Project 1: Letters Network

Abhishek Rao

## Table of Contents

1. Introduction
2. Overview of the code
3. Results
4. Analysis of Results
5. Future work
6. References
7. Code
8. Result Graphs

## Introduction

Here we have constructed a weighted directed network where the nodes are letters of alphabets. An input sentence is taken and an edge is formed between letters by parsing through the input sentence, forming an edge between letters if one letter comes after another in the given sentence. If it occurs again, the weight is incremented.

A code is written in python and NetworkX. The input sentence for this is “*hello how are you? Im fine. Hope you are fine too. Im going to college now*”

This network is weighted and directed network with weight equal to the number of occurrences of transition from character u to character v in the given input sentence.

## Overview of the Code

The program consists of following files.

1. **Charnet.py**  - The main file, calls the other two
2. **strfuncgraph.py**  - Contains most of the functions written for this project, specific to letter graph
3. **graph\_analyze\_mod\_dir.py** - Modified version of the graph analyzer written last week, a general graph analyzing module

Below a brief overview of the code is given. For more details please check the full python code given in the later part.

#### Open Source

The source code for the entire project is available online at [https://github.com/abhishekraok/NetworkX\_study](https://github.com/abhishekraok/networkx_study)

### File 1: Charnet.py

The main file of the code named “charnet.py” contains the functions to do the following.

1. Create a list of characters called 'alphalist' (string) and a sentence called 'chars' (string)
2. Create a Graph 'G' using this 'alphalist' as nodes and building edges using 'chars'
3. Analyze the created graph 'G' wih the analyze function written last week (with some modifications for directed neworks)
4. Get results

### File 2: strfuncgraph.py

#### String Creation *'createstr()'*

Let the input sentence be = InpS

let cp = last character of InpS

#### Graph Creation ‘*createstrGraph(alphalist,chars)’*

Create nodes named after alphalist,

for each character 'c' in chars

Create edge from (c,cp), if there already is an edge increase its weight

cp = c #Update the previous character to present

#### Displaying the graph ‘*drawstrGraph(G)*’

A custom code is written to display edge weights of the graph

### **File 3: graph\_analyze\_mod\_dir.py**

Same as last weeks except for clustering measurements the graph is converted into undirected graph.

## Results

Start of analyzing network

The number of nodes is 18 and the number of edges is 41

The average degree is 4.55555555556

The average shortest path length is 2.58496732026

The betweenness centrality is {'a': 0.11734068627450979, ' ': 0.0, 'c': 0.0, 'e': 0.37647058823529406, 'g': 0.07169117647058824, 'f': 0.042279411764705885, 'i': 0.17346813725490198, 'h': 0.09485294117647057, 'm': 0.022855392156862744, 'l': 0.03602941176470588, 'o': 0.5006127450980392, 'n': 0.13651960784313721, 'p': 0.018872549019607842, 'r': 0.1136642156862745, 'u': 0.05330882352941176, 't': 0.01715686274509804, 'w': 0.05330882352941176, 'y': 0.01715686274509804}

The clustering clustering coefficient is {'a': 0.0, ' ': 0.0, 'c': 0.0, 'e': 0.05555555555555555, 'g': 0.4, 'f': 0.3333333333333333, 'i': 0.3, 'h': 0.0, 'm': 0.3333333333333333, 'l': 0.0, 'o': 0.07272727272727272, 'n': 0.5, 'p': 0.0, 'r': 0.0, 'u': 0.3333333333333333, 't': 0.6666666666666666, 'w': 0.0, 'y': 0.0}

End of analyzing network

## Analysis of Results

Differences:

Note that all the distributions are given as histograms which we feel is more convenient for small number of node networks to get the number of nodes in that value. The distance distribution takes the distance to self too, so N 0 values are added.

The network has 18 nodes and 41 edges. The number of edges is more that what we have seen in the usual networks. So many cycles are to be expected.

Looking at the degree distribution it's hard to see any curve that can be fit in there since it's only few nodes.

{'a': 3, ' ': 1, 'c': 2, 'e': 13, 'g': 6, 'f': 3, 'i': 5, 'h': 5, 'm': 3, 'l': 6, 'o': 16, 'n': 5, 'p': 2, 'r': 2, 'u': 3, 't': 3, 'w': 2, 'y': 2}

There is a large number of nodes with degree 3-4. No nodes in the range of 7 - 12 and then there is one at 13 and one at 16. This suggests most characters occur few times except special characters which occur many times. We see that as expected the vowels have higher degree, e has 13 and o has 16. The consonants occur less frequently.

The average shortest path is 2.6. This is to be expected as the number of edges is high.

Looking at the betweenness centrality, most are less than 0.2 except ‘o’ and ‘e’ with 0.5 and 0.3. This indicates that ‘o’ and ‘e’ are visited several times during the sentence formation. ‘o’ and ‘e’ are like hubs of network.

Looking at the clustering co-efficients some 10 are 0 indicating that they are on the spokes of network. Unlike previously ‘o’ and ‘e’ do not have high clustering values, instead nodes which are connected to both like ‘t’ and ‘n’ have high clustering.

Comparision with real networks:

Comparing this to the real networks which have power law degree distributions, constant clustering and logarithmic distances we see here that the degree distributions are somewhat similar in the sense that most of the values are in the lower region. There are few in the very high region and none in the mid region which is different from real networks. Clustering distribution is not constant. Distance distribution does not follow logarithmic distribution but rather looks like normal distribution.

## Future work

Using this network it seems that one should be able to construct words or characters that sound similar to the given input sentence. Using markov chain from the network constructed, we create a string by traversing through the network with next node chosen based on the probability equal to the weight of that edge. Using this we can also try to understand unknown language like the one used to characterize Bengalese finch bird songs [1].

So given an English input sentence the output should sound similar to English and given a French input sentence the output should sound similar to French.

## References

1. A Compact Statistical Model of the Song Syntax in Bengalese Finch - Dezhe Z. Jin\*, Alexay A. Kozhevnikov - Department of Physics, The Pennsylvania State University - <http://users.phys.psu.edu/~djin/PDFs/JinKozhevnikovPLoSComp2011.pdf>