Programming project - Graph Partitioning

Anssi Moisio
anssi.moisio@aalto.fi

Nikolas Erkinheimo
nikolas.erkinheimo@aalto.fi

Algorithmic Methods of Data Mining — December 20, 2018

Introduction

The topic of this programming exercise is graph partitioning. The datasets being used are provided by SNAP (Stanford Network Analysis project). We will use the following five undirected networks:

- ca-GrQc
- Oregon-1
- soc-Epinions1
- web-NotreDame
- roadNet-CA

The networks have substantial size differences: the amount of vertices range from about 4000 (ca-GrQc) to about 2000000 (roadNet-CA). We will be using mostly the two smallest graphs when validating our algorithms to save time. The graphs have defined values of k (numbers of clusters), that will be used in the competition, from 2 to 50. We will use these same k values throughout the project.

Graph partitioning is a classical NP-hard problem which means polynomial time algorithms for this problem may not even exist and at the very least have not been found yet. We will use spectral algorithms. First, we will generate the Laplacian matrix of all of the graphs after which we calculate the corresponding eigenvectors. We will then apply a clustering algorithm to these eigenvectors to partition the nodes into sets.

We used the following loss function:

$$\phi(V_1, ..., V_k) = \frac{|E(V_i, ..., V_k)|}{min_{1 \leq i \leq k}|V_i|} \tag{1}$$

Where the nominator corresponds to the amount of cut edges and the denominator is the size of the smallest cluster.

1 Literature review

Daniel Spielman [?] has written a good introduction to spectral graph theory. Spielman exaplains the basic algorithm of computing the laplacian of the adjacency matrix and then taking the eigenvectors of the laplacian and using these to form the clusters.

2 Implementation

We are using Python 3 as our implementation language along with the following packages: scikit-learn, networkx, numpy, scipy and pandas. We intend to use spectral algorithms for this problem.

The contributions of each team member:
The project was done mostly with both of us being in the same room, which means the planning and solution was very much a team effort with no clear division of work. That being said, there was some tasks that were mostly done by one team member. Nikolas wrote most of the code for the algorithm and Anssi wrote the code for plotting the figures. The project report was written mostly by Nikolas. Anssi wrote some of the introduction and describing the figures.

2.1 Data loading and preprocessing

The data is in an edgelist format which will need to be loaded from the filesystem. We used pandas to load the files as DataFrames which we converted into adjacency matrices using networkx. We used scipy to morph the adjacency matrix into a normed Laplacian matrix.

2.2 Clustering the eigenvectors

Every row represents a node in the graph. Now that the graph has been converted into a matrix we can apply numerical algorithms to it. We used k-means to cluster the eigenvectors into sets and calculated the loss function.

3 Results

From Figure 1 we can see that iterating the algorithm with new initializations can make the results better. For the smallest network the differences are not large (from 3.26 to 3.20) but with the larger networks there is noticeable differences (see Figure 3).

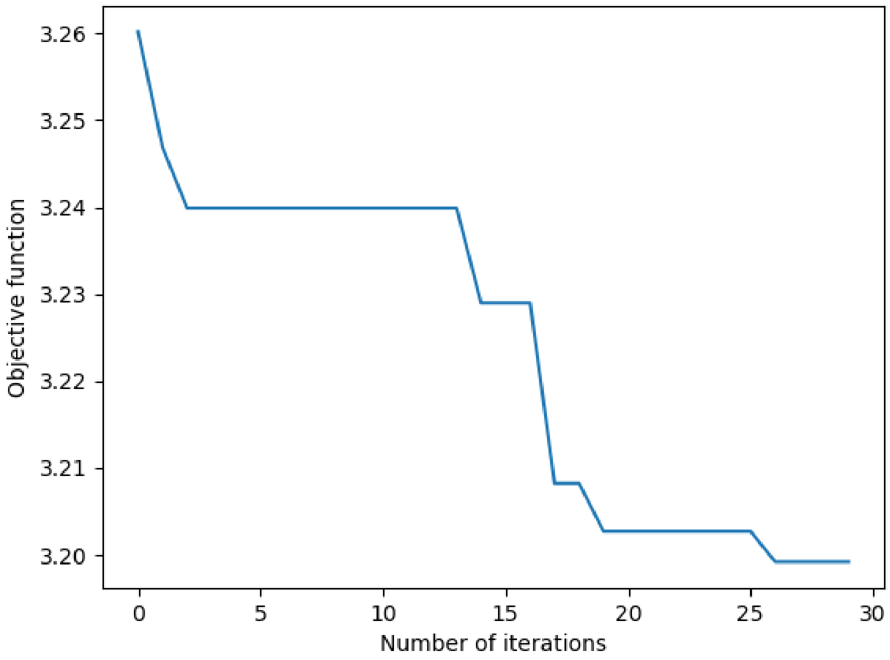

Figure 1: The objective function as a function of the number of iterations of the algorithm. The eigenvectors are normalized here. This figure is for the smallest graph ca-GrQc, with k=2.

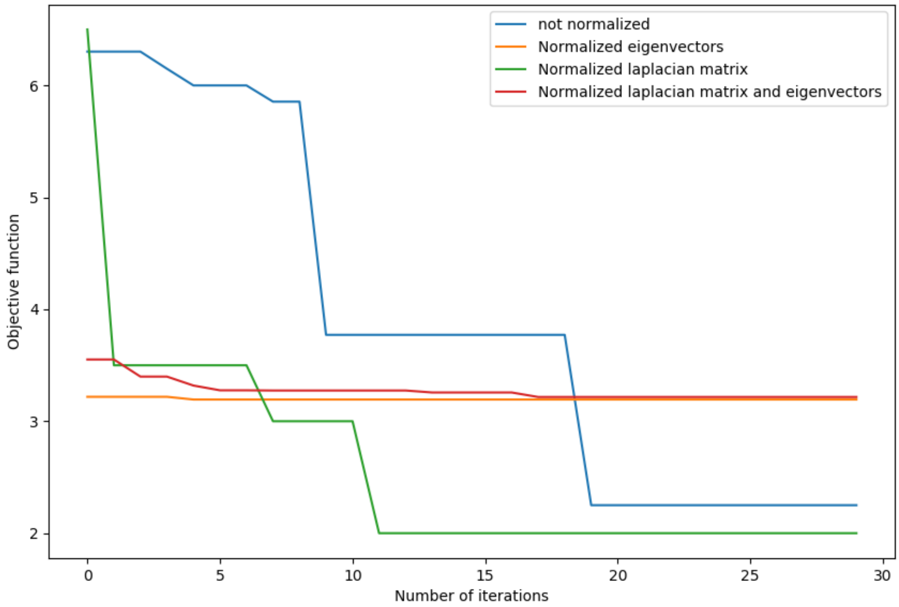

Figure 2: Results for normalized eigenvectors or Laplacian matrix or both This figure is for the smallest graph ca-GrQc, with k=2.

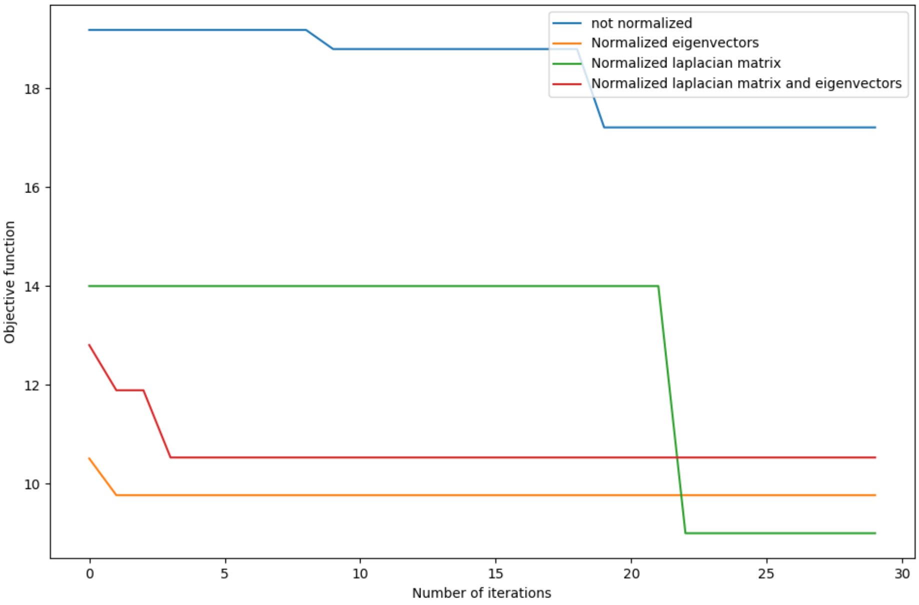

Figure 3: Results for normalized eigenvectors or Laplacian matrix or both. This figure is for the second smallest graph Oregon-1, with k=5.

We wanted to see that to what degree does normalization affect the end result. Our program has two boolean parameters which normalize different parts of the numerical data: the laplacian matrix and the eigenvector matrix. These parameters give us a total of four different permutations. We decided to plot the objective function with every parameter combination as a function of iterations.

From Figure 2 we can see that normalizing the eigenvectors makes the results stable in the smallest graph. Normalizing the Laplacian matrix gives the best results and normalizing the eigenvectors can actually make the results somewhat worse than not normalizing anything.

As can be seen from figure 3, the unaltered data performs quite poorly in graph Oregon-1 when compared with any normalization. Normalizing the eigenvectors generally yields better results with this graph given that the sample size is big enough and normalizing both the Laplacian matrix and the eigenvector matrix gave a slightly worse performance compared to only normalizing eigenvectors.

Looking at figures 2 and 3, normalizing the laplacian matrix will be sufficient given that enough iterations are applied in the clustering. Normalizing both Laplacian and the eigenvector matrix will result in faster convergence.

These graphs show us, however, that any data normalization will generally yield better results when compared to the raw data. This is related to the objective function as the data to be clustered can and usually is in a quite unbalanced state and in our case it results in a small denominatitor, making the objective function larger, even when the amount of cut edge count is a little smaller than after normalization.

4 Conclusions

Weird stuff

References

[1] Daniel Spielman, Spectral Graph Theory, Chapter 16, `http://www.cs.yale.edu/homes/spielman/PAPERS/SGTChapter.pdf`