Data Analytics Assignment 2

Kalyan Reddy, 21361

1 Implementation summary

1.1 Importing data

1. `import_facebook_data(data_path)`:
   - Reads data from a file specified by `data_path`.
   - Using with open() parsing the file, splitting each line into lists based on spaces and removes duplicate edges returns a np array containing unique edges.
   - Dataset Details:
     Number of unique edges: 88234
     Number of nodes: 4039
2. `import_bitcoin_data(data_path)`:
   - Reads CSV data from a file specified by `data_path`.
   - Unweighting and undirecting the graph i.e, extracts the first two columns from the CSV and converts unique values in these columns to integers and creates a mapping.
   - Sorts and removes duplicates edges and returns a list of unique, edges.
   - In this dataset the nodes are numbered from 1 to 6005 and some are missing . To make all node numbers are continuous integers they are mapped from 1,..,6005 to 0,...,5881.
   - Dataset Details:(after unweighting and undirected)
     Number of unique edges: 21492
     Number of nodes: 5881

1.2 Spectral Decompostion Implementation

1.2.1 One Iteration

1. `spectralDecomp_OneIter(nodes_connectivity_list)`:
   - Takes a list of node connectivity, `nodes_connectivity_list`, as input and creates an adjacency matrix, degree matrix(D), and Laplacian matrix(L).
   - Solves Lx=λDx i.e, generalized eigen value probelm using scipy.linalg.eigh and sorts the eigenvalues and eigenvectors.
   - Extracts the second smallest eigenvector and uses its sign of each value to assign nodes to one of two communities and returns the second smallest eigenvector (`fielder_vec_fb`), adjacency matrix (`Adjacency_matrix`), and graph partition (`graph_partition`).
   - If the number of communities formed is not equal to 2 i.e, if all the eigen vector values are positive or all are negative, the function returns `None` for all values.

1.2.2 Multi Iteration

1. `spectralDecomposition_multi_iter(nodes_connectivity_list)`:
   - Calls the `spectralDecomp_OneIter` function to perform spectral decomposition and community detection.

- Checks if the fielder vector is `None`(which means only one community is formed instead of 2). If it is, returns an empty list.
- Calculates the differences between consecutive values in the sorted fielder vector and determines whether to proceed with further iterations based on stopping criteria(which is written later in this report).
- Divides the network into sub-networks and recursively calls itself on each sub-network.
- Merges the results from sub-networks into a sorted graph partition.

2. **`spectralDecomposition(nodes_connectivity_list)`:**
   - Initiates the spectral decomposition process by calling `spectralDecomposition_multi_iter` which return final graph partition
   - Prints the number of communities and their IDs and computes the sorted adjacency matrix for plotting based on the graph partition
   - Visualizes the resulting graph partition using the `plot_graph_vis` function and above adjacency matrix and returns the sorted graph partition.

1.3 Louvain Algorithm Implementation

Louvain Algorithm has been implemented as a Python class, 'Louvain_Algorithm', implementing the Louvain community detection algorithm. Below are the some key details and attributes:

1. `__init__(self, nodes_connectivity_list)`: Initializes the Louvain algorithm class. Constructs a graph and initializes various attributes related to the input data. Determines the dataset name based on the number of nodes. Calculates the adjacency matrix, degree matrix and graph neighbors.

2. `get_Adjacency_matrix_scaled(self, nodes_connectivity_list_fb, Node_count)`: - Computes the scaled adjacency matrix based on the input connectivity list.This is useful for calculating q_merge and q_demerge faster than with regular(unscaled) adjacency matrix.

3. `get_degree_matrix(self, A)`: - Computes the degree matrix(stored as a degree vector as it is diagonal) based on the adjacency matrix.

4. `get_Adjacency_matrix_plot(self, nodes_connectivity_list_fb, Node_count)`: - Computes the adjacency matrix for plotting purposes.

5. `plot_adjacency_matrix_sorted(self, graph_partition)`: - Plots a sorted adjacency matrix as an image. - Saves the plot with a filename determined by the dataset name.

6. `plot_graph_vis(self, graph_partition)`: - Visualizes the graph based on the adjacency matrix for plotting. - Colors nodes according to the provided partition and saves the plot with an appropriate filename.

7. `get_modularity(self)`: - Computes the modularity of the current partition.

8. `get_sigma_in(self, community_nodes)`: - Computes the sum of internal edges within a community.

9. `get_q_merge(self, moving_community, node)`: - Calculates the change in modularity when merging a node into another community.

10. `get_q_demerge(self, node)`: - Calculates the change in modularity when removing a node from its current community.

11. `get_best_community(self, node, neighbour_communities, maximum_q, q_demerging_value, best_community)`: - Determines the best community for a node to maximize modularity.

12. `Run_LA_Phase1(self)`: - Executes the first phase of the Louvain algorithm, iteratively optimizing modularity by moving nodes between communities and when gets stopped when changes become 0(i.e, no change of nodes between communities will increase in modularity) it returns `graph_partition`

2 Visualization Implementation Details

2.1 Spectral Decomposition Plots

2.1.1 One Iteration

The below functions are called in `plot` function which plots all 3 plots after `spectralDecomp_OneIter`:

1. `plot_fielder_vector(fielder_vector, dataset_name):`
   - Plots a scatter plot of sorted values from the `fielder_vector` against a range of x-values(nodes).
   - The function saves the plot as "Fielder_Vector_Facebook.png" or "Fielder_Vector_bitcoin.png" based on the `dataset_name` input.
2. `plot_adjacency_matrix_one_iter(nodes_list, fielder_vector, dataset_name):`
   - Sorts the `fielder_vector` values and rearranges rows and columns of Adjacency Matrix accordingly.
   - Plots the sorted adjacency matrix and saves the plot as "Facebook_Adj_matrix_spectral_sorted_one_iter.png" or "Bitcoin_Adj_matrix_spectral_sorted_one_iter.png" based on the `dataset_name` input.
3. `plot_graph_vis_one_iter(A, graph_partition, dataset_name):`
   - Converts the input matrix `A` which is Adjacency matrix to a np array and constructs a graph `G` using NetworkX with `A` as the adjacency matrix.
   - Extracts node colors from the `graph_partition` array, assigns them to `node_colors` and generates a visualization of the graph using NetworkX, with nodes colored according to `node_colors`.
   - Depending on the `dataset_name`, it saves the plot as "Facebook_Graph_visualization_spectral_one_iter.png" or "Bitcoin_Graph_visualization_spectral_sorted_one_iter.png".

2.1.2 Multi Iterations

The below functions are called in `SpectralDecomposition` function

1. `plot_adjacency_matrix(graph_partition, nodes_connectivity_list):`
   - Obtains an adjacency matrix `A` using a separate function `get_Adjacency_matrix_plot` with `nodes_connectivity_list` as input.
   - Sorts the indices of `graph_partition` based on the second column and Sorts the array `sorted_g` based on the sorted indices.
   - Creates a copy of `A` called `A_new` and rearranges its rows and columns according to `sorted_g` and generates a `A_new`.
   - Depending on the size of `A`, it saves the plot as "Facebook_Adj_matrix_spectral_sorted.png" or "Bitcoin_Adj_matrix_spectral_sorted.png".
   - Finally, it returns `A_new`.
2. `plot_graph_vis(A, graph_partition):`
   - Constructs a graph `G` with `A` as the adjacency matrix.
   - Computes node positions and generates a visualization of the graph using NetworkX, with nodes colored according to `graph_partition`.
   - Depending on the size of `A` i.e, dataset, it saves the plot as "Facebook_Graph_visualization_spectral.png" or "Bitcoin_Graph_visualization_spectral.png".

2.2 Louvain Algorithm Plots

Louvain Algorithm has been implemented in a **class Louvain_Algorithm** for ease of understanding/implementation

1. `plot_adjacency_matrix_sorted(self, graph_partition):`
   - Sorts the indices of `graph_partition` based on the second column and sorts `graph_partition` based on the sorted indices.
   - Obtains a new adjacency matrix `A_new` using the sorted indices Adjacency matrix and plots `A_new` as an image.

- Depending on the dataset name (`self.dataset_name`), it saves the plot as "facebook_Adj_matrix_lovian.png" or "bitcoin_Adj_matrix_lovian.png".

2. `plot_graph_vis(self, graph_partition):`

- Constructs a graph `G` using adjacency matrix and converts the input `graph_partition` into a np array and extracts node colors from `graph_partition`.
- Generates a visualization of the graph using NetworkX, with nodes colored according to `node_colors`.
- Depending on the dataset name (`self.dataset_name`), it saves the plot as "louvain_facebook_plot.png" or "louvain_bitcoin_plot.png".

3 Visualization Results

3.1 Facebook Dataset

3.1.1 Spectral Decompostion

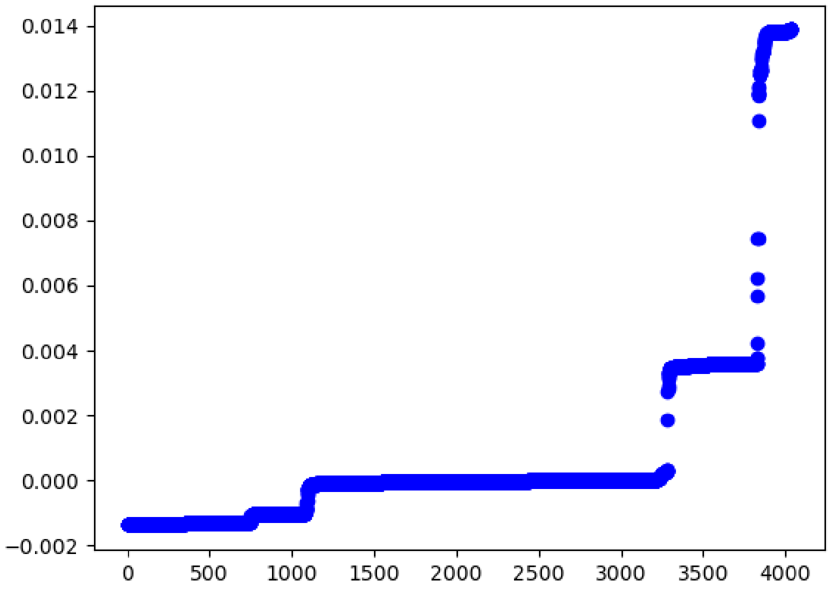

Figure 1: Fielder Vector

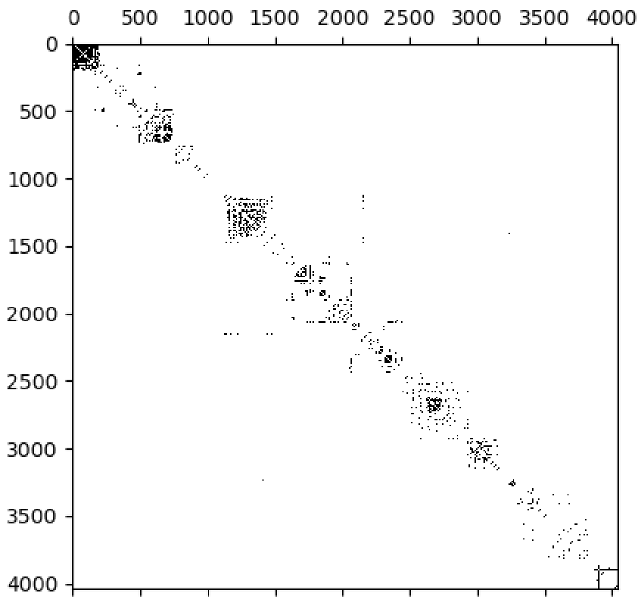

Figure 2: Adjacency matrix for one iter

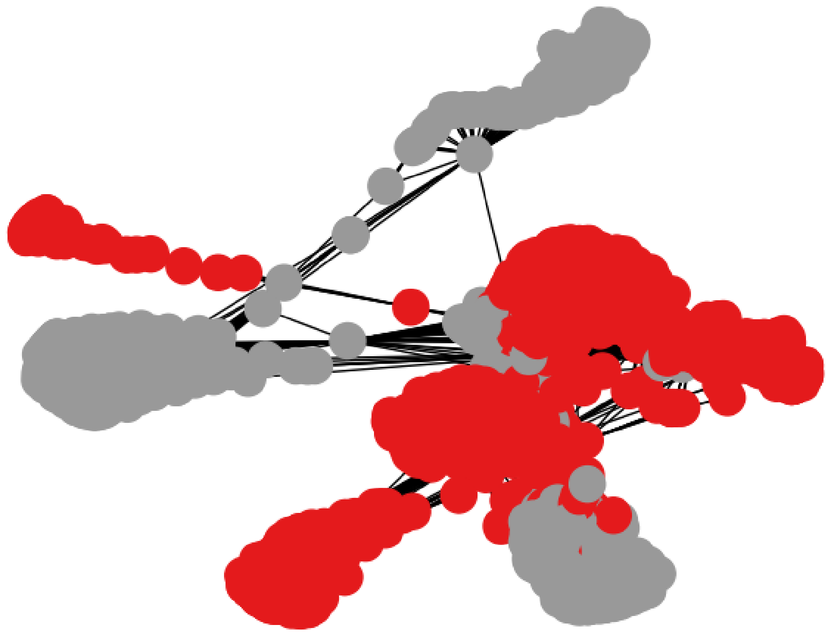

Figure 3: Graph Visualization for one iter

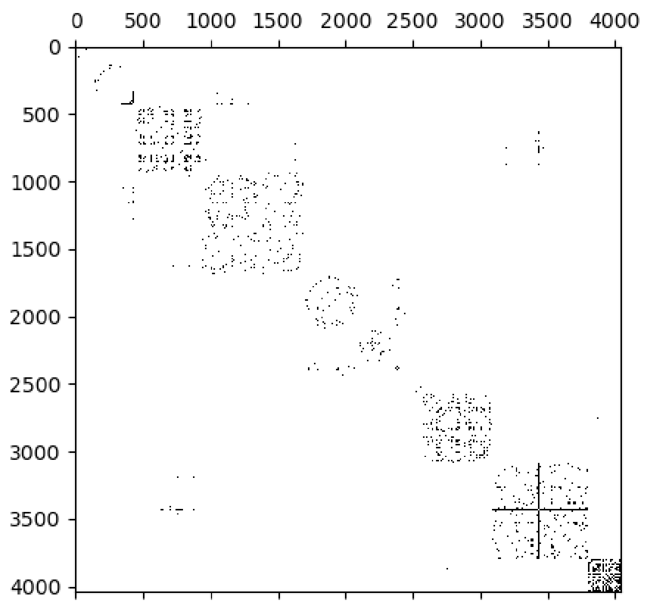

Figure 4: Adjacency matrix for multi iter

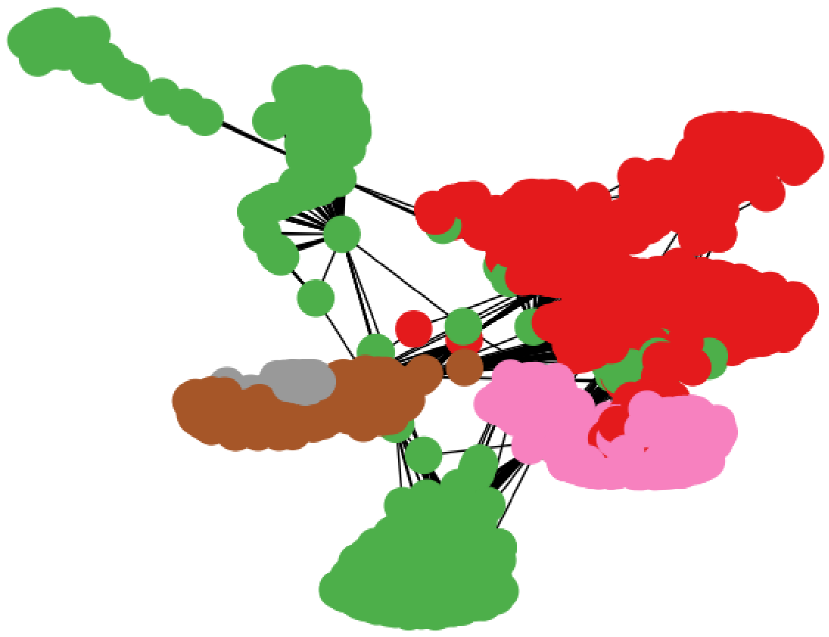

Figure 5: Graph Visualization for multi iter

3.1.2 Louvain Algorithm

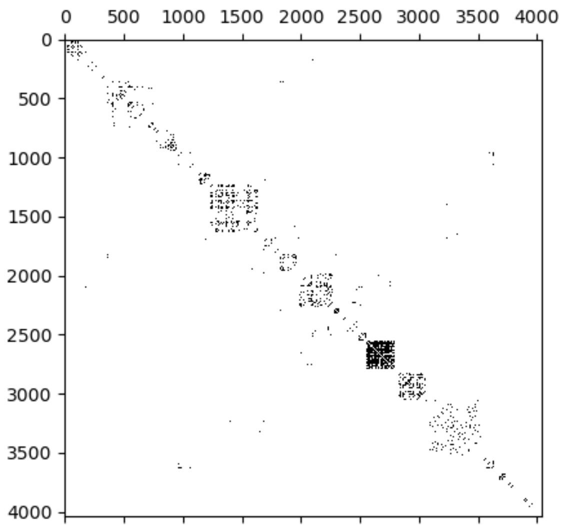

Figure 6: Adjacency Matrix for Louvain Algorithm

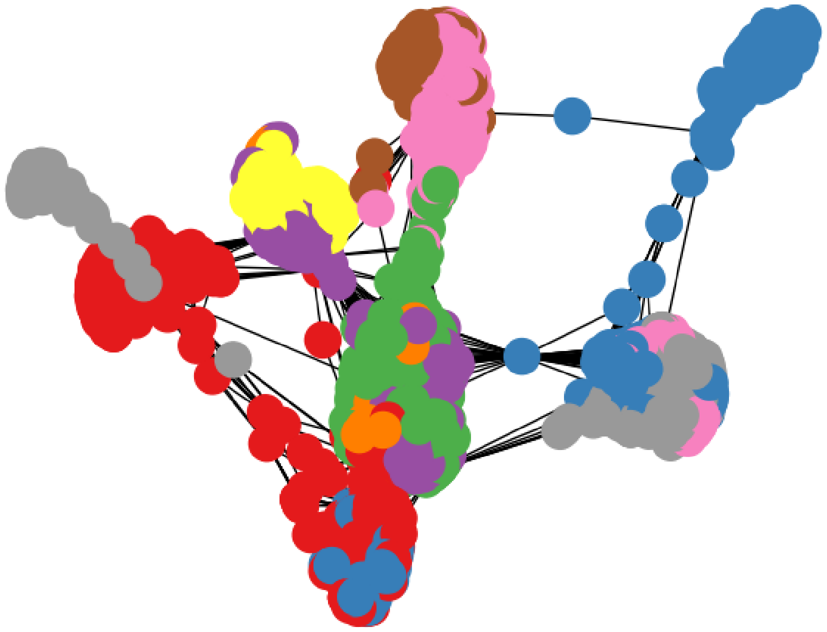

Figure 7: Graph Visualization for Louvain algorithm

3.2 Bitcoin Dataset

3.2.1 Spectral Decompostion

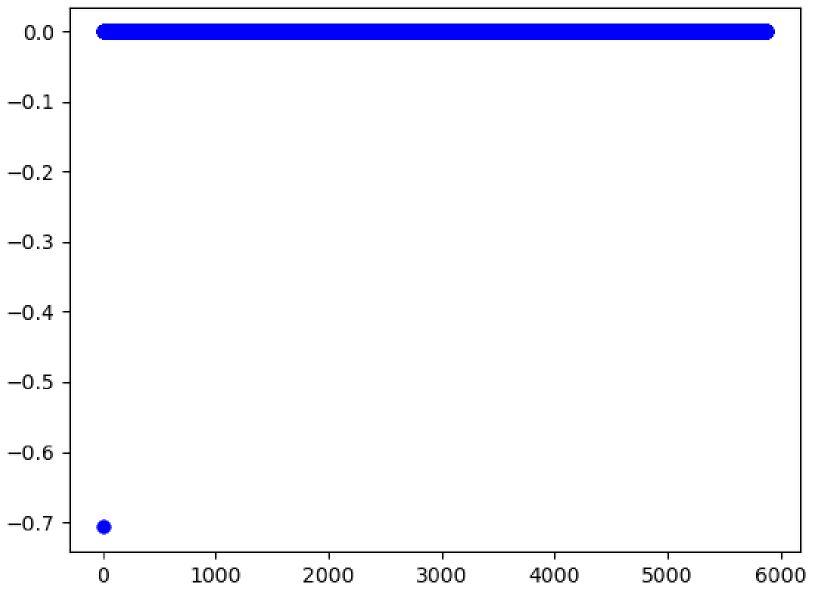

Figure 8: Fielder Vector

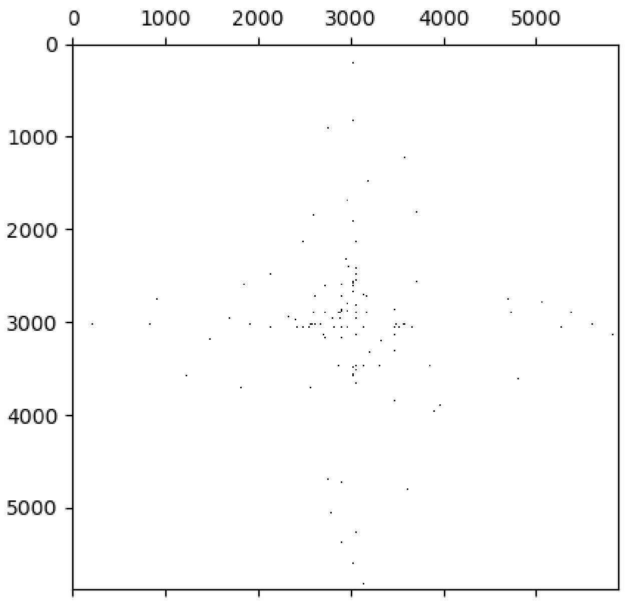

Figure 9: Adjacency matrix for one iter

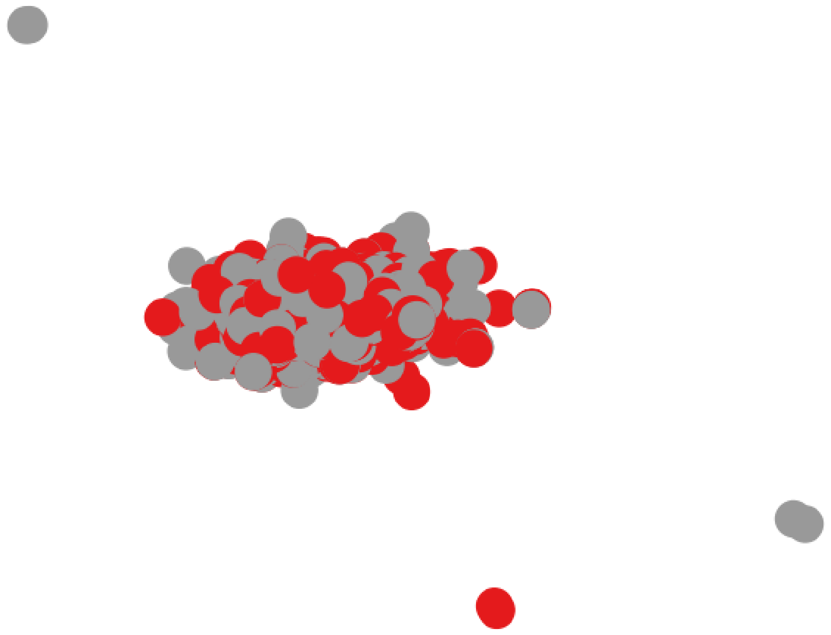

Figure 10: Graph Visualization for one iter

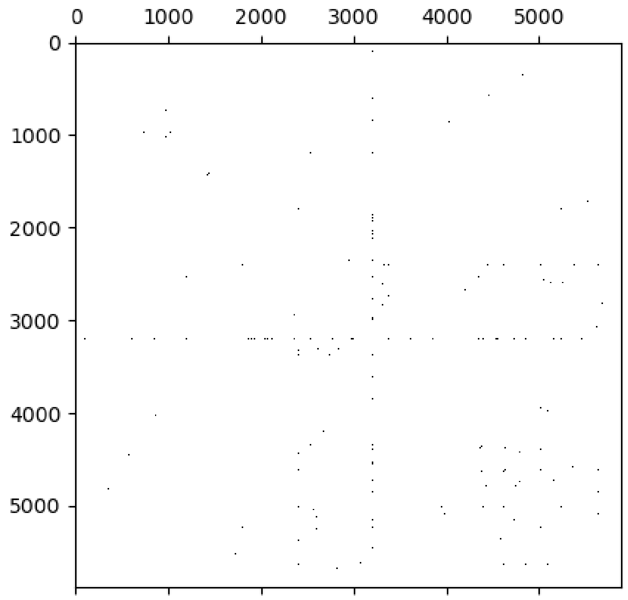

Figure 11: Adjacency matrix for multi iter

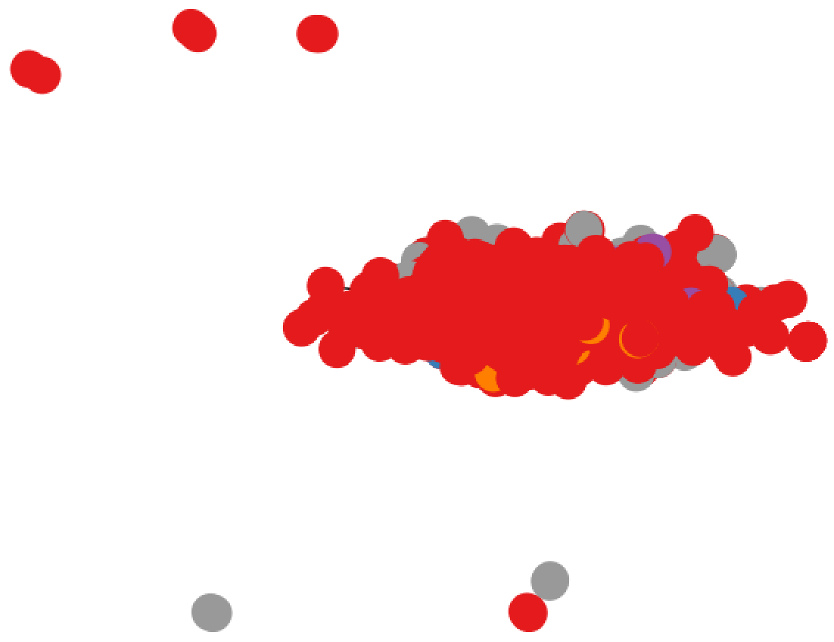

Figure 12: Graph Visualization for multi iter beta=25

3.2.2 Louvain Algorithm

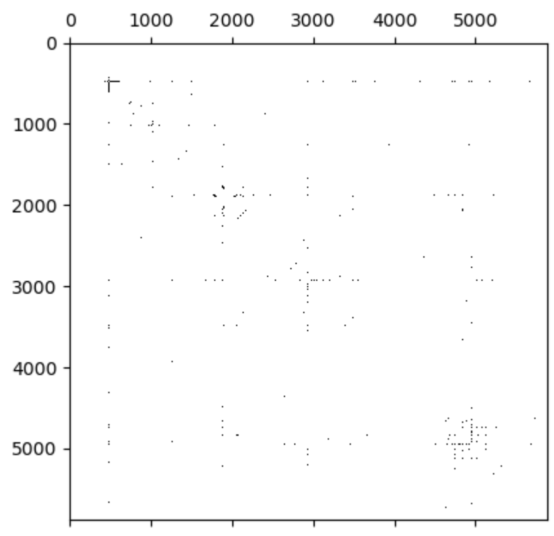

Figure 13: Adjacency Matrix for Louvain Algorithm

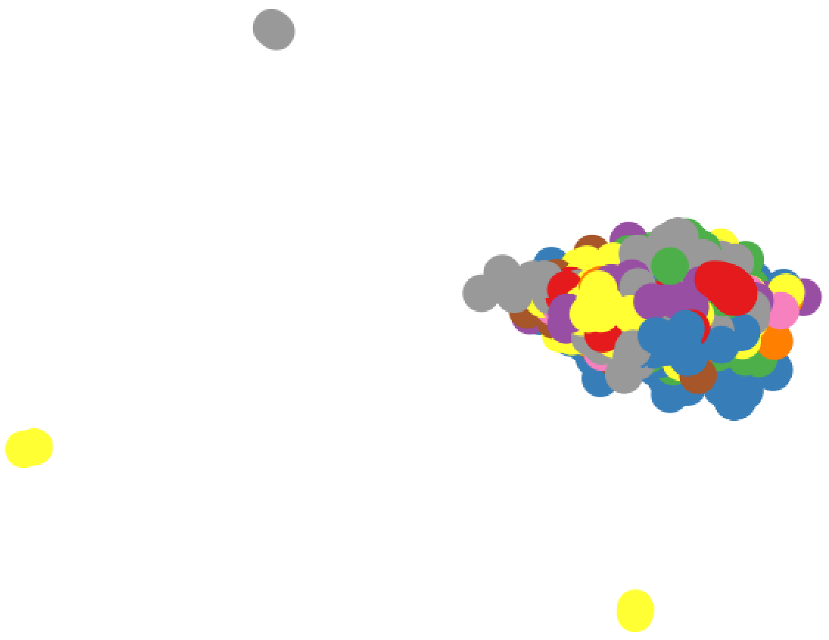
Figure 14: Graph Visualization for Louvain algorithm

4 Results

(Q2)
Stopping Criterion:

One of the possibilities I have used:

If the maximum difference (`diff_value_max`) among the calculated differences is less than β times the mean difference, then the algorithm stops and returns `graph_partition`.

In simpler terms, this stopping criterion is used to check if the differences between values in `sorted_fielder_vector` are relatively small compared to their average (`mean`). If they are, it suggests that the data has relatively little variation between consecutive values, and the algorithm stops. If the differences are large compared to the mean, the algorithm continues its iterations.

The choice of `beta` (I have used 200 and 25) determines how strict the stopping criterion is. We can adjust `beta` to control when the algorithm should stop based on the variation.

Hence, from the above, a higher `beta` leads to fewer communities.

For `beta` = 200, the following results are achieved:

Dataset	No. of Communities	Modularity of obtained graph partition	Final Community IDs
Facebook	10	0.795	[0, 1, 34, 58, 107, 472, 594, 1465, 1505, 1917]
Bitcoin	21	0.071	[0, 1, 2, 3, 6, 10, 11, 13, 17, 19, 29, 31, 40, 47, 75, 103, 279, 524, 586, 960, 5381]

Table 1: Results of Spectral Decomposition

For `beta` = 25, the following results are achieved:

Dataset	No. of Communities	Modularity	Final Community IDs
Facebook	100	0.434	[0, 1, 2, 3, 4, 6, 19, 23, 34, 35, 58, 78, 107, 166, 350, 352, 353, 357, 472, 573, 576, 583, 594, 596, 641, 686, 687, 688, 689, 698, 708, 717, 729, 897, 899, 900, 901, 906, 907, 909, 913, 920, 921, 922, 923, 925, 926, 927, 959, 960, 961, 966, 979, 999, 1002, 1054, 1135, 1405, 1465, 1505, 1577, 1684, 1726, 1912, 1914, 1915, 1916, 1917, 1918, 1921, 1923, 1924, 1927, 1930, 1934, 1939, 1942, 1943, 1956, 1968, 1970, 1974, 2081, 2661, 2662, 2663, 2664, 2670, 2672, 2673, 2679, 2681, 2683, 2697, 2718, 2805, 3437, 3980, 3981, 3983]
Bitcoin	112	0.075	[0, 1, 2, 3, 6, 10, 11, 13, 14, 16, 17, 18, 19, 24, 26, 29, 31, 40, 42, 47, 50, 54, 56, 64, 68, 69, 72, 75, 91, 98, 103, 105, 116, 134, 145, 147, 148, 167, 175, 178, 187, 206, 211, 214, 216, 243, 244, 257, 264, 279, 285, 292, 299, 316, 335, 347, 368, 375, 377, 414, 424, 426, 429, 440, 508, 524, 547, 584, 586, 660, 716, 721, 858, 902, 921, 960, 963, 1011, 1021, 1055, 1127, 1166, 1189, 1197, 1204, 1230, 1379, 1461, 1501, 1794, 1998, 2028, 2042, 2097, 2195, 2302, 2464, 2720, 2724, 2744, 3396, 3487, 3527, 3958, 4216, 4482, 4519, 4738, 5186, 5381, 5384, 5487]

Table 2: Results of Spectral Decomposition

(Q4- Louvain Algorithm-Results)

Dataset	No. of Communities	Modularity	Final Community IDs
Bitcoin	397	0.443	[12, 33, 46, 61, 143, 208, 213, 345, 404, 410, 445, 470, 493, 512, 526, 531, 558, 563, 579, 595, 596, 605, 624, 638, 644, 657, 658, 665, 676, 678, 687, 698, 706, 711, 727, 733, 796, 798, 809, 842, 852, 873, 889, 903, ...5177, 5193, 5262, 5270, 5272, 5296, 5322, 5327, 5335, 5348, 5354, 5359, 5362, 5396, 5426, 5470, 5485, 5518, 5537, 5553, 5558, 5561, 5575, 5624, 5645, 5648, 5688, 5706, 5718, 5726, 5729, 5764, 5775, 5815, 5818, 5823, 5830, 5834, 5835, 5847, 5877]
Facebook	101	0.8105	[42 69 95 135 151 152 174 179 182 189 192 201 241 256 278 282 300 307 351 382 401 453 505 572 612 642 650 702 757 831 852 865 877 882 895 1016 1044 1069 1111 1178 1208 1224 1240 1245 1253 1486 1586 1627 1657 1698 1748 1760 1775 1829 1892 1905 2113 2167 2168 2401 2424 2431 2435 2536 2565 2595 2721 2792 2801 2817 2902 2917 3003 3012 3122 3194 3290 3310 3311 3407 3543 3581 3585 3598 3641 3657 3668 3675 3676 3699 3700 3742 3787 3792 3846 3885 3925 4007 4012 4018 4029]

Table 3: Results of Louvain Algorithm

(Q5) How would you pick the best decomposition of nodes into communities? Below are some evaluation criteria which we can use:

- Modularity: Measures the quality of a partition based on the density of connections within communities compared to connections between communities. Maximized modularity indicates a good community structure.
- NMI (Normalized Mutual Information): Measures the similarity between the true community structure (if available) and the detected structure.
- Coverage: The proportion of nodes included in communities. High coverage may be important if we want to capture most of the network.

(Q6) Running Times for Algorithms (Time taken in seconds):
For Facebook Data:

1) For Spectral Decomposition: 21.753588438034058 seconds

1) For Louvain Algorithm: 2.69828200340271 seconds

For Bitcoin Data:

1) For Spectral Decomposition: 34.1813149452209 seconds

1) For Louvain Algorithm: 8.517959117889404 seconds

Note that the above numbers may vary slightly during runtime.
Inference: From the above, we can infer that the Louvain algorithm is faster than spectral decomposition (7 times faster for Facebook and 4 times faster for Bitcoin data).

(Q7) In your opinion, which algorithm gave rise to better communities, and why?
Louvain Algorithm has done a better job in my opinion due to:

- It took less time to run.(from above)
- We have not run Louvain to the full iterations (we have run only one iteration), and still, it has higher modularity(from the table above) compared to Spectral Decomposition.