Neighbourhood approximation using ANF

Social Network Analysis for Computer Scientists — Course Project Paper

Atish Kulkarni
LIACS, Leiden University
s2483122@umail.leidenuniv.nl

Tushar Mandloi
LIACS, Leiden University
s2502585@umail.leidenuniv.nl

ABSTRACT

In today's world graphs can be seen in almost all the technical and non technical domains. With increasing dimensionality of graphs there is need of a method for mining these graphs which is accurate and fast. The authors came up with a fast data mining tool called *approximate neighbourhood function*(ANF), which uses neighbourhood approximation. This tool solves the time complexity problem present in large scale graph mining. To achieve this authors have used a bit-based calculation algorithm which helps in fast and accurate approximation making it 700 times faster than the exhaustive computation. In this paper we discuss about algorithm proposed by authors and compare similar neighborhood approximation methods which help in fast similarity measurement.

General Terms

Graph Theory, Nearest neighbors, social networks

Keywords

Neighborhood approximation, LSH, graph similarity

1. INTRODUCTION

~~In today's world~~ with huge amount of data being produced every day, there's a need for a method to organise it in such a way that if a query is to be executed one can find what they are looking for within a small amount of time. Any binary relational data set can be represented using a graph and by using the graph properties and methods various queries can be performed. Lets take ~~an~~ example ~~of~~ the Internet, where a computer, or a web page can be a node and the edge ~~will~~ represent the connection (network/hyperlink) between these computers/web pages [1]. Search engines such as "Google" make use of this approach for finding the most nodes in the graph [2].

Another example for graph based methods is citation graphs, where each node is a publication and each citation is an edge linking one publication with another and using this we can find the most important publications [3]. The graph based methods can be applied for mining information out of these graphs as well, let's say if a CAD system in circuit design is considered as a graph then finding the common components, point of failures etc becomes the data mining task.

In a graph, in order to look for similar nodes and to find patterns, we have to look at the neighborhood structure of these node. Lets look at some of the data mining questions, which can be answered using neighborhood properties.

1. Is the Internet of two different locations/countries similar?
2. Does the structure of the Internet hierarchical?
3. Are patterns of phone calls in two countries similar?
4. Comparing a new design with the patented designs to check its similarity?
5. What are the most influential publications in a certain field?
6. Closing of which streets would have the least affect on traffic?
7. What would be the best opening in the game of tic-tac-toe?
8. Clustering of the movie genres.
9. What are the chances of Internet failures?

The questions above can be answered using graph properties such as **Graph Similarity**, **Subgraph similarity** and **Vertex Importance**. You can answer ~~the~~ question 3, 4 and 9 using the graph similarity, which compares the connectivity of two graphs based on their neighborhood structure. ~~The~~ questions 1, 3, 7, and 8 can be answered using subgraph similarity, which compares how ~~these~~ two subgraphs are connected in the graph. The vertex importance can be used to answer the questions 2, 5, and 6, where each node is assigned an importance value based on their connectivity.

~~The authors of the paper ANF [1]~~ tried to answer ~~one question from each type~~ using the Neighborhood function. ~~For answering the~~ questions, authors designed a tool which ~~can be used to compute~~ the three graph properties i.e. graph

SNACS '20 Social Network Analysis for Computer Scientists, Master CS, Leiden University (liacs.leidenuniv.nl/~takesfw/SNACS).

similarity, subgraph similarity and vertex importance and the tool was called *Approximate Neighborhood Function*. It has set of properties such as it is **fast**, because it scales linearly with number of nodes and edges. It has a **low storage requirements**, is **parallelizable**, instead of random access uses **sequential access** of the edge file. It also provides **estimates per nodes**, and an **highly-accurate estimates** [1].

In the recent years, numerous approaches for approximation of nearest neighbors (ANN) have been developed. This approximation is required as the exhaustive search is computationally expensive with increased dimensionality. These approximation reduce the time complexity from $O(n^{2.38})$ to $O((n+m)d)$, where n is the number of nodes, m is the number of edges and d is a constant with a small value [1, 4, 5]. ANN are being used in Visual searches, recommender systems, etc [4, 5]. These methods make use of **vector transformation**, **vector encoding** and **avoids** exhaustive search [4]. These methods differ from each other majorly in the way they perform vector encoding by using LSH, Quantization or Trees [5], we discuss these approaches in detail in section 2.

In this paper we compare ANF and one of the nearest neighbor approximation method which uses LSH presented in [5] and [6]. For ANF we will be using `SNAP`, a python package, which is a high performance system for analysing and manipulating large networks. It is written using C++ but is optimized for giving maximum performance and compact representation of graphs. For LSH discussed in the papers mentioned does not have a open source implementation, therefore we attempted to have our own implementation. For both the methods we will look at the speed and accuracy of both the methods by comparing them with actual computation. In section 3, the algorithms and the datasets selected for this study are described. In section 4, we present the experiments performed and discuss the results. In section 5 we conclude the paper.

2. RELATED WORK

Since computing exact neighborhood for each node is computationally infeasible with increased dimensionality, approximation methods were developed. The ANN methods reduces the time complexity by preprocessing the data using an efficient index by performing the following techniques [4]

- **Vector Transformation**: It's applied to the vector for dimensionality reduction and vector rotation
- **Vector Encoding**: It constructs the actual index for faster query execution. Most used vector encoding methods are LSH, Quantization, Trees etc. further compacting the data structure
- **None Exhaustive Search Component**: This is applied to avoid exhaustive searches

Tree based algorithm constructs a forest i.e. many trees, by randomly selecting two points and splitting the space into two hyperplane and repeating until certain size of cluster is reached [4, 5]. Implementation of this technique can be seen in Spotify's Annoy algorithm which can be found here.

Quantization based algorithms reduce the dataset size by replacing every vector with k-centroid [4]. Whereas, LSH constructs a hash table as a data structure and maps nearby points in same buckets. *FAISS* from facebook is one of the implementations of LSH and can be found here [4, 5]. The paper, *An Investigation of Practical Approximate Nearest Neighbor Algorithms* [5], compared all the three methods i.e. LSH, Quantization and Trees, and found that LSH was comparatively better.

ANF was extended by HyperANF [7] which uses *HyperLoLog* counters in combination with broadword programming. HyperANF was proved to be faster and accurate than original ANF. Another distributed implementation of ANF using map-reduce called HADI, could compute neighborhood function for a graph with 2 billion links in half an hour [8]. However HyperANF can compute this in just 15 mins, making HADI faster than ANF but not with HyperANF [7]. We will only be using ANF as the main focus of the paper.

3. ALGORITHM

Here, we explain ANF and LSH, two algorithms we will be comparing in the paper.

3.1 ANF

Graph traversal of the edge file can be computationally complex hence authors iterate over the edge set instead. However this is still time consuming approach hence authors used bit masking membership, where each node is given one of `n` bits and a set of length `n` which is a bit string. Union of two sets is taken using bitwise-OR operator in order to add a node to the set [1]. To further reduce the time complexity, k parallel approximations are performed by considering `M(x,h)` as a bit string, `k(log n+r)` bits long. `M(x,h)` is the set of nodes which are neighbors of `x` with a distance `h`.

Algorithm 1: Approximate neighborhood function [1]

for *each node* x **do**
  if $x \in C$ **then**
    Mcur(x) = concatenate k bitmasks each with 1 bit set (P(bit i) = .5i+1)
  end
end
for *each distance from 1 to* h **do**
  for *each node* x **do**
    Mlast(x) = Mcur(x)
  end
  for *edge* (x, y) **do**
    Mcur(x) = (Mcur(x) BITWISE-OR Mlast(y))
  end
  for *each node* x **do**
    $I\hat{N}^{+}(x,h,C) = (2b)/.77351$ where b is the average position of the least zero bits in the k bitmasks
  end
  $\hat{N}^{+}(h,S,C) = \sum_{x \in S} I\hat{N}(x,h,C)$
end

The ANF algorithm can be seen above (Algorithm 1), where `Mcur` and `Mlast` are the `M(x,h)` and `M(y,h-1)` for `h` respec-

tively. This helps in reducing the memory usage by the algorithm. S, and C are the starting nodes and the concluding nodes respectively. The algorithm approximates the individual neighbourhood of each node using $2b/0.77351$ where b is the bit positions in the bit string. This method ~~have~~ a run time O((n+m)d), and requires storage for only M_{curr} and M_{last} which allows ANF to adapt to available memory.

3.2 Locality sensitive hashing (LSH)

Creating a N*N sparse matrix representing node to node relation would be a very large size for large graphs and makes iterative row processing inefficient and computationally expensive. For numerous data mining problem dealing with approximation make use of Locality sensitive hashing (LSH). It is the best suited algorithm when the data sets are high dimensional and large in size [5, 6].

In this paper we apply similar concept of LSH which uses minhash. The key idea behind this combination is that to use a hash function that tells us whether x and y are a candidate pair or not by picking a threshold value s, and if similarity is more than s then they are candidate pair. This threshold can be set as a float value between 0 to 1. LSH has three main steps when it comes to similarity measurement. These steps are as follows-

1. **Shingling:** In this step each user or node is represented as set of elements. In case of graph this is adjacency list of every edge existing for a particular node
2. **Min-Hashing:** Min hashing helps to convert large shingle set into hashes or signatures which saves the similarity between nodes. Key idea behind min-hashing is that if we hash every column to a tiny signature such that the similarity is preserved and if the similarity is high for two columns then there is a high probability of both the columns should fall into same bucket. And if the similarity of two columns is low then there is high probability of both the columns shouldn't fall into same bucket
3. **Locality sensitive hashing:** Using the signatures goal is to find nodes that are most likely to be above threshold s. If These candidate pairs are falling into same hash in more than one band i.e. are similar, then pairs are selected and their jaccard similarity is calculated.

4. DATASETS

In this paper we use two different type of data sets. The first data set is Movie lens data set and second data set is Enron email records.

4.1 Movielens Data

In this data set every user and movie act as a node and link exists if the user has watched a movie. In this type of structure a user can be connected to multiple movies but a movie can not be connected to other movies.

This data set is very similar to ~~data set~~ used ~~by authors of ANF~~, ~~which is~~ movie-actor data set ~~and link exits~~ if actor ~~appears~~ in a movie. With large number of users this data set is perfect to test the node similarity in large graphs. Further more if k-clustering is applied to this data set then graph similarity of sub components of graph can also be checked.

This data set has 65 million edges and about 100K unique nodes. With such a large size this data set serves as a good problem statement for similarity measurement. Graph generated with this data set will be un-directed. For a user if information like how many times a movie has been watched is available then weight can be also associated but this information is not currently available. Due to large number of edges Gephi is unable to load this data set hence visualisation has not been successful.

4.2 Enron email data

Enron was the leading energy company and due to a financial fraud an investigation was conducted on this company and all the emails were made available for research.

This data set has more than 30 features but for this scope of work we do not need any information more than From, To, Content. This data set is spread over 6 years and needs to be pre-processed. Edges of this data set are less than the movielens data set hence visualization is possible for a subset of the data. Depending on From and To features directed graph can be created for this data set.

5. EXPERIMENTS

In this section we will ~~be comparing~~ ANF and LSH based on accuracy and speed.

5.1 Exploratory data analysis

We performed exploratory data analysis on both the data sets. Movie lens data has a total of 103703 nodes and 65225506 edges. Due to these many edges it is way too big for Gephi to process, (even 1% of the dataset is still big for gephi to process). The Enron data set has a total of 41915 nodes and 148444 edges and the density of the network is 0.000084. In the figure 1 and 2 we show the network using Gephi. In table ?? we see the different properties of these networks, average clustering coefficient for user movie network is missing for now but will add it in the final report.

Properties	User-Movie	Enron
Nodes	103703	41915
Edges	65225506	148444
Density	0.006065	0.000084
Strongly connected (SCC)	85944	37206
Largest SCC, nodes	17760	4686
Largest SCC, edges	38167136	98731
Weakly connected (WCC)	1	533
Largest WCC, nodes	103703	40623
Largest WCC, edges	65225506	147490
Average clustering coefficient	-	0.1012

Table 1: Properties of the networks

In figure 3, we see the in and out degree distribution of both the networks.

5.2 LSH Experiments

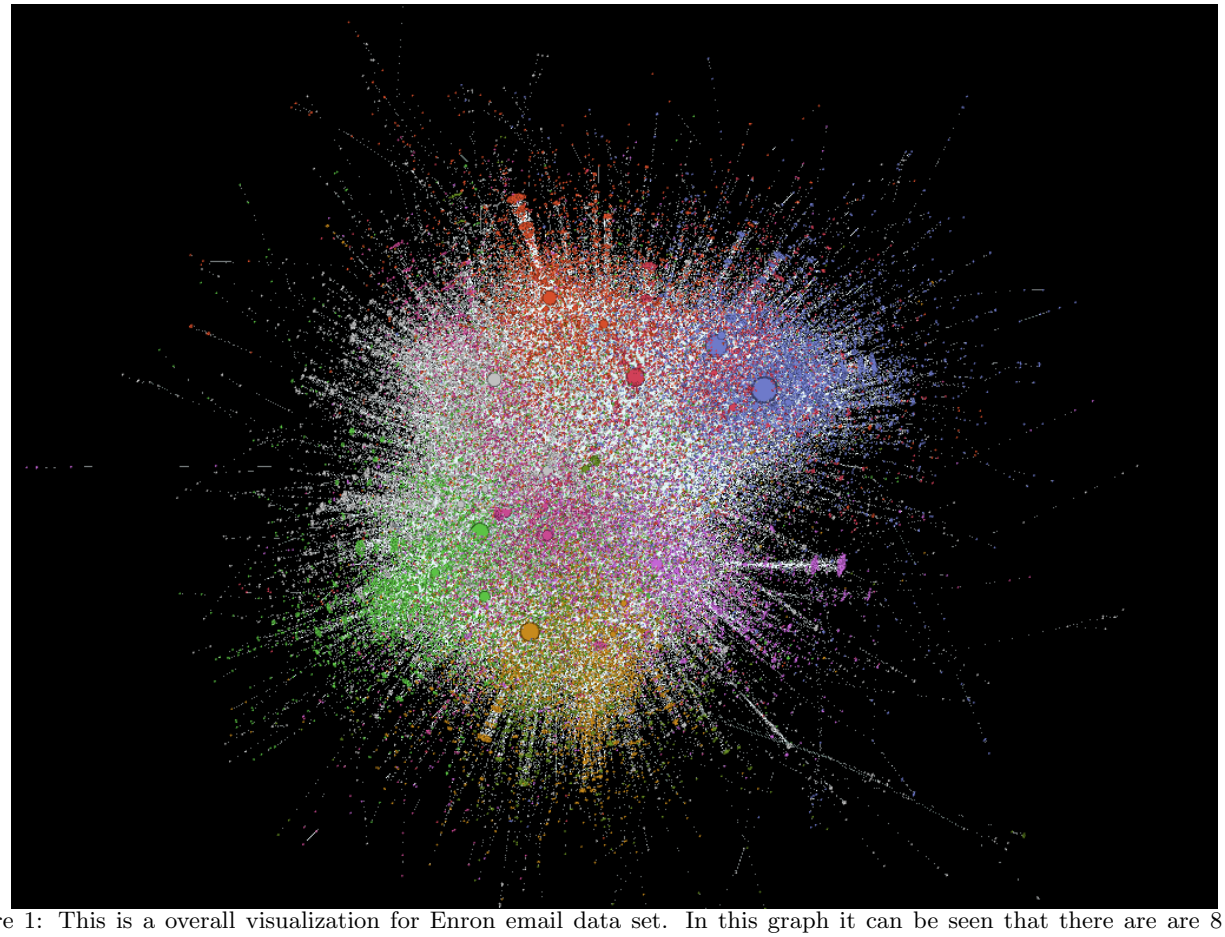


Figure 1: This is a overall visualization for Enron email data set. In this graph it can be seen that there are are 8 clear communities. These communities are obtained using algorithm [9]. Resolution value used is 3 and modularity score obtained is 0.67. Size of nodes was set based on degree of the node to get overall idea of email traffic at a node. From the graph it can be seen that in every community there is one large sized node representing manager or an important position holder for that community.

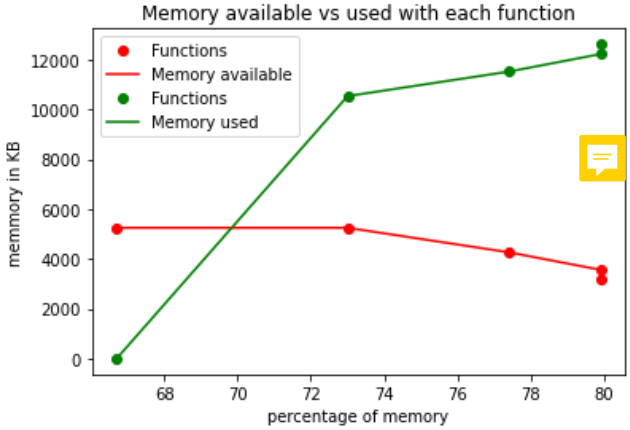


Figure 4: This image shows the change in memory plot for LSH algorithm for every step in algorithm. Even with large size of edges totaling to 65 million use of LSH algorithm allows to processes such a graph on local system consuming very low memory when compare to other methods.

For the LSH experiments similarity between every user will be calculated. Based on the number of pairs found and the total time taken quality of the algorithm and hyper parameters can be tested. The main performance metric

5.3 ANF experiments

This section is incomplete.

5.4 Comparison experiments

This section is incomplete.

6. CONCLUSION

This section is incomplete.

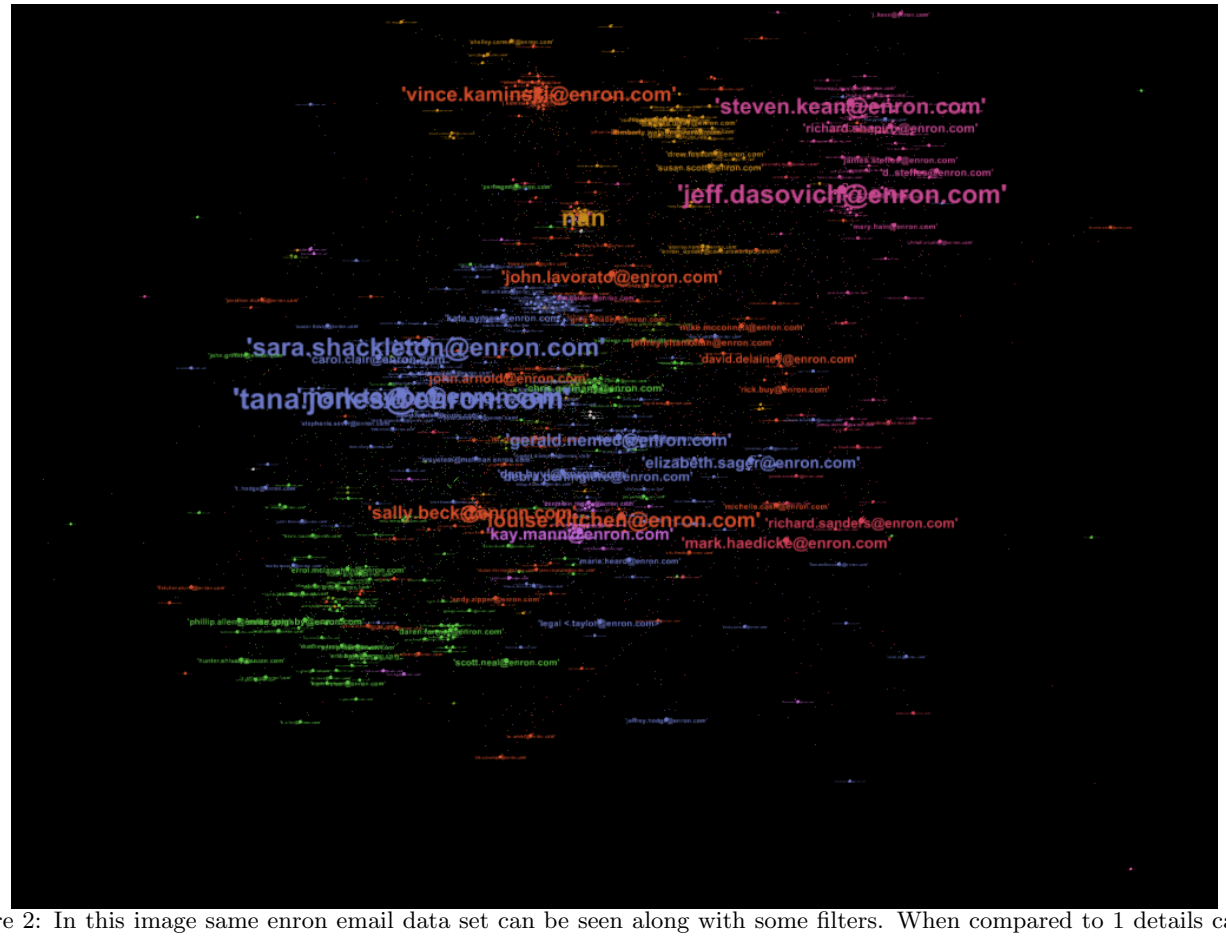


Figure 2: In this image same enron email data set can be seen along with some filters. When compared to 1 details can be seen better in this image. For this image giant component filter was set along with range filter for degree. From the filtered results names "jeff dsovich", tana jones", "sara shackelton" have the highest degree. It is surprising to see Employee list does not have any record for two of these names but have record for all other employees.

References

[1] Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos. "ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs". In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '02. Edmonton, Alberta, Canada: Association for Computing Machinery, 2002, pp. 81–90. ISBN: 158113567X. DOI: 10.1145/775047.775059.

[2] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30.1 (1998). Proceedings of the Seventh International World Wide Web Conference, pp. 107–117. ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00110-X.

[3] Gerard Salton and Michael McGill. *Introduction to modern information retrieval*. New York, NY: McGraw-Hill, 1983.

[4] Eyal Trabelsi. *Comprehensive Guide To Approximate Nearest Neighbors Algorithms*. Feb. 2014. URL: https://towardsdatascience.com/comprehensive-guide-to-approximate-nearest-neighbors-algorithms-8b94f057d6b6.

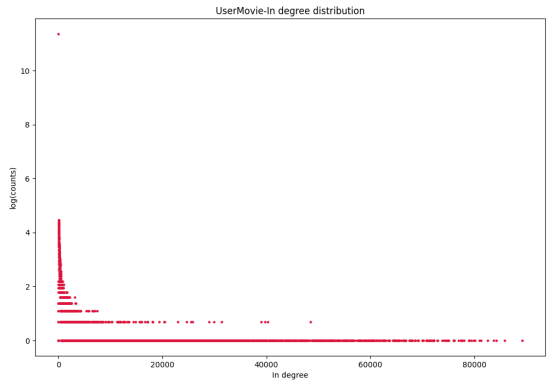
[5] Ting Liu et al. "An Investigation of Practical Approximate Nearest Neighbor Algorithms". In: *Proceedings of the 17th International Conference on Neural Information Processing Systems*. NIPS'04. Vancouver, British Columbia, Canada: MIT Press, 2004, pp. 825–832.

[6] Wen Li et al. "Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement (v1.0)". In: *CoRR* abs/1610.02455 (2016). arXiv: 1610.02455. URL: http://arxiv.org/abs/1610.02455.

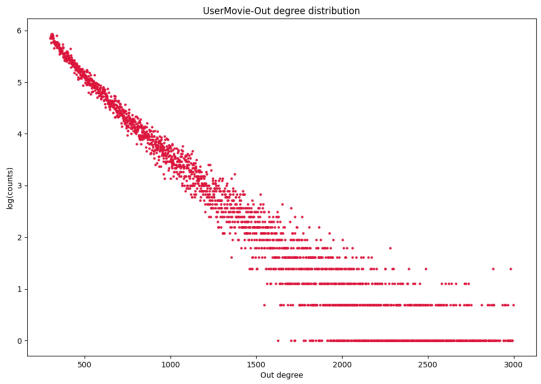
[7] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. "HyperANF: Approximating the Neighbourhood Function of Very Large Graphs on a Budget". In: *CoRR* abs/1011.5599 (2010). arXiv: 1011.5599.

[8] U. Kang et al. "HADI: Mining Radii of Large Graphs". In: *ACM Trans. Knowl. Discov. Data* 5.2 (Feb. 2011). ISSN: 1556-4681. DOI: 10.1145/1921632.1921634.

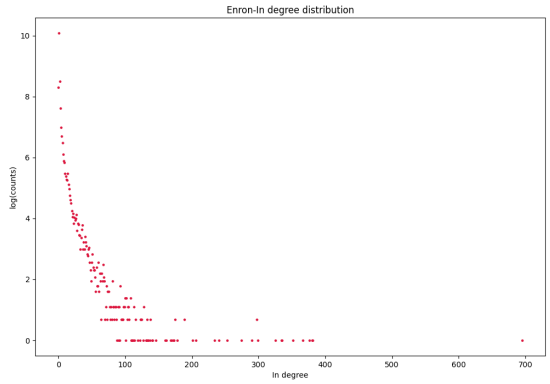
[9] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (Oct. 2008), P10008. DOI: 10.1088/1742-5468/2008/10/p10008. URL: https://doi.org/10.1088%2F1742-5468%2F2008%2F10%2Fp10008.



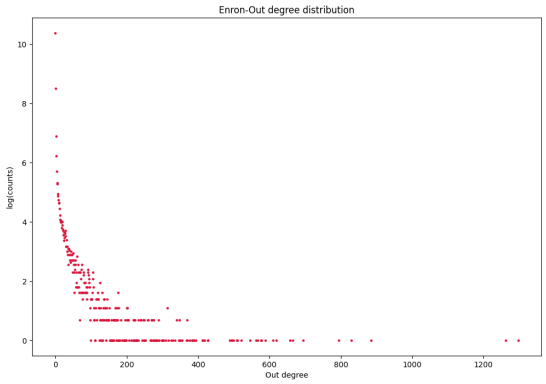
(a) User-movie In degree



(b) User-movie Out degree



(c) Enron In degree



(d) Enron Out degree

Figure 3: In and Out degree distribution of both the networks

APPENDIX

A. SOURCE CODE

Below you see the source code developed during the project work.

```
## Required packages
import glob
import pandas as pd
import os
from wordcloud import WordCloud
import numpy as np
import nltk
from sklearn.decomposition import
    LatentDirichletAllocation as LDA
from sklearn.feature_extraction.text import
    CountVectorizer
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from keras.models import Model
from keras.utils.vis_utils import plot_model
import tensorflow as tf
from keras.layers import Conv2D, Flatten, Dense,
    Activation, MaxPooling2D, Dropout, Input,
    BatchNormalization
import keras
import networkx as nx
import matplotlib.pyplot as plt
from math import log
from random import sample
from collections import Counter
import itertools

## Function definitions
def Plot(deg, cnt, ptype, title, ylab, xlab):
    plt.figure(figsize=(12,8))
    if ptype == 'bar':
        plt.bar(deg, cnt, color='crimson', width=0.5)
    elif ptype == 'scatter':
        plt.scatter(deg, cnt, color='crimson', s=5)
    plt.title(str(title))
    plt.ylabel(ylab)
    plt.xlabel(xlab)
    plt.savefig(str(title + ptype+'.png'))
    plt.show()


def NetworkX(net, filename):

    ## Network information
    edges = net.number_of_edges()
    nodes = net.number_of_nodes()
    print("Number of edges: %d\nNumber of nodes: %d"
        %(edges, nodes))
```

```
    print("Density of the network is %f" %
          nx.density(net))
    ## Outdegree distribution
    dg = Counter(sorted([n[1] for n in
          net.out_degree()], reverse=True))
    d, c = zip(*dg.items())
    Plot(d, [log(n) for n in c], 'scatter',
          str(filename + "-Out degree distribution"),
          'log(counts)', 'Out degree')

    dg = Counter(sorted([n[1] for n in
          net.in_degree()], reverse=True))
    d, c = zip(*dg.items())
    Plot(d, [log(n) for n in c], 'scatter',
          str(filename + "-In degree distribution"),
          'log(counts)', 'In degree')

    ## Strongly connected components
    sc = [len(n) for n in
          sorted(nx.strongly_connected_components(net))]
    sc_nodes = [n for n in
          sorted(nx.strongly_connected_components(net))]
    sc_deg = net.degree(sc_nodes[sc.index(max(sc))])
    sc_deg = [n[1] for n in sc_deg]

    print("Number of strongly connected components:
          %d\nNodes in the largest strongly connected
          components: %d\nNumber of links in largest
          strongly connected components: %d" %(len(sc),
          max(sc), sum(sc_deg)/2))

    ## Weakly connected components
    wc = [len(n) for n in
          sorted(nx.weakly_connected_components(net))]
    wc_nodes = [n for n in
          sorted(nx.weakly_connected_components(net))]
    wc_deg = net.degree(wc_nodes[wc.index(max(wc))])
    wc_deg = [n[1] for n in wc_deg]

    print("Number of weakly connected components:
          %d\nNodes in the largest weakly connected
          components: %d\nNumber of links in largest
          weakly connected components: %d" %(len(wc),
          max(wc), sum(wc_deg)/2))

    ## Clustering coefficient
    print("Average clustering coefficient for the
          network: %.4f" % nx.average_clustering(net))

## EDA - User movie
um = pd.DataFrame(np.load('user_movie.npy'),
     columns=['source', 'target'])
net = nx.from_pandas_edgelist(um, 'source', 'target',
     create_using=nx.DiGraph())
NetworkX(net, 'UserMovie')


# EDA ENRON

## Preprocessing the enron data set
path = 'SNACS'
os.chdir(path)

## Merging multiple files into a single dataframe
data = pd.DataFrame()
for file_name in glob.glob(path+'*.csv'):
    temp_data = pd.read_csv(file_name)
    data = data.append\
        (temp_data.sample(frac=0.1, replace=True,
             random_state=1))

data.to_csv('Enron.csv')
data = pd.read_csv('Enron.csv')
data['content'] = data['content'].astype(str)

## Sampling the data set in order to read into Gephi
data_sample = data.iloc[:10000, :]
data_sample = data_sample[['from', 'to', 'content']]
data_sample = data_sample.reset_index()
data[['from', 'to', 'content']] = data[['from', 'to',
     'content']].astype(str)
data_sample = data_sample.fillna('0')
content = []
origin = []
to = []
## Preprocessing the data set
data_sample_preprocessed = pd.DataFrame()
for i in range(len(data_sample)):
    data_sample['to'][i] =
          data_sample['to'][i].replace('frozenset({', '')
    data_sample['to'][i] =
          data_sample['to'][i].replace('})', '')
    data_sample['from'][i] =
          data_sample['from'][i].replace('frozenset({',
          '')
    data_sample['from'][i] =
          data_sample['from'][i].replace('})', '')
    content.append(data_sample['content'][i].split())
    to.append(data_sample['to'][i].split())
    origin.append(data_sample['from'][i].split())
    if len(content[-1]) < 2500:
        data_sample_preprocessed =
            data_sample_preprocessed.append\
           ([content[i] + to[i] + origin[i]])

data_sample.iloc[:, 1:3].to_csv('enron_gephi.csv',
     index=False)
en =
     pd.read_csv('Enron.csv').iloc[:,[2,3]].replace('frozenset\(\{
     regex=True).replace('\}\)','', regex=True)
en.to = en.to.str.split(', ')
en =
     en.explode('to',ignore_index=True).reset_index(drop=True)
net = nx.from_pandas_edgelist(en,'from', 'to',
     create_using=nx.DiGraph())
NetworkX(net, 'Enron')
```