Aalto School of Science

Aalto University
School of Science

CS-E4600 - Algorithmic Methods of Data Mining

Programming project

Antoine Louis (784915)
antoinebenoitnathancharles.louis@aalto.fi

Olivier Moitroux (784928)
olivier.moitroux@aalto.fi

Academic year 2019-2020

Contents

1 Introduction

The goal of the project is to implement our own method to partition a graph into multiple communities. To experiment our implementation, we used five undirected graphs from the Stanford Network Analysis Project (SNAP), namely `ca-GrQc, Oregon-1, roadNet-CA, soc-Epinions1` and `web-NotreDame`.

Given an undirected graph $G = (V, E)$ and an integer $k > 1$, our task is to partition the set of vertices V into k communities $V_1, ..., V_k$, so that $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset$, for all $i \neq j$, and the communities V_i are as much separated as possible.

By following the suggestion of the statement, we investigated the spectral clustering algorithm with all its variants, explained in Section 2, to achieve the task. Then, we tried to improve it in order to optimize the objective function. Our implemented methods are detailed in Section 3, and the corresponding experiments in Section 4.

2 Literature review

Spectral clustering has first been introduced by the work of Donath and Hoffman (1973), who suggested to construct graph partitions based on eigenvectors of the adjacency matrix. In the same year, Fiedler (1973) discovered that bi-partitions of a graph are closely related the second eigenvector of the graph Laplacian, and he suggested to use this eigenvector (named the Fiedler vector) to partition a graph. Since then, spectral clustering has re-discovered and extended many times in different fields. In the machine learning community for example, spectral clustering has been made popular mainly by the works of Shi and Malik (2000) and Ng et al. (2001).

The success of spectral clustering is mainly due to the better results they offer compared to more traditional clustering algorithms such as k-means or single linkage. Moreover, spectral clustering can be implemented efficiently even for large data sets, and can be solved by standard linear algebra methods, ensuring that there are no issues of getting stuck in local minima or restarting the algorithm for several times with different initializations.

In this section, we first introduce the necessary vocabulary and notations that will be used in the rest of this report. Then, we explain the basic spectral bi-partition algorithm presented by Fiedler and its recursive extension for k-clustering. Finally, we detail the different spectral-partition algorithms using multiple eigenvectors.

2.1 Notation and Preliminary terms

Let $G = (V, E)$ be an undirected graph with a set of vertices V of size $|V| = n$, and a set of edges E. Vertices will each be assigned a unique positive integer i, where $1 \leq i \leq n$. The degree of a vertex $i \in V$, written $d(i)$, is defined as the number of edges incident on i, i.e. the number of edges $(i, j) \in E$ where $j \in V$. Edges connect vertices and are labeled by their endpoints, i.e. for $i, j \in V$, $(i, j) \in E$ if an edge connects vertices i and j.

Let's now define three matrices that are derived from the graph G. The *adjacency matrix* $A(G)$ of the graph is a square symmetric matrix of size $n \times n$ such that

$$A(G) = \{a_{ij}\} \text{ where } a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

The *degree matrix* $D(G)$ of the graph is a square diagonal matrix D of size $n \times n$ such that

$$D(G) = \{d_{ij}\} \text{ where } d_{ij} = \begin{cases} d(i) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Finally, the *Laplacian matrix* $L(G)$ of the graph is a square matrix of size $n \times n$ that can be computed in three different ways:

1. *Unormalized Laplacian matrix*: $L_u(G) = D(G) - A(G)$
2. *Normalized Laplacian matrix*: $L(G) = I - D^{-\frac{1}{2}}(G)A(G)D^{-\frac{1}{2}}(G)$
3. *Normalized "random-walk" Laplacian matrix*: $L_{rw}(G) = I - D^{-1}(G)A(G)$

2.2 Basic spectral-partition algorithm

The use of spectral methods to compute edge separators in graphs was first considered by Donath and Hoffman, who suggested using the eigenvectors of adjacency matrices of graphs to find partitions. Later, Fiedler associated the second smallest eigenvalue of the Laplacian matrix with its connectivity and suggested partitioning by splitting vertices according to their value in the corresponding eigenvector. This algorithm, known as the spectral bi-partioning, is quite simple.

For a graph $G = (V, E)$ which is to be partitioned, the Laplacian matrix $L(G)$ is formed, and the eigenvector-eigenvalue pairs of $L(G)$ are computed. After labeling the eigenvalues so that $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$, the second eigenvector $\mathbf{x_2}$ (i.e., the one corresponding to the second-smallest eigenvalue λ_2), known as the Fiedler vector, is selected and sorted according to its coefficient values. Then, to form the partition, the optimal sweeping cut S (i.e., cut that respect the order) is chosen as it minimizes $\phi(S)$. Such cuts might for example be the median value of the vector, or the frontier between positive and negative values. Finally, all vertices whose value in sorted $\mathbf{x_2}$ is from one side of he cut are assigned to the first cluster V_1, and the others to V_2. The pseudo-code of the algorithm is presented in Algorithm 1.

Algorithm 1 Spectral bi-partitioning

```
    Input: Adjacency matrix $A(G) \in \mathbb{R}^{n \times n}$ of a graph $G$.
    Output: Clusters $V_1$ and $V_2$ of graph $G$.
1: function SPECTRAL-BIPARTITIONING($A(G)$)
2:     Form the degree matrix $D(G)$.
3:     Form the unormalized laplacian matrix $L_u(G)$.
4:     Compute the eigenvectors $\mathbf{x_2}$ of $L_u(G)$ (Fiedler vector).
5:     Order vertices according to their coefficient value on $\mathbf{x_2}$.
6:     Consider only sweeping cuts (i.e., cuts that respect the order).
7:     Take the sweeping cut $S$ that minimizes $\phi(S)$.
8:     return $V_1$ and $V_2$ according to the cut.
```

Now, this algorithm can be used to partition a graph in k clusters. One simply needs to run it recursively on each partitions until k clusters were created. The pseudo-code of this recursive algorithm is detailed in Algorithm 2. However, it has been shown that this algorithm was quite inefficient and unstable. Hence, alternatives were proposed, and are described in the next section.

Algorithm 2 Recursive spectral bi-partitioning

```
    Input: Adjacency matrix $A(G) \in \mathbb{R}^{n \times n}$ of a graph $G$, number $k$ of clusters to construct.
    Output: Clusters $V_1, ..., V_k$ of graph $G$.
1: function RECURSIVE-SPECTRAL-BIPARTITIONING($A(G), k$)
2:     Apply SPECTRAL-BIPARTITIONING($A(G)$) to find $V_1$ and $V_2$.
3:     if ($k/2 > 1$) then
4:         Apply RECURSIVE-SPECTRAL-BIPARTITIONING($A(V_1), \frac{k}{2}$)
5:         Apply RECURSIVE-SPECTRAL-BIPARTITIONING($A(V_2), \frac{k}{2}$)
6:     return $V_1, ..., V_k$
```

2.3 Spectral-partition algorithm with multiple eigenvectors

Another approach consists in using k eigenvectors instead of just the Fielder vector. However, there exists three different methods that are quite similar except that they use different versions of the Laplacian matrix. The different algorithms presented below were all implemented and tested, results are described in Section 4.

2.3.1 Unormalized spectral clustering

The idea behind unormalized spectral clustering is quite straightforward. The first step consists in forming the degree matrix $D(G)$ of the graph and use it to compute the unormalized laplacian $L_u(G)$. Next, the first k eigenvectors of the matrix L_u are calculated. Note that by "the first k eigenvectors", we refer to the eigenvectors corresponding to the k smallest eigenvalues. The matrix $\mathbf{X}$ is then formed by concatenating these column vectors (first column corresponds to the eigenvector of the smallest eigenvalue). By representing each node i as the vector $\mathbf{y}_i$ being simply the i-th row of the matrix $\mathbf{X}$, the vectors $\mathbf{y}_i$ are clustered into k different partitions with a clustering algorithm such as k-means for example. The pseudo-code of the algorithm is presented in Algorithm 3.

Algorithm 3 Unormalized spectral clustering

```
    Input: Adjacency matrix $A(G) \in \mathbb{R}^{n \times n}$ of a graph $G$, number $k$ of clusters to construct.
    Output: Clusters $V_1, ..., V_k$ of graph $G$.
1: function UNORMALIZED-SPECTRAL-CLUSTERING($A(G), k$)
2:     Form the degree matrix $D(G)$.
3:     Form the unormalized laplacian matrix $L_u(G)$.
4:     Compute the first $k$ eigenvectors $\mathbf{x}_1, ..., \mathbf{x}_k$ of $L_u(G)$.
5:     Form the matrix $\mathbf{X} \in \mathbb{R}^{n \times k}$ with columns $\mathbf{x}_1, ..., \mathbf{x}_k$.
6:     for ($i = 1, ..., n$) do
7:         Consider the $i$-th row of $\mathbf{X}$ as point $\mathbf{y}_i \in \mathbb{R}^k$, $i = 1, ..., n$.
8:         Cluster the points $(\mathbf{y}_i)_{i=1,...,n}$ into clusters $C_1, ..., C_k$ (e.g., with k-means clustering).
9:     return $V_1, ..., V_k$ with $V_i = \{j | y_j \in C_i\}$
```

2.3.2 Normalized spectral clustering (Shi and Malik, 2000)

The algorithm presented below is nearly the same as Algorithm 3, except that it uses normalized random-walk laplacian matrix L_{rw} instead of the unormalized laplacian.

Formally, the transition probability of jumping in one step from vertex i to vertex j is given by $p_{ij} = \frac{a_{ij}}{d(i)}$. The transition matrix $P = (p_{ij})_{i,j=1,...,n}$ of the random walk is thus defined by $P = D^{-1}A$. Hence, one can notice that there is a tight connection between P and L_{rw}, as $L_{rw} = I - D^{-1}A = I - P$. In this way, spectral clustering can be interpreted as trying to find a partition of the graph such that the random walk stays long within the same cluster and seldom jumps between clusters.

Algorithm 4 Normalized spectral clustering (Shi and Malik, 2000)

```
    Input: Adjacency matrix $A(G) \in \mathbb{R}^{n \times n}$ of a graph $G$, number $k$ of clusters to construct.
    Output: Clusters $V_1, ..., V_k$ of graph $G$.
1: function NORMALIZED-SPECTRAL-CLUSTERING-SHI($A(G), k$)
2:     Form the degree matrix $D(G)$.
3:     Form the normalized random-walk laplacian matrix $L_{rw}(G)$.
4:     Compute the first $k$ eigenvectors $\mathbf{x}_1, ..., \mathbf{x}_k$ of $L_{rw}(G)$.
5:     Form the matrix $\mathbf{X} \in \mathbb{R}^{n \times k}$ with columns $\mathbf{x}_1, ..., \mathbf{x}_k$.
6:     for ($i = 1, ..., n$) do
7:         Consider the $i$-th row of $\mathbf{X}$ as point $\mathbf{y}_i \in \mathbb{R}^k$, $i = 1, ..., n$.
8:         Cluster the points $(\mathbf{y}_i)_{i=1,...,n}$ into clusters $C_1, ..., C_k$ (e.g., with k-means clustering).
9:     return $V_1, ..., V_k$ with $V_i = \{j | y_j \in C_i\}$
```

2.3.3 Normalized spectral clustering (Ng et al., 2001)

Once again, the algorithm presented here is very similar to the two previous ones, except that it uses the normalized laplacian matrix $L(G)$, and introduces an additional row normalization step after computing the matrix $\mathbf{X}$. This step comes to counteract the fact that for the matrix L, the eigenvectors in the ideal situation are not indicator vectors. If the degrees of some vertices are particularly low, the corresponding entries in the eigenvectors are very small, near to zero. The problem arises when some perturbation is introduced in the eigenvectors, meaning that the vectors will usually not have any non-zero components any more, but many near-zero ones. Then, it becomes unclear how we should interpret a value near zero in the eigenvector k: is it a result of the noise and the node does not belong to cluster k, or the node belongs well to the cluster but its small degree makes its value close to zero? The normalization step of each row in the matrix $\mathbf{X}$ allows to solve this problem. The pseudo-code of the normalized spectral clustering is described in Algorithm 5.

Algorithm 5 Normalized spectral clustering (Ng et al., 2001)

```
    Input: Adjacency matrix $A(G) \in \mathbb{R}^{n \times n}$ of a graph $G$, number $k$ of clusters to construct.
    Output: Clusters $V_1, ..., V_k$ of graph $G$.
1: function NORMALIZED-SPECTRAL-CLUSTERING-NG($A(G), k$)
2:     Form the degree matrix $D(G)$.
3:     Form the normalized laplacian matrix $L(G)$.
4:     Compute the first $k$ eigenvectors $\mathbf{x}_1, ..., \mathbf{x}_k$ of $L(G)$.
5:     Form the matrix $\mathbf{X} \in \mathbb{R}^{n \times k}$ with columns $\mathbf{x}_1, ..., \mathbf{x}_k$.
6:     Normalize $\mathbf{X}$ so that rows have norm 1.
7:     for ($i = 1, ..., n$) do
8:         Consider the $i$-th row of $\mathbf{X}$ as point $\mathbf{y}_i \in \mathbb{R}^k$, $i = 1, ..., n$.
9:         Cluster the points $(\mathbf{y}_i)_{i=1,...,n}$ into clusters $C_1, ..., C_k$ (e.g., with k-means clustering).
10:    return $V_1, ..., V_k$ with $V_i = \{j | y_j \in C_i\}$
```

3 Algorithms and methods

As a starting, we implemented the three spectral-partition algorithms presented in Section 2.3. From the technical side, we used Python scripts with library such as `networkx` for creating the graphs and `scipy` to compute the eigenvalues/eigenvectors.

Now, from the algorithmic side, we implemented the algorithms as described in the literature, by deriving a bit from it as we introduced the normalization step of the rows of the matrix **X** (with different type of normalization) for the three kinds of algorithms, as shown in Table 1.

4 Experimental results

4.1 Parameters of the tested algorithms

As a first step, all the algorithms explained in Section 2.3 were tested in their different possible configurations. Table 1 describes these configurations by attributing to each of them a label of the type *"algo x"*, that will be used to refer to the given configuration in the result graphs. It should be noted that some configurations are expected to not perform very well. Indeed, like exposed in [1], only the *Normalized Spectral Clustering* algorithm of Ng et Al., 2001 requires an additional step of regularization of the rows of **X**.

Algorithms	**Type of Laplacian**	**Rows' normalization of X**
alg 1	unormalized	-
alg 2	unormalized	L_1
alg 3	unormalized	L_2
alg 4	normalized	-
alg 5	normalized	L_1
alg 6	normalized	L_2
alg 7	normalized "random walk"	-
alg 8	normalized "random walk"	L_1
alg 9	normalized "random walk"	L_2

Table 1: Description of the different algorithms tested in our experiments.

4.2 Defining scores

Now, in order to evaluate the "goodness" of the partitions created by the different algorithms, a partitioning score needs to be given to each one of them. Such score can be defined in multiple ways. The simpler is probably the *Cut* score defined by

$$Cut(V_1, ..., V_k) = \sum_{i=1}^{k} \left| E\left(V_i, \bar{V_i}\right) \right|$$

where for $S, T \subseteq V$ with $S \cap T = \emptyset$, 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., $E(S,T) = \{(u,v) \in E | u \in S \text{ and } v \in T\}$, and we also define $\bar{V_i} = V \backslash V_i$.

However, minimizing this score usually doesn't lead to satisfactory partitions. The problem is that in many cases, the solution of the minimization of the *Cut* score simply separates one individual vertex from the rest of the graph, which is obviously not desired as we would like out clusters to be reasonably large groups of points. Hence, the *Cut* score is not often used in practice.

In order to counteract this problem, other objectives functions were introduced. The two most common ones are the *RatioCut* (Hagen and Kahng, 1992) and the *NormalizedRatioCut* (Shi and Malik, 2000) defined by

$$RatioCut\left(A_1, \ldots, A_k\right) = \sum_{i=1}^{k} \frac{Cut\left(V_i, \bar{V_i}\right)}{|V_i|}$$
$$NormalizedRatioCut\left(V_1, \ldots, V_k\right) = \sum_{i=1}^{k} \frac{Cut\left(V_i, \bar{V_i}\right)}{vol\left(V_i\right)}$$

which both penalizes unbalanced partitions. In the case of the *RatioCut*, this is due to the division by the size of the clusters V_i, while for the *NormalizedRatioCut*, the balancing is forced by the division of the volume of the clusters V_i. Note that the volume of a partition V_i is defined by $vol(V_i) = \sum_{j \in V_i} d(j)$.

In general, Clustering strives to optimize two objectives:

- Finding a partition such that points in different clusters are dissimilar, a.k.a, minimizing the cut
- Find a partition such that points in the same cluster are similar to each other which is maximized if the cut is small and the volume large.

It should be noted that Normalized spectral clustering consider both objectives, while unormalized spectral clustering only implements the first one [1]. For this reason, we decided to log the normalized ratio cut too.

As requested in the statement of the project, the different algorithms were evaluated and optimized with the *RatioCut*. Note, however, that the *NormalizedRatioCut* was also considered and the results for this score are shown in Appendix A.

4.3 Results

Our results are discussed on 4 graphs of 5 requested graphs, `road_Net-CA` being the missing one. The time required to compute the eigenvalues/vectors of the laplacian being superior to one night with a Microsoft Surface Pro (2.2 Ghz clock, 2 cores), we decided to stop the computation in order not to damage the laptop. Introducing a very large tolerance on the eigenvalue computation wasn't sufficient for this graph but helped for web-notreDame (tol=1e-10). The eigenvectors were dumped to memory for later usage to speed up the grid search.

4.3.1 Evaluation of the ratio-cut scores

Figure 1 summarizes the results in terms of ratio-cut we obtained by performing a grid search on the 5 requested graphs. A first glance at the results reveals that the algorithm 1, which consider an unormalized laplacian whose eigen vectors are not normalized, performs the best on all the graphs except `web-NotreDame`. This is not especially surprising, considering that this setup is designed explicitly to optimize the ratio-cut problem [1].

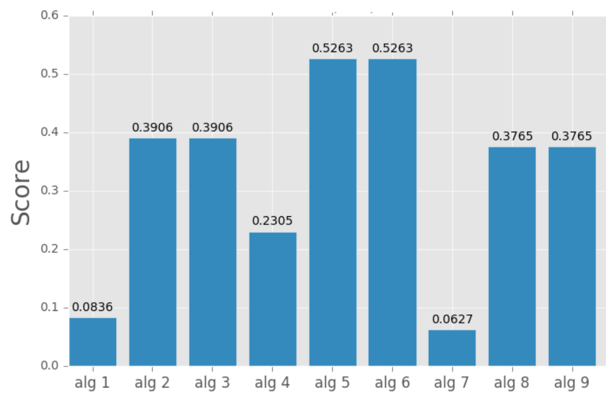
Following *alg 1* is *alg 7* with the regular normalized random-walk laplacian. In fact, as discussed in details in Appendix A, this algorithm is designed to optimize *the normalized ratio cut*. This emphases the fact that the algorithm 1 might not be the "in-every-case" best choice but depends on the metric we want to optimize. In this regards, *alg 7* can be considered to be very good since it gives good results, if not the best, in both cases.

On average, algorithms 2, 3, 8 and 9 do not perform very well, unsurprisingly, since they include an additional normalization step for the eigen vectors that is not that well theoretically motivated.

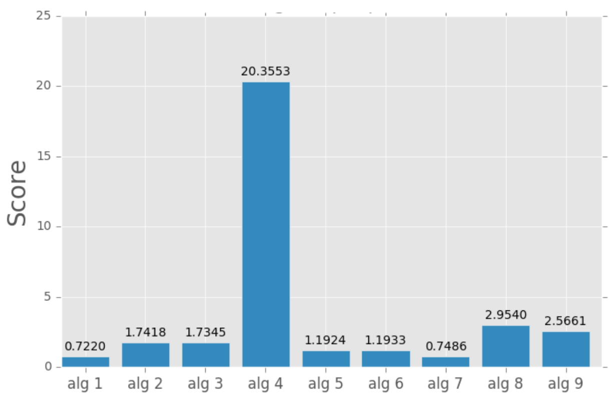
The 3 symmetric normalized laplacian (*alg 4-6*) do not give good results for `ca-GrQc` and `soc-Epinions`. In fact, the normalization step introduced for *alg5/6* is, surprisingly, even harmful in this setup (even sometimes for the normalized ratio cut). Eventhough this normalization step is well theoretically motivated for this particular algorithm (see sect. 2.3.2), it might not work properly in practice. A possible explanation is that introducing this step can be problematic if the eigenvectors contain particularly small entries: a phenomenon that occur when some of the vertices have a particularly low degree. There is no clear distinction on which of l_1 or l_2 norms perform the best overall.

However, for the two other graphs, the non-normalized version (*alg4*) prevented the convergence towards a good partitioning, which is particularly striking for `web-notreDame`. In fact, for this particular graph, we observe that the normalization of the eigen vectors (be it l_1 or l_2) is crucial to achieve good results since even the $alg7$, that usually perform well, is not satisfactory and requires also an additional step (unplanned in the original design of the algorithm) of normalization. We can thus conclude that *Alg4-5-6* suffer from poor robustness, a possible consequence of the fact that the eigenvectors of L are additionally multiplied with $D^{1/2}$, which might introduce undesired artifacts and prevent convergence [1].

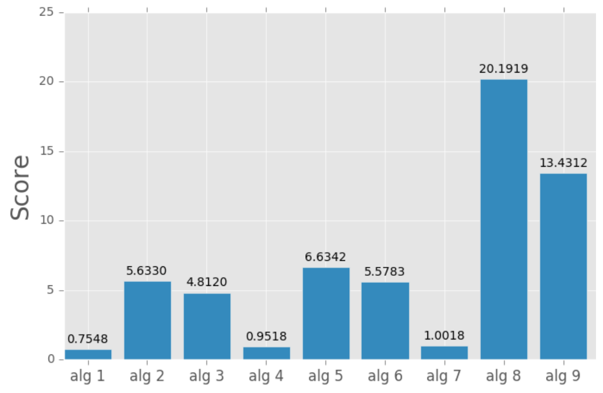
For `web-NotreDame`, we notice that l_1 norm, proposed in the original algorithm of Ng et al., 2001 gives slightly better results but it is l_2 norm that is best for the algorithm with the normalized random walk Laplacian (alg9). Finally, comparing these results with the one for the normalized ratio-cut, $alg5$ seems to be the final winner in both cases.



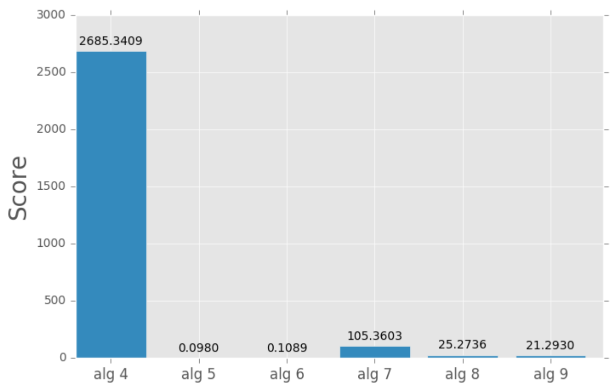
(a) Ratio Cut on `ca-GrQc`



(b) Ratio Cut on `Oregon-1`



(c) Ratio Cut on `soc-Epinions1`



(d) Ratio Cut on `web-NotreDame`

Figure 1: Ratio Cut scores of each algorithm on the different graphs.

4.3.2 Evaluation of the balancing between partitions

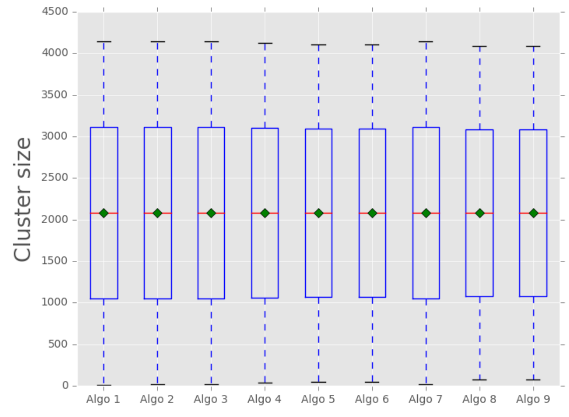
In order to evaluate visually how balanced the size of the resulting clusters are, we generated one boxplot for each output of our tested algorithms. The green diamond represent the mean cluster size= $|V|/k$ and, logically, the same everytime.

Looking at, figure 2a, we observe that all algorithms were able to find communities of "balanced size" and that the median is the same as the mean.

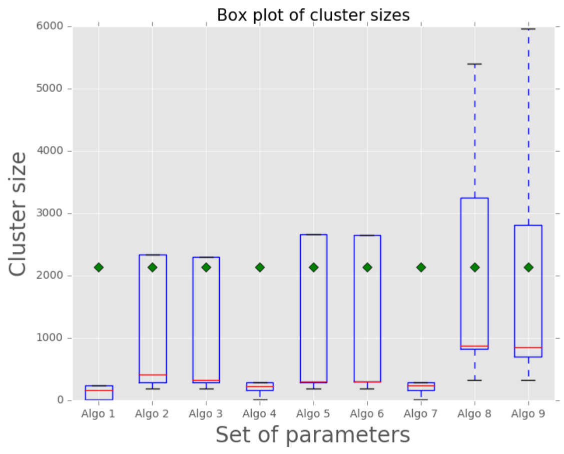
The other graph problems highlight the fact that it is not always the case. Many algorithms produce a clustering such that one cluster is very big which can be noticed visually when the median is a lot below the mean. When the box itself expands more vertically, the cluster sizes are less centered towards a specific value and more diverse.

For `web-NotreDame`, we notice that the more balanced clustering proposed by *alg8* and *alg9*, penalizes more the normalized ratio-cut, eventhough they perform well with the ratio-cut. This suggest that the optimal clustering sizes should be in such a way that the median should be in the lower third of the mean (|v|/k) like for the best performer: *alg*5 in this particular case.

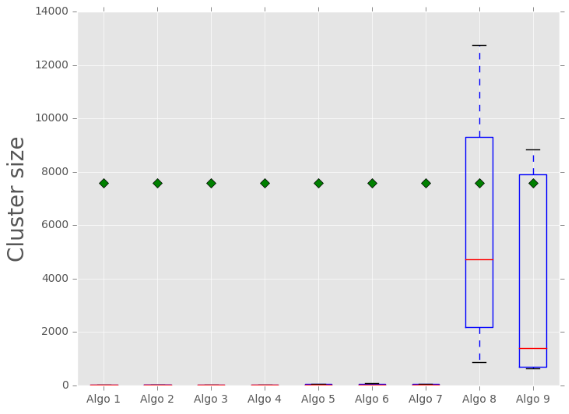
By comparing these results with the one of section 4.3.1, we notice that the best clusterings are often the one where one or few cluster(s) owns most of the vertices. Indeed, when the median is extremely low, one or some clusters should be very big to compensate and obtain the same mean.



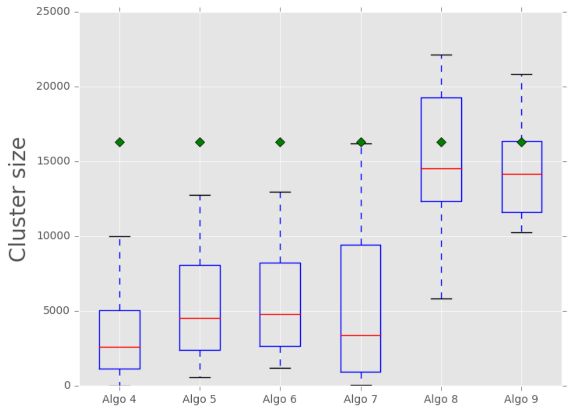
(a) Box plots for `ca-GrQc`



(b) Box plots `Oregon-1`



(c) Box plots for `soc-Epinions1`



(d) Box plots for `web-NotreDame`

Figure 2: Box plots for of each algorithm for the different graphs.

4.3.3 Other metrics

Other metrics have been gathered for the implemented algorithms, such as the conductance, minimum and maximum cluster size, expansion as well as the variance of the cluster size, and are presented in Table 2 below.

Graph	Algo	Conductance	min C size	max C size	expansion	var{C size}
ca-GrQc	L_u (alg1)	0.1667	12	4146	0.0834	4272489
	L (alg5)	1.04	50	4108	0.52	4116841
	L_{rw} (alg7)	0.125	16	4142	0.0625	15344007
Oregon-1	L_u (alg1)	0.9397	6	10260	0.262715	16515723
	L (alg5)	1.3398	188	7226	0.3287	7348666
	L_{rw} (alg7)	0.9453	10	9966	0.2551	15344007
soc-Epinions1	L_u (alg1)	0.8131	7	75723	0.1428	515824439
	L (alg5)	7.5963	10	75584	2.1628	513722160
	L_{rw} (alg7)	7.5963	10	75584	2.1628	513722160
web-NotreDame	L (alg5)	0.1014	557	220536	0.0166	2216369742
	L_{rw} (alg7)	105.7105	46	219889	76.3982	2220858908
	L_{rw} (alg9)	25.2735	5826	38019	4.2831	51794482

Table 2: Additional metrics computed for the most relevant algorithms

5 Conclusion

To conclude, spectral partitioning algorithms have been a major improvement in the literature with a lot of applications and are not just restricted to communities extraction but also as general clustering technique in unsupervised learning settings. While being simple to implement thanks to standard linear algebra methods, they offer great processing speed (at the cost of knowing the number of clusters k) and can handle non-convex clusters, a significant advantage compared to some traditional methods that make hypothesizes on the cluster shapes beforehand.

Our implementation and experiments demonstrated the best performance with unormalized Laplacian on the ratio cut and the best overall score in terms of both ratio cut and normalized ratio cut with normalized random walk Laplacian. Future improvements would include speeding up or trying to replace (see [6] power clustering with a mixture of spectral analysis and maximum spanning tree) the resource-demanding eigen computation of the laplacian. Some further investigation with regards to other final clustering technique as kmean could also be performed, eventhough the literature mainly suggested explicitely it was the best one.

The following table summarizes our final scores for the requested task:

Graph	RatioCut	Algorithm
ca-GrQc	0.0627	Spectral-partition with L_{rw} laplacian and no normalization step on $\mathbf{X}$
Oregon-1	0.7220	Spectral-partition with L_u laplacian and no normalization step on $\mathbf{X}$
soc-Epinions1	0.7548	Spectral-partition with L_u laplacian and no normalization step on $\mathbf{X}$
web-NotreDame	0.0980	Spectral-partition with L laplacian and L_1 normalization step on $\mathbf{X}$

Table 3: Summary of the ratio-cut scores obtained for the different graphs.

References

[1] U. Von Luxburg. (2007). A Tutorial on Spectral Clustering. *Statistics and Computing, 17 (4)*. Available at `http://www.tml.cs.uni-tuebingen.de/team/luxburg/publications/Luxburg07_tutorial.pdf`.

[2] B. Slininger. Fiedler's Theory of Spectral Graph Partitioning. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.592.1730&rep=rep1&type=pdf`.

[3] P. Kabelikova. (2006). Graph Partitioning Using Spectral Methods. Available at `https://pdfs.semanticscholar.org/ab34/1258fbab7b2e9a719c6bbeb96fc204356a82.pdf`.

[4] A. Gionis. (2019). Spectral graph analysis. Available at `https://mycourses.aalto.fi/pluginfile.php/1120056/mod_resource/content/2/11-spectral-graph-analysis.pdf`.

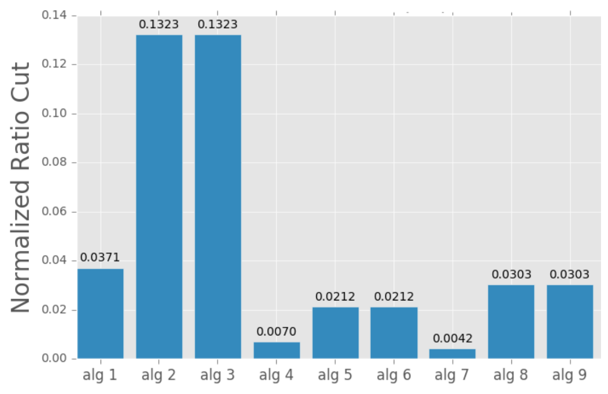
[5] S.O. Gharan. (2016). Clustering and the Spectral Partitioning Algorithm. Available at `https://courses.cs.washington.edu/courses/cse521/16sp/521-lecture-11.pdf`.

[6] Challa et. Al (2018), Power Spectral Clustering, Available at `https://hal.archives-ouvertes.fr/hal-01516649`

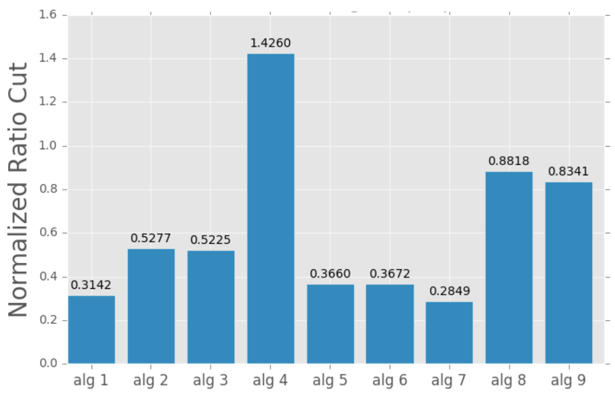
Appendices

A Additional experimental results

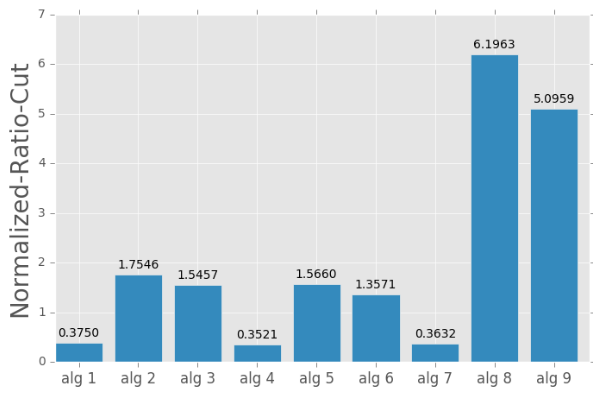
Focusing on the normalized ratio reveals that, unsurprisingly, the normalized random-walk laplacian performs the best on average. *Alg4* is however able to perform the best on `soc-Epinions1`, but, yet again demonstrate poor robustness since it score the worst on `Oregon-1` and `web-NotreDame`. Again the normalization step of the eigen vectors are very disapointing for algorithm 5 and 6 while they should improve algorithm 4. This is however not the case with `web-notreDame` where the additional normalization step give very good results, even way better than for the normalized random-walk laplacian.



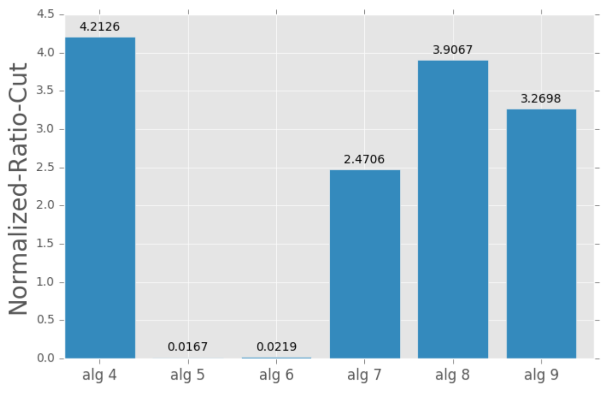
(a) Normalized Ratio Cut on `ca-GrQc`



(b) Normalized Ratio Cut on `Oregon-1`



(c) Normalized Ratio Cut on `soc-Epinions1`



(d) Normalized Ratio Cut on `web-NotreDame`

Figure 3: Normalized Ratio Cut scores of each algorithm on the different graphs.

Graph	**Ncut**	**Algorithm**
`ca-GrQc`	0.0042	Spectral-partition with L_{rw} laplacian and no normalization step on $\mathbf{X}$
`Oregon-1`	0.2849	Spectral-partition with L_{rw} laplacian and no normalization step on $\mathbf{X}$
`soc-Epinions1`	0.3521	Spectral-partition with L laplacian and no normalization step on $\mathbf{X}$
`web-NotreDame`	0.0167	Spectral-partition with L laplacian and L_1 normalization step on $\mathbf{X}$

Table 4: Summary of the normalized ratio-cut scores obtained for the different graphs.

B Contribution of team members

- **Antoine**: Literature review, design of the implemented methods, writing of the report.
- **Olivier**: Design and implementation of the methods, discussion of the results.

C Participation in competition

We did not participate in the competitions. Unfortunately for us, our results show that we have the same score as the top performer for `ca-GrQc` (0.0627).