M2 – NSD (Practical Work 1 - Sessions 1,2,3)

Computation of network structural properties

Lionel Tabourier, Fabien Tarissan

This first practical work aims at getting used to basic statistical notions on large interaction networks and at having more insights about analysis and computation of these properties. This is achieved through two practical cases.

In the following (and in future practical works), we represent networks by graphs. We denote a graph $G = (V, E)$, where V is the set of vertices (also called nodes) of the graph and numbered from 0 to $|V| - 1$ and E is the set of links (also called edges).

1 To get things started

Exercise 1 — *Preparation* Create (manually) a few graphs that will enable you to check the results of your programs.

Store them in files where each line is of the form :

x y

which indicates that a link exists between nodes x and y.

2 Basic operations and properties

2.1 Load a graph in memory : compact storage version

In the following (and in future sessions), *program* stands for a program or a combination of programs in the language of your choice.

Exercise 2 — *Size of a graph* Create a program that counts the number of nodes in a graph (without storing it in memory) and writes this value on the standard output.

Exercise 3 — *Node degree* Create a program which counts the degree (*i.e.* the number of edges) of each node of a graph (without storing it in memory) and writes this value on the standard output.

Exercise 4 — *Loading a graph in memory* Using the two former exercises, create a program reading a graph and storing it in memory in the optimal way memory-wise seen during the course, that is a table of tables.

If possible, the tables will be contiguous in terms of storage (for most of the rest of this course, compact storage will not be necessary).

2.2 A few basic statistical properties

Exercise 5 — *Number of isolated nodes* Create a program that computes the number of nodes of degree 0 of a graph, as well as the density, the average degree, the minimum and maximum degree of the graph.

Is it necessary to store the graph in memory for this purpose ?

Let's remind that the *degree distribution* of a graph G is the function which associates to an integer k the number of nodes which have a degree k.

Exercise 6 — *Degree distribution* Create a program which calculates the degree distribution of a graph and writes it in a file in the following format :
d n
where n is the number of nodes in the graph having degree d (you can use unix commands to do this very fast).

2.3 Applications to two real-world networks

Get the protein-protein interaction network of the fly drosophila [1], which will be denoted PPI-DROSO from now on, and the network of interactions between routers of the Internet [2], which will be denoted INET from now on. These two datasets are stored on the public directory `/Enseignants-Public/tarissan/sdr/2014/` and on the web page of the course. Careful : these graphs can contain loops.

Exercise 7 — *Deleting loops* Create a program that deletes loops existing in the graph.

Exercise 8 — *Application* We apply now what has been implemented until now to the cases of networks PPI-DROSO and INET.

1. Using the programs formerly defined, compute the characteristics of networks PPI-DROSO and INET, as well as their degree distributions.
2. Display these distributions, using (for example) GNUPLOT (see section 4.3)
3. Comment the results.

Exercise 9 — *Cumulative distribution* Create a program that computes the cumulative degree distribution for a given graph. Use this program on PPI-DROSO and INET and display the result using GNUPLOT. Is this consistent with the distribution of the previous exercise ?

3 Local density vs global density

The notion of local density aims at capturing how set nodes which are *close* are connected to each other. The ratio between triangles and the connected triplets of nodes is a usual way to assess local density. More precisely, two measures are commonly used :

Definition 1 (Clustering Coefficient) *Let $G = (V, E)$ be a graph and $v \in V$ a node. Let $N(v)$ be the set of neighbours of v and $E(N(v)) = \{(u_1, u_2) \in E \mid u_1, u_2 \in N(v)\}$ the set of edges among these neighbours. The clustering coefficient* $\mathrm{cc}(v)$ *of v is defined by :*

$$\mathrm{cc}(v) = \frac{2|E(N(v))|}{d^\circ(v)(d^\circ(v)-1)}.$$

The clustering coefficient of the graph G is then the average of the clustering coefficients of every nodes of G with degree $\geqslant 2$.

Definition 2 (Transitive Ratio) *Let $G = (V, E)$ be a graph. The* transitive ratio $\mathrm{tr}(G)$ of G *is defined by :*

$$\mathrm{tr}(G) = \frac{3N_\Delta}{N_\vee},$$

where N_Δ is the number of triangles of the graph and $N_\vee$ is the number of connected triplets.

Exercise 10 — *Clustering coefficient and transitive ratio*

1. Create a program which enables to compute these coefficients for a given network.
2. Apply this program to the files PPI-DROSO and INET.
3. How to analyse these values ?
4. Test the efficiency of your program (the time needed to achieve the computation).

1. obtained via the online data repository `http://160.80.34.4/mint/Welcome.do`
2. obtained via the online data repository `http://svnet.u-strasbg.fr/merlin`

4 Breadth first search

We recall here the breadth first search (BFS) algorithm for a given graph and a given seed node.

Algorithm 1: BFS of a graph G starting from node s.

```
F ← CreateEmptyQueue()
Enqueue(F,s)
Mark(s)
while F not empty do
    u ← DequeueFirstElement(F)
    Show u
    for v neighbour of u in G do
        if NotMarked(v) then
            Enqueue(F,v)
            Mark(v)
        end
    end
end
```

Exercise 11 — *Breadth first search* Create a program which, given a graph G and a node s, display the BFS of G from node s.

The following exercises use this algorithm to compute some properties of a given graph.

4.1 Connected components

Exercise 12 — *Size of the connected components* By modifying the previous program, create a program which computes the size of each connected components of a given graph. Use this program to display the distribution of sizes of the connected components of networks PPI-DROSO and INET. Comment the results.

Exercise 13 — *Principal connected component* By modifying the previous program, create a program that isolates the principal connected component of a given graph. Use it on PPI-DROSO and INET.

4.2 Centrality

Betweenness centrality is a property that aims at capturing how important is a node (or a link) to propagate a signal/message, in other words it assesses its importance as a relay in the structure of the network. It relies mostly on the notion of shortest path.
By convention, we denote σ_{st} the number of shortest paths from s to t of a graph G and $\sigma_{st}(v)$ the number of shortest paths from s to t going through v (we assume here that $s \neq t \neq v$). The betweenness centrality of a node v is then defined by the following formula :

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

Notice that the computation of $BC(v)$ is particularly simple when there only exists one shortest path for any pair s, t of G. In this case $BC(v)$ is exactly $\sum_{s \neq t \neq v} \sigma_{st}(v)$ that is the number of shortest paths in G going through v.

Exercise 14 — *Set of shortest paths*
Modify the BFS algorithm so that it gives *all* the shortest paths. Does the complexity of the algorithm change ?

Exercise 15 — *Number of shortest paths*

Given a set of shortest paths from a node s, how should we compute the number of shortest paths going through $v \neq s$ in this set ? Compute this number for each node of the graph.

Exercise 16 — *Betweenness centrality*

1. Using the previous exercise, give an algorithm that enables to compute the centrality of each node in the graph.
2. What is its complexity ?
3. Apply this algorithm to PPI-DROSO and INET.
4. Display the distributions obtained.
5. Display the corresponding cumulative distributions.

Mini-tutorial `gnuplot`

GNUPLOT is a program that allows to obtain a graphical representation of a structured dataset. There are a lot of online documentations and tutorials [3]. Here are only a few basic commands which will be useful to display your results.

- Launch and plot. You just need to execute the command `gnuplot` in a terminal and then `plot "file.txt" using i:j` in gnuplot command prompt to display the information contained in file *file.txt* using the values in column i (abscissa) and column j (ordinates).

- Ranges. If the minimal and maximal values on the axes are not appropriate :
  `plot [x1:x2][y1:y2] "file.txt"`
  allows to plot with x-axis values between $x1$, $x2$ and y-axis values between $y1$, $y2$.
  Another option is the command :
  `set xrange` $[x1 : x2]$: display data corresponding to values of x between $x1$ and $x2$ (same goes with `yrange`)

- Scales and labels. Turn to a logarithmic scale on both axes : `set logscale xy`
  Turn to a linear scale on both axes : `unset logscale xy`
  Give a specific label to an axis : `set xlabel` *name* : give the name *name* to x axis (same goes with `ylabel`).

- Saving. To save a picture in a file, you must first choose the kind of file with the command `set terminal`, for example for a postscript file :
  `set terminal postscript`
  Then you can choose a file name :
  `set output "my_plot.ps"`
  so that the plot is created in the corresponding file (in the current directory). Then use the corresponding `plot` command to the curve that you want to plot.
  Other format are available, for example : png, pdf, ...

  Remark : notice that after an output file has been declared with `set output`, all the following commands will write in this output (so that the first one may be overwritten). To avoid that, create a new output, or if you want to print on screen, use :
  `set output "/dev/null"`
  `set term x11`

3. `http://www.gnuplot.info/`
`http://www.info.univ-angers.fr/~gh/tuteurs/tutgnuplot.htm`