

## CASE STUDY 2 : COMMUNITY FINDING

### ALGORITHMS (LOUVAIN, HIERARCHICAL CLUSTERING, LAPLACIAN & GIRVAN-NEWMAN)

**Author :** *Paula Dwan*  
**Email :** *[paula.dwan@gmail.com](mailto:paula.dwan@gmail.com)*  
**Student ID :** *13208660*  
**Course :** *MSc Advanced Software Engineering*  
**Module :** *COMP-47270 Computational Network Analysis and Modelling*  
**Lecturer :** *Dr. Neil Hurley*  
**Email :** *[neil.hurley@ucd.ie](mailto:neil.hurley@ucd.ie)*  
**Due Date :** *20 April 2015*



## TABLE OF CONTENTS

<b>1</b>	<b>Case Study Requirements.....</b>	<b>3</b>
1.1	Case Study Requirements.....	3
1.2	Understanding of Requirements from Class Discussions.....	3
<b>2</b>	<b>Introduction.....</b>	<b>3</b>
2.1	Community Finding.....	3
2.2	Datasets Used.....	3
2.3	Evaluations Used.....	3
2.3.1	Modularity Function.....	4
2.3.2	Best Partition.....	4
2.3.3	Calculate Run-time.....	4
2.3.4	Impact of Dataset Size.....	4
<b>3</b>	<b>Louvain Algorithm (Community Detection in Large Networks).....</b>	<b>4</b>
3.1	Overview.....	4
3.1.1	What is the Louvain Algorithm?.....	4
3.1.2	Components.....	5
3.1.3	How Calculated Using Python?.....	5
3.2	Results obtained.....	6
3.2.1	Facebook : Small Dataset $\rightarrow \leq 10k$ Nodes (Louvain).....	6
3.2.2	cit-HepTh : Medium Dataset $\rightarrow 10k$ to $100k$ Nodes (Louvain).....	6
3.2.3	Twitter : Medium Dataset $\rightarrow 10k$ to $100k$ Nodes (Louvain).....	7
3.2.4	Google + : Large Dataset $\rightarrow > 100k$ Nodes (Louvain).....	7
<b>4</b>	<b>Hierarchical Clustering Algorithm.....</b>	<b>7</b>
4.1	Overview.....	7
4.1.1	What is Hierarchical Clustering Algorithm?.....	7
4.1.2	Components.....	8
4.1.3	How Calculated Using Python?.....	8
4.2	Results obtained.....	10
4.2.1	Facebook : Small Dataset $\rightarrow \leq 10k$ Nodes (Hierarchical Clustering).....	10
4.2.2	p2p-Gnutella06 : Small Dataset $\rightarrow 10k$ to $100k$ Nodes (Hierarchical Clustering).....	11
4.2.3	Medium & Large Datasets (Hierarchical Clustering).....	11
<b>5</b>	<b>Laplacian Algorithm.....</b>	<b>12</b>
5.1.1	Components.....	12
5.1.2	How Calculated Using Python?.....	13
5.2	Results obtained.....	13
5.2.1	Facebook : Small Dataset $\rightarrow \leq 10k$ Nodes (Laplacian Algorithm).....	13
5.2.2	p2p-Gnutella06 : Small Dataset $\rightarrow 10k$ to $100k$ Nodes (Laplacian Algorithm).....	15
5.2.3	Medium & Large Datasets (Laplacian Algorithm).....	15
<b>6</b>	<b>Girvan &amp; Newman Algorithm.....</b>	<b>16</b>
<b>7</b>	<b>Conclusions.....</b>	<b>16</b>
<b>8</b>	<b>References / Acknowledgements.....</b>	<b>17</b>
8.1	References – Networks Datasets.....	17
8.1.1	Acknowledgements – Datasets.....	17
8.1.2	Statistics – Datasets.....	17
8.2	Citations / Acknowledgements.....	18

## 1 CASE STUDY REQUIREMENTS

### 1.1 CASE STUDY REQUIREMENTS

Evaluate some community-finding methods on SNAP data and on simulated networks with embedded communities:

1. Implement your own community-finding algorithm.
2. Compare with at least two other algorithms.
3. Compute their relative performance in terms of quality (NMI, modularity) and run-time.

### 1.2 UNDERSTANDING OF REQUIREMENTS FROM CLASS DISCUSSIONS

1. Write a new community finding algorithm or use Laplacian as covered in class.
2. Compare to at least two others.
3. Evaluate one against the other two by performance / scalability:  
NMI score (0 totally different  $\rightarrow$  1 = very similar)  
Newman's Modularity / Modularity  
Time taken to calculate calculations  
Size of dataset used

## 2 INTRODUCTION

### 2.1 COMMUNITY FINDING

To start, what is a community?

A community of a graph  $G$  is a sub-graph  $C$  of  $G$ , such that the number of edges joining two vertices in  $C$  is greater than the number of edges joining vertices in  $C$  to vertices in  $G \setminus C$ .

So basically, a community is a sub-set of a graph where groups of nodes are very similar to each other, all of which are close neighbours of each other. Quite a few means of creating and evaluating communities exist.

### 2.2 DATASETS USED

Each of the directed graphs chosen deal with Social media and vary in the number of nodes present. For more information on each, please see the section : [References – Datasets](#)

### 2.3 EVALUATIONS USED

Once each community /partition is created using the community-finding algorithms, it is evaluated using :

- Modularity
- Best partition
- Time taken to complete evaluation
- Impact of dataset size

### 2.3.1 MODULARITY FUNCTION

<b>Overview</b>	Uses Louvain to best calculate the partition of the graph nodes which maximises modularity. This enables us to evaluate the dense connectivity between nodes within the same module / clusters. Smaller communities are unfortunately not evaluated very well by this function. Higher the value, the more closely connected are the clusters.
<b>How calculated?</b>	<pre>G = graph() partition = community.best_partition(G)  modularity = community.modularity(partition, G)</pre>

### 2.3.2 BEST PARTITION

<b>Overview</b>	Compute the modularity of a partition of a graph
<b>How calculated?</b>	<pre>G = graph() partition = community.best_partition(G)</pre>

### 2.3.3 CALCULATE RUN-TIME

<b>Overview</b>	Basically, how long does it take algorithm to produce the clustered graphs for the communities?
<b>How calculated?</b>	<p>I used an existing python module called <b>timeit</b>. This calculates the start and end time and subtraction produces the difference. Time taken may then be compared across all algorithms and all datasets.</p> <pre>import timeit                                # import inbuilt python method  def getStartTime():                           # return the start time     return timeit.default_timer()  def showTimeTaken(start):                     # end time &amp; calculate total time taken     logging.info(cs_ref, 'calculate Time Taken to run partitioning')     time_end = timeit.default_timer()     print 'Time taken to run = {0} - {1} = {2}'.format(end, start, end-start)</pre>

### 2.3.4 IMPACT OF DATASET SIZE

<b>Overview</b>	Does a larger dataset produce better communities? Does a larger dataset significantly increase the time taken? Is there a ratio between the dataset size and the time taken? In short, does the size of the dataset rather than algorithm used impact the time taken?
<b>How calculated?</b>	It is not but is based on the results obtained for each algorithm.

## 3 LOUVAIN ALGORITHM (COMMUNITY DETECTION IN LARGE NETWORKS)

### 3.1 OVERVIEW

#### 3.1.1 WHAT IS THE LOUVAIN ALGORITHM?

The Louvain Algorithm was developed by Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, when all were based in Louvain. It computes the best partitions of large scale networks and works as follows :

1. First, it looks for "small" communities by optimizing modularity in a local way.
2. Second, it aggregates nodes of the same community and builds a new network whose nodes are the communities.
3. Repeat until a maximum of modularity is attained.

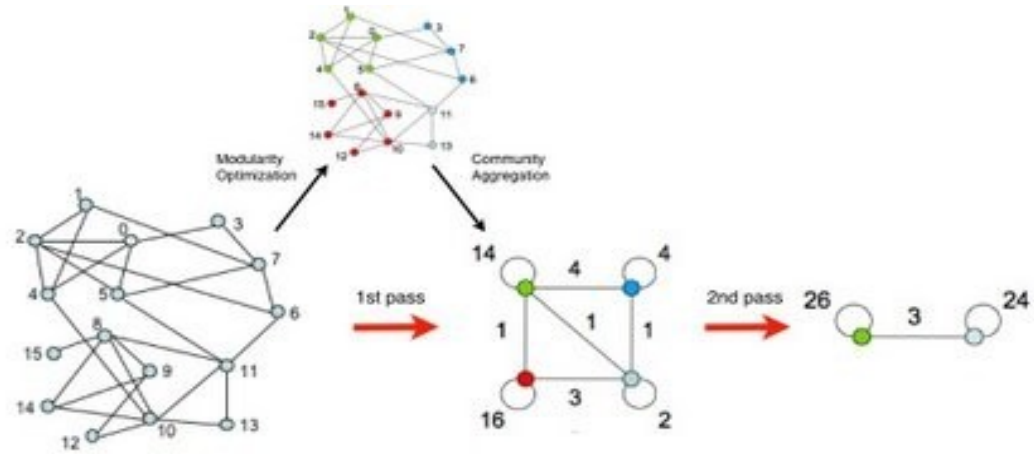


Figure 1 : Louvain Process (src : )

Initially, many communities are created, each of which is small in size. After each iteration, the communities become larger and larger until no more partitions can be created. This is also call a greedy optimization method.

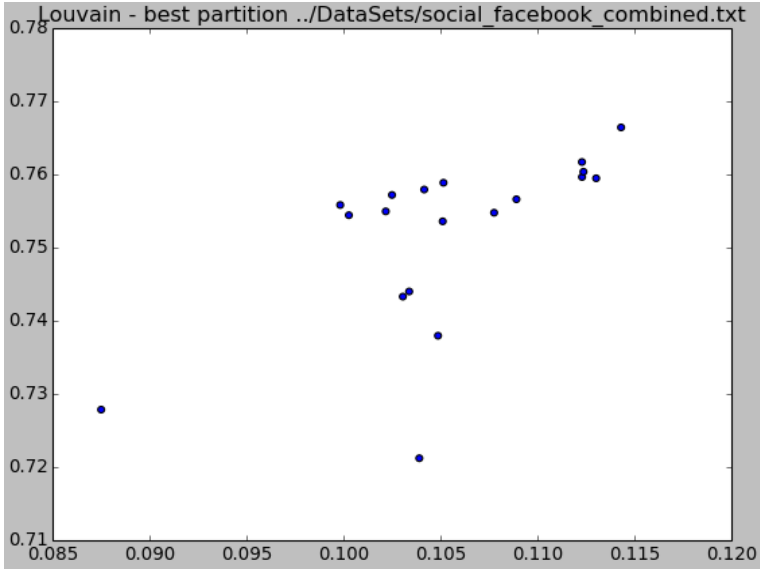
### 3.1.2 COMPONENTS

Component	Explanation
Modularity Best partition	Used <b>community</b> functions, as explained previously (see : <a href="#">Evaluations Used</a> ).
timing	Used <b>timeit</b> functions,, as explained previously (see : <a href="#">Evaluations Used</a> ).
All communities	Detail all communities present in the graph, if time permits also plot them using matlab.
Best community	Detail the community which forms the best partition of the graph.

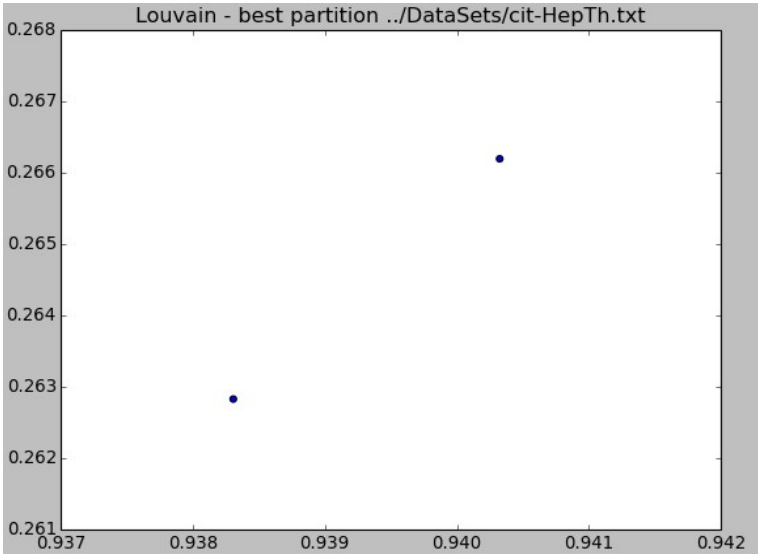
### 3.1.3 HOW CALCULATED USING PYTHON?

Component	Implementation & Explanation
Louvain – all communities	<pre> for i in set(partition.values()):     members = [nodes for nodes in partition.keys()                 if partition[nodes] == i]     members_info = 'Community[' + str(i) + ']' : ', members     with open(dest_file, "a") as dat_file:         dat_file.write("\n\t" + str(members_info))     print "\t" + str(members_info)     for com_ref, members in com_dict:         com_values["community_{0}".format(i)].append(members) </pre>
Louvain – best community	<p>Knowing the best partition of the graph, retrieve all nodes present in that cluster.</p> <pre> count = 0. for com in set(bp.values()):     count += 1.     list_nodes = [nodes for nodes in partition.keys()                   if partition[nodes] == com]     nodes_info = "\tNodes in best partitioned Community[" + str(com) + "]"                 = " + str(list_nodes) </pre>

## 3.2.1 FACEBOOK : SMALL DATASET → ≤ 10K NODES (LOUVAIN)

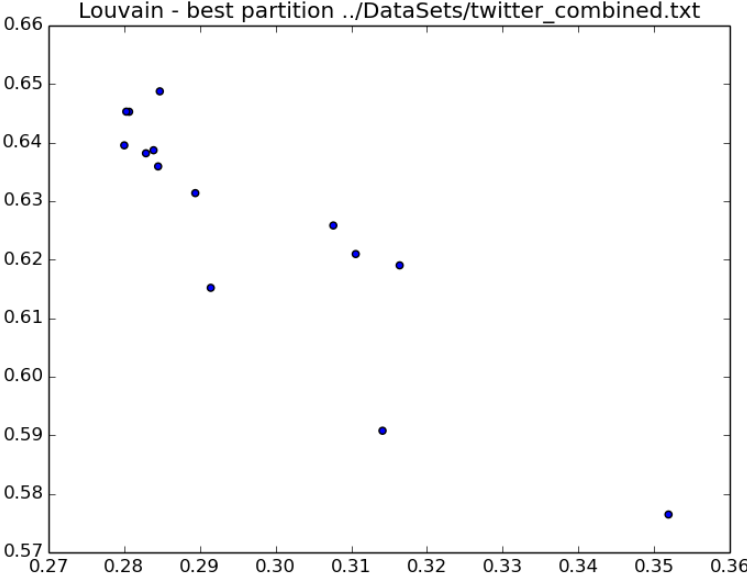
<b>Best partitions</b>			
	<p><i>Figure 2 : best partition as plotted</i></p> <p>Community[15] = [2670, 2699, 2703, 2767, 2834, 2879, 2889, 2959, 2972, 2982, 3008, 3283, 3309, 3311, 3314, 3318, 3325, 3382, 3423]</p>		
<b>Modularity</b>	Modularity : : 0.83497475803		
<b>All Communities</b>	Community[0] to Community[15] of various sizes		
<b>Time taken</b>	<b>w/ Plot :</b>	End – Start =	509365079.165 usec
	<b>w/o Plot :</b>	End – Start =	1560554.02756 usec

## 3.2.2 CIT-HEP'TH : MEDIUM DATASET → 10K TO 100K NODES (LOUVAIN)

<b>Best partitions</b>			
	<p><i>Figure 3 : best partition for community</i></p> <p>Community[172] = [9302016, 9303169]</p>		
<b>Modularity</b>	Modularity : : 0.649750739315		
<b>All Communities</b>	Community[0] to Community[172] of various sizes		

<b>Time taken</b>	<b>w/ Plot :</b>	End – Start =	2265986901.04 usec
	<b>w/o Plot :</b>	End – Start =	13264205.9326 usec

### 3.2.3 TWITTER : MEDIUM DATASET → 10K TO 100K NODES (LOUVAIN)

<b>Best partitions</b>			
	<p>Figure 4 : best partition for community</p> <p>Community[74] =  [16018368, 16806252, 227197639, 20765038, 228911535, 521746689, 203768124, 322524776, 23167567, 203268821, 107456903, 48132194, 167432593, 164087020]</p>		
<b>Modularity</b>	Modularity : : 0.805019461044		
<b>All Communities</b>	Community[0] to Community[74] of various sizes		
<b>Time taken</b>	<b>w/ Plot :</b>	End – Start =	n/a
	<b>w/o Plot :</b>	End – Start =	51262211.7996 usec

### 3.2.4 GOOGLE + : LARGE DATASET → > 100K NODES (LOUVAIN)

Not processed due to memory error :

```
File "build/bdist.linux-x86_64/egg/community/__init__.py", line 186, in best_partition
File "build/bdist.linux-x86_64/egg/community/__init__.py", line 248, in generate_dendrogram
... ..
File "/usr/lib/python2.7/copy.py", line 192, in deepcopy
    memo[d] = y
MemoryError
```

## 4 HIERARCHICAL CLUSTERING ALGORITHM

### 4.1 OVERVIEW

#### 4.1.1 WHAT IS HIERARCHICAL CLUSTERING ALGORITHM?

Let N be a set of items which need to be combined where similar (clustered), where there also exists a matrix of size NxN. This matrix details the similarities / differences between each item.

1. Initially, create N clusters containing one item per cluster. Apply the same distance between each cluster as exists between the items each contains.
2. Merge the two most similar pair of clusters (If more than one exists that are practically indistinguishable, then choose one at random. One less cluster now exists.

3. Recalculate all distances to include the new cluster.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N.

There are different ways to calculate the distances. Also, it is possible to complete this computation in reverse, with all nodes assigned to a single cluster and dividing at each iteration. As implemented, this is an **Agglomerative Clustering** algorithm which joins from the bottom up rather than divides from the top down. As I did not apply connectivity constraints where each cluster may only join an adjacent one, computation is somewhat significant for larger datasets (actually, out of memory errors arose).

There are also different means of calculating the optimal number of clusters, including :

- calculate **elbow criterion** → number of clusters -v- sum of squared distance,
- **visually inspect** the dendrogram → where does similarities = dissimilarities for existing clusters.

Basically, the dendrogram is cut where there is space to cut them, where there is an obvious large jump in levels of consecutive nodes.

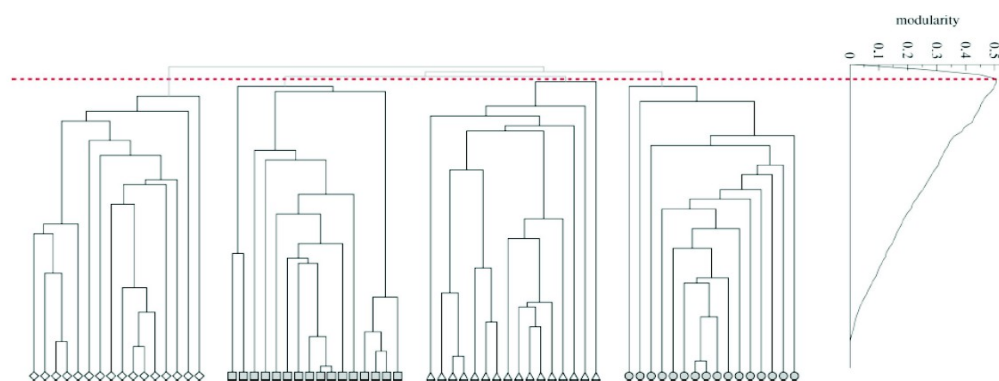


FIG. 6. Plot of the modularity and dendrogram for a 64-vertex random community-structured graph generated as described in the text with, in this case,  $z_{in}=6$  and  $z_{out}=2$ . The shapes at the bottom denote the four communities in the graph and, as we can see, the peak in the modularity (dotted line) corresponds to a perfect identification of the communities.

Figure 5 : Sample Dendrogram from lecture slides

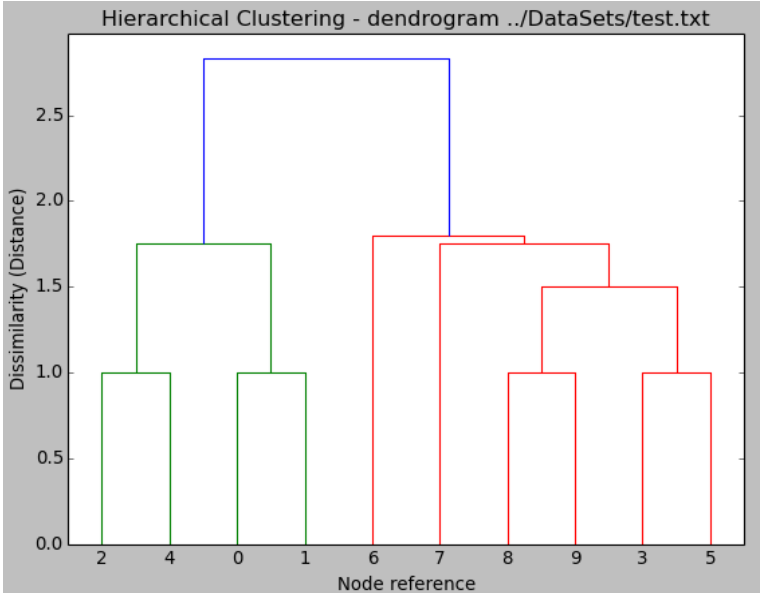
#### 4.1.2 COMPONENTS

Component	Explanation
get dendrogram	Parse the graph and retrieve the dendrogram tree structure, using <code>scipy.cluster &gt; hierarchy</code> and <code>scipy.spatial &gt; distance</code> to generate the dendrogram.
Plot results	Used hierarchy functionality to plot the tree / dendrogram
Modularity Best partition	Used community functions, as explained previously (see : <a href="#">Evaluations Used</a> ).
timing	Used <code>timeit</code> functions,, as explained previously (see : <a href="#">Evaluations Used</a> ).

#### 4.1.3 HOW CALCULATED USING PYTHON?

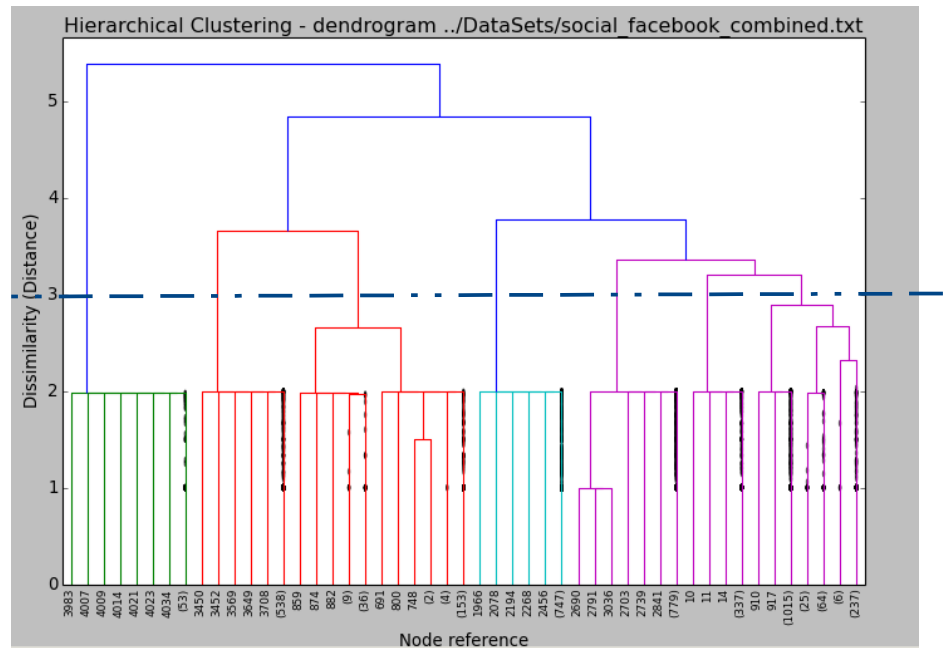
Component	Implementation & Explanation
Get dendrogram	<pre> # calculate the shortest distance between each node in the graph path_length = nx.all_pairs_shortest_path_length(g) n = len(g.nodes()) distances = np.zeros((n, n))  # merge the two nodes closest together into the same cluster &amp; # recalculate distances for u, p in path_length.iteritems():     for v, d in p.iteritems():         distances[int(u) - 1][int(v) - 1] = d sd = distance.squareform(distances)  hier = hierarchy.average(sd) return hier </pre>



Component	Implementation & Explanation
Plot results	<p>I used scipy hierarchy.dendrogram to plot the results:</p> <pre> hierarchy.dendrogram(hier, show_leaf_counts=True, truncate_mode='level', p=7, show_contracted=True) plt.title("Hierarchical Clustering - dendrogram " + src_file) plt.ylabel("Dissimilarity (Distance)") plt.xlabel("Node reference") plt.savefig("plots/hierarchical_clustering.png") plt.show() </pre>
	<p>The x-axes contains the node references and this is more clearly seen in the following sample. For larger dendrograms, the x-axis is illegible.</p>  <p>Figure 6 : Sample dendrogram – 10 nodes &amp; 13 edges.</p> <p>As a result, I restricted the number of levels to produce a cleaner output :</p> <pre> show_leaf_counts=True, truncate_mode='level', p=7, show_contracted=True </pre>

4.2.1 FACEBOOK : SMALL DATASET  $\rightarrow \leq 10K$  NODES (HIERARCHICAL CLUSTERING)

dendrogram

Figure 7 : resulting dendrogram – with set number of levels ( $p=7$ ) for clarity.

Visually cut dendrogram at 3.0 to optimize clusters

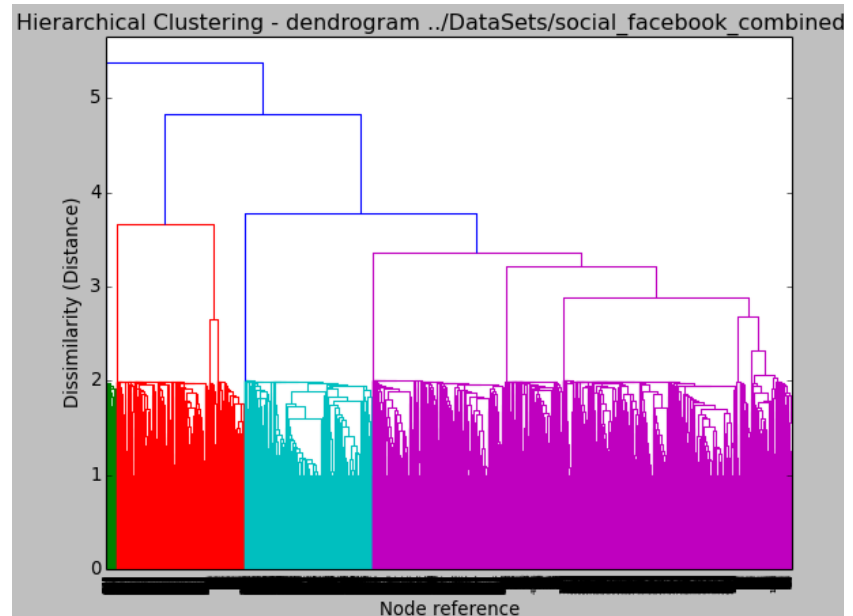
dendrogram –  
full hierarchy

Figure 8 : resulting dendrogram – with all levels included.

<b>Best partitions</b>	Community[15] [2670, 2699, 2703, 2767, 2834, 2879, 2889, 2959, 2972, 2982, 3008, 3283, 3309, 3311, 3314, 3318, 3325, 3382, 3423]
<b>Modularity</b>	0.83497475803
<b>Time taken</b>	<b>w/ Plot :</b> End – Start = $1.43147966635e+15 - 1.43147958886e+15 = 77489745.1401$ usec <b>w/oPlot :</b> End – Start = $1.43145102074e+15 - 1.4314507998e+15 = 220930594.206$ usec

#### 4.2.2 P2P-GNUTELLA06 : SMALL DATASET → 10K TO 100K NODES (HIERARCHICAL CLUSTERING)

##### dendrogram

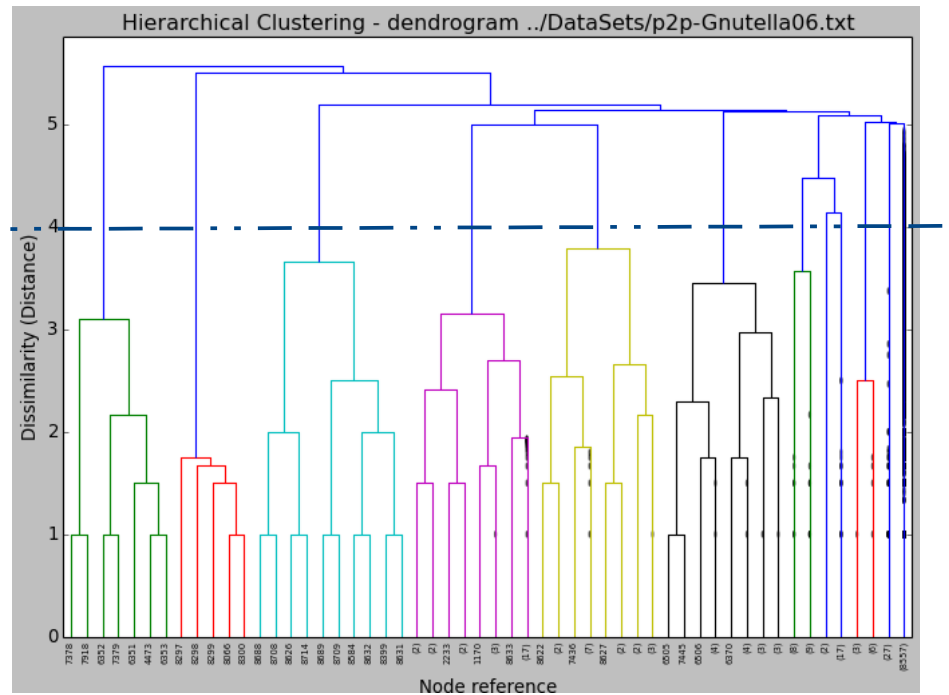


Figure 9 : resulting dendrogram – with set number of levels ( $p=7$ ) for clarity

Visually cut dendrogram at 4.0 to optimize clusters

##### Best partitions

Community[21] =

[348, 553, 597, 598, 602, 603, 604, 605, 748, 1036, 1037, 1041, 1042, 1043, 1044, 1045, 1112, 1114, 1115, 1116, 1128, 1169, 1208, 1330, 1332, 1333, 1334, 1335, 1336, 1342, 1471, 1473, 1474, 1475, 1476, 1477, 1479, 1625, 2149, 2150, 2152, 2153, 2155, 2212, 2249, 2250, 2251, 2252, 2255, 2373, 2519, 2530, 2590, 2592, 2594, 2595, 2597, 2757, 2828, 2976, 3019, 3054, 3069, 3188, 3199, 3275, 3294, 3299, 3314, 3364, 3420, 3494, 3511, 3533, 3607, 3629, 3673, 3708, 3740, 3742, 3743, 3745, 3770, 3778, 3787, 3798, 3824, 3895, 3986, 3996, 4073, 4074, 4075, 4076, 4083, 4253, 4254, 4256, 4257, 4258, 4259, 4263, 4390, 4402, 4404, 4443, 4456, 4522, 4532, 4581, 4648, 4741, 4933, 4951, 4968, 5067, 5144, 5145, 5205, 5206, 5208, 5209, 5236, 5352, 5394, 5395, 5428, 5479, 5554, 5555, 5599, 5674, 5675, 5676, 5693, 5704, 5726, 5757, 5760, 5780, 5803, 5804, 5805, 5856, 5868, 5939, 5945, 5946, 5947, 5948, 5949, 5968, 5969, 5970, 5971, 5972, 6051, 6062, 6097, 6106, 6226, 6334, 6415, 6417, 6440, 6494, 6579, 6640, 6641, 6642, 6643, 6688, 6711, 6804, 6805, 6896, 7004, 7011, 7017, 7034, 7035, 7036, 7037, 7063, 7064, 7067, 7080, 7081, 7123, 7312, 7313, 7314, 7315, 7324, 7325, 7326, 7374, 7486, 7490, 7493, 7495, 7497, 7499, 7552, 7570, 7572, 7573, 7610, 7642, 7704, 7708, 7733, 8094, 8143, 8198, 8259, 8286, 8327, 8332, 8334, 8368, 8411, 8412, 8413, 8422, 8423, 8426, 8495, 8520]

##### Modularity

0.393810635543

##### Time taken

w/ Plot : End – Start =  $1.43146545306e+15 - 1.43146492412e+15 = 528944584.846$  usec

w/O Plot : End – Start =  $1.4314641486e+15 - 1.43146372118e+15 = 427414832.115$  usec

#### 4.2.3 MEDIUM & LARGE DATASETS (HIERARCHICAL CLUSTERING)

Again, a memory time-out occurred so I decided to use two small datasets for comparison, Facebook and *Gnutella peer-to-peer network, August 6 2002* (<http://snap.stanford.edu/data/p2p-Gnutella06.html>) :

Details	≤ 10k : Facebook	≤ 10k : p2p-Gnutella06
Node	4,039	8,717
Edges	88,234	31,525
Nodes in largest WCC	4,039 (1.000)	8,717 (1.000)

Details	≤ 10k : Facebook	≤ 10k : p2p-Gnutella06
Edges in largest WCC	88,234 (1.000)	31,525 (1.000)
Nodes in largest SCC	4,039 (1.000)	3,226 (0.370)
Edges in largest SCC	88,234 (1.000)	13,589 (0.431)
Average clustering coefficient (ACE)	0.6055	0.0067
Number of triangles	1,612,010	1,142
Fraction of closed triangles	0.2647	0.002717
Diameter (longest shortest path)	8	10
90-percentile effective diameter	4.7	5.3

Again, please see relevant SNAP web-page for more information on each dataset.

## 5

### LAPLACIAN ALGORITHM

Spectral clustering algorithm which uses the eigenvectors of the graph Laplacian matrices to cluster the social network. Kmeans to associate nodes based on shortest distance to the **centre** of each cluster.

<b>Laplacian</b>	Having a graph $G = (V, E)$ , where $G$ is undirected, un-weighted with no graph loops (i.e.: $v^i \not\rightarrow v^i$ ). There are different types of Laplacian :
	<i>laplacian_matrix</i> ( $G[, nodelist, weight]$ )      Return Laplacian matrix $G$
	<i>normalized_laplacian_matrix</i> ( $G[, nodelist, ...]$ )      Return normalized Laplacian matrix $G$
	<i>directed_laplacian_matrix</i> ( $G[, nodelist, ...]$ )      Return directed Laplacian matrix $G$
<b>K-Means</b>	K-Means is calculated as follows :
	1. Decide K for each cluster at random.
	2. Assign all objects to the nearest K (centre of respective cluster).
	3. Move each K (centre) to the average of all members of that cluster.
	4. After moving each K (centre), re-assign all other objects to the nearest K.
	5. Repeat steps 2 to 4 until each K (centre) no longer needs to move.

#### 5.1.1 COMPONENTS

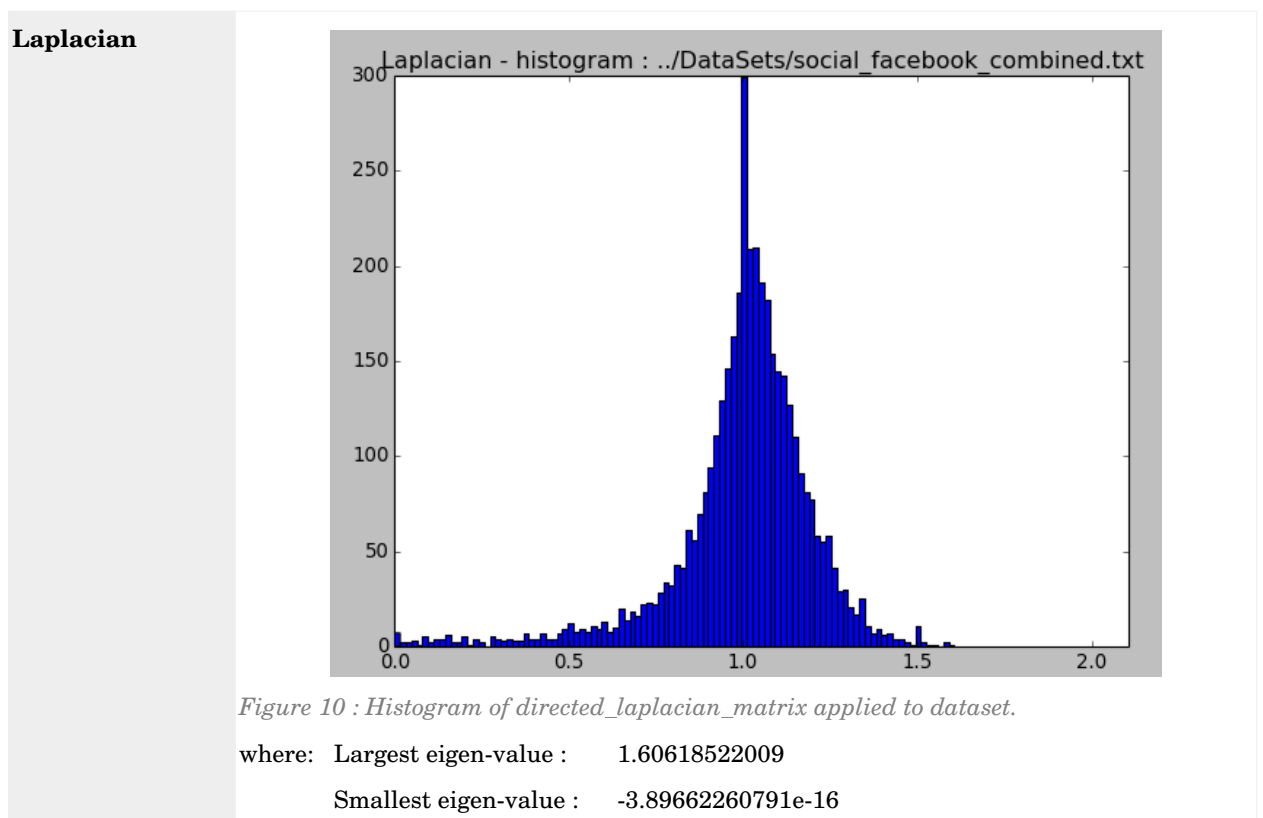
Component	Explanation
Calculate laplacian	Use <b>networkx</b> inbuilt function
Calculate k-means	Use k-means functions <code>vq</code> , <code>kmeans</code> , <code>whiten</code> from <code>scipy.cluster.vq</code> import
Plot results	Use <b>networkx</b> inbuilt function
Modularity Best partition	Used <b>community</b> functions, as explained previously (see : <a href="#">Evaluations Used</a> ).
timing	Used <b>timeit</b> functions,, as explained previously (see : <a href="#">Evaluations Used</a> ).

### 5.1.2 HOW CALCULATED USING PYTHON?

Component	Implementation & Explanation
laplacian	<p>Used NetworkX to return eigen values for the Laplacian matrix created from source graph.</p> <pre> g2 = cv.convert_to_directed(g) laplacian = nx.directed_laplacian_matrix(g2) eigen_values= lg.eigvals(laplacian.A) eigen_info = "\tLaplacian : \n\tLargest eigen-value : " + str(max(eigen_values)) + "\n\tSmallest eigen-value : " \ + str(min(eigen_values)) with open(dest_file, "a") as dat_file:     dat_file.write("\n" + eigen_info) print(eigen_info) return eigen_values </pre>
K-Means	<p>Used scipy to calculate :</p> <pre> g_matrix= nx.to_numpy_matrix(g) g_array = np.asarray(g_matrix).reshape(-1) whitened = whiten(g_array) clusters = 3 centroids,_ = kmeans(whitened, clusters) idx,_ = vq(g_array, centroids) </pre>
Plot results	<p>Plot a scatter graph containing all clusters</p> <pre> plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y_pred); </pre>

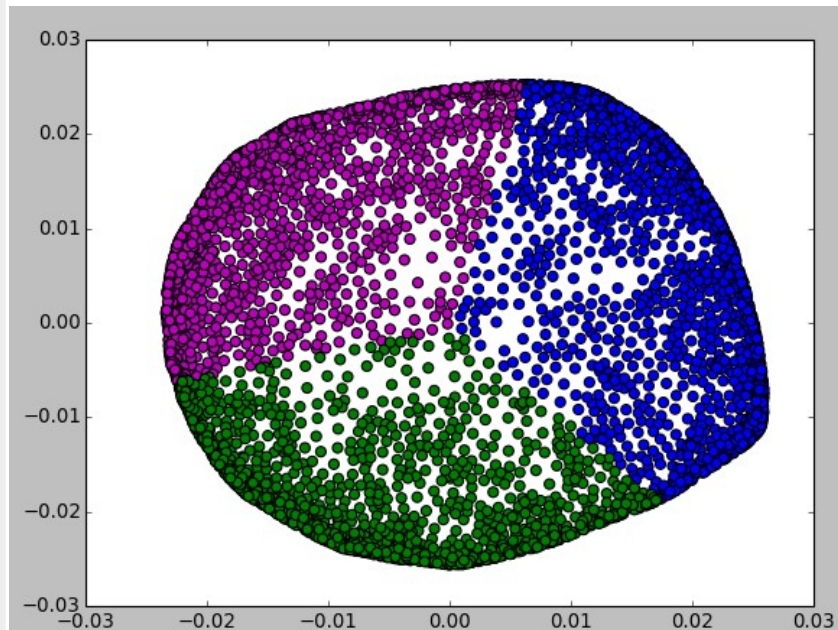
## 5.2 RESULTS OBTAINED

### 5.2.1 FACEBOOK : SMALL DATASET → ≤ 10K NODES (LAPLACIAN ALGORITHM)

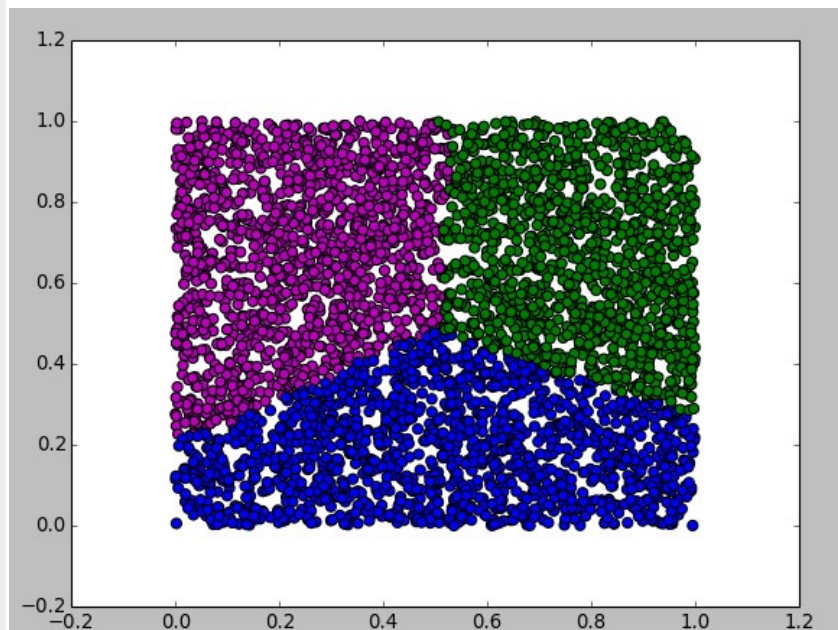


**KMeans**

Applying kmeans and eigen-vectors in cs2\_laplacian\_amended.py results in :



and K-Means and Delauney results in :

**Best partitions**

Community[15] =  
[2670, 2699, 2703, 2767, 2834, 2879, 2889, 2959, 2972, 2982, 3008, 3283, 3309, 3311, 3314, 3318, 3325, 3382, 3423]

**Modularity**

0.83497475803

**Time Taken**

**w/ Plot :** End – Start = 1.43149743004e+15 – 1.43149731495e+15 = 115089203.119 usec

**w/o Plot :** End – Start = 1.43149777696e+15 - 1.43149767847e+15 = 98486344.8143 usec

### 5.2.2 P2P-GNUTELLA06 : SMALL DATASET → 10K TO 100K NODES (LAPLACIAN ALGORITHM)

#### Laplacian

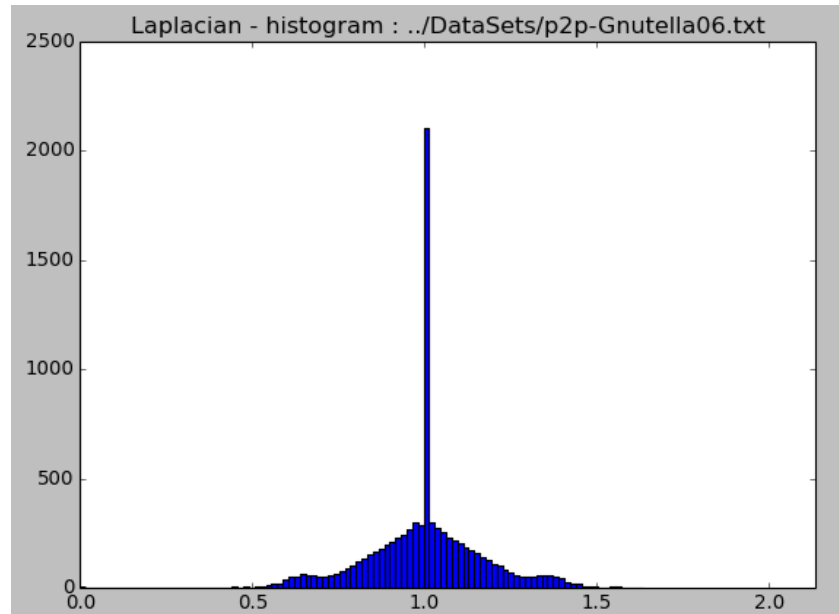


Figure 11 : Histogram of directed\_laplacian\_matrix applied to dataset.

where: Largest eigen-value : 1.63497817956  
Smallest eigen-value : -9.2699337316e-14

#### KMeans

##### Best partitions

Community[21] =  
[348, 553, 597, 598, 602, 603, 604, 605, 748, 1036, 1037, 1041, 1042, 1043, 1044, 1045, 1112, 1114, 1115, 1116, 1128, 1169, 1208, 1330, 1332, 1333, 1334, 1335, 1336, 1342, 1471, 1473, 1474, 1475, 1476, 1477, 1479, 1625, 2149, 2150, 2152, 2153, 2155, 2212, 2249, 2250, 2251, 2252, 2255, 2373, 2519, 2530, 2590, 2592, 2594, 2595, 2597, 2757, 2828, 2976, 3019, 3054, 3069, 3188, 3199, 3275, 3294, 3299, 3314, 3364, 3420, 3494, 3511, 3533, 3607, 3629, 3673, 3708, 3740, 3742, 3743, 3745, 3770, 3778, 3787, 3798, 3824, 3895, 3986, 3996, 4073, 4074, 4075, 4076, 4083, 4253, 4254, 4256, 4257, 4258, 4259, 4263, 4390, 4402, 4404, 4443, 4456, 4522, 4532, 4581, 4648, 4741, 4933, 4951, 4968, 5067, 5144, 5145, 5205, 5206, 5208, 5209, 5236, 5352, 5394, 5395, 5428, 5479, 5554, 5555, 5599, 5674, 5675, 5676, 5693, 5704, 5726, 5757, 5760, 5780, 5803, 5804, 5805, 5856, 5868, 5939, 5945, 5946, 5947, 5948, 5949, 5968, 5969, 5970, 5971, 5972, 6051, 6062, 6097, 6106, 6226, 6334, 6415, 6417, 6440, 6494, 6579, 6640, 6641, 6642, 6643, 6688, 6711, 6804, 6805, 6896, 7004, 7011, 7017, 7034, 7035, 7036, 7037, 7063, 7064, 7067, 7080, 7081, 7123, 7312, 7313, 7314, 7315, 7324, 7325, 7326, 7374, 7486, 7490, 7493, 7495, 7497, 7499, 7552, 7570, 7572, 7573, 7610, 7642, 7704, 7708, 7733, 8094, 8143, 8198, 8259, 8286, 8327, 8332, 8334, 8368, 8411, 8412, 8413, 8422, 8423, 8426, 8495, 8520]

##### Modularity

0.393810635543

##### Time Taken

w/ Plot : End – Start = 100066301.076 usec

w/o Plot : End – Start = ---

### 5.2.3 MEDIUM & LARGE DATASETS (LAPLACIAN ALGORITHM)

Again due to memory issues, it was not possible to calculate for medium and large datasets.

I was going to use the SNAP implementation of the Girvan and Newman algorithm, however I decided that I had used enough and any more might not add any more to the resulting data nor to the end conclusions. Girvan & Newman is summarised as follows :

1. First, calculate the betweenness of all existing edges in the network.
2. Remove the edge with the highest betweenness
3. Recalculate the betweenness of any edge impacted by step 2.
4. Repeat until no edges remain in the network.

Overall, this was a very interesting exercise as it showed the differences between hierarchical clustering, Laplacian Eigen-Values, K-Means, Louvain. As a python learning experience, I decided where practical to use third party functions.

The purpose of the algorithms chosen was to see if the type of algorithm would improve on the resulting clusters. Did the type of dataset rather than the size of the dataset affect the results. (Larger datasets did affect the results as in some instances there were none due to memory issues)

<b>Louvain</b>	<p>This algorithm uses modularity of the graph to calculate the appropriate clusters (communities). As a network with a high modularity will have good (strong) connections between the nodes within the same cluster. Connections to other clusters will be sparse in number.</p> <p>If we take two social networks Facebook (4k nodes) and Twitter (81k nodes), both have modularity of about 80% (actually 81% and 83% respectively). While a citation dataset has a modularity of 65%. The resulting number of clusters is significantly higher for the citation network cit-HepTh, than the social networks.</p>
<b>Hierarchical Clustering</b>	<p>In the algorithm as used, all nodes were assigned to individual clusters and then merged based on a similarity measure, usually proximity. This is from the bottom up implementation (top-down implementation also exists)</p> <p>Again, the social network appears to work better with Hierarchical Clustering than does the citation network, cutting dendrogram at 3 rather than 4. Also it is worth noting that the citation network has many small clusters of similar sizes.</p> <p>Due to the time taken to calculate only small networks were used – all medium and large networks failed due to memory errors.</p>
<b>Laplacian</b>	<p>A directed Laplacian Matrix was created using NetworkX using as source the dataset. From this, the eigenvalues were calculated and the minimum and maximum noted.</p> <p>Again, there were memory issues for medium and large datasets.</p> <p>To finalise this algorithm, it necessary to apply K-Means.</p> <p>From the results to date, again social networks appear to fair better in the creation of strong communities.</p>
<b>KMeans</b>	<p>While the results are calculated, they are not yet graphed. It will be interesting if the results seen so far also apply to K-Means.</p>

No one algorithm is better than the other, it depends on the requirements when creating the clusters. Do we know how many clusters we need, then perhaps k-means would be an appropriate option to define them. If we don't then perhaps hierarchical clustering would be a better choice, allowing the user to decide what number of clusters apply once all are known. If a network has a high modularity then Louvain would be more appropriate, however if the resulting clusters will be small in size then Louvain could sometimes assign the wrong entity to the wrong cluster.



## 8.1 REFERENCES – NETWORKS DATASETS

Datasets : <http://snap.stanford.edu/data/index.html>

The information in this section is taken directly from <http://snap.stanford.edu/> and is included for informational purposes only.

## 8.1.1 ACKNOWLEDGEMENTS – DATASETS

Name	Size	Link
Social circles: Facebook	Small → ≤ 10k nodes	<a href="http://snap.stanford.edu/data/egonets-Facebook.html">http://snap.stanford.edu/data/egonets-Facebook.html</a>
Social circles: Twitter	Medium → 10k to 100k nodes	<a href="http://snap.stanford.edu/data/egonets-Twitter.html">http://snap.stanford.edu/data/egonets-Twitter.html</a>
High-energy physics citation network	Medium → 10k to 100k nodes	<a href="http://snap.stanford.edu/data/cit-HepPh.html">http://snap.stanford.edu/data/cit-HepPh.html</a>
High-energy physics theory citation network	Medium → 10k to 100k nodes	<a href="http://snap.stanford.edu/data/cit-HepTh.html">http://snap.stanford.edu/data/cit-HepTh.html</a>
Social circles: Google+	Large → > 100k nodes	<a href="http://snap.stanford.edu/data/egonets-Gplus.html">http://snap.stanford.edu/data/egonets-Gplus.html</a>

## 8.1.2 STATISTICS – DATASETS

Details	≤ 10k : Facebook	10k → 100k : cit-HepTh	10k → 100k cit-HepPh	10k → 100k : Twitter	> 100k : Google +
Nodes	4,039	27,770	34,546	81,306	107,614
Edges	88,234	352,807	421,578	1,768,149	13,673,453
Nodes in largest WCC	4,039 (1.000)	27,400 (0.987)	34,401 (0.996)	81,306 (1.000)	107,614 (1.000)
Edges in largest WCC	88,234 (1.000)	352,542 (0.999)	421,485 (1.000)	1,768,149 (1.000)	13,673,453 (1.000)
Nodes in largest SCC	4,039 (1.000)	7,464 (0.269)	12,711 (0.368)	68,413 (0.841)	69,501 (0.646)
Edges in largest SCC	88,234 (1.000)	116,268 (0.330)	139,981 (0.332)	1,685,163 (0.953)	9,168,660 (0.671)
Average clustering coefficient (ACE)	0.6055	0.3120	0.2848	0.5653	0.4901
Number of triangles	1,612,010	1,478,735	1,276,868	13,082,506	1,073,677,742
Fraction of closed triangles	0.2647	0.04331	0.05377	0.06415	0.6552
Diameter (longest shortest path)	8	13	12	7	6
90-percentile effective diameter	4.7	5.3	5	4.5	3

Reference	Acknowledgement
SNAP → datasets	Sourced from : <a href="http://snap.stanford.edu/data">http://snap.stanford.edu/data</a> Please see each dataset page link provided for specific link.
Best partition using Louvain	<a href="https://bitbucket.org/taynaud/python-louvain">https://bitbucket.org/taynaud/python-louvain</a> <a href="http://perso.crans.org/aynaud/communities/">http://perso.crans.org/aynaud/communities/</a> <a href="https://sites.google.com/site/findcommunities/">https://sites.google.com/site/findcommunities/</a>
Python timeit module	<a href="http://pymotw.com/2/timeit/">http://pymotw.com/2/timeit/</a> <a href="https://docs.python.org/2/library/timeit.html">https://docs.python.org/2/library/timeit.html</a>
Graph Clustering and Community Finding	Comp-47270 lectures → Section 4 Dr. Neil Hurley
Hierarchical Clustering	<a href="http://en.wikipedia.org/wiki/Hierarchical_clustering">http://en.wikipedia.org/wiki/Hierarchical_clustering</a> Web Science Summer School 2011 <a href="http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hierarchy.dendrogram.html">http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hierarchy.dendrogram.html</a>
Girvan & Newman's Algorithm	
General overview	<a href="http://scikit-learn.org/">http://scikit-learn.org/</a>
Laplacian - amended	Used Lab 2 file as amended on the day
Laplacian	<a href="http://www.networkx.org">http://www.networkx.org</a>
Laplacian - kmeans	<a href="http://scikit-learn.org/">http://scikit-learn.org/</a> <a href="http://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.vq.kmeans.htm">http://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.vq.kmeans.htm</a>
Different algorithms and queries	<a href="http://wiki.cns.iu.edu/">http://wiki.cns.iu.edu/</a>