CS-E4600 — Graph Partition

Jun Zhang 801940 Zhiheng Qian 765714
<jun.1.zhang@aalto.fi> <zhiheng.qian@aalto.fi>

December 6, 2019

1 Introduction

This report is for the programming project of course Algorithmic Methods of Data Mining at Aalto University. In this project, our task is to design and implement our own method to partition a graph into communities. We use pre-processed graphs from the Stanford Network Analysis Project (SNAP). Given an undirected graph $G = (V, E)$ and an integer $k > 1$, we need to partition the set of vertices V into k communities $V_1, ..., V_k$, so that$\cup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset$ for all $i \neq j$. We want the communities $V_1, ..., V_k$ to be as much separated from each other as possible. We also want that the communities have roughly equal size. Thus, we will evaluate the goodness of a partition $V_1, ..., V_k$ using the objective function

$$\phi(V_1, ..., V_k) = \sum_{i=1} k \frac{|E(V_i, \overline{V_i})|}{|V_i|}$$

where for $S, T \subseteq V$ with $S \cap T = \phi$ we define $E(S, T)$ to be the set of edges of G with one endpoint in S and the other endpoint in T, i.e., $E(S, T) = (u, v) \in E | u \in S \; and \; v \in T$, and we also define $\overline{V_i} = V \backslash V_i$.

At the first glance, the objective function is almost identical to the RatioCut (except the scaling factor 1/2 !). So our idea is to solve this problem using spectral clustering. We first study in detail how and why exactly spectral clustering can be used to minimize RatioCut. We later explored a couple of ways to optimize the general spectral clustering and do some comparisons.

The report is structured as follows: Section 2 gives the literature review and some fundamental concepts in order to understand the algorithm. Section 3 describes the algorithm and the explorations in detail. Section 4 presents the results and finding of the experiments. Finally, we give the conclusions in section 5 and attach a table of individual contributions in section 6.

2 Literature Review

Spectral clustering is widely used to solve clustering problems related to graph. Specifically, it can be used to identify communities of nodes in a graph based on the edges connecting

them. The general approach of spectral clustering is to define Laplacian matrix of the graph, embedded the vertices using the eigenvectors of the Laplacian matrix and use a traditional clustering method such as k-means on the relevant eigenvectors of the Laplacian matrix. In this project, we use [1] and [2] our main references. In the remaining part of this section, we will introduce some very important concepts.

2.1 Adjacency Matrix

Adjacency matrix is one way to represent a graph. It uses row and column indices to represent graph nodes, and the entries to represent if there is an edge or the weight of the edge between nodes. In the case of the graph that has more vertices and less edges, we can find the adjacency matrix of the graph is very sparse and has a lot of 0. Adjacency matrix should have a diagonal with only zero if there is no self-loop in the graph.

General spectral clustering problem which intends to group the points has three common methods to construct adjacency matrix of a graph: ϵ-nearest neighbor, k-nearest neighbor and fully connected. However, in this case, we are given a graph whose edges are know. So the weight between two points are 1 if there is direct point between them and 0 if not. In this way, we construct the adjacency matrix W.

2.2 Degree Matrix

Given a undirected graph $G = (V, E)$, the degree of a vertex v_i equals to how many edges it connects with. The degree matrix D of the graph is defined as the diagonal matrix with the degrees $d_1, ..., d_n$ on the diagonal.

2.3 Laplacian Matrix

We define the unnormalized Laplacian matrix of a graph as

$$L = D - W$$

Each element of the Laplacian matrix of G is

$$L_{ij} = \begin{cases} d_i & if\ i = j \\ -1 & if\ (i,j) \in E, i \neq j \\ 0 & if\ (i,j) \notin E, i \neq j \end{cases}$$

The good thing about Laplacian Matrix is that it has many useful properties that we can use in the spectral clustering. We summarizes the most important properties as follow:

- For every vector $f \in \mathbb{R}^n$, we have

$$f'Lf = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(f_i = f_j)^2$$

.

- L is symmetric and positive semi-definite.
- The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector 1.
- L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$.

3 Method

There are three basic algorithms in the spectral clustering family that use unnormalized and normalized Laplacian matrices. In this case, in order to minize the objective value, we need to use unnormalized spectral clustering. Given the dataset of edges (v_i, v_j), where $i, j \in N$, the first step is to construct adjacency matrix. To attend the competition, we implemented different algorithm for different graph. In the case of large dataset such as roadNet-CA, sparse matrix has to be used to represent the graph, which can save a huge amount of memory.

3.1 Algorithm: Unnormalized spectral clustering

Let A_i for $i = 1, ..., n$ be the clusters/classes of the vertices set. Let's define *cut* between A_i and $\overline{A_i}$ as

$$cut(A_i, \overline{A_i}) = W(A_i, \overline{A_i})$$

The ratiocut is defined as

$$ratiocut(A_1, ..., A_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A_i})}{|A_i|}$$

which is basically the objective function multiply by 2. To solve the RatioCut minimization problem, we define k indicator vectors

$$h_{ij} = \begin{cases} 1/\sqrt{|A_j|} & if\ v_i \in A_j \\ 0 & otherwise \end{cases}$$

where $(i = 1, ..., n; j = 1, ..., k)$. Then for $h_i^T L h_i$, we get

$$
\begin{aligned}
h_i^T L h_i &= \frac{1}{2}\sum_{m=1}\sum_{n=1} w_{mn}\left(h_{im}-h_{in}\right)^2 \\
&= \frac{1}{2}\left(\sum_{m\in A_i, n\notin A_i} w_{mn}\left(\frac{1}{\sqrt{|A_i|}}-0\right)^2 + \sum_{m\notin A_i, n\in A_I} w_{mn}\left(0-\frac{1}{\sqrt{|A_i|}}\right)^2\right) \\
&= \frac{1}{2}\left(\sum_{m\in A_i, n\in A_i} w_{mn}\frac{1}{|A_i|} + \sum_{m\in A_i, n\in A_i} w_{mn}\frac{1}{|A_i|}\right) \\
&= \frac{1}{2}\left(\operatorname{cut}\left(A_i,\bar{A}_i\right)\frac{1}{|A_i|} + \operatorname{cut}\left(\bar{A}_i,A_i\right)\frac{1}{|A_i|}\right) \\
&= \frac{\operatorname{cut}\left(A_i,\bar{A}_i\right)}{|A_i|}
\end{aligned}
$$

So for k subsets, it's RatioCut:

$$
\operatorname{RatioCut}\left(A_1,A_2,\ldots A_k\right) = \sum_{i=1}^{k} h_i^T L h_i = \sum_{i=1}^{k}\left(H^T L H\right)_{ii} = \operatorname{tr}\left(H^T L H\right)
$$

where $\operatorname{tr}\left(H^T L H\right)$ is the trace of the matrix. Thus, minimizing the Ratiocut also means to minimize the trace. Notice that $H^T H = I$, so we need to minimize

$$
\underbrace{\arg\min}_{H} \operatorname{tr}\left(H^T L H\right) \text{ s.t. } H^T H = I
$$

This is the standard form of a trace minimization problem, and again a version of the Rayleigh-Ritz theorem (e.g., see Section 5.2.2.(6) of Lutkepohl, 1997) tells us that the solution is given by choosing H as the matrix which contains the k smallest eigenvectors of L as columns.The shape of H is $n \times K$ where n indicates the number of vertices and K indicates the number of community. Each row of H is the embedding of the corresponding vertex. Note that it is a NP hard problem that the entries of solution vector (i.e. eigenvector) h_i is only allow to take two particular values. The most obvious relaxation in this setting is to discard the discreteness condition and instead allow that h_i takes arbitrary values in R. This allow us to compute the solution vectors much more efficiently. However, this also breaks the assumption that we uses to get the conclusion

$$
\operatorname{RatioCut}\left(A_1,A_2,\ldots A_k\right) = \operatorname{tr}\left(H^T L H\right)
$$

So we needs to use kmeans algorithm on the rows of H to group them into K clusters.

The following is the pseudocode of the algorithm:

- Construct the adjacency matrix of the given graph
- Construct the degree matrix D
- Compute the unnormalized Laplacian matrix L.
- Compute the k smallest eigenvectors $u_1, ..., u_k$ of L.

- Form matrix $H \in \mathbb{R}^{nk}$ with columns vectors $h_1, ..., h_k$
- For $i = 1, ..., n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i - th$ row of H, y_i is actually the embedding of vertex i.
- Cluster the points $\{y_i\}_{i=1,...,n}$ into clusters $A_1, ..., A_k$

3.2 Implementation and Exploration

In this section we will walk through our implementation. This includes the implementation of the baseline algorithm with Kmeans and some explorations using find the maximum elements of the embedding of the each vector and other clustering methods to give the labels. As we discuss in the last section, the problem of using spectral clustering methods to partition the graph is that we discard the discreteness condition when computing the eigenvectors. So the eigvenvectors we computed may not satisfy the assumption. This is also why we need to use KMeans to do the clustering. But the other hand, this part also leave some space for us to potentially optimize the problem. We include some of our explorations in the subsection 3.2.4.

3.2.1 Constructing the adjacency matrix

At the beginning, we use dense matrix to represent the graphs, which is sufficient for ca-GrQc, Oregon-1 graph that are small. When we attempt to use dense mxtrix to represent the larger graph, the ram is running out due to huge memory usage. Hence, we use sparse matrix to construct graph:

```
# Construct the sparse matrix for the graph
graph_matrix = sparse.lil_matrix((vertices_n, vertices_n))
def adjacent_matrix_construction(x,y):
  # Here we ignore the self-loop edges
  if x!=y:
    graph_matrix[x, y] = 1
    graph_matrix[y, x] = 1
edges_df = pd.read_csv(filepath, sep = '', skiprows=1, header=None, names=['V
edges_df.apply(lambda x: adjacent_matrix_construction(x[0],x[1]), axis=1)
```

3.2.2 Construct the Laplacian matrix

We use scipy library to calculate Laplacian graph:

```
# Construct the unnormalized Laplacian matrix
Laplacian_graph = csgraph.laplacian(graph_matrix, normed= False)
```

3.2.3 Calculate eigenvectors

To minimize the ratiocut, according to Rayleigh-Ritz theorem, the solution is given by choosing H as the matrix which contains the k smallest eigenvectors of L as columns. Hence, it is sufficient to calculate the k smallest eigenvectors.

```
# Calculate the first k eigenvectors
vals, vecs = linalg.eigs(Laplacian_graph, k=k, sigma = 0)
vecs = np.real(vecs)
```

Note that in the library scipy, if input parameter "sigma" is set, we are telling the program to find eigenvalues near sigma using shift-invert mode. As the eigenvalues has to be great or equal 0. We can find k smallest eigenvalues and the corresponding the eigenvector.

3.2.4 Clustering the points into clusters

In this part, we first try with Kmeans.

Then we tried with another method called $findMax$, which is basically finding the maximum elements in the embedding for each vertex and assign the vertex to the corresponding class. The motivation behind this is that, if the vectors we compute is close to the "true vector" (i.e vector contain elements from discrete space), then we can assume that maximum element of each embedding is close to $1/\sqrt{|A_j|}$. In the subsection 3.1 we know that if we follow the assumption, for $x_j \in y_i$(embedding of vertex i) of U, we have

$$x_j = \begin{cases} 1/\sqrt{|A_j|} & if\ v_i \in A_j \\ 0 & otherwise \end{cases}$$

Then we can assign the vertex to class in corresponding to the maximum element in the element.

The second method we explored is agglomerative clustering. The main idea of agglomerative clustering is that we let each point to be a unique cluster in the first iteration and merge them according to the similarity between them until we have K clusters. We use agglomerative clustering from sklearn. By default, it uses ward as the linkage rule. It basically chooses to merge the clusters that minimize the increment of the variance in all the clusters.

The third method we explored is Mini Batch KMeans. It is one variant of standard KMeans. The main difference of Mini Batch Kmeans is to sample a small batch for training in each iteration instead of using the whole dataset. It speeds up the training process a lot but also decreases the performance a bit. Consider that we have some huge graphs, it will be worthy if a relative small decrement of the performance can be exchanged into a huge speedup of computing.

```
# k-means clustering
kmeans = KMeans(n_clusters=k, n_init=20, max_iter=500).fit(vecs)
classes, class_numbers = np.unique(kmeans.labels_, return_counts=True)

# agglomerative Clustering
clustering = AgglomerativeClustering(n_clusters=k).fit(vecs)
classes, class_numbers_agg = np.unique(clustering.labels_, return_counts=True

# Mini batch k-means clustering
clustering_MBKmeans = MiniBatchKMeans(n_clusters=k).fit(vecs)
classes, class_numbers_MBK = np.unique(clustering_MBKmeans.labels_, return_co
```

3.2.5 Calculate ratiocut

To test how our algorithms perform on the graph partition, we implemented the objective function of Ratiocut:

```
# Objective function
def objective(edges, labels, k, class_numbers):
    cut_edges = {}
    for i in range(k):
      cut_edges[i]= 0
    def accumulate(vertex1, vertex2):
      if labels[vertex1] != labels[vertex2]:
        cut_edges[labels[vertex1]] += 1
        cut_edges[labels[vertex2]] += 1
    edges.apply(lambda x: accumulate(x[0],x[1]), axis = 1)
    objective = 0
    for j in cut_edges:
      objective += cut_edges[j]/class_numbers[j]
    return objective
```

4 Results and Analysis

4.1 Results with Kmeans

We ran our program using the baseline algorithm with Kmeans on five required networks i.e., ca-GrQc, Oregon-1, roadNet-CA, soc-Epinions1, and web-NotreDame. The result is shown as Figure 1. Even though the equation takes into account that each community cannot be too small, we can see that the partition is not balanced in some of the experiments. We use standard deviation(STD) of the normalized partitions to measure the level of balance. In the first data set, the STD is up to 0.49 with a partition of 12 : 4146. The partition on network roadNet-CA is pretty balanced that the STD is below 0.01.

Dataset	ca-GrQc	Oregon-1	roadNet-CA	soc-Epinions1	web-NotreDame
Vertice number	4158	10670	1957027	75877	325729
Edge number	13428	22002	2760388	405739	1117563
K	2	5	50	10	20
STD	0.49711	0.38088	0.00673	0.29933	0.17553
Objective	0.0835745297	0.7219871624	0.2678069743	0.7548013939	0.0294942602

Figure 1: Experimental Results of five datasets with Kmeans

4.2 Compare with explorations

In section 3.2.4, we propose three additional methods for exploration. Note that in the experiments we also include the comparison with SpectralClustering(see the result in the last row of the table) from sklearn to see if our implementation is comparable to the that in sklearn. Due to the time issue, we mainly conducted the experiments on network ca-GrQc and Oregon-1, as Figure 2 and Figure 3. In the experiments on ca-GrQc, spectral clustering yielded the best result in terms of the objective. Methods with Kmeans and agglomerative clustering had the second best result. The objective of experiment with findMax is zero because it one community has zero vertex. This shows that the idea descirbed in section 3.2.4 won't hold, i.e. there is no guarantee that how close the vectors will be to the "true solution vectors". This is the same case for experiment on Oregon-1. In terms of standard deviation, it is quite similar no matter which method we use. We observe the same change in experiments on Oregon-1. A difference is now agglomerative clustering has the smallest objective value.

We notice that when we ran the experiments on Oregon-1, we got almost 2X speedup if we used Mini Batch Kmeans instead of Kmeans. It takes 0.045s for Mini Batch Kmeans to finish while it take 0.067s for Kmeans to finish. The cost for MiniBatchKmeans is a increment of objective value from 0.72 to 1.74, which is quite big.

We also did the experiments on roadNet-CA and web-NotreDame. While in roadNet-CA it took 13min 19s to finish the Kmeans subroutine but it only costed 30s to run the Mini Batch Kmeans! And the ojective value obtained with Mini Batch Kmeans is even better (0.269 vs 0.267)! But we noticed that the Mini Batch Kmeans is not very stable in terms of objective value. For experiment on web-NotreDame, Kmeans used 6.60s while Mini Batch Kmeans used 0.97s. The objective values are 0.029 and 0.309 for Kmeans method and Mini Batch Kmeans method respectively. So in general method with Kmeans still give a better result.

Network	Vertex Number	Edges Number	K	
ca-GrQc	4158	13428	2	
Name	**Objective**	**Class Number**	**Mean**	**STD**
Kmeans	0.08357452966714905	[4146 12]	0.5	0.49711399711399706
findMax	0	[0, 4158]	0.5	0.5
MiniBatchKmeans	0.19043111871030777	[4094 64]	0.5	0.4846079846079846
Agg	0.08357452966714905	[12 4146]	0.5	0.49711399711399706
Spectral	0.07572850898494414	[4118 40]	0.5	0.49037999037999036

Figure 2: Experimental Results on ca-GrQc with different methods

5 Conclusion

In this project, we did a comprehensive study on how to apply spectral clustering on graph partition problem. We implemented unnormalized spectral clustering with Kmeans on our

Network	Vertex Number	Edges Number	K	
Oregon-1	10670	22002	5	
Name	**Objective**	**Class Number**	**Mean**	**STD**
Kmeans	0.7219871623562948	[236 158 10260 6 10]	0.19999999999999998	0.3808766760636779
findMax	0	[10143, 0, 287, 194, 46]	0.2	0.3754278840170863
MiniBatchKmeans	1.738331181846053	[10113 216 150 45 146]	0.2	0.3739338644667905
Agg	0.6930296085066402	[10211 278 165 10 6]	0.2	0.3786116941472779
Spectral	0.7221762437912969	[288 9966 10 158 248]	0.2	0.36711944561153775

Figure 3: Experimental Results on Oregon-1 with different methods

own. We further proposed three additional methods in order to explore how we can optimize the algorithm, which apply findMax, agglomerative clustering, Mini Batch Kmeans on the eigenvectors we found respectively. Results show our implementation has more or less the result with that one in sklearn. And in general spectral clustering with kmeans still has the best performance in terms of the tradeoff between speed and objective.

6 Individual Contributions

We worked together on this project. The below shows our contribution.

Task	Jun Zhang	Zhiheng Qian
Algorithm	60%	40%
Implementation	55%	45%
Configuration	45%	55%
Report	40%	60%

Figure 4: Contribution to the project

References

[1] Ulrike von Luxburg,
A Tutorial on Spectral Clustering

[2] Steve Butler, Fan Chung,
Spectral Graph Theory