Networks Structure and Dynamics
2. Graph algorithms

Maximilien Danisch, Marwan Ghanem, Lionel Tabourier

LIP6 – CNRS and Sorbonne Université

first_name.last_name@lip6.fr

September 25th 2018

Outline

Outline

Definitions and notations

A graph $G = (V, E)$ is a couple of sets.

- V is the set of *nodes*
- $E \subseteq (V \times V)$ is the set of *edges*

We denote :

- $n = |V|$ the number of nodes
- $m = |E|$ the number of edges

u and v are neighbors if there is an edge between them.

Degree: (k_i or $d°(i)$) number of neighbors of i

Definitions and notations

A graph $G = (V, E)$ is a couple of sets.

- V is the set of *nodes*
- $E \subseteq (V \times V)$ is the set of *edges*

We denote :

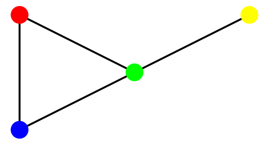
- $n = |V|$ the number of nodes
- $m = |E|$ the number of edges

u and v are neighbors if there is an edge between them.

Degree: (k_i or $d^\circ(i)$) number of neighbors of i

Average degree, density

- Average degree of a graph, $d^\circ(G) = \frac{\sum_v d^\circ(v)}{n}$
- density of a graph, $\delta = \frac{2m}{n(n-1)}$

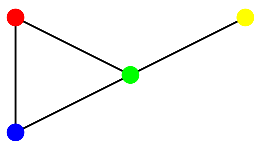


degrees : **2, 2, 3, 1** ; average degree 2

$n = 4$, $m = 4$, $\delta = \frac{8}{12} = 0.66..$

Average degree, density

- Average degree of a graph, $d^\circ(G) = \frac{\sum_v d^\circ(v)}{n}$
- density of a graph, $\delta = \frac{2m}{n(n-1)}$



degrees : **2, 2, 3, 1** ; average degree 2

$n = 4$, $m = 4$, $\delta = \frac{8}{12} = 0.66..$

Connectedness

Path from u to v: sequence of edges $(u, v_1), (v_1, v_2), \ldots, (v_{\alpha-1}, v)$
Length: number of edges in the path (here α)

Connected component: maximal set of nodes such that $\exists$ a path between any pair of nodes

Connected graph: only one connected component

Connectedness

[Path]() from u to v: sequence of edges
$(u, v_1), (v_1, v_2), \ldots, (v_{\alpha-1}, v)$
Length: number of edges in the path (here α)

Connected component: maximal set of nodes such that $\exists$ a path between any pair of nodes

Connected graph: only one connected component

Distributions

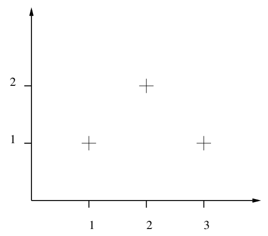
Distribution : synthetic way to represent a sequence of values.
$\rightarrow$ how many times a value occurs in the sequence?

Distributions

Distribution : synthetic way to represent a sequence of values.
$\rightarrow$ how many times a value occurs in the sequence?

Example/reminder of the degree distribution:
4 nodes, degrees : 2 2 3 1

1 $\rightarrow$ 1, 2 $\rightarrow$ 2, 3 $\rightarrow$1



Distribution characteristics

One benefit: characterize qualitatively a sequence of values.

Power law

- degree distribution follows $N_k \sim k^{-\alpha}$
- line in échelle log-log scale

heterogeneous distribution: non-homogeneous (various types of behaviors), in practice often close to a power law

Distribution characteristics

One benefit: characterize qualitatively a sequence of values.

Power law

- degree distribution follows $N_k \sim k^{-\alpha}$
- line in échelle log-log scale

heterogeneous distribution: non-homogeneous (various types of behaviors), in practice often close to a power law

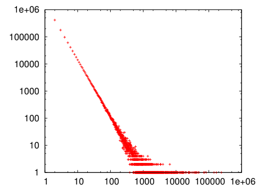
Distribution characteristics

One benefit: characterize qualitatively a sequence of values.

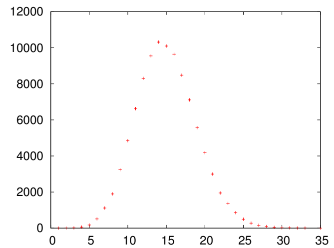
Power law

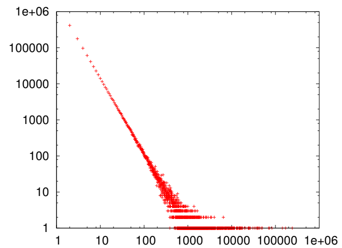
- degree distribution follows $N_k \sim k^{-\alpha}$
- line in échelle log-log scale

heterogeneous distribution: non-homogeneous (various types of behaviors), in practice often close to a power law



Heterogeneous vs homogeneous distributions





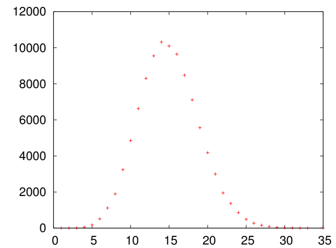
Homogeneous

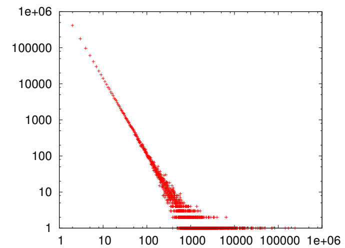
Idea of normality (and of exceptions)

Heterogeneous

Any kind of behaviours $\rightarrow$ no notion of normality

Heterogeneous vs homogeneous distributions





Homogeneous

Idea of normality (and of exceptions)

Heterogeneous

Any kind of behaviours $\rightarrow$ no notion of normality

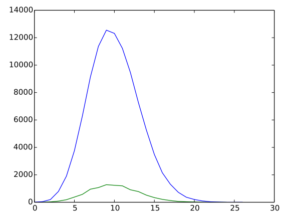
Two choices:

- N_k: number of occurrences of value k in the sequence
- p_k: proportion of the value k in the sequence
  $\rightarrow$ Normalized distribution

$$p_k = \frac{N_k}{n}$$

Just a change of the value on the Y-axis.

Allow to compare graphs with different sizes:



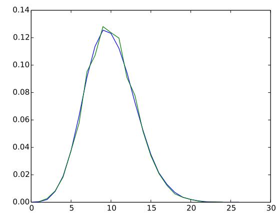
Two choices:

- N_k: number of occurrences of value k in the sequence
- p_k: proportion of the value k in the sequence
  $\rightarrow$ Normalized distribution

$$p_k = \frac{N_k}{n}$$

Just a change of the value on the Y-axis.

Allow to compare graphs with different sizes:



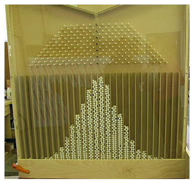
Example of homogeneous distribution: the normal law

Normal law (Gaussian law)

$$P(x) \sim exp\left(-\frac{(x - <x>)^2}{2\sigma^2}\right)$$

Normal law often found in “nature”, good model for:

- asymptotic limit of binomial law: repeated coin flipping experiments (heads:+1, tails:-1), Galton board, ...



- complex phenomena: human height distribution etc.

Characterizing a distribution

First and second moments

- $\langle k \rangle = \sum k p_k$: first moment (mean)
- $\langle k^2 \rangle = \sum k^2 p_k$: second raw moment
  we rather use the second central moment or variance (σ^2):

$$\sum (k - \langle k \rangle)^2 p_k = \langle k^2 \rangle - \langle k \rangle^2$$

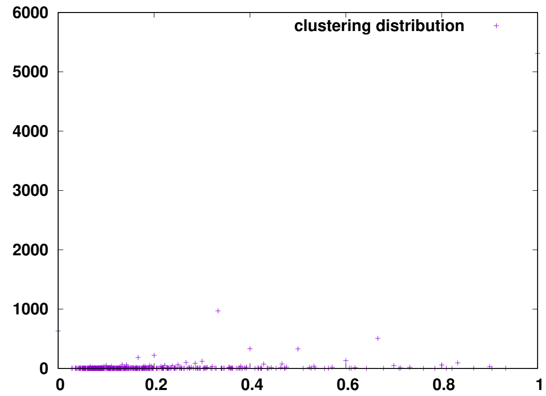
and higher order moments
(skewness - *dissymétrie*, kurtosis - *aplatissement*, ...)

Notion of cumulative distributions

- Distribution of k:
  N_k: number of occurrences equal to k
- Cumulative distribution of k:
  C_k: number of occurrences lower or equal to k
- Inverse cumulative distribution of k:
  IC_k: number of occurrences greater or equal to k

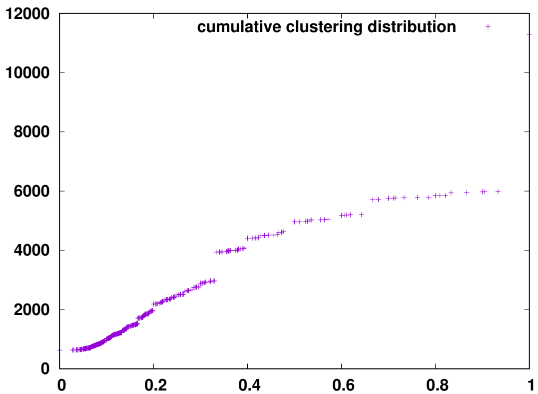
Relevance of cumulative distributions

Illustration: clustering distribution of a coauthoring network



Relevance of cumulative distributions

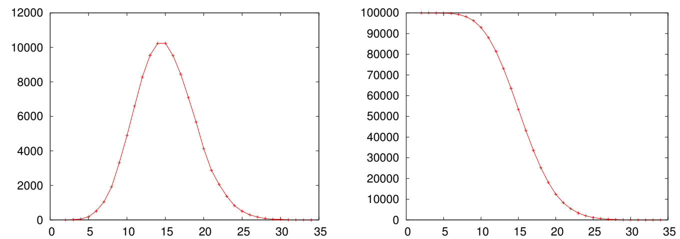
Illustration: clustering distribution of a coauthoring network



Cumulative and inverse cumulative distribution

- N_k: number of occurrences equal to k
- C_k: number of occurrences lower or equal to k
- IC_k: number of occurrences greater or equal to k

N_k and IC_k for a homogeneous distribution:

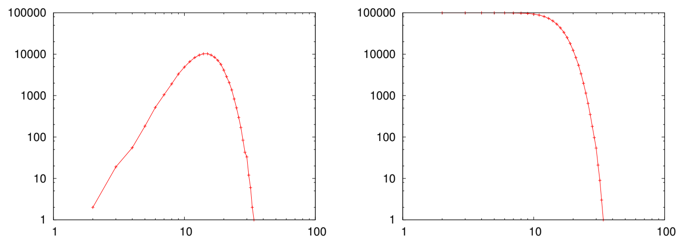


linear scale

Cumulative and inverse cumulative distribution

- N_k: number of occurrences equal to k
- C_k: number of occurrences lower or equal to k
- IC_k: number of occurrences greater or equal to k

N_k and IC_k for a homogeneous distribution:

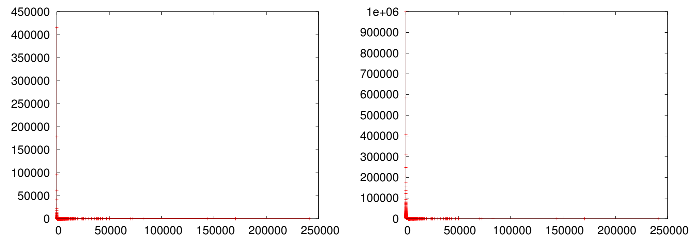


log-log scale

Cumulative and inverse cumulative distribution

- N_k: number of occurrences equal to k
- C_k: number of occurrences lower or equal to k
- IC_k: number of occurrences greater or equal to k

N_k and IC_k for a heterogeneous distribution:

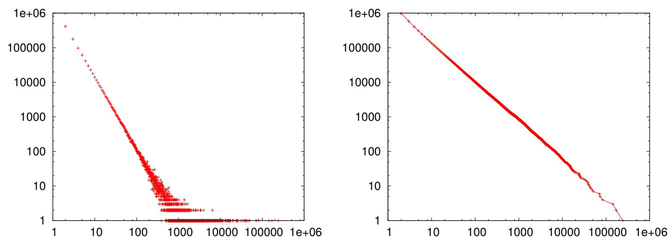


linear scale

Cumulative and inverse cumulative distribution

- N_k: number of occurrences equal to k
- C_k: number of occurrences lower or equal to k
- IC_k: number of occurrences greater or equal to k

N_k and IC_k for a heterogeneous distribution:



log-log scale

Cumulative and inverse cumulative distribution

- N_k: number of occurrences equal to k
- C_k: number of occurrences lower or equal to k
- IC_k: number of occurrences greater or equal to k

Homogeneous and heterogeneous

- can be distinguished on both normal and cumulative distributions

Ex: power-law

- $N_k \sim k^{-\alpha} \Longrightarrow C_k \sim k^{-\alpha+1}$
  Remark: same idea as $\int x^{-\alpha}\,dx \sim x^{-\alpha+1}$

Cumulative and inverse cumulative distribution

- N_k: number of occurrences equal to k
- C_k: number of occurrences lower or equal to k
- IC_k: number of occurrences greater or equal to k

Homogeneous and heterogeneous

- can be distinguished on both normal and cumulative distributions

Ex: power-law

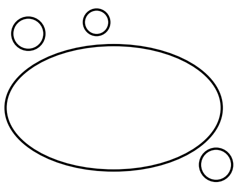
- $N_k \sim k^{-\alpha} \Longrightarrow C_k \sim k^{-\alpha+1}$
  Remark: same idea as $\int x^{-\alpha}\,\mathrm{d}x \sim x^{-\alpha+1}$

Outline

Connectedness (reminder)

For complex networks

In general, giant component
$\rightarrow$ contains most nodes



Q : How to identify the giant component? How to count the connected components?

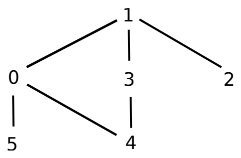
Breadth First Search algorithm (BFS)

Parcours en largeur

Algorithm 1: Breadth First Search of a graph G from node s.

```
begin
  F ← CreateEmptyQueue()
  Enqueue(F,s)
  Mark(s)
  while F not empty do
    u ← DequeueFirstElement(F)
    Display u
    for v neighbor of u in G do
      if Unmarked(v) then
        Enqueue(F,v)
        Mark(v)
      end
    end
  end
end
```

Example



1. Apply this algorithm to the previous example from node 3.
2. How to modify it so that it returns the tree of shortest paths from a node? Draw the corresponding tree.

Breadth First Search algorithm (BFS)

Properties of a BFS:

- From a node : we detect its connected component
  $\rightarrow$ 1 BFS per component
- Complexity: $\mathcal{O}(m)$
- With the alternative with parentage memorization: spanning tree *(fr: arbre couvrant)* of shortest paths

Breadth First Search algorithm (BFS)

Properties of a BFS:

- From a node : we detect its connected component
  $\rightarrow$ 1 BFS per component
- Complexity: $\mathcal{O}(m)$
- With the alternative with parentage memorization: spanning tree *(fr: arbre couvrant)* of shortest paths

Going back to local density

Several means to capture this idea, for example:

- clustering coefficient: $\mathtt{cc}(G) = \frac{\sum_v \frac{\Delta(v)}{\wedge(v)}}{n}$
- transitive ratio: $\mathtt{tr}(G) = \frac{3\Delta(G)}{\wedge(G)}$

In other words:

- clustering coefficient: compute a value for each node (with degree ≥ 2) then average
- transitive ratio: direct computation

Going back to local density

Several means to capture this idea, for example:

- clustering coefficient: $\mathtt{cc}(G) = \frac{\sum_v \frac{\Delta(v)}{\wedge(v)}}{n}$
- transitive ratio: $\mathtt{tr}(G) = \frac{3\Delta(G)}{\wedge(G)}$

In other words:

- clustering coefficient: compute a value for each node (with degree ≥ 2) then average
- transitive ratio: direct computation

Transitive ratio *vs* Clustering coefficient

We have $n \in \mathbb{N}$, let's G_n be the graphs of $2n+1$ nodes and $3n$ edges such that:

- a unique node n_0 is connected to all other nodes in the network
- all other nodes have degree 2

Exercise :

1. Draw the cases of G_3, G_4.
2. Compute the local density coefficients for G_4.
3. How do these coefficients evolve when n goes to ∞?
4. Deduce how to interpret these coefficients in terms of probability.

Computing the number of triangles

Both coefficients rely on the number of triangles. How to enumerate the number of triangles that node n belongs to?

Naive answer: For any pair of neighbors (u_1, u_2) of v, test if (u_1, u_2) exists.

Questions :

1. Complexity of the algorithm for node v?
2. In which case is it expensive?
3. Is it a problem for the networks we are working on?

Computing the number of triangles

Both coefficients rely on the number of triangles. How to enumerate the number of triangles that node n belongs to?

Naive answer: For any pair of neighbors (u_1, u_2) of v, test if (u_1, u_2) exists.

Questions :

1. Complexity of the algorithm for node v? $\Sigma_v \frac{k(v).(k(v)-1)}{2}$
2. In which case is it expensive?
3. Is it a problem for the networks we are working on?

Computing the number of triangles

Both coefficients rely on the number of triangles. How to enumerate the number of triangles that node n belongs to?

Naive answer: For any pair of neighbors (u_1, u_2) of v, test if (u_1, u_2) exists.

Questions :

1. Complexity of the algorithm for node v? $\Sigma_v \frac{k(v).(k(v)-1)}{2}$
2. In which case is it expensive? if $k(v)$ is large
3. Is it a problem for the networks we are working on?

Definitions and metrics
Algorithms
Connected components
Local density
Distances and diameter
Centralities

Computing the number of triangles

Both coefficients rely on the number of triangles. How to enumerate the number of triangles that node n belongs to?

Naive answer: For any pair of neighbors (u_1, u_2) of v, test if (u_1, u_2) exists.

Questions :

1. Complexity of the algorithm for node v? $\Sigma_v \frac{k(v).(k(v)-1)}{2}$
2. In which case is it expensive? if $k(v)$ is large
3. Is it a problem for the networks we are working on? yes, because heterogeneous degree distribution

Definitions and metrics
Algorithms
Connected components
Local density
Distances and diameter
Centralities

(Faster) computation of the number of triangles

Other point view: Considering edge (u, v), how many triangles contain it?
Idea: There is a triangle if there exists a node w neighbor of u and v.
Solution: We need to compute the size of the intersection of the neighborhood of u and v. Efficient if lists of neighbors are ordered.

Algorithm 2: Size of the intersection of two ordered lists U and V, of size $d^\circ(u)$ and $d^\circ(v)$

```
i_u = 0
i_v = 0
nb = 0
while (i_u < d°(u)) and (i_v < d°(v)) do
    if U[i_u] < V[i_v] then
        i_u++
    else
        if U[i_u] > V[i_v] then
            i_v ++
        else
            nb++ // triangle found
            i_u++
            i_v++
        end
    end
end
```


Definitions and metrics
Algorithms
Connected components
Local density
Distances and diameter
Centralities

(Faster) computation of the number of triangles

Other point view: Considering edge (u, v), how many triangles contain it?
Idea: There is a triangle if there exists a node w neighbor of u and v.
Solution: We need to compute the size of the intersection of the neighborhood of u and v. Efficient if lists of neighbors are ordered.

Algorithm 3: Size of the intersection of two ordered lists U and V, of size $d^\circ(u)$ and $d^\circ(v)$

```
i_u = 0
i_v = 0
nb = 0
while (i_u < d°(u)) and (i_v < d°(v)) do
    if U[i_u] < V[i_v] then
        i_u++
    else
        if U[i_u] > V[i_v] then
            i_v ++
        else
            nb++ // triangle found
            i_u++
            i_v++
        end
    end
end
```

(Even faster) computation of the number of triangles

Remarks:

- Algorithm slow if u or v have high degree
- Triangle (u, v, w) may be detected from the 3 list intersections: U with V, U with W, V with W

Idea: Reduce as much as possible the size of the lists involved in the intersection computation:

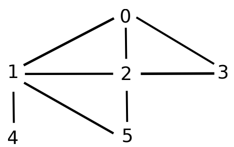
1. sort nodes per decreasing degree and re-index the graph
2. consider edges (u, v) such that $u < v$
3. we look for nodes w neighbor of u and v only if $w < u$ (and so $w < v$).

(Fast) computation of the number of triangles

```
Algorithm 4: Compute the number of triangles
Result: Table tr contains the # of triangles of each node
tr: table of size n initialized at 0;
for u = 0; u < n; u++ do
    for i=0; i < d°(u); i++ do
        v = i-th neighbor of u
        if u < v then
            U = list of neighbors of u
            V = list of neighbors of v
            i_u = 0 , i_v = 0
            while (i_u < d°(u)) and (i_v < d°(v))
            and (U[i_u] < u) and (V[i_v] < u) do
                if U[i_u] < V[i_v] then
                    i_u++;
                else
                    if U[i_u] > V[i_v] then
                        i_v ++;
                    else
                        tr[u]++;
                        tr[v]++;
                        tr[U[i_u]]++;
                        i_u++;
                        i_v++;
                    end
                end
            end
        end
    end
end
```


Exercise

Apply the previous algorithm to the following graph:



Higher order cliques

Clique

Triangles are 3-nodes cliques. A clique is a complete subgraph.

- subgraph: graph obtained considering a subset of nodes and the edges between these nodes *(fr: sous-graphe)*
- complete: any node is connected to all others *(fr: complet)*

Maximal cliques

Decomposition of a graph into its maximal cliques

Obtaining the list of all maximal cliques is known to be computationally hard
$\rightarrow$ we favor search with fixed size

Higher order cliques

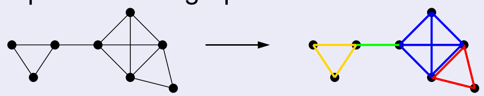
Clique

Triangles are 3-nodes cliques. A clique is a complete subgraph.

- subgraph: graph obtained considering a subset of nodes and the edges between these nodes *(fr: sous-graphe)*
- complete: any node is connected to all others *(fr: complet)*

Maximal cliques

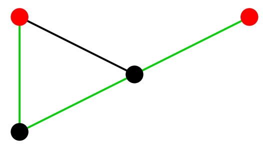
Decomposition of a graph into its maximal cliques



Obtaining the list of all maximal cliques is known to be computationally hard
$\rightarrow$ we favor search with fixed size

Distances and diameter (reminder)

path from u to v = sequence of edges u...v

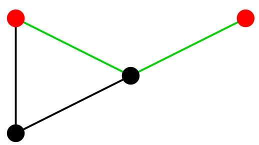


one path of length 3

Distances and diameter (reminder)

path from u to v = sequence of edges u...v

distance $d(u, v)$ = length of *one* shortest path



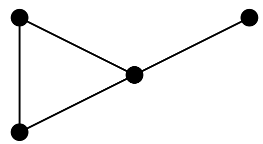
a shortest path; length 2 $\Rightarrow$ distance = 2

Distances and diameter (reminder)

path from u to v = sequence of edges u...v

distance $d(u, v)$ = length of *one* shortest path

average distance = average of the distances between all pairs of nodes



Average distance = $8/6$

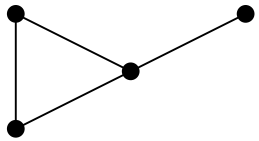
Distances and diameter (reminder)

path from u to v = sequence of edges u...v

distance $d(u, v)$ = length of *one* shortest path

average distance = average of the distances between all pairs of nodes

diameter Δ = highest distance (on any set of pairs)



diameter = 2

Distances computation

Distance from a node to all others:
modified breadth first search $\rightarrow$ Complexity: $\mathcal{O}(m)$

Average distance, diameter

Need all distances $\rightarrow \mathcal{O}(nm)$
possible to approximate or to give bounds on the diameter

Example: approximation of average distance

Distance is a homogeneous property
Possible to use a random sampling
!: not if heterogeneous property

Distances computation

Distance from a node to all others:
modified breadth first search $\rightarrow$ Complexity: $\mathcal{O}(m)$

Average distance, diameter

Need all distances $\rightarrow \mathcal{O}(nm)$
possible to approximate or to give bounds on the diameter

Example: approximation of average distance

Distance is a homogeneous property
Possible to use a random sampling
!: not if heterogeneous property

Distances computation

Distance from a node to all others:
modified breadth first search $\rightarrow$ Complexity: $\mathcal{O}(m)$

Average distance, diameter

Need all distances $\rightarrow \mathcal{O}(nm)$
possible to approximate or to give bounds on the diameter

Example: approximation of average distance

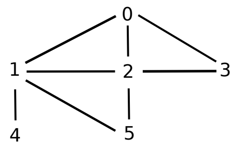
Distance is a homogeneous property
Possible to use a random sampling
!: not if heterogeneous property

Exercise

Whether it be for the average distance or the diameter, one needs to be able to compute the distance between two nodes.

Exercise :

1. Propose an algorithm, based on the BFS principle, enabling to compute the distance between one given node towards **all** nodes of the graph.
2. Apply the algorithm on the following graph (taking 5 as the initial node):



Modified BFS

Algorithm 5: Distance from node *s* in graph *G*.

```
begin
    F ← CreateEmptyQueue()
    Enqueue(F,s)
    ∀v Dist(v) initialized at -1
    Dist(s) ← 0
    while F not empty do
        u ← DequeueFirstElement(F)
        Display u
        for v neighbor of u in G do
            if Dist(v) = -1 then
                Enqueue(F,v)
                Dist(v) ← Dist(u) +1
            end
        end
    end
end
```


Approximations

Approximation: given a property P, how to estimate this property on a given graph.
Examples: average degree, average distance, diameter, . . .

One possible approach (sampling)

1. Pick a node v of G at random
2. Estimate the property for v
3. Go back to step 1 while the estimation is not good enough

Questions:

- How to express the notion of "good enough"?
- How to know if this approach provides a good approximation or not?

Definitions and metrics
Algorithms
Connected components
Local density
Distances and diameter
Centralities

Approximations

Approximation: given a property P, how to estimate this property on a given graph.
Examples: average degree, average distance, diameter, . . .

One possible approach (sampling)

1. Pick a node v of G at random
2. Estimate the property for v
3. Go back to step 1 while the estimation is not good enough

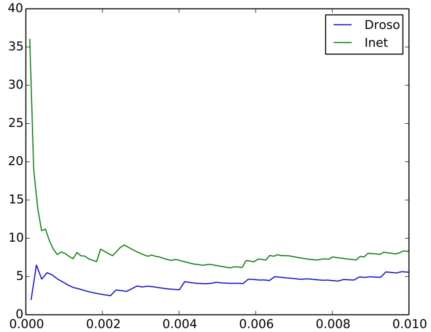
Questions:

- How to express the notion of "good enough"?
- How to know if this approach provides a good approximation or not?

33/46

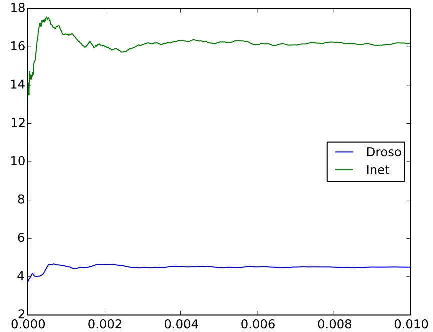
Average degree

Application of the former method to the average degree:
X-axis : fraction of measured values



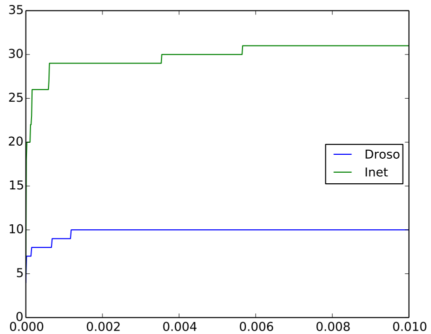
Average distance

Application of the former method to the average degree:
X-axis : fraction of measured values



Diameter

Application of the former method to the average degree:
X-axis : fraction of measured values



Approximations

Quality of the approximation: depends on the nature of the property.

Other possible approach:

- Compute (lower and upper) bounds of the property
- Rely on the property to drive the computations

Example: For every node v, let max_v be the greatest distance from v to a node of G. Then the diameter D of G is such that:

$$max_v \leq D \leq 2max_v$$

Exercise : explain why.

Approximations

Quality of the approximation: depends on the nature of the property.

Other possible approach:

- Compute (lower and upper) bounds of the property
- Rely on the property to drive the computations

Example: For every node v, let max_v be the greatest distance from v to a node of G. Then the diameter D of G is such that:

$$max_v \leq D \leq 2max_v$$

Exercise : explain why.

Centrality measures

Question: how to quantify the importance of a node in a network?

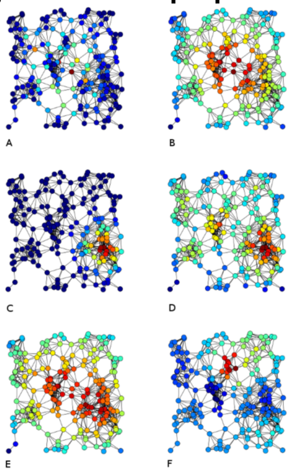
Centrality measures: “key” nodes of the network.

- Degree centrality (degree)
- Katz centrality
- Closeness (and harmonic) centralities
- Eigenvector and Pagerank centralities
- Betweenness centrality
- etc.

Centrality measures

Different metrics highlight differents properties

A. betweenness
B. closeness
C. eigenvector
D. degree
E. harmonic
F. Katz



About betweenness centrality

Let G be a graph and v, s and t be nodes of G.
We call *betweenness centrality* the value:

$$BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

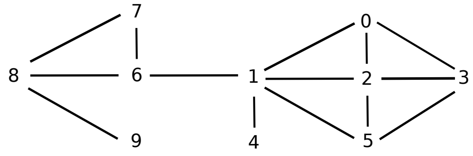
with:

- σ_{st} : number of shortest paths between s and t
- $\sigma_{st}(v)$: number of shortest paths between s and t going through v

Meaning

Capture the importance of a node v in a graph as a relay for diffusing information:

- propagation of signals/messages/virus
- connectivity of the network



Exercise : compute $BC(0)$, $BC(1)$ and $BC(4)$.

Link betweenness centrality

In the former example:

- 1 has a high centrality
- but all links starting from 1 don't have the same importance

There exists a similar definition to compute the importance of the links:

$$BC(u, v) = \sum_{s \neq t \neq u \neq v} \frac{\sigma_{st}(u, v)}{\sigma_{st}}$$

with:

- σ_{st} : nb of shortest paths between s and t
- $\sigma_{st}(u, v)$: nb of shortest paths between s and t using (u, v)

Relation to communities

What do you think about the relation between link betweenness and the communities of a network?

Can you think of an algorithm that would use betweenness to infer communities?

1. (Re)compute the centrality of all links
2. Delete the link with highest centrality
3. Test if an isolated connected component appears
4. Repeat steps 1 – 3 until there are no more links.

Relation to communities

What do you think about the relation between link betweenness and the communities of a network?

Can you think of an algorithm that would use betweenness to infer communities?

1. (Re)compute the centrality of all links
2. Delete the link with highest centrality
3. Test if an isolated connected component appears
4. Repeat steps 1 – 3 until there are no more links.

Relation to communities

What do you think about the relation between link betweenness and the communities of a network?

Can you think of an algorithm that would use betweenness to infer communities?

1. (Re)compute the centrality of all links
2. Delete the link with highest centrality
3. Test if an isolated connected component appears
4. Repeat steps $1-3$ until there are no more links.

About sparse matrix product

Many measurement on graphs demand matrix product
ex: Katz, PageRank, eigenvector centralities

Matrix product

Given a matrix M ($n \times n$) and a vector V
we compute the vector W ($n \times 1$) such that:

$$W = M \times V$$

where $\forall i \in [\![1,n]\!]$, $W_i = \sum_{j=1}^{n} M_{ij} \times V_j$

Note that for a $n \times n$ sparse matrix M with m nonzero values,
we take into account only the nonzero values
$\Rightarrow$ compute W is in $\mathcal{O}(m+n)$ and not in $\mathcal{O}(n^2)$.

About sparse matrix product

Many measurement on graphs demand matrix product
ex: Katz, PageRank, eigenvector centralities

Matrix product

Given a matrix M ($n \times n$) and a vector V
we compute the vector W ($n \times 1$) such that:

$$W = M \times V$$

where $\forall i \in [\![1,n]\!]$, $W_i = \sum_{j=1}^{n} M_{ij} \times V_j$

Note that for a $n \times n$ sparse matrix M with m nonzero values,
we take into account only the nonzero values
$\Rightarrow$ compute W is in $\mathcal{O}(m+n)$ and not in $\mathcal{O}(n^2)$.

About sparse matrix product

Many measurement on graphs demand matrix product
ex: Katz, PageRank, eigenvector centralities

Matrix product

Given a matrix M ($n \times n$) and a vector V
we compute the vector W ($n \times 1$) such that:

$$W = M \times V$$

where $\forall i \in [\![1,n]\!]$, $W_i = \sum_{j=1}^{n} M_{ij} \times V_j$

Note that for a $n \times n$ sparse matrix M with m nonzero values,
we take into account only the nonzero values
$\Rightarrow$ compute W is in $\mathcal{O}(m+n)$ and not in $\mathcal{O}(n^2)$.

Classic Version

Algorithm 6: Classic sparse Matrix/Vector multiplication

input: M, V;
for i *from* 1 *to* n **do**
  $W[i] \leftarrow 0$
  for *each* j *such that* $M[i][j] \neq 0$ **do**
    $W[i] = W[i] + M[i][j] \times V[j]$
  end
end
return W;

Should be implemented with “the adjacency list” graph data structure.

to be continued in the course on PageRank