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 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.

1.1 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.

1.2 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.

2 Results

Kuvaajat:

- $\phi(max\ iters)$
- $\phi_{min}(number\ of\ sampled\ kmeans\ results)$
- Effect of normalization on results

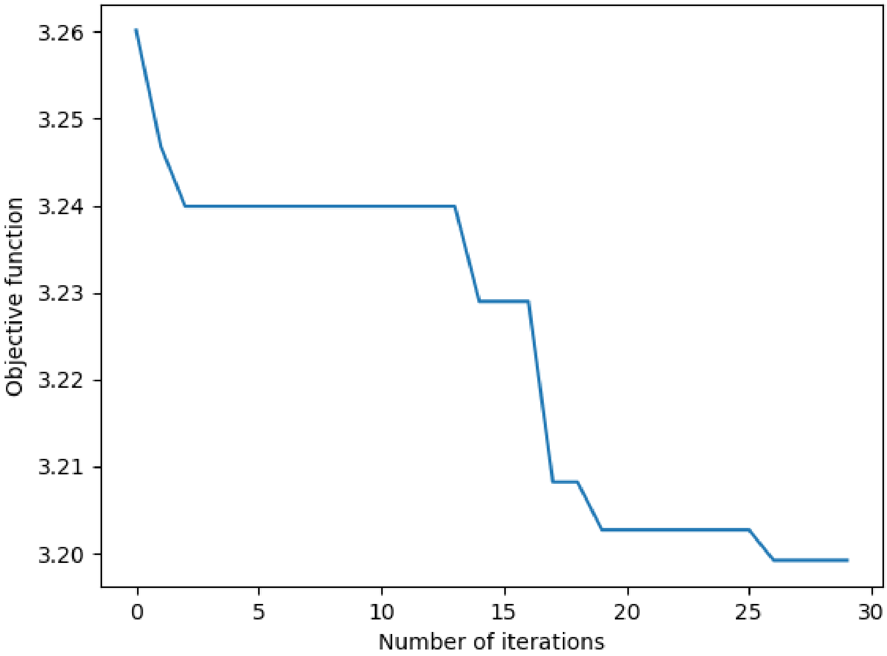

Figure 1: The objective function as a function of the number of iterations of the algorithm. The eigenvectors are normalized here. This figure is for the smallest graph ca-GrQc, with k=2.

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).

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

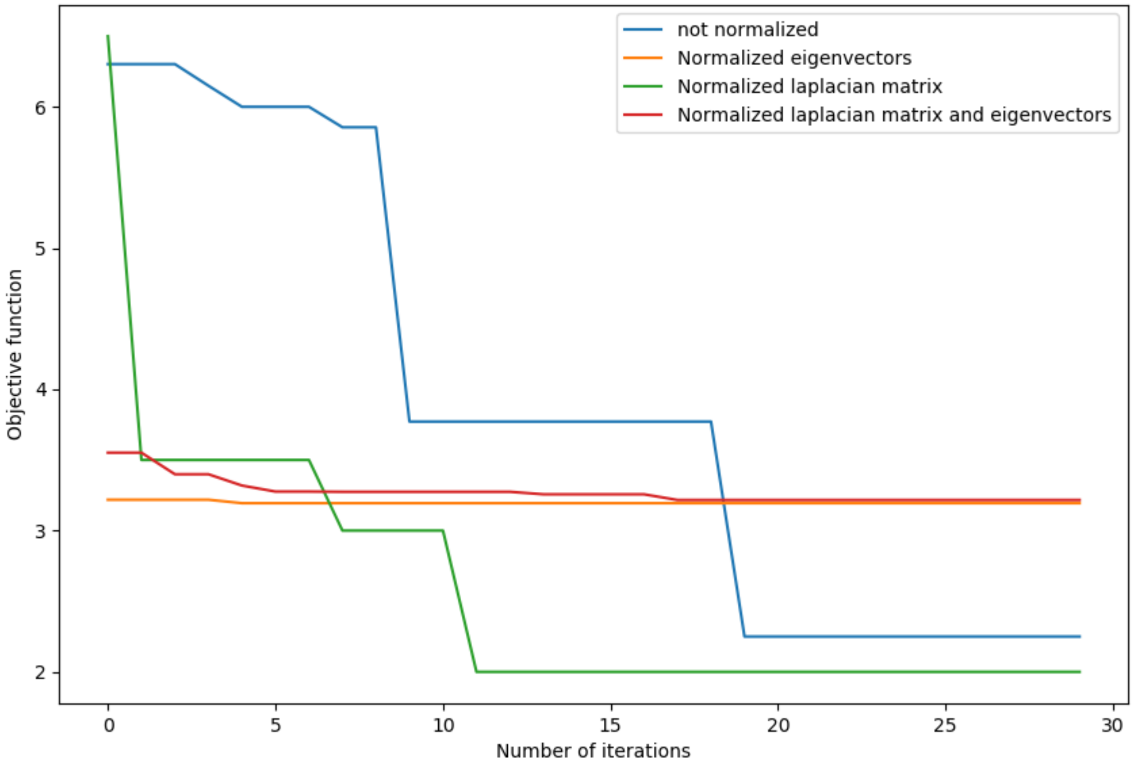

Figure 2: Results for normalized eigenvectors or Laplacian matrix or both This figure is for the smallest graph ca-GrQc, with k=2.

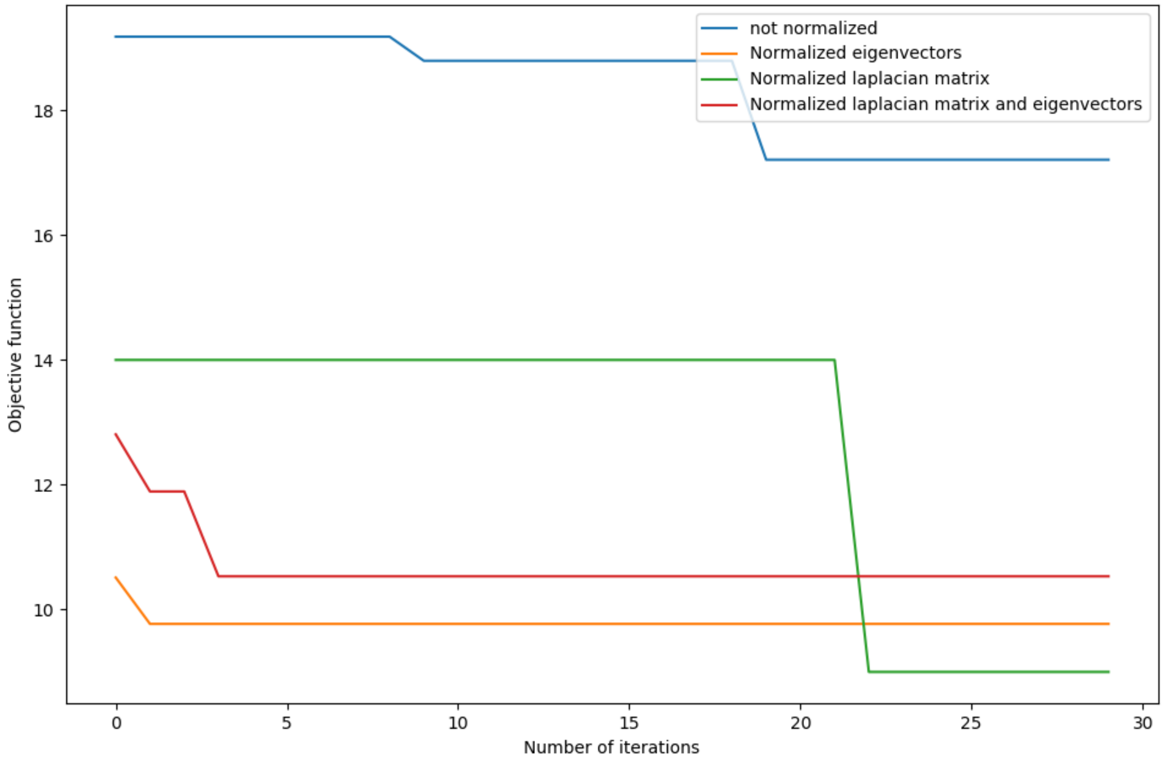

Figure 3: Results for normalized eigenvectors or Laplacian matrix or both. This figure is for the second smallest graph Oregon-1, with k=5.

3 Conclusions

Conclusions