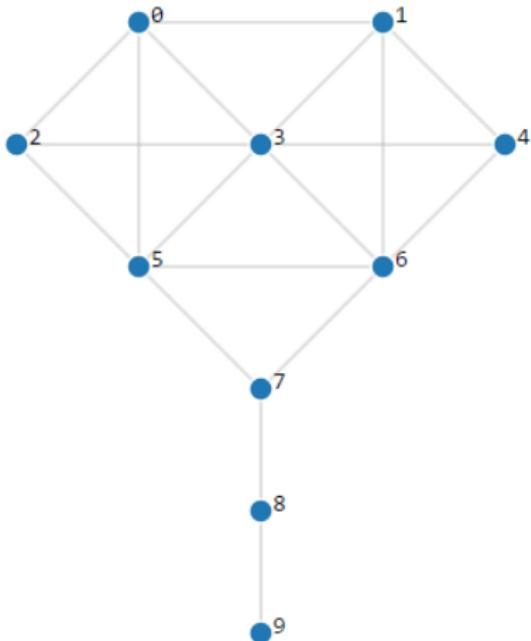
$$x_i = \frac{1}{N-1} \sum_{j \in V, j \neq i} \frac{1}{l_{ij}}.$$



## betweenness centrality

Let define  $n_{st}^i$  such that it is equal to 1 when  $i$  is belonging to the shortest path between  $s$  and  $t$ , and 0 otherwise. Let define  $n_{st}$  the total number of shortest path between  $s$  and  $t$ .

$$x_i = \sum_{s,t \in V} \frac{n_{st}^i}{n_{st}}$$



# Eigenvector centrality

The eigenvector centrality is measuring the influence of a node on the graph. This is quantified by the spectrum of the adjacency matrix of the graph.

## Definition

Let denote  $\mathbf{x}$  the eigenvectors associated to  $\lambda$  the largest eigenvalues for the adjacency matrix (This implies that  $\mathbf{x}$  has only positive values.)

$$x_i = \frac{1}{\lambda} \sum_{j \in V} A_{ij} x_j$$

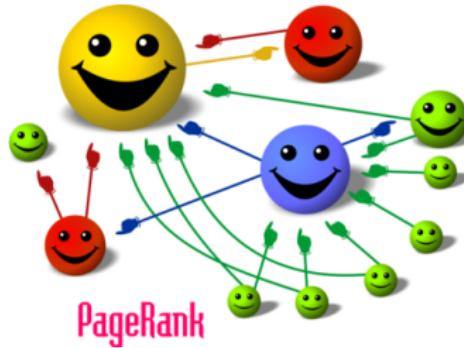
A node  $i$  has a high centrality if it has many neighbours or if its neighbours have a high centrality or both. Pay attention, the values have to be normalised for being able to compare them.

# PageRank (Google)

From Google: *PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.*

This is defined recursively. The output is a distribution used to represent the likelihood that a person randomly clicking on links will arrive at a particular page.

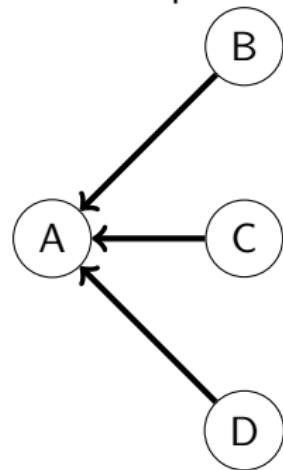
When Page Rank equal 0.5, there is a 50% chance that a person clicking on a random link will de directed to the said document.



# PageRank (Google): computation

Let us take 4 web pages: A, B, C and D. We initialise  
 $PR(A) = PR(B) = PR(C) = PR(D) = 0.25$ .

First example:

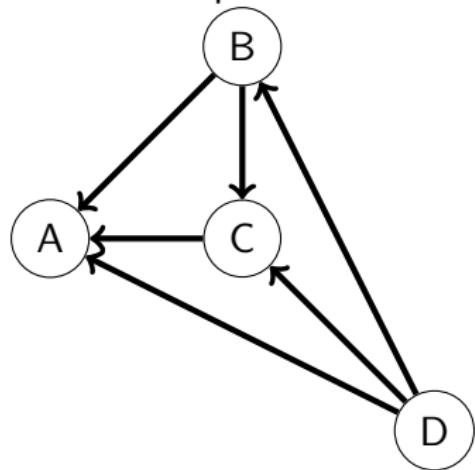


$$PR(A) = PR(B) + PR(C) + PR(D) = 0.75$$

# PageRank (Google): computation

Let us take 4 web pages: A, B, C and D. We initialise  $PR(A) = PR(B) = PR(C) = PR(D) = 0.25$ .

Second example:



$$PR(A) = \frac{PR(B)}{2} + PR(C) + \frac{PR(D)}{3} = 0.458$$

## PageRank (Google): computation

Page Rank conferred by an outbound link is equal to the document's own Page Rank score divided by the number of outbound links  $L()$ : for all  $u$ , node of the graph,

$$PR(u) = \sum_{v \in Bu} \frac{PR(v)}{v}$$

Finally, to take into account the fact that one person can stop clicking:

$$PR(u) = \frac{d - 1}{N} + d \left( \sum_{v \in Bu} \frac{PR(v)}{v} \right)$$

Because of the iteration process, Page Rank favors the old pages. A new page will likely have very few links despite its quality. Page Rank measures high valued page by the web community, not necessarily linked to quality and can be manipulated by monetise advertising links. Existence of link farms.

# PageRank (Google): algorithm

**Iterative**  $p_i$  is page  $i$ ,  $N$  the total number of pages.

At  $t = 0$ ,  $PR(p_i; 0) = \frac{1}{N}$

At each time step, the computation is,

$$PR(p_i; t + 1) = \frac{1 - d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{L(p_j)},$$

$d$  is the damping factor.

The computation ends when for small  $\varepsilon$  and for all  $p_i$ ,

$$|PR(p_i; t + 1) - PR(p_i; t)| < \varepsilon$$

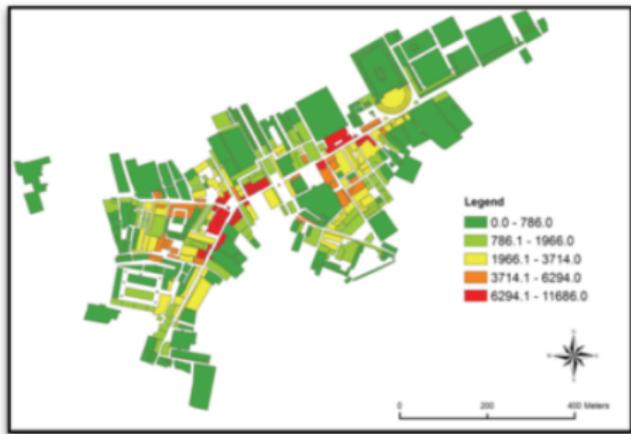
# Examples of real data sets

## Urban networks

### Closeness Centrality



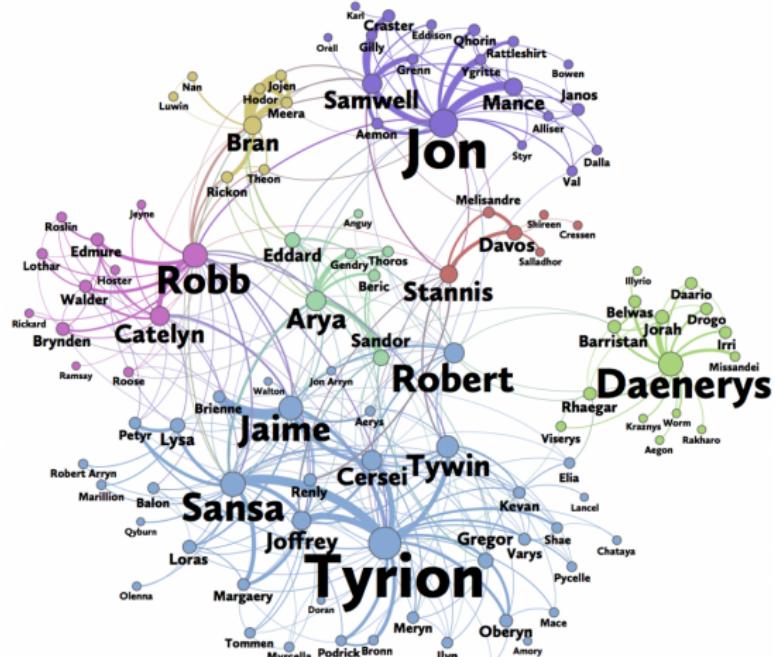
### Betweenness Centrality



*Rethinking Approaches for the Study of Urban Movement at Asia* K. A. Crawford 2019

# Examples on real data sets

## Game of thrones



<https://predictivehacks.com/social-network-analysis-of-game-of-thrones/>

## Examples on real data sets

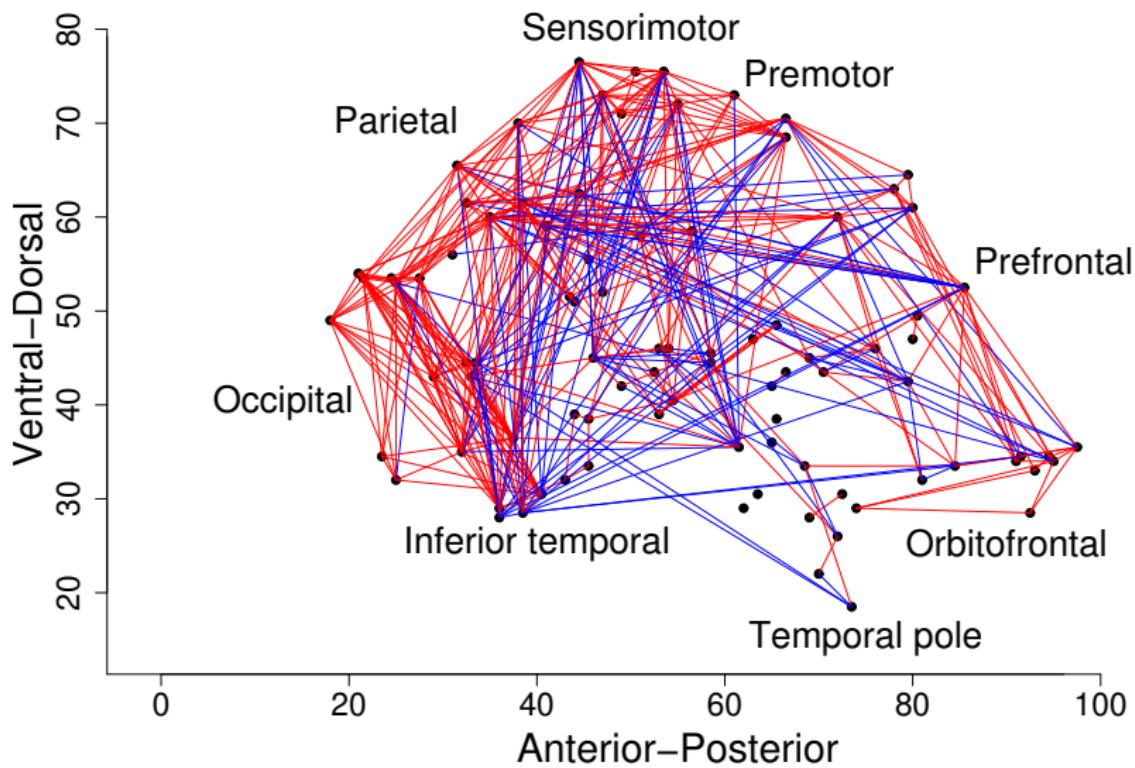


The brain network of the  
*C. Elegans* worm  
Data compiled by D. Watts and S. Strogatz.

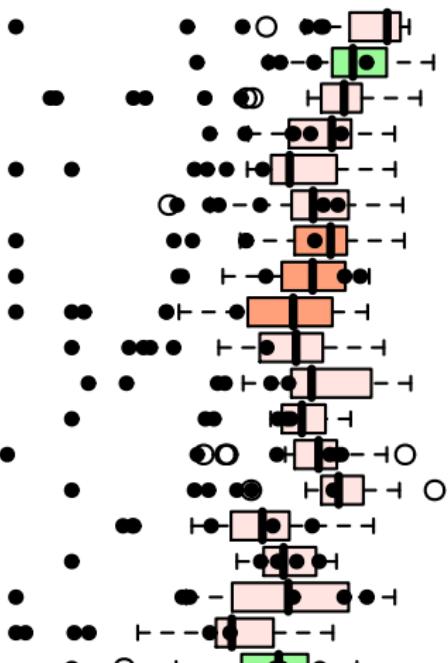
Map spatialized and exported from Gephi

# Examples on real data sets

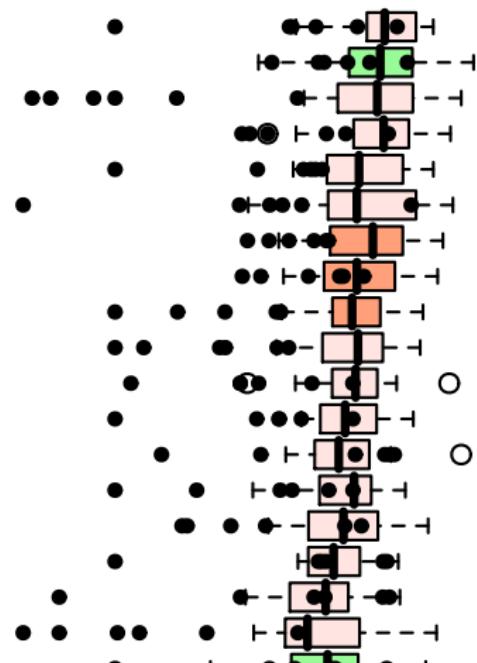
Brain data



# Left



# Right



## Examples on real data sets

