Homework 5 - Lab

Achille Salaün

17 février 2016

Question 1

There are $N = 6301$ nodes and $M = 20777$ edges in the network.

To create the *adjacency matrix* ADJ, I built the following method :

```
# build the adjacency matrix
def buildAdjacencyMatrix(gnutella):
    N = numberNodes(gnutella)
    matrix = [[0 for j in range(0,N)] for i in range(0,N)]
    for l in range(0,len(gnutella)):
        line = gnutella[l]
        i = line[0]
        j = line[1]
        matrix[i-1][j-1]=1
        matrix[j-1][i-1]=1
    return matrix
```

Where the `gnutella` input is a list of pairs.

Question 2

I would choose $p = \frac{d}{N}$, where d is the *average degree* of the network.

Question 3

The degree distribution is, we could plot it in the chart of the FIGURE 1.

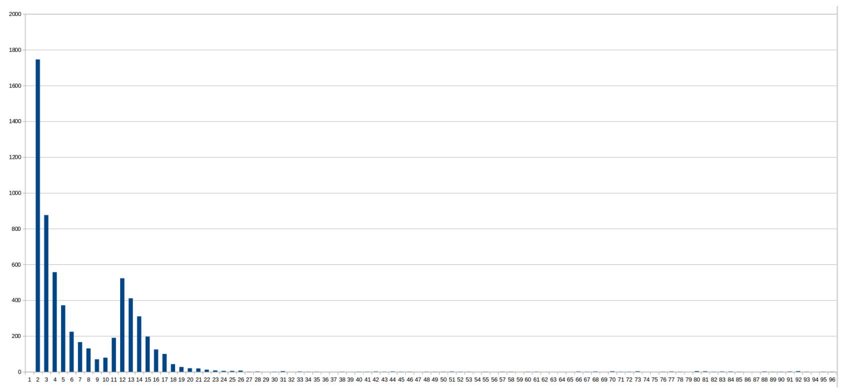

FIGURE 1 – Degree distribution of the whole network

Furthermore, the *average degree* is 6.595, the *maximum degree* is 97, while the *minimum degree* is 1.

The degree distribution is not *power-law*, indeed, we can see two local maxima which is not the shape of a power-law degree distribution.
When we look at the graph (cf. FIGURE 2), we can see the existence of a "core" : there the average degree is different and explains the second maximum.

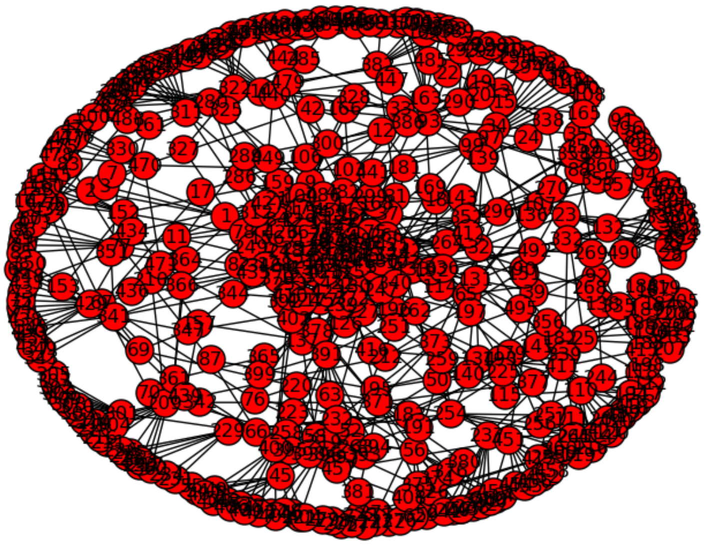

FIGURE 2 – Subnetwork's whole view

Question 4

The clustering coefficient is $CC = 0.021$. He is computed with the following method :

```
# compute the clustering coefficient (float)
def clusteringCoefficient(graph,matrix):
        triangles = 0
        triples = 0
        for n in range(0,len(graph)):
              n_neighbours = len(graph[n])
              triples += n_neighbours*(n_neighbours-1)/2
              if(len(graph[n])>1):
                    for i in range(0,n_neighbours-1):
                          for j in range(i+1,n_neighbours):
                                node_i = graph[n][i]
                                node_j = graph[n][j]
                                if(matrix[node_i][node_j]==1):
                                      triangles += 1
        return float(triangles)/float(triples)
```

Where `graph` is the list of the lists of neighbours of each node (one line is all the neigbours of the node whose the index is the line number), and `matrix` is the adjacency matrix.

The Clustering Coefficient of the respective Poisson graph is $CC_p = p = 0.001$, which is far smaller than CC. Thus, a Poisson graph is a bad model for our network.

Question 5

Here is the `randomWalk()` method :

```
# implement a random walk during "steps" steps, starting from "start" and given the "nextOne" rule.
# this method returns the list of all the steps
def randomWalk(matrix,graph,start,steps,nextOne):
        if(graph[start]==[]):
              print("WARNING : random walk from a lonely node!")
              return [start]
        listSteps = [start]
        node = start
        for n in range(0,steps):
              line = graph[node]
              node = nextOne(matrix,line)
              listSteps.append(node)
        return listSteps
```

using the `uniform()` method :

```
# choose a node following an uniform law
def uniform(matrix,line):
        return line[random.randint(0,len(line)-1)]
```

Then, we just have to call `randomWalk(ADJ,G,N0,400,uniform)`, and we get so the Figure 3.

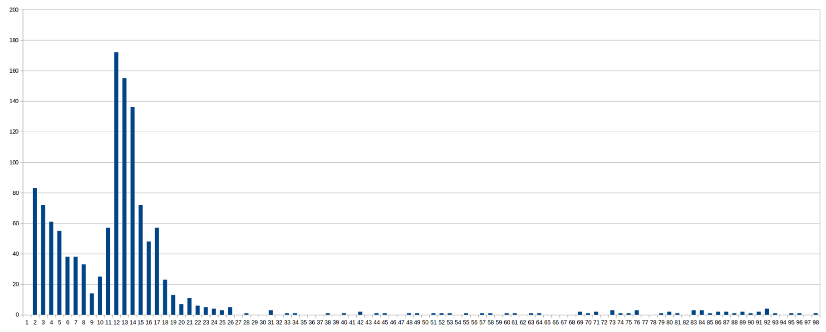

Figure 3 – Sampled (uniform law) degree distribution of the subnetwork

The two maxima have not the same importance in the total degree distribution and in the sampled one. Indeed, with the random walk, we spend statistically more time in the "core" of the network.

Question 6

Then, it's sufficient to implement the *monteCarlo* method :

```
# choose a node following the Monte Carlo rule
def monteCarlo(matrix,line):
       l = len(line)
       neighbourNumber = sum(nodeDegree(matrix,line))
       listProba = list()
       for i in range(0,l):
             listNodes = list()
             listNodes.append(line[i])
             p = float(nodeDegree(matrix,listNodes)[0])/float(neighbourNumber)
             listProba.append(p)
       normalisation = 0
       for i in range(0,l):
             listProba[i]=1./float(listProba[i])
             normalisation += listProba[i]
       for i in range(0,l):
             listProba[i] = float(listProba[i])/float(normalisation)
       for i in range(1,l):
             listProba[i]+=listProba[i-1]
       a = random.random()
       for i in range(0,l):
             if(a<listProba[i]):
                   return line[i]
       return line[l]
```

Then, we just have to call `randomWalk(ADJ,G,N0,400,monteCarlo)`, and we get the FIGURE 4.

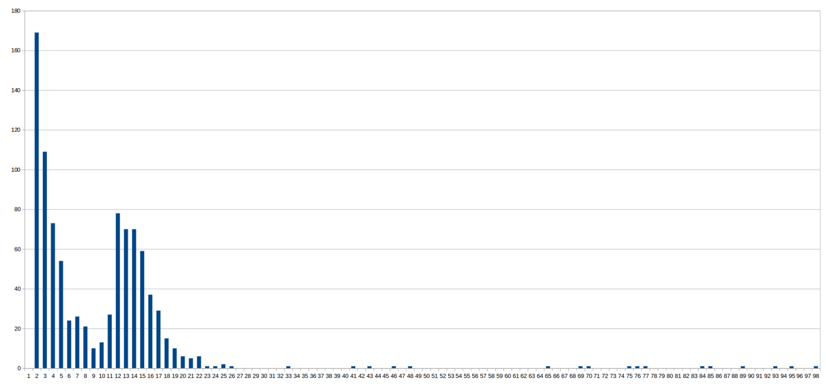

FIGURE 4 – Sampled (Monte Carlo method) degree distribution of the subnetwork

By forcing to explore the less dense areas of the network, the shape of the new sampled degree distribution is closer to the total one.

Question 7

Thanks to `networkx`, we have the Figure 5.

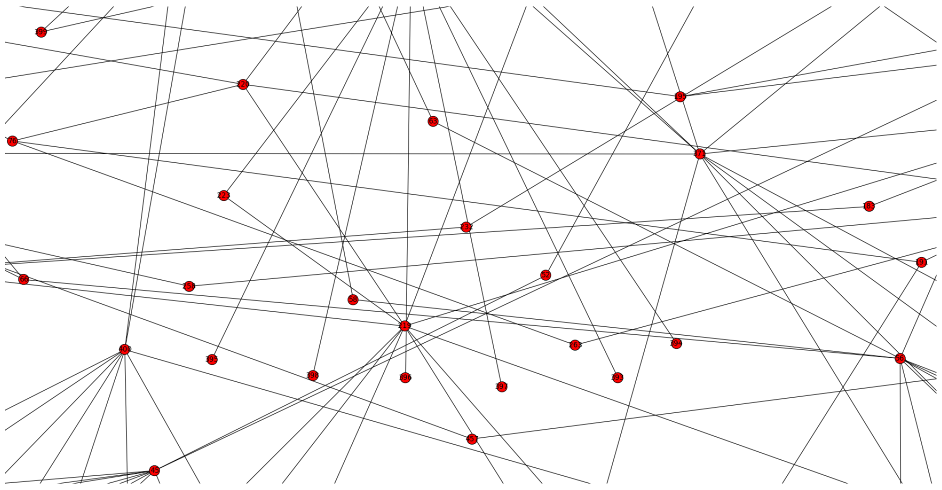

Figure 5 – Sample of the subnetwork

Question 8

The *diameter* of the subnetwork is $D_S = 13$.

We could pick randomly two nodes, compute the distance. If we iterate many times and we keep the maximum of the computed distances, we should converge to the diameter.

Question 9

The *average distance* of the subnetwork is 5.612, which is strictly greater than $log(500) = 2.699$. Thus, the subnetwork is not *small world*.

Question 10

The first eigenvalue is $\lambda_1 = 1$. The second one is ... .

The smaller is the second biggest eigenvalue (all smaller than 1), the quicker is the convergence to the stationary distribution.