AMDM Project

Shaghayegh Safar (613183)
email shaghayegh.safar@aalto.fi

Kavya Sreekumar(796042)
email kavya.sreekumar@aalto.fi

December 7, 2019

1 Introduction

This project involves designing and implementing a graph partitioning method. The data set contains graphs from Standard Network Analysis Project (SNAP). The data set used for our project is the largest connected component in each graph. The objective of the project is to partition the input graph G with the node set V into well separated communities. This is achieved by minimizing the objective function:

$$\phi(V_1, V_2, ..., V_k) = \sum_{i=1}^{k} \frac{|E(V_i, \overline{V_i})|}{|V_i|}$$

where $V_1, V_2, ..., V_k$ are the different non-overlapping partitions obtained post the algorithm (note that $V = \cup_i^k V_i$). 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. We also define $\overline{V}_i$ as set of nodes in V other than those V_i. Minimizing the objective function implies that the number of edges going across the clusters is minimised. In addition to this, we also need to ensure that the graphs are well balanced (since there is a division by the size of each partition in the error function).

We minimize the objective function by computing laplacian matrix of the graph, followed by computing its second eigen vector i.e fiedler's vector and finally performing k-means on it to cluster the nodes in one dimension. We would be performing a modified version of k-means as an attempt to balance the graph and minimise our objective function.

2 Literature Review

Graph partitioning algorithms abstract a big graph into smaller components and are useful in a variety of applications from detecting communities, for example

in social media, pathological, or biological networks, to partitioning a huge power network area into several maximally independent partitions so that if one network failed, there would not be a power blackout (see [1] for a comprehensive review).

There are many algorithmic solutions for graph partitioning. In this section, we very briefly summarize some of the common ones that have been extensively discussed in [1]. In particular there are two family of solutions:

- **Global Algorithms**: These methods work with the entire graph and find the solution directly on that. As they are working with the entire graph, they usually cannot scale up to very big graphs. These methods can be divided into exact algorithms (the algorithms that finds the solution to graph partitioning optimally but have usually bad time complexities and cannot scale up), Spectral Partitioning (which was briefly discussed in the previous section), Graph Growing (that use breadth-first search types of algorithms), Flows types algorithms (that use max-flow min-cut theorem), streaming Graph Partitioning (that consider the inputs as data stream and therefore have low space complexity; enabling them to handle big graphs very fast but usually with higher error).

- **Iterative Improvement Heuristics**: These methods start with an initial solution and then try to sequentially improve the solution, for example by swapping neighbour nodes between two partitions. These local updates can follow different approaches that produce different algorithms.

- **Multilevel Graph Partitioning**: These methods create a hierarchy of different partitioning. For example at the first level all nodes are in one partition and then each levels divides the partitions appropriately based on a criterion.

In this project we use spectral algorithm which first project the graph in a continuous feature space and then use standard clustering algorithms such as K-means [2] to partition the data (nodes). K-means is not directly minimizing the objective function introduced in the previous section so we propose different modifications (similar to iterative improvement heuristics methods) with the hope to fill its shortcomings.

3 Implementation of Basic K-means Algorithm

3.1 Storing graphs

Any graph can be represented by its adjacency matrix. Adjacency matrix is a matrix which has value 1 when there is an edge between two nodes and value 0 when there are no edges between them. The graph datasets we are working with have size up to 2 million nodes and due to this reason, we would be using our adjacency matrices as sparse matrices to store our data. Sparse matrices are matrices that store only the location when an edge occurs along with the

value of the edge. These can be used when number of edges between the nodes is much less than the total number of possible edges between all the nodes in a graph. A quick analysis can show that this indeed is the case for our huge graphs in these data sets (see table 1 in the project description document).

3.2 Eigenvector of Laplacian Matrix

Before going ahead with the algorithm, let us define a laplacian matrix. Laplacian matrix is a matrix which has values in the ith row and the jth column as:

$$L[i,j] = \begin{cases} deg(i) & \text{if i = j} \\ -1 & else \end{cases} \tag{1}$$

We would be working with Laplacian instead of adjacency matrix because Laplacian gives information regarding the degree of each node as well, which is not directly conveyed in adjacency matrix. We do the spectral analysis using Laplacian matrix when it is an irregular graph and use adjacency matrix if it is d-regular graph. Taking this into account, we have irreguar graphs in our problem statement and thus would be using the Laplacian matrix. Further, Laplacian matrix has properties like it is positive semi-definite, its row sum is 0 and sum of the square of each element in a row equals to 1. We also know that the first smallest eigen vector is the vector [1,1,1..] and its eigen value is 0.
The next question comes whether we use normalized Laplacian or unnormalized Laplacian matrix. Unnormalized Laplacian serve in the approximation of the minimization of RatioCut, while normalized Laplacian serve in the approximation of the minimization of NCut. Basically, in the unnormalized case, you optimize an objective relative to the number of nodes in each cluster. And, in the normalized case, you optimize the objective relative to the volume of each cluster. Since we are trying to optimise the number of nodes in each cluster, we would be using unnormalised laplacian matrix.
So, once we have stored these graph in sparse matrix format, we compute the unnormalized laplacian matrix for the same using the formula:

$$L = D - A$$

where D is the diagonal matrix in which the diagonal element represents the degree of each node and A represents the adjacency matrix.
On computing Laplacian matrix, we find the second smallest eigen vector or fiedler's vector. This vector is a representation of how our nodes in a graph are converted to one dimensional space. Considering number of partitions as n, we have also looked into choosing smallest n eigen vectors (excluding trivial eigen vector with eigen value 0) through eigen decomposition. This means that we are converting the graph space into n dimensional space. This brought a slight improvement in our objective function but the time complexity was much more considering there were large graph datatsets. Thus we will be choosing the second eigen vector for the scope of this project. Please note that as per our code we would be using both first and second eigenvector to compute K-means

but since first eigen vector is trivial, no additional information is imparted into clustering.

3.3 Basic K-means Algorithm

Next, we apply basic K-means++ algorithm on the eigenvectors with 1000 iterations to the embedding of these graphs to form the required number of clusters. The time complexity of K-means algorithm is $O(|V|^2)$.

4 Implementation of Modified K-means Algorithm

As mentioned, K-means does not aim to directly minimize the objective error introduced at the beginning of this report, but rather it tries to assign each node to the cluster with the nearest mean. This characteristic of K-means is desirable also for objective function but unfortunately it ignores the fact that we are interested to have balanced clusters (since having balanced clusters will generally decrease the denominator of the objective function). Given this intuition, we implemented different modifications for the K-means algorithm that try to balance K-means. The list of ideas are as followings:

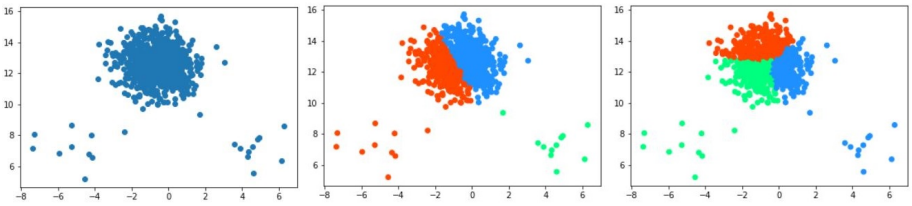
- **Idea 1**: Restricting the size of each cluster to be at maximum equal to number of nodes divided by number of partitions, $|V|/k$, in a standard K-means algorithm.
- **Idea 2**: Starting from a standard K-means clustering solution and then distributing the nodes from the largest cluster to the closest cluster to that node that has less than $|V|/k$ nodes.
- **Idea 3**: Changing the definition of distance to a cluster center by penalizing the distance with respect to the size of the cluster. In other words, when we are updating the cluster assignments, rather than assigning x to $argmin_{i=1,..,k} d(x, V_i)$ we use a the new criterion $argmin_{i=1,..,k} d(x, V_i) + \alpha|v_i|$, where α is an open parameter that needs to be carefully tuned. This is a regularized K-means which has an open parameter alpha indicating how much we care about balanceness of the data.
- **Idea 4**: defining a priority for each node and then assigning nodes based on the priority to the closest cluster that has less than $|V|/k$ elements.

Unfortunately, out of these prototype ideas, only the last one (idea 4) was able to produce sensible results on some test tasks. Implementation for all these ideas is available in the attached python notebook but from here on we only concentrate on the last idea.

The general idea is adapted from [4]. The algorithm starts by initializing the cluster centers based on K-means++ initialization idea which initializes cluster

centers from the data sequentially such that they are far away from each other[1]. Then the algorithm computes a priority value for each data being clustered in each cluster which is based on the distance of data to that cluster minus the distance of the data to the furthest away cluster. The lower this priority value, the better to cluster that data faster. Then, the data with the lowest priority to a cluster is selected and is assigned to that cluster. If a cluster reaches $|V|/k$ limit then, the data is assigned to the next best cluster. The process continues until all data are assigned to a cluster. In the next step, the cluster centers are updated based on the data in each cluster. This process repeats for certain number of iterations. Figure 1 shows a toy example of how this algorithm differs from a standard K-means algorithm in a simple 2D dataset.

Figure 1: Comparison between standard K-means (implemented in *sklearn.cluster* library in python; shown in the middle panel) to the proposed modified K-means that balances the clusters in (right panel). In the modified K-means all clusters have 270 data members while in K-means the numbers are 325, 474, and 11.



The algorithm is more expensive than the standard K-means as it requires to sort the data based on their priority values for cluster assignment phase, which if the sorting is implemented efficiently will add $O(|v|log|v|)$, with to the time complexity of standard K-means.

5 Results

Table 1 shows the objective function for the standard K-means and the modified one (introduced in the previous section) for the five given datasets.

[1]We used the implemented function in `https://www.geeksforgeeks.org/ml-k-means-algorithm/` for initialization.

Graph	**#Nodes**	**#Edges**	k	**K-means**	**K-means-modified**
ca_GrQc	4158	13428	2	0.0627	0.9370
oregon_1	10670	22002	5	0.72199	7.8932
soc_Epinions1	75877	405739	10	0.754801	71.607
web_NotreDame	325729	1117563	20	0.2491	11.53
roadNet-CA	1957027	2760388	50	0.524	NA

Table 1: The result of objective function for standard K-means and K-means-modified.

In K-means,we aimed in minimising the objective function but there was no balance between clusters. In K-means-modified , we attain equal number of nodes for each cluster and make an attempt to attain a minimum objective function at the same time and ended with these results. The NA value is missing since the computation was too expensive for our laptops and we ran out of time.

6 Conclusion

Graph partitioning is usually an NP-hard problem. For the objective function introduced in this report, it is very difficult to find the partitioning that would optimally minimize it. To tackle the problem we used spectral algorithm that maps the graph to a continuous feature space which allowed us to use K-means algorithm which is one of the widely used classical clustering algorithms. The problem with K-means (as shown in figure 1) is that it does not try to balance the data set while in our objective function, balanceness could have a positive effect. To overcome this, we proposed modified K-means algorithm that behaves similarly to K-means but enforces the clusters to be balanced.

Still, our results in the test data provided show that K-means performed the best reaching errors always smaller than one. For example for ca_GrQc our error is similar to the best error that was achieved in the online competition. Unfortunately, the modified K-means algorithm did not reach the desirable error values. This may be because, it forces the clustering to be exactly balanced, but in reality, the exact balanceness may not be optimal. Future works should consider soft balanceness which would allow a bit unbalanceness if it would decrease the objective.

7 Competition

We participated in the competition as the name "Jugaad" for four graphs: ca-GrQc (cost = 0.0627), oregan-1 (cost = 0.722), soc-Epinions1 (cost = 0.755), and web-NotreDame (cost = 0 249).

8 References

1. Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C. (2016). Recent advances in graph partitioning. In Algorithm Engineering (pp. 117-158). Springer, Cham.

2. MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 14, pp. 281-297).

3. `https://elki-project.github.io/tutorial/same-size_k_means`

4. `https://math.stackexchange.com/questions/1113467why-laplacian-matrix-need-normalization-and-how-come-the-sqrt-of-degree-matrix`

9 Acknowledgements

A special mention to the following people who have helped in making this project a success:

- Pedram Daee
- Alejandro Ponce de Leon
- Wen Xiang

Both authors contributed equally to implementation and writing the ideas.