EE 232E

Graphs and Network Flows

Homework 1 Report

Pingyuan Yue: 504737715

Ke Xu: 604761427

Yuanyi Ding: 404773978

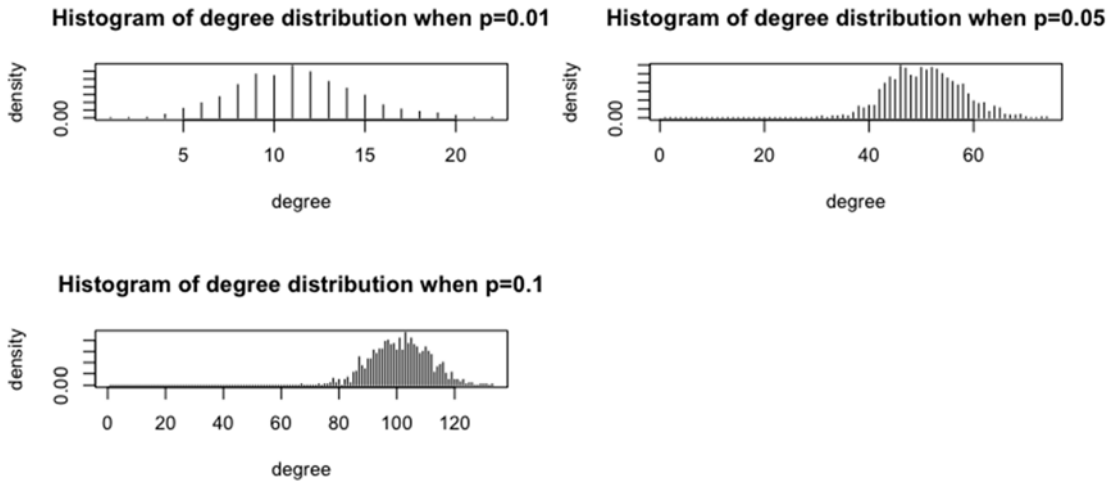
4/17/2017

Table of Contents

1. Create Random Networks

Part (a):

We are asked to create three undirected random networks with 1000 nodes, and the probability p for connecting two arbitrary vertices 0.01, 0.05 and 0.1 respectively. Below are three figures of degree distribution:



Figrue1.1: degree distribution for different value of p

We observed from the graphs above that the networks trend to have high degree if the probability p for connecting two arbitrary vertices is high.

Part (b):

Due to fact that our graphs are randomly generated, we decide to repeat the process for 100 times and compute the average for connectivity and diameter. The results are shown as following:

	P = 0.01	P = 0.05	P = 0.1
Connectivity	0.99	1	1
Diameters	5.34	3	3

Table 1.1: connectivity and diameters for different value of p

Part (c):

We are asked to find out a value of p_c, so that any networks we generated with a p that is smaller than p_c are disconnected, and any networks generated with a p that is larger than p_c are connected. Considering the results in part b, we decide to perform a binary search between 0 and 0.01 to find the p_c. The algorithm stops whenever the graph we generate is disconnected with possibility of (p - ε) and is connected with possibility of (p + ε). That value of p is the p_c we try to find. The p_c we find is 0.00728.

Part (d):

We are asked to derive the value of p_c analytically. We assume the with certain value of p, we expect the find one single isolated node. The expectation of finding this node can be represented as:

$$1 = N * (1-p) * (1-p) * \dots \dots * (1-p) = N * (1-p)^N, \text{ where N = 1000}$$

$$\downarrow$$

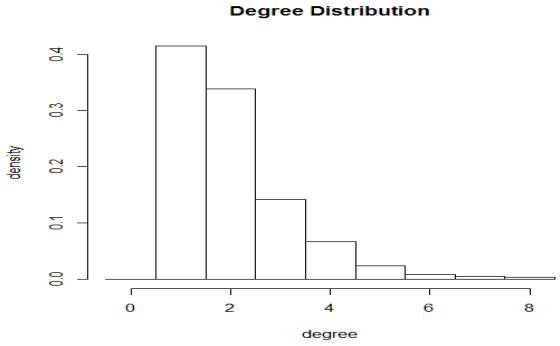
$$p = 1 - \left(\frac{1}{N}\right)^{\frac{1}{N}}$$

The p we calculate from the above equation is equal to 0.00688, which is close to what we get from part c.

2. Dataset and Problem Statement

Part (a):

Here we use function sample_pa to create the network and the diameter of the graph is 29.56 for which we create the network 100 times and take the mean value. The degree distribution is as following:



part (b):

Here we implement the function is.connected to test the connectivity of the network, which is always connected when we repeated for 100 times.

For GCC: Since the entire graph is connected and GCC is the largest connected component, so the GCC here is the whole graph.

Community Structure: We use the function cluster_fast_greedy to find the structure and use modularity() to get the modularity of the graph.

Number of communities: 32.

Modularity: 0.9345737.

Discussion: Why is it so large?

The modularity for the graph is 0.9345737. The reason that it has such a large modularity is because the graph is generated based on Barabasi model, which has preferential attachment mechanism, in which the new added nodes tend to be linked to the nodes with high degree. So it has dense connections between the nodes within certain communities but sparse connections between nodes in different communities. Thus the modularity is large for this graph.

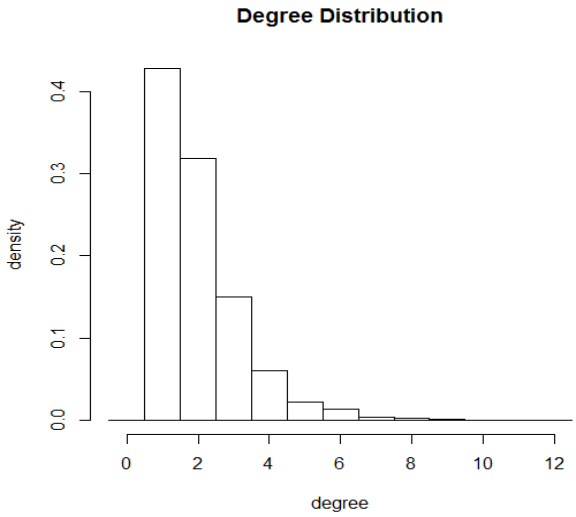
Part (c):

Also we use sample_pa to generate the network and the structure as well as modularity:

Number of communities: 105

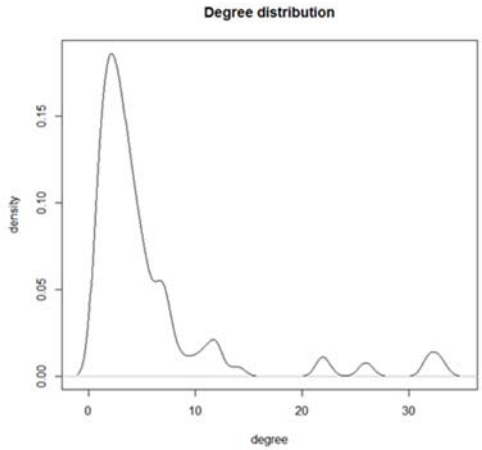
Modularity: 0.9791046

Obviously, the modularity is slightly larger than the 1000 nodes graph. For the graph which has 10000 nodes, degree distribution is as following:



Part (d):

Here we create the network based on function neighborhood() and the degree distribution of nodes j is as following:



3. Creates a Random Graph by Simulating Its Evolution

Part (a):

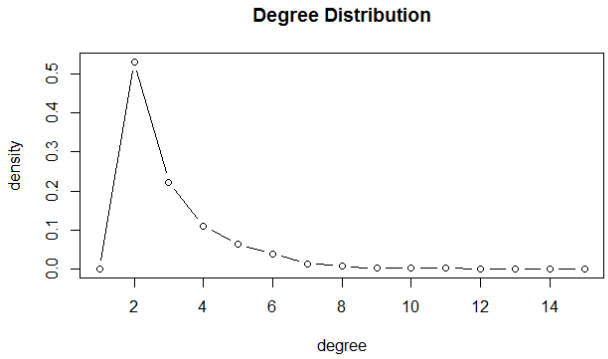
By definition, a preferential attachment process is any of a class of processes in which some quantity, typically some form of wealth or credit, is distributed among a number of individuals or

objects according to how much they already have, so that those who are already wealthy receive more than those who are not.

In this problem, we created a random graph with 1000 nodes by simulating its evolution. A new vertex is added and creates a number of links to old vertices and the probability that an old vertex is cited depends on its in-degree. We can simply simulate the evolution of the graph by calling the sample_pa_age function in igraph library.

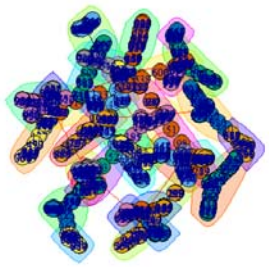
```
g <- sample_pa_age(1000, pa.exp=1, aging.exp=-1, aging.bin=1000, directed=FALSE)
```

The degree distribution of the graph is shown as follows:



Part (b):

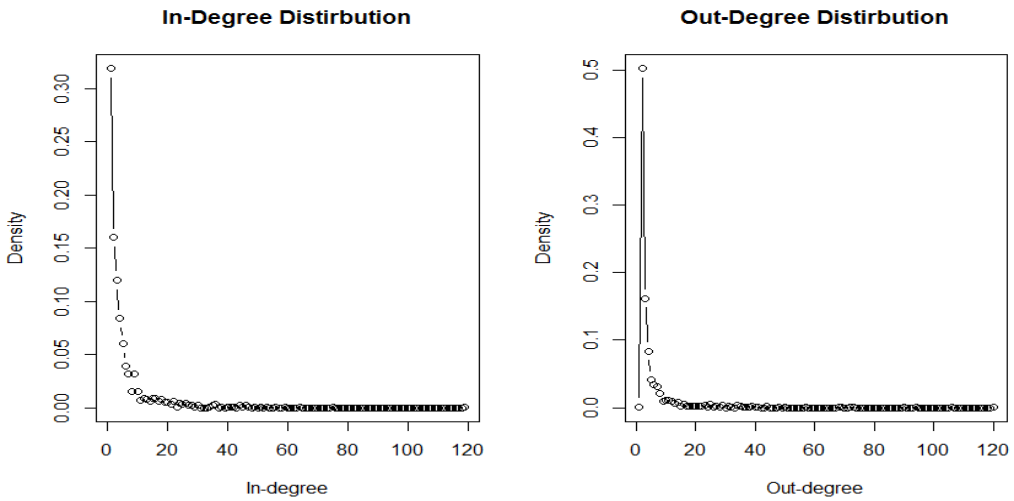
Use fast greedy method to find the community structure. We can find that the number of communities of this graph is around 32. The modularity is 0.93597. The structure of the graph is shown below:



4. Forest Fire Model

Part (a):

In this problem, we generate a forest fire model by using forest.fire.game function. The forward burning probability is set to be 0.37, and the backward burning probability is set to be 0.32/0.37 = 0.86. Both in-degree distribution and out-degree distribution of the graph are shown below. It could be concluded that both degree decrease quite fast.

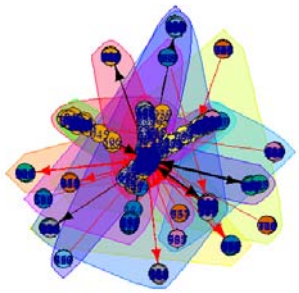


Part (b):

We generate the graph for 50 times and calculate the average diameter which is around: 10.56.

Part (c):

In this problem, the structure of the graph is calculated using spinglass method, since fast greedy method is only applicable to undirected graph. The number of communities is around 24 and the modularity of the graph is calculated to be 0.47009. The structure of the graph is shown below:



5. Conclusion

In this assignment, we learn the basic knowledge of several graph theories and models and get familiar with igraph library and R language programming. The main challenges in this assignment is to figure out the meaning of different models (e.g. fat-tailed distribution, preferential attachment, forest fire) find the correct method to generate different models.