Neighbourhood approximation competition

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 dimensions of graphs there is need of a method for mining these graphs which is accurate and fast. In paper [1] a fast data mining tool called *approximate neighbourhood function*(ANF), which uses neighbourhood approximation was contributed. This tool solves the time complexity problem present in large scale graph neighbourhood detection. 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. Information obtained with this method is used to calculate similarity between nodes, sub graphs and also find out other important statistical aspects of graph. Similar to this tool LSH(Locality sensitive hashing) is one more method which is used to find nearest neighbours and calculate similarity between nodes. In this paper we discuss about algorithms used in these two methods and compare similar neighborhood approximation methods which help in fast similarity measurement. With this comparison it was found that *ANF* is less memory efficient than *LSH* but is also a bit faster for approximate neighbourhood computation.

Keywords

Neighborhood approximation, LSH, graph similarity

1. INTRODUCTION

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. For example the Internet, where a computer, or a web page can be a node and the edge represents the connection (network/hyperlink) between these computers/web pages [1]. Search engines such as "Google" handle data stored as a graph and use graph mining techniques for finding the most relevant nodes in the graph [2]. Another example for this type of data representation is citation graphs, where each node is a publication and each citation is an edge linking one publication with another. Setting up citation graph data in such a way helps to find important publications [3] using graph mining techniques.

In a graph, in order to look for similar sub-graphs, or nodes and to find patterns, neighborhood structure of the node has to be considered. Some of the real world questions listed below, can be answered based on the neighborhood structure.

1. is structure of twitter network same to a company email network
2. How similar are two users in twitter network?
3. Which movies have most similar audience?
4. Based on movies watch how similar is watching pattern for two users?
5. For a email address how many nodes are within certain distance?
6. Who is most influential in a company network?
7. Which two departments are most similar in a company network?

These questions can be answered using graph properties such as Graph Similarity, subgraph similarity and Vertex Importance. You can answer question 1 using the graph similarity, which compares the connectivity of two graphs based on their neighborhood structure. The questions 2, 3, 4 and 7 can be answered using subgraph similarity, which compares how well these two subgraphs are connected in the parent graph. The vertex importance can be used to answer the question 6. Where each node is assigned an importance value based on their connectivity. Question 5 is the general graph mining question which is about finding out which node has highest number of similarity with other nodes.

Traditional approaches such as Breadth-first search for computing these graph properties are computationally expensive as it drills down on every node and edge. Moreover popular packages like `Networkx` with combination to python are very helpful to process graphs but as the size of the graph

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

increases use of such package gets limited due to the time and memory consumed. Therefore there is a need for faster computation of graph properties and various approximation approaches have been developed such as Approximate Neighborhood function (ANF)[1], approximation of nearest neighbors (ANN) [4] etc. These approximation are computationally inexpensive with increased dimensionality and 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, 5, 4].

In [1] authors tried to answer one question from each type by approximating Neighborhood function. Their algorithm can compute the above mentioned graph properties (i.e. graph similarity, subgraph similarity and vertex importance) efficiently, and the algorithm is 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 *accurate estimates* [1]. On the other hand, ANN are being used in Visual searches, recommender systems, etc [5, 4]. These methods make use of *vector transformation*, *vector encoding* and *avoids* exhaustive search [5]. These methods differ from each other majorly in the way they perform vector encoding by using LSH, Quantization or Trees [4], we discuss these approaches in detail in section 2.

In this paper we study two different graph mining methods based on ANF and LSH. We compare these methods for 5 different data sets allowing us to study their time and memory complexity for different graphs. With this comparison of ANF and LSH we find out the relation between time, memory complexity and size of the graph. In [1] it is discussed that algorithm adopts to memory available and LSH is a method which uses sparse data frames in order to make processing efficient and fast. Given this similarity we aim to test adaptability as well. For ANF we used `SNAP`, a python package, which is a high performance system for analysing and manipulating large networks [6]. It is written using C++ but is optimized for giving maximum performance and compact representation of graphs. For LSH we use `Nearpy` a python package, which uses LSH vector encoding. 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 large size of nodes and edges, approximation methods were developed. The ANN methods reduces the time complexity by pre processing the data using an efficient index by the following techniques [5]

- *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 [5, 4]. Implementation of this technique can be seen in Spotify's Annoy algorithm. Quantization based algorithms reduce the dataset size by replacing every vector with k-centroid [5]. 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 [5, 4]. The paper, *An Investigation of Practical Approximate Nearest Neighbor Algorithms* [4], compared all 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]. These algorithms are implemented in newer languages like *Rust*. For the scope of this paper we will be only be using ANF as the main focus of the comparison.

3. ALGORITHM

This section explains ANF and LSH, two algorithms which will be compared in the paper.

3.1 ANF

As the exact computation, using breadth first search access the edge file randomly making it computationally complex hence [1] iterate over the edge set instead. However this is still time consuming approach hence bit masking membership is used, 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 bit-wise-OR operator in order to add a node to the set. 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`.

The ANF algorithm can be seen in Algorithm 1, where `Mcur` and `Mlast` are the `M(x,h)` and `M(y,h-1)` for `h` respectively which 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 `Mcurr` and `Mlast` which allows ANF to adapt to available memory.

Example: Figure 1 shows a undirected cycle and the table 1 the bitmasks and approximations for the cycle using ANF algorithm. Here we use a bitmask of size 3 ($k = 3$). In the first `FOR` loop, the algorithm produces bitmask for *M(x, 0)*. Then in the next iteration `OR` operator is applied to the

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 M$last$(y))
  end
  for *each node* x **do**
    $I\hat{N}^{+}(x, h, C) = (2^{b})/.77351$ 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

nodes reachable in *h-1* steps from the neighboring nodes. For example, for *M(2,1)*, `OR` operator is applied to *M(2,1)* OR *M(1,1)* OR *M(3,1)* since 1 and 3 are the neighbors. Then IN(2,1) is computed using $b = 4/3$ (since the average least zero bit position are 2, 1, 1). Therefore $IN(2, 1) = b^{4/3}/0.77359 = 3.25$.

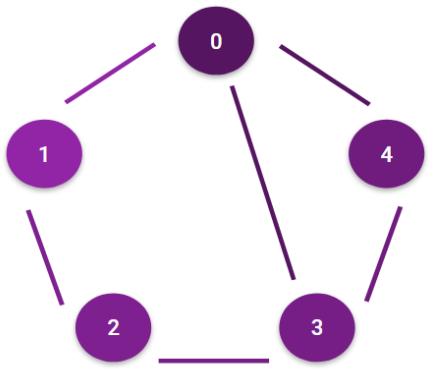


Figure 1: 5 node undirected cycle

x	M(x,0)	M(x,1)	IN(x,1)	M(x,2)	IN(x,2)
0	100100001	110110101	4.1	110111101	5.2
1	010100100	110101101	3.25	110111101	5.2
2	100001100	110101100	3.25	110111101	5.2
3	100100100	100111100	4.1	110111101	5.2
4	100010100	100110101	3.25	110111101	5.2

Table 1: Simple example for ANF

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 data mining problems where comparison of an element with entire data set is not possible due to large size of data, LSH is used. For example - finding a song name based on short part of it. It is the best suited algorithm when the data sets are high dimensional and large in size [4, 9].

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 Euclidean distance is calculated.

To adapt this method for graph mining each node can be converted to a vector representing its links with other nodes. Based on LSH euclidean distance(with `nearpy`) or jaccard similarity (with`self implemented`) can be calculated. Compared to ANF which gives you nodes within certain hop distance, both of these methods compute the distance between one node with all the other nodes. Using custom input threshold can be set allowing us to only get idea of neighbourhood within certain distance.

4. DATASETS

In this paper we use a total of five different type of data sets. In this section we discuss the datasets in detail.

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 and there is also no link between any two users [10].

This data set is very similar to data set used by authors of ANF[1], 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

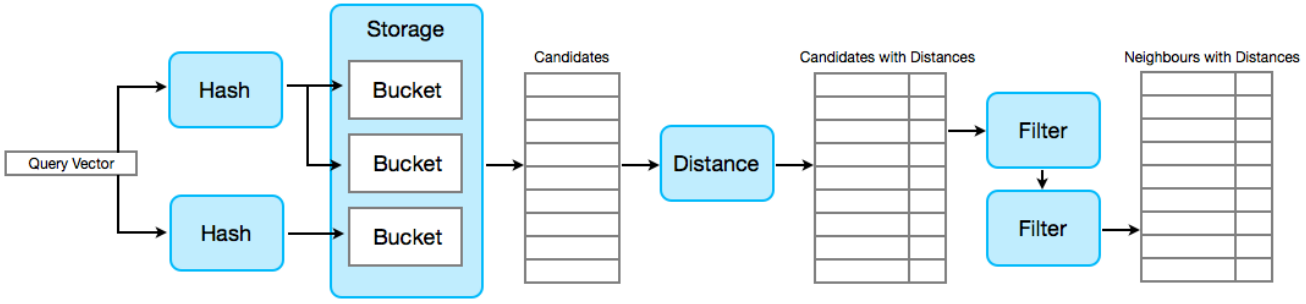


Figure 2: This figure explains the steps involved in a nearpy LSH algorithm. hashing generates one or more bucket keys. Storage stores and gives out bucket contents. Filters can be set to nearest, threshold distance h or unique elements. [4] [9]

Properties	Twitter small	Enron	Slashdot	Twitter large	Movielens
Nodes	35690	41915	82168	650714	103703
Edges	65690	148444	948464	1029439	65225506
Largest SCC, nodes	51	4686	71307	31314	17760
Largest SCC, edges	1814	98731	912381	275328	38167136
Largest WCC, nodes	3995	40623	82168	53459	103703
Largest WCC, edges	53459	147490	948464	275328	65225506

Table 2: Basic data statistics of each dataset

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 [11].

4.3 Twitter Data

This dataset is set of crawled tweets from June 2009 to December 2009 and has the timestamps along with the user who tweeted and what was tweeted. This dataset was divided into two datasets, 'small' and 'large', where *small* dataset is a subset of the whole dataset (i.e. *large*).

This dataset was preprocessed to check the tweet lengths whether they were inside the character limit defined by Twitter i.e. 250 character length. Tweets with data encoding inaccuracy had a length more than 250 and were pre-processed appended to the dataset. Tweets which had no user mentions were removed from the dataset and only the tweets which had a user mention were kept.

4.4 SlashDot

Slashdot is a website knows for technology-related news, it features technology oriented news posted by users and evaluated by the editors. The network cotnais links between the users of Slashdot as friends and this data was collected in February 2009 [11].

4.5 Data statistics

We performed exploratory data analysis on each data sets. In Appendix section B, degree distribution along with the visualization of the networks using Gephi can be seen. In table 2 we see the statistics of the datasets used for experiments.

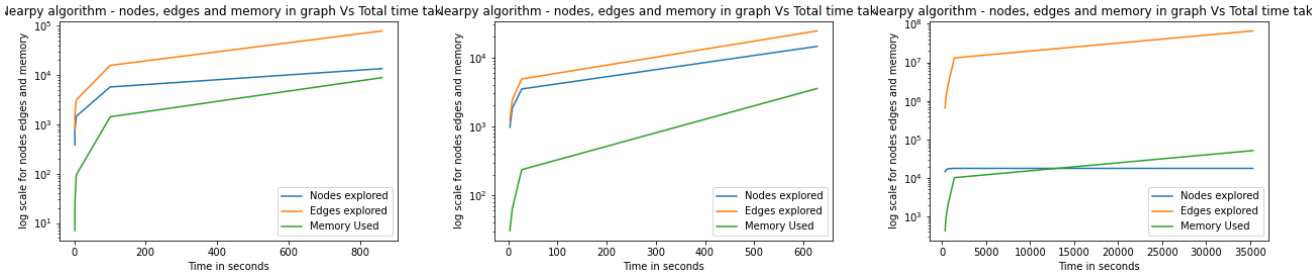
5. EXPERIMENTS

In this section we discuss the results to compare ANF and LSH based on their speed and computational efficiency. We answer the graph mining questions, such as which node is the most important node in the graph, nodes similar to other nodes. Along with this Exploratory data analysis is also presented helping to visualize data and get statistics. For this visualization of data sets `Gephi` was used.

5.1 LSH Experiments

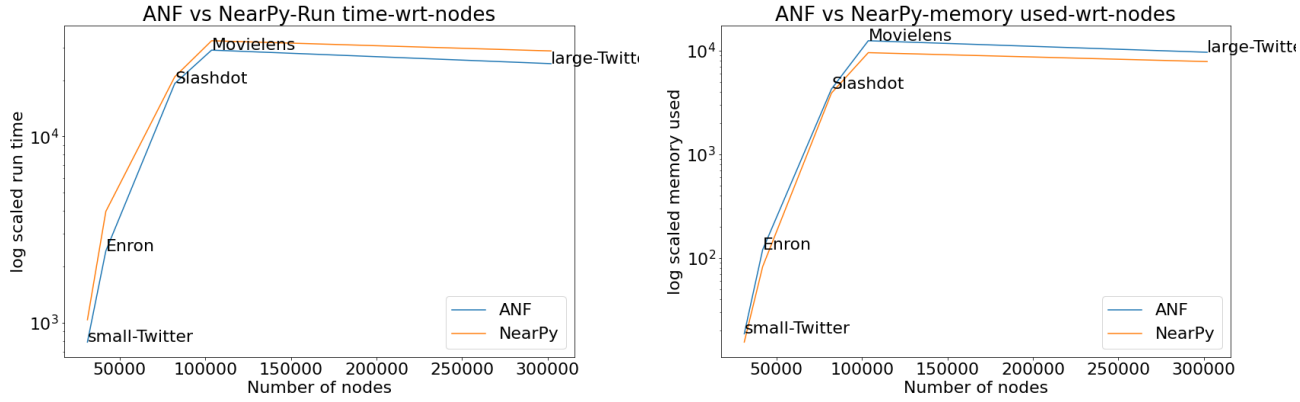
This section explains the experimental setup for LSH and gives basic idea of how well it stands when compared to ANF experiments in [1]. In this experiment we are comparing time and memory used by algorithm allowing us to check their speed, storage requirements and co relation or dependency of these parameters on number of nodes, edges.

Using a vector having information about the edges of a node is stored sequentially in an `nearpy` engine. In the engine

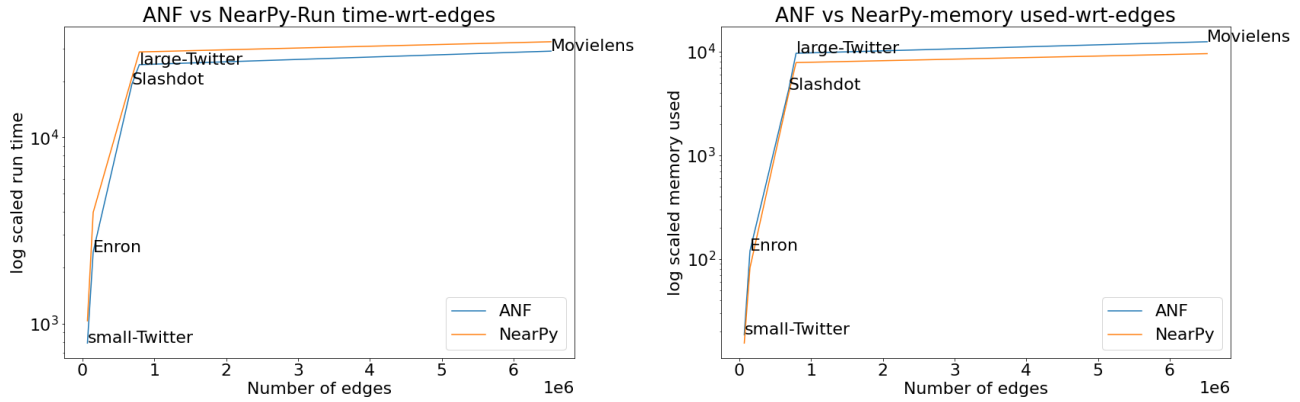


(a) Twitter (b) Enron (c) Movielens

Figure 3: NearPy experiment results



(a) Run time wrt Nodes (b) Memory wrt Nodes



(c) Run time wrt Edges (d) Memory wrt Edges

Figure 4: Run time and memory used by ANF and NearPy using 5 datasets.

steps mentioned in LSH algorithm section will be applied on every vector. After hashing every vector, for every node neighbourhood structure i.e. nodes within certain distance h can be explored. To trace exact memory used by the LSH algorithm `Tracemalloc` can be used. This package allows to monitor current and peak usage of memory. Every data set can be split into n parts representing some percent of data with final part being the whole data itself. LSH algorithm is then applied to these n parts. Splitting data in such way allows to track whether algorithm scales linearly or not.

Results of this experiment can be seen in 3. In 3 sub figure (a) results are for the nearpy and small twitter data set. Blue line shows the total number of unique nodes processed and orange line is total number of edges explored.The total memory used can be seen with the green line plot.Looking at the figure it can be concluded that for this data set all the three parameters are in the linear ratio when compared with time. In sub figure (b) results are same as (a) but algorithm changes to linear scaling quickly unlike to (a). Results on twitter data and Enron data shown in Sub figures (a) and

(b) do not help in deciding whether memory and time used by LSH is directly proportional to nodes or edges. But in sub figure 3 (C) Results of Movie-lens data set can be seen, which indicate that memory and time consumed is directly proportional to edges and not total nodes in graph. Results of this individual LSH experiment show that LSH is a good competitor for ANF algorithm and it would be interesting to see how well these perform.

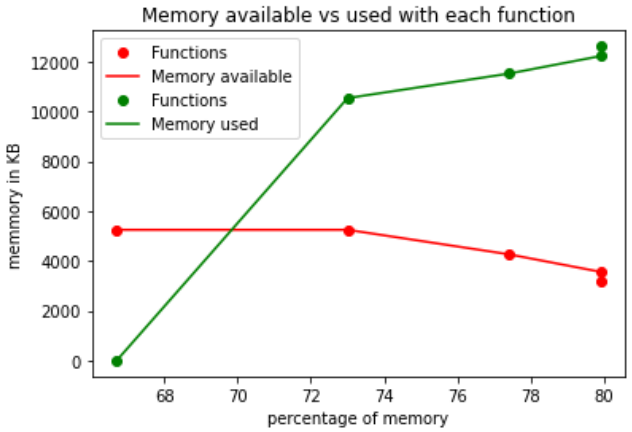


Figure 5: This image shows the change in memory plot for LSH algorithm for every step in algorithm. Even with large size of edges totaling to 65 million use of LSH algorithm allows to processes such a graph on local system consuming very low memory when compare to other methods.

5.2 Comparison Experiments

To find the most important nodes and similar nodes in the graph, we compute individual neighborhood up to distance of 3 for each node using ANF and nearest neighbors using Nearpy. The ANF returns a list of number of neighbors at each distance i.e. at $h = 1, 2, 3$ whereas NearPy returns the neighboring nodes. We evaluate both the algorithms based on total time taken and memory used for the computation. In figure 4 we see the runtime and memory used by each algorithm using the 5 mentioned datasets.

In figure 4 a and b the runtime and memory used by both the algorithm with respect to nodes respectively can be seen. We see that for *Movielens* runtime is more than in figure 4 (a and b) we see for *Movielens* run time and memory used is more than the *large Twitter* this is due to the difference in edges in both dataset which we can see in figure 4 c and d.

Both the algorithms produced identical results i.e the most important nodes and similar nodes based on the query, node IDs `456` for *small-Twitter*, `2453` for *Enron*, `44341` for *slashdot*, `1123` for *large-Twitter* and `663213` for *Movielens* were most important nodes. We see that ANF always took less time compared to NearPy however ANF had a higher memory requirement. ANF computes the node importance based on the number of neighbors at each distance `h`. Whereas, NearPy computes node importance by taking in account the times a node appear in nearest neighbor for other nodes. We can see that both methods scale linearly with increasing nodes and edges.

We wanted to see the time complexity and memory requirement for exact computation of individual neighborhood of each node on Movielens. Using `networkx` neighborhood of each node for a distance of 3 requires $60GB$ memory and `more than 32 hours` of run time. This shows that the approximation methods require less time and memory complexities.

6. CONCLUSION

In this paper we discussed about two algorithms which solve the problem of graph neighbourhood approximation. The approximation is necessary as it reduces time and memory complexity. From the experiments and results it can be seen that both algorithms have memory and time consumed directly proportional to the number of edges existing in graph. As compared to [1], LSH algorithm also scales linearly and does not show consumption of time or memory in an exponential way. ANF allows to compute neighbourhood within a certain hop distance. In comparison to this LSH computes euclidean distance of all nodes from a node. Depending on the type of problem both methods have advantages and disadvantages of this type of return methods. We were able to run both algorithms on a local machine having 16 GB of ram. Hence it can be concluded that both methods scale to the availability of the memory. Compared to ANF it was observed that LSH consumes less memory but is slower than ANF. In order to run LSH on on a data set, nodes have to be en numerated. Depending on number of unique nodes this can force algorithm to use higher int types such as int32 causing a memory overflow problem on small machine. Both algorithms can be parallelized. As said in [1] synchronization points can be added to achieve this and for LSH number of `Engines` can be increased. Both algorithms perform sequential scan of the data avoiding any random access. For the future work it would be an interesting to see the the comparison between these two algorithms when they are forced to run in parallel on multiple threads of CPU. Along with this combination of k means clustering and LSH algorithm can be used to obtain direct sub graph similarity. This comparison helped us to learn about working of algorithms used for neighbourhood approximation. Using results of this paper an algorithm can be chosen that suits best for a specific graph data mining problem.

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] 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.

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

[6] Jure Leskovec and Rok Sosič. "SNAP: A General-Purpose Network Analysis and Graph-Mining Library". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 8.1 (2016), p. 1.

[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] 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.

[10] Julian McAuley and Jure Leskovec. *From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews*. 2013. arXiv: 1303.4402 [cs.SI].

[11] Jure Leskovec et al. *Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters*. 2008. arXiv: 0810.1355 [cs.DS].

[12] 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.

APPENDIX

A. SOURCE CODE

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

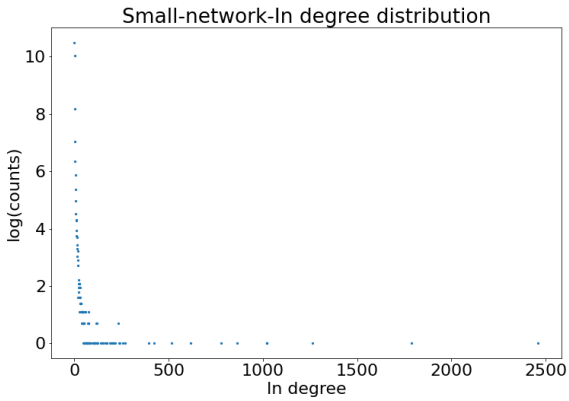
```
## Required packages
import glob
import pandas as pd
import os
from wordcloud import WordCloud
import numpy as np
import nltk
from sklearn.decomposition import
    LatentDirichletAllocation as LDA
from sklearn.feature_extraction.text import
    CountVectorizer
from sklearn.cluster import KMeans
from 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')
```

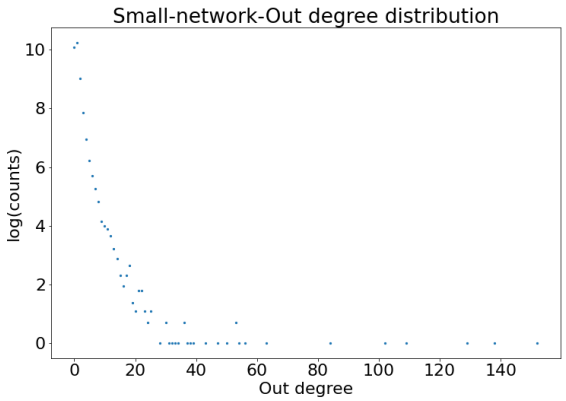
B. FIGURES

B.1 Degree distribution

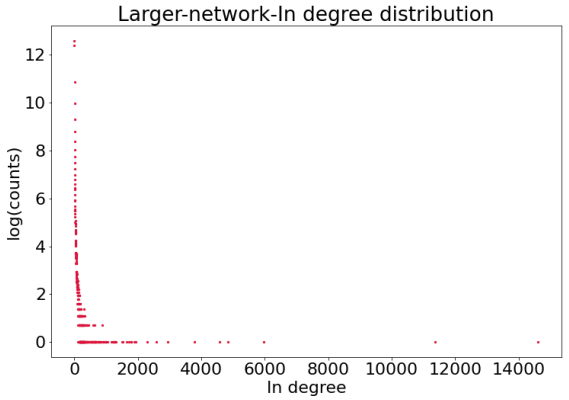
B.2 Enron visualization



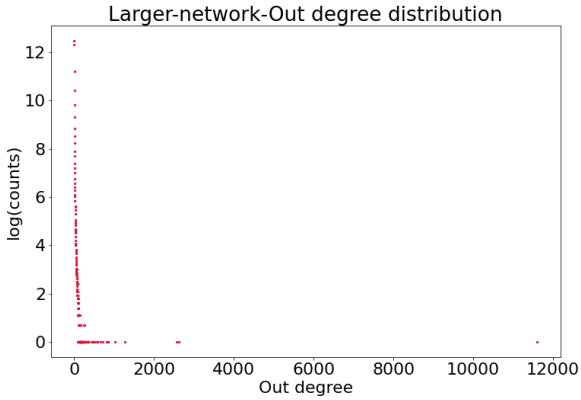
(a) Small Twitter In degree



(b) Small Twitter Out degree

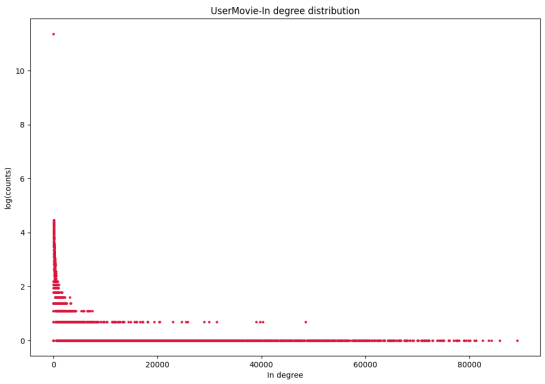


(c) Large Twitter In degree

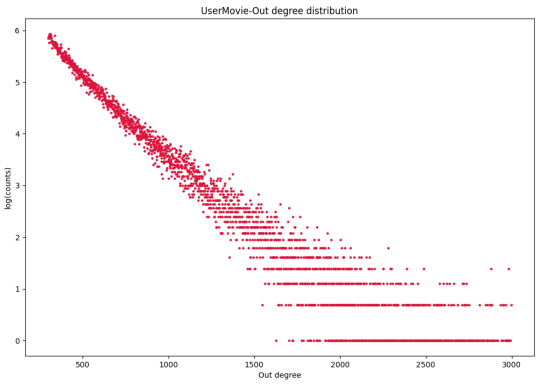


(d) Large Twitter Out degree

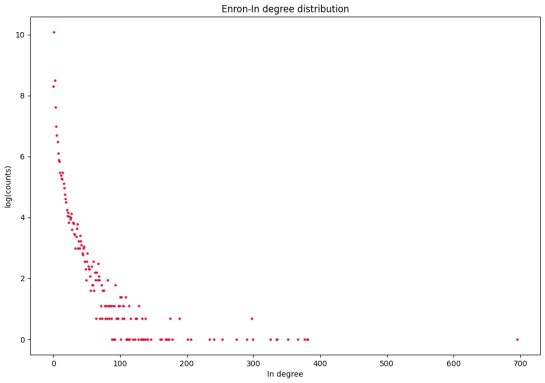
Figure 6: In and Out degree distribution



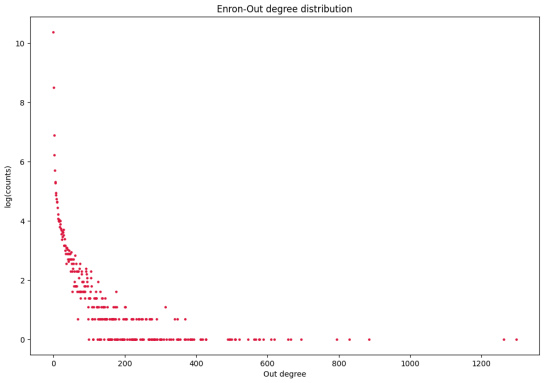
(a) User-movie In degree



(b) User-movie Out degree



(c) Enron In degree



(d) Enron Out degree

Figure 7: In and Out degree distribution

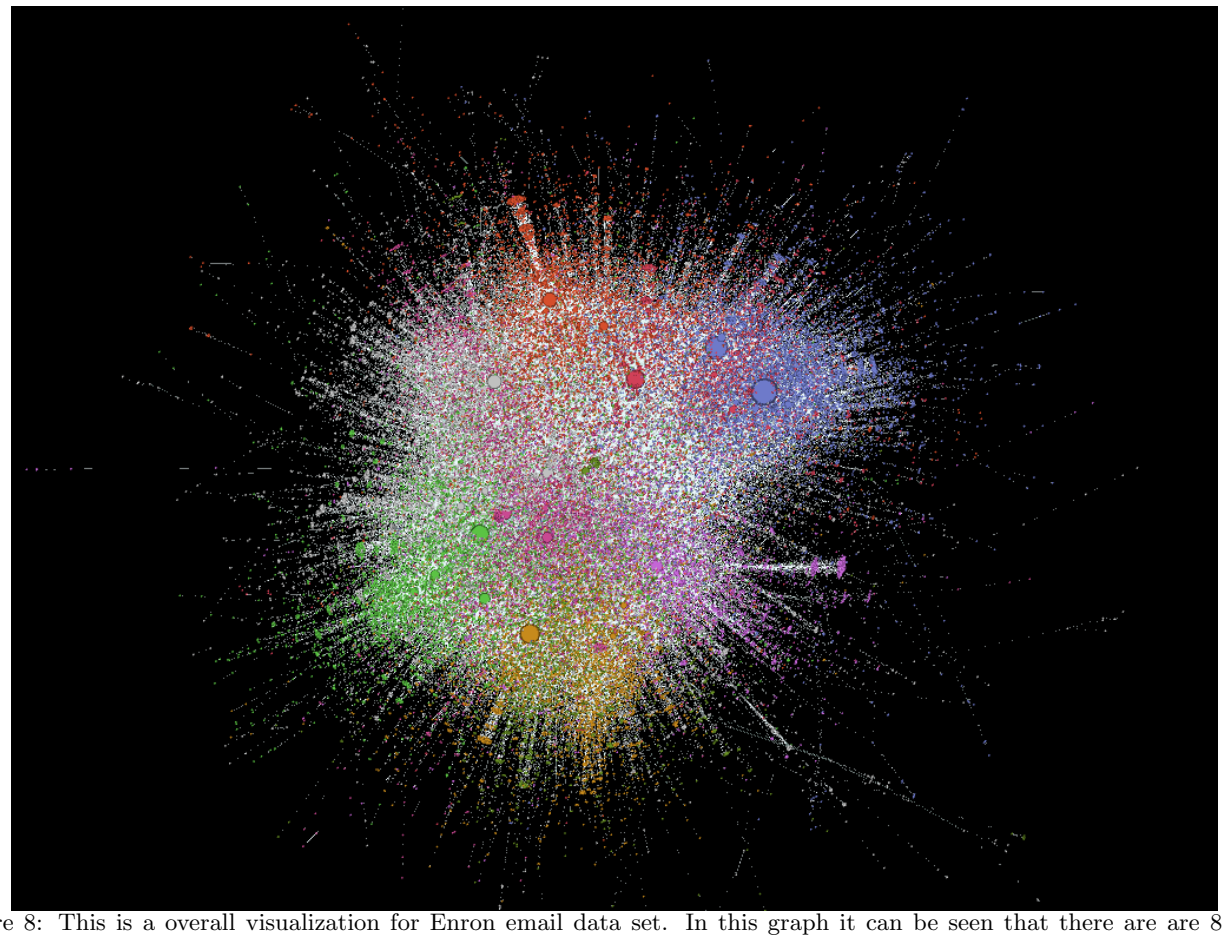


Figure 8: This is a overall visualization for Enron email data set. In this graph it can be seen that there are are 8 clear communities. These communities are obtained using algorithm [12]. Resolution value used is 3 and modularity score obtained is 0.67. Size of nodes was set based on degree of the 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.

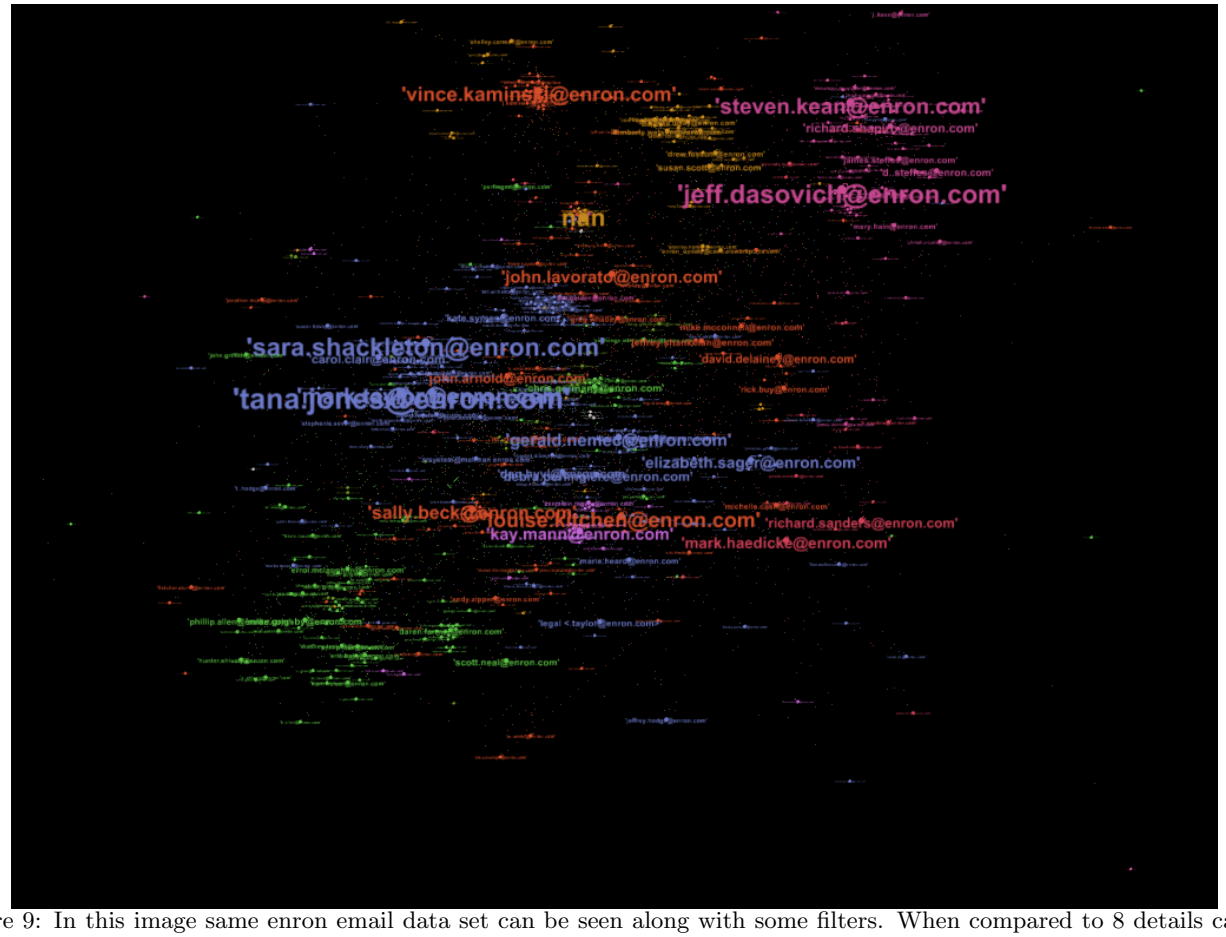


Figure 9: In this image same enron email data set can be seen along with some filters. When compared to 8 details can be seen better in this image. For this image giant component filter was set along with range filter for degree. From the filtered results names "jeff dsovich", tana jones", "sara shackelton" 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.