Algorithmic Methods of Data Mining - Project - 2018

Team26:

Alexandru Dumitrescu - 720623

alexandru.dumitrescu@aalto.fi (mailto:alexandru.dumitrescu@aalto.fi)

Dimitrios Papatheodorou - 721033

dimitrios.papatheodorou@aalto.fi (mailto:dimitrios.papatheodorou@aalto.fi)

Contents

Work split

Most of our work was done while together in the lab. Individual work was mostly like this:

Alexandru did most of the experiments and results and the implementation of Spectral Clustering with Unnormalized Laplacian and Spectral Clustering with Normalized Laplacian by Shi & Malik.

Dimitris implemented Bridge Cut Clustering, Spectral Clustering with Normalized Laplacian and wrote literature review and the theoretical analyses.

Acknowledgements

We discussed a lot with *Ananth Mahadevan* and *Abhilash Jain* about theory, ideas, implementations and optimizations. We really helped each other understand the concepts in-depth, understand the problem better and optimize code. Our discussions were mostly about the nature of our metric, its connection to the variants of spectral clustering, the bridge cutting algorithm and convergence issues in our eigendecomposition approximation algorithm.

Introduction

In this project we were asked to analyze and perform clustering on social network graphs with the goal of minizing cuts while maximizing cluster size balance. We started by analyzing the graphs themselves trying to extract information about their structure, behaviour and size to set realistic expectations for our algorithms and understand the limitations. The graphs are large and highly connected so needed to consider space-efficient implementations and sparse representations - we even parallelized some computations to achieve the highest speed. Then we performed some literature review to understand the problem setting better and get ideas from state-of-the-art implementations. We dived into the specific problem and tried to analyze the behaviours of the different types of Laplacians and their connection with our metric. We first implemented a simple **Spectral Clustering** technique, both *normalized*, by Shi & Malik and by Ng, and *unnormalized* versions to accomodate for the different degree distributions, computation time and fit to our objective function, using simple KMeans on the spectral embedding of the Laplacian. Then we tried a different approach, **Bridge Cutting**, by cutting bridges using simple heuristics. The results were mostly in favour of bridge cutting and Spectral Clustering with Unnormalized Laplacian due to the fact that it is connected with the *Ratio Cut* problem that seems to fit better our objective function instead of of the *Normalized Cut* problem that is connected to the Normalized Laplacian. Our scores are very good compared to the leaderboard and sum of them were even the best ones that can be achieved.

Problem Statement and Literature review

Clustering

Any nonuniform data contains underlying structure due to the heterogeneity of the data. The unsupervised process of identifying this structure in terms of grouping the data elements is called clustering. The grouping is usually based on a *similarity measure* defined for the data elements. [1]

Graph Clustering

Graph clustering is the task of grouping the vertices of a graph into clusters, often called communities, taking into consideration the edge structure of the graph in such a way that there should be many edges within each cluster and relatively few between the clusters.

That means that the internal density of a good clustering should be notably higher than the density of the whole graph, and the intercluster density of the clustering should be lower than the graph density [184]. Considering this, the loosest possible definition of a graph cluster is that of a connected component, and the strictest definition is that each cluster should be a maximal clique (i.e. a subgraph into which no vertex could be added without losing the clique property). In most occasions, the semantically useful clusters lie somewhere in between these two extremes. Connected components are easily computed in O(n+m)-time with a breadth-first search, whereas clique detection is NP-complete [3,4]. Typically, interesting clustering measures tend to correspond to NP-hard decision problems [5,6].

Graph Clustering Similarity Measures

Vertex Similarity: There are many clustering algorithms based on similarities between the vertices - the higher the similarity, the stronger theneed to cluster the vertices together. Some adjacency-based measures of vertex similarity are the following:

- Jaccard similarity of the neighbourhoods or the vertices, which essentially measures the value of neighbourhood overlap between them
- Pearson correlation of columns or rows of the modified adjacency matrix that includes reflexive edges (self loops) as well. This can be easily derived by adding the identity matrix to the adjacency matrix. This way we can construct a symmetrical similarity matrix where each value is an edge weight. This measure again focuses on neighbourhood overlap.

Connectivity measures: Clusters in graphs can also be defined through connectivity by calculating the number of (edge-distinct) paths that exist between each pair of vertices. For some vertices to belong to the same cluster, they should be highly connected to each other [7]. Edachery et al. [8] proposed that in a good cluster, it is not absolutely necessary that two included vertices v and u are connected by a direct edge, if they are at least connected by a short path. Hence a matrix containing the distance for each vertex pair could serve as a similarity matrix, treating two vertices as similar if they have a low distance.

Graph Clustering Approaches

Hierarchical Clustering: A global clustering does not have to be a single partition or cover, but it may also be defined as a hierarchical structure, where each top-level cluster is composed of subclusters and so forth. This is useful in situations where the graph structure itself is hierarchical, and a single cluster can naturally be composed further to obtain a more fine-grained clustering or alternatively merged with another cluster to obtain a coarser division into clusters. Hierarchical clustering algorithms can be further divided into two classes, depending on whether the partition is refined or coarsened during each iteration:

- top-down or divisive algorithms that split the dataset iteratively or recursively into smaller and smaller clusters, and
- bottom-up or agglomerative algorithms that start with each data element in its own singleton cluster or another set of small initial clusters, iteratively merging these clusters into larger ones.

Divisive Clustering:

- ***Cuts***: We wish to split the graph in two by removing a cut, a set of edges that when removed creates two subgraphs, which we wish to be minimal. To well-known variants of the minimum-cut problem are the Ratio Cut and the Normalized Cut. We will discuss more about them next. *Minimum bisection* is the problem of dividing a 2n-vertex graph into two n-vertex subgraphs such that the cut size is minimized [9]. Deciding whether such a cut exists remains NP-complete even for regular graphs [10] and for graphs with bounded maximum degree [11], but the problem is polynomial for trees [11] and graphs with bounded treewidth [12].
- ***Spectral Clustering***: *Spectrum* of a graph is defined as the list of eigenvalues (together with their multiplicities) of its adjacency matrix or the laplacian matrix (this is widely used in physics), which is often more convenient and informative to study. We can actually use the eigenvectors of the laplacian as the input of a simple general clustering algorithm like KMeans and get results that actually discover underlying structures of the graph. We will talk more about it later.
- ***Betweenness***: Betweenness of an edge is the number of shortest paths connecting any pair of vertices that pass through the edge and betweenness of a node is defined as the number of shortest paths in the graph that pass through that vertex [19]. In order

to cluster an unweighted graph we can impose weights on the edges based on structural properties of the graph G, such as betweenness. Edges with high betweenness probably are links between clusters instead of internal links within a cluster: the several shortest paths passing through these edges are the shortest paths connecting the members of one cluster to those of another. Hence you can split the network into clusters by removing one by one edges with high betweenness values. If more than one edge has the highest betweenness value, one of them is chosen randomly and removed. The removal is followed by recalculation of the betweenness values, as the shortest paths have possibly been altered. This gives a clustering algorithm polynomial in n and m. This could be also used as a heuristic for finding the best bridge to cut for maximizing size balance between the two new subgraphs.

Our specific problem setting

For this project the task is to cluster big graphs by trying to minimize cuts (between-clusters edges) in order to minimize the between-cluster similarity, while maximizing the within-cluster similarity, while maintaining size balance among the clusters. The metric/objective function to be used is the following:

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

where $E(V_1, \cdots, V_k)$ is the set of edges of the graph that is cut by the k communities found by the clustering algorithm.

The metric tries to take into acount both the balance of the clusters and the number of cuts between those. We noticed however that, because of the graphs being so densely connected and with a big number of edges compared to nodes as we will show later, the metric actually motivates you towards focusing more on the minimization of cuts rather than the balance. That is because dense areas create many cuts if split into subgraphs just for the sake of balancing, and these graphs have such dense structures. In order for the metric to incentivize balance, it should have a denominator that rewards balance more, at least to the same scale as the nominator. Furthermore, the denominator, although surely promotes balance, it doesn't reward balance as it should, because it disregards the differences of sizes between clusters and focuses only on the size of the smallest, which doesn't feel like a universal measurement of balance. For example, one can argue that a 4-clustering of 65 points with these clusters sizes $[5, 20, 20, 20]$ is more balanced than $[7, 15, 19, 23]$, but the metric believes the latter is better, because $7 > 5$.

Graphs and Exploration

The graphs used for experimentation were of two sets, one for each part of the project. All of them were taken from the *Stanford Network Analysis Project (SNAP)* (http://snap.stanford.edu/data) (http://snap.stanford.edu/data)). Some information about the graphs:

Statistics of the graphs

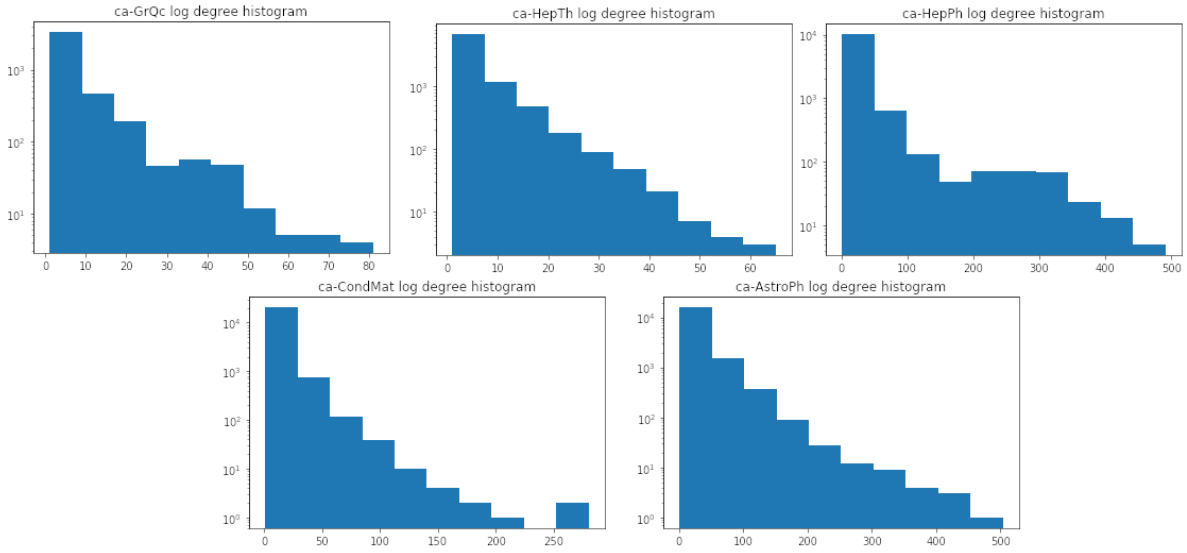
Part 1 graphs:

.	ca-GrQc	ca-HepTh	ca-Heph	ca-CondMat	ca-AstroPh
Number of nodes	41584	8638	11204	21363	17903
Number of edges	13428	24827	117649	91342	197031
Connected components	1	1	1	1	1
Average clustering coeff	0.55	0.48	0.62	0.64	0.63
Network density ($\times 10^{-4}$)	1.553	1.87	6.65	4	1.22

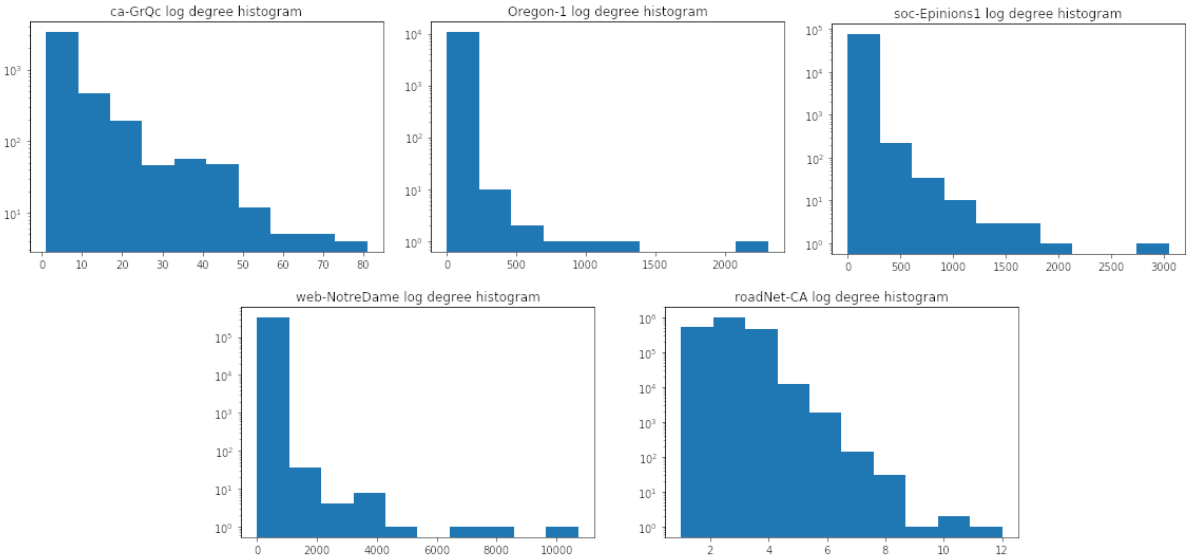
Part 2 graphs:

.	ca-GrQc	Oregon-1	soc-Epininons1	web-NotreDame	roadNet-CA
Number of nodes	41584	10670	75879	325729	1957027
Number of edges	13428	22002	405740	1117563	2760388
Connected components	1	1	2	1	1
Average clustering coeff	0.55	0.29	0.13	0.23	0.046
Network density ($\times 10^{-4}$)	1.553	3.86	1.4	0.21	0.0144

Below you can check the degree distributions of the 5 part-1 graphs, starting from the smallest one and ending with the biggest one. The y-axis (number of nodes in the degree bin) has been log-ed for visual purposes. As it's expected the degree distributions follow either a negative log or a decreasing linear fashion. *ca-AstroPh* and *ca-HepPh* have a bigger variety of degree values reaching to 500 for their most central nodes, while *ca-CondMat* reaches 250 and the smaller ones, *ca-GrQc* and *ca-HepTh* reach 80 and 60 respectively.



Below you can check the degree distributions of the 5 competition graphs, starting from the smallest one and ending with the biggest one again. The y-axis has been log-ed for visual purposes as previously. As it's expected again the degree distributions follow either a negative log or a decreasing linear fashion, but you shouldn't be fooled by the just the look of them as they have vastly different maximum number of degree values, more than in the previous case. Interestingly, the biggest graph, *roadNet-CA* has the smallest number of maximum degree, only 12, with the smallest one, *ca_GrQc* following as a close second at a maximum of 80, whereas the other ones have huge numbers of degrees - Oregon-1 reaches more than 2000, soc-Epinions1 reaches 3000 and impressively, web-NotreDame reaches higher than 10000. These graphs with big degree values have very dense areas with nodes with very high centrality, while *ca-GrQc* and especially *roadNet-CA* are much more sparse.



Algorithms and Approaches

Spectral Clustering

Laplacian Matrices and their Properties

As said previously, using the spectral properties of a graph one can encode the underlying structure of the graph and yield good clustering results. A common way of doing so is eigendecomposing the laplacian matrix of the graph and use the k first/smallest eigenvector as a vector representation of the graph that can be clustered directly by a clustering algorithm such as KMeans, Gaussian Mixtures, Agglomerative etc. These are the laplacian matrices that are generally used:

- **Unnormalized Laplacian**:
$$\mathcal{L} = I - A$$
[14].

Interesting properties of the Unnormalized $\mathcal{L}$ [17]:

1. For every vector $x \in \mathbb{R}^n$ it holds:
$$x' \mathcal{L} x = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2,$$
where w_{ij} is the ij-th entry of the weighted adjacency matrix W. W can equal to A in an unweighted setting.
2. $\mathcal{L}$ is symmetric and positive semi-definite.
3. The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector $\mathbf{1}$.
4. $\mathcal{L}$ has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

- ***Symmetric* Normalized Laplacian** is defined as
$$\mathcal{L}_{sym} = I - D^{-1/2} A D^{-1/2}$$

An element-wise definition for the normalized Laplacian, which is easier to understand intuitively than the matrix version, is the following:

$$\mathcal{L}_{uv} = \begin{cases} 1, & \text{if } u = v \text{ and } \deg(v) > 0 \\ -\frac{1}{\sqrt{\deg(u) \cdot \deg(v)}}, & \text{if } u \in \Gamma(v) \\ 0, & \text{otherwise} \end{cases}$$

Using the normalized Laplacian is convenient as the eigenvalues of L all fall within the interval [0, 2]. The smallest eigenvalue is always zero, as the matrix is singular, and the corresponding eigenvector is simply a vector with each element being the square-root of the degree of the corresponding vertex. A pleasant consequence of having the spectra limited to the interval [0, 2] is that it makes comparing the spectra of two graphs easier. However, even non-isomorphic graphs can share the same spectrum [15]. Graphs that have the same spectrum are called cospectral (also isospectral) [63].

- ***Random-Walk* Normalized Laplacian**:

$$\mathcal{L}_{rw} = I - D^{-1}A$$

Interesting properties of the Normalized Laplacians $\mathcal{L}_{rw}$ and $\mathcal{L}_{sym}$ [17]:

1. For every vector $x \in \mathbb{R}^n$ it holds:
$$x' \mathcal{L}_{sym} x = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}} \right)^2,$$
where d_i is the degree of the i-th node.
2. λ is an eigenvalue of $\mathcal{L}_{rw}$ with eigenvector u if and only if λ is an eigenvalue of $\mathcal{L}_{sym}$ with eigenvector $w = D^{1/2}u$.
3. λ is an eigenvalue of $\mathcal{L}_{rw}$ with eigenvector u if and only if λ and u solve the generalized eigenproblem $\mathcal{L}u = \lambda Du$.
4. 0 is an eigenvalue of $\mathcal{L}_{rw}$ with the constant one vector $\mathbf{1}$ as eigenvector. 0 is an eigenvalue of $\mathcal{L}_{sym}$ with eigenvector $D^{1/2}\mathbf{1}$.

To give a more clear intuition about the normalized laplacian think about the following case:

When a graph is formed by a collection of k disjoint cliques, the symmetric normalized Laplacian is a block-diagonal matrix that has eigenvalue 0 with multiplicity k and the corresponding eigenvectors serve as indicator functions of membership in the corresponding cliques: the elements of the clique have a different value (of larger magnitude) than the other vertices. Any deviations caused by introducing edges between the cliques causes $k - 1$ of the k eigenvalues that were zero to become slightly larger than zero and also the corresponding eigenvectors change. However, some of the underlying structure can still be seen in the eigenvectors of the Laplacian even when edges are added to connect the cliques and when some edges are removed from within the original cliques. This phenomenon is the basis of spectral clustering, where an eigenvector or a combination of several eigenvectors is used a vertex similarity measure for computing the clusters.

Also other matrices than the Laplacian can be used to compute such spectral measures; if instead of the adjacency matrix of a simple graph, the input is some kind of a similarity matrix for a complete graph, similar computations still may yield good results. The

downside is that computing or even approximating eigenvalues and eigenvectors is not fast for all graphs and hence such methods may face scalability issues when applied to massive graphs.

Spectral clustering is typically based on computing the eigenvectors corresponding to the second-smallest eigenvalue of the normalized Laplacian or some eigenvector of some other matrix representing the graph structure. Possible matrices include modifications of the adjacency matrix such as the transition matrix of a blind random walk on the graph. The component values of the resulting eigenvector are used as vertex-similarity values to determine the clustering.

Spectral Clustering with KMeans

In our implementations an experiments we used the below spectral algorithms combined with KMeans. We tried a naive KMeans with balance constraints but gave out very bad results, so we kept the simple one. A way to compute Spectral Clustering in a fast way can be found here [18].

All the below implementations use Scipy's Compressed Sparse Column (CSC) matrix representations that also have the advantages of efficient arithmetic operations with other CSC matrices, efficient column slicing and fast matrix vector products. The eigendecomposition was performed using the `scipy.sparse.linalg.eigs` function which is much much faster than the simple eigendecomposition of NumPy.

Unnormalized Spectral Clustering with KMeans

Algorithm:

1. Input: Graph G, k number of clusters
2. Compute adjacency matrix A
3. Compute modified diagonal degree matrix D
4. Compute unnormalized Laplacian matrix $L = D - A$
5. **Compute the first k eigenvectors (smallest k)** $u_1, u_2, \ldots, u_k$ **of** L
6. Form $U \in R^{n \times k}$ with columns $u_1, \ldots, u_k$
7. Cluster the points $\{y_i\}$ into $C_1, \ldots, C_k$ using KMeans
8. Return clusters $A_1, \ldots, A_k$, such that $A_i = \{j | y_j \in C_i\}$

Normalized Spectral Clustering (Shi & Malik) with KMeans

Algorithm:

1. Input: Graph G, k number of clusters
2. Computer adjacency matrix A
3. Compute modified diagonal degree matrix D
4. Compute unnormalized Laplacian matrix $L = D - A$
5. **Compute the first k eigenvectors (smallest k) $u_1, u_2, \ldots, u_k$ of the generalized eigenproblem $Lu = \lambda Du$**
   ---- *eigenvectors of the random walk Laplacian*: $I - D^{-1}W$
6. Form $U \in R^{n \times k}$ with columns $u_1, \ldots, u_k$
7. Normalized U, st. rows will have norm 1
8. Consider the i^{th} row of U as point $y_i \in R^k, i = 1, \ldots, n$
9. Cluster the points $\{y_i\}$ into $C_1, \ldots, C_k$
10. Return clusters $A_1, \ldots, A_k$, such that $A_i = \{j | y_j \in C_i\}$

Normalized Spectral Clustering (Ng, Jordan & Weiss) with KMeans

Algorithm:

1. Input: Graph G (or adjacency matrix), k number of clusters
2. Compute adjacency matrix A
3. Compute modified diagonal degree matrix D
4. Compute normalized symmetric Laplacian matrix $L = I - D^{-1/2}AD^{-1/2}$
5. **Compute the first k eigenvectors (smallest k) $u_1, u_2, \ldots, u_k$ of L**
6. Form $U \in R^{n \times k}$ with columns $u_1, \ldots, u_k$
7. **Normalize U, st. rows will have norm 1**
8. Consider the i^{th} row of U as point $y_i \in R^k, i = 1, \ldots, n$
9. Cluster the points $\{y_i\}$ into $C_1, \ldots, C_k$
10. Return clusters $A_1, \ldots, A_k$, such that $A_i = \{j | y_j \in C_i\}$

Diving deeper into the problem

Performing graph clustering using a minimum cut without a size contraint on the clusters often leads in creating clusters of 1 node, which is not desirable. There are 3 main types of cuts that help us circumvent this problem: *RatioCut* or *RCut* [20], *NormalizedCut* or *NCut* [21] and *MinMaxCut* [23]. In $RatioCut$, the size of a subset C of a graph $G = (V, E)$,

with W a weight matrix or simply an adjacency matrix, is measured by its cardinality (number of vertices) $|C|$, while in $NCut$ the size is measured by the weights of its edges $vol(C) = \sum_C d_i$, where d_i is the degree of the i-th node, and in *MinMaxCut* by just the weight matrix.

For k partitions $C_1, C_2, \cdots, C_k$, and $\overline{C_1}, \overline{C_2}, \cdots, \overline{C_k}$ their k complements, as $\overline{C_i} = V \backslash C_i$:

$$\text{RatioCut}(C_1, \cdots, C_k) := \sum_{i=1}^{k} \frac{\text{cut}\left(C_i, \overline{C}_i\right)}{|C_i|}$$

$$\text{NCut}(C_1, \cdots, C_k) := \sum_{i=1}^{k} \frac{\text{cut}\left(C_i, \overline{C}_i\right)}{\text{vol}(C_i)}$$

$$\text{MinMaxCut}(C_1, \cdots, C_k) := \sum_{i=1}^{k} \frac{\text{cut}\left(C_i, \overline{C}_i\right)}{W(C_i, C_i)}$$

In [17] it proved that *Unnormalized Spectral Clustering* is a relaxation of *RatioCut* and both *Normalized Spectral Clustering* and *MinMaxCut* are relaxations of $NCut$, so there is a connection here that can help us draw prior expectations for out tests. Although the normalized version is praised by the literature and as previously mentioned has a lot of advantages and interesting properties we can make an educated guess that it will lack in performance against unnormalized and even Shi & Malik's normalized. The reason is that our metric resembles the RatioCut problem much more than the NCut or the MinMaxCut, because its balance constraint is only based on the cluster cardinality, just as RatioCut, but the focus is only on the smallest cluster. Essentialy, our metric seems like an approximation of RatioCut. On the other hand, Ncut assumes in-betweenes connections of the graph - $vol(C_i)$ represents how many edges connect the vertixes inside each cluster, so this function represents a different form of balance, that might end up in creating many cuts to achieve it on a dense graph. In graphs created from objects such as images, this is obviously way better. You want to prioritize the similarity given by some pixels and so, the in-betweenes connection of those is really important. In the case of undirected, unweighted and dense graphs this may not be a good choice.

Bridge Cutting Clustering

After the spectral algorithms we wanted to try something else, simpler in conception but very effective in our setting, and that was finding the best subset of bridges that can be cut in order to achieve a k-clustering. Essentialy, it's like a minimum-cut k-partition problem that searches over bridges only. This way the cuts that are performed are O(k), which

means at maximum k - fewer than k in the case of already existing disjoint connected components, as for example the *soc-Epinions1* graph. This means that cuts as minimized, so we can try to motivate for trying to balance clusters out *according to the metric*. It's obvious that this method only works if there are bridges, as it doesn't consider any other minimum subset of edges that can cut the graphs.

We came up with some simple heuristics that would help in the balance aspect of the problem, speed up our computations and, in fact, perform an approximation of the perfect solution.

Heuristics:

- We consider only bridges that both their nodes have a *higher degree than 1*, thus remove all bridges that simply connect individual nodes with the graph, as they would create a very unbalanced clustering. This could be changed to keep only one such bridge, just in case it's a good bridge for an extreme solution.
- After every split, we recurse with the subgraphs till this point as input and the graph that the next split is performed on the *biggest one*, the one with the highest number of nodes. This is good way to motivate the algorithm create more balanced sets and not having to search over all subgraphs created so far, as then the complexity would grow exponentially.
- The implementation gives the choice of calculating a good subset of bridge to use in order to reduce the heavy load of exhaustively checking every non-trivial bridge, which is based on the *edge betweenness centrality*. Reminder: The edge betweenness centrality is defined as the number of the shortest paths that go through an edge in a graph [19]. Edges with high betweenness probably are links between dense areas, so we choose the bridge with the highest centrality values. The way the betweenness centrality is calculated is again an approximation that is done by choosing a subset of nodes as sources and targets. We have to be careful here as the computation of the centrality may be very costly - for small graphs with small k maybe costlier than checking for every bridge. This heuristic was not tested properly during this project because the idea came to us the last day, but it's implemented.
- In every subroutine that iterates over the bridges, the bridge is removed from the graph and the two created connected components are tested using the following idea: we find the components with the *maximum of minimum sizes* (as this is tested by the metric). So for every bridge we cut, we keep the size of the smallest component (this will be tested in the metric) and then find the maximum of all these sizes (we want this size to be the largest possible).

High-level sum-up of the algorithm:
The algorithm is performed through a recursion that test every *useful* bridge (according to heuristics) in the biggest subgraphs returned by the previous recursion, so the recursion starts with 1 graph, on the next iteration it's repeated on 2 graphs (2 subgraphs of the

previous one), on the next on 3 graph (1 smallest subgraphs from previously and the 2 new ones - the subgraphs of the previous biggest one), and so on and so forth. The recursion stops when the numbers of graph is eqaul to k.

Parallelization: The implementation supports parallelization of the betweenness centrality calculation and the bridge cutting calculation by splitting the workload to all the threads of the computer through a MapReduce scheme using the *multiprocess* module.

Algorithm

BridgeCuttingClustering:

1. Input: G graph, k number of clusters, $maxIter$ maximum number of iterations, $betweennessHeuristic$ whether you want to use it, $parallel$ whether you want it parallelized
2. $bestClustering$ = BridgeCuttingRecursive([G], k, $maxIter$, $betweennessHeuristic$, $parallel$)
3. **if** $bestClustering$ is Null: **return** Null;
4. $finalClustering$ = {};
5. **for** $label$, $cluster$ **in** enumerate($bestClustering$):
6. **for** $node$ **in** $cluster$.nodes(): $clustering$[$node$] = $label$;
7. **return** $clustering$;

BridgeCut:

1. Input: G graph, $bridges$ bridges, $maxIter$ maximum number of iterations
2. $bestSize$, $bestClustering$, $bestBridge$, it = -1, **Null**, **Null**, 0;
3. **for** $bridge$ **in** $bridges$:
   3.1. G.removeEdge(bridge);
   3.2. $clusters$ = G.connectedComponents();
   3.3. $minClusterSize$ = $min\text{-}length$ {$clusters$};
   3.4. **if** $bestSize$ < $minClusterSize$:
   $bestSize$ = $minClusterSize$; $BestClustering$ = $clusters$; $bestBridge$ = $bridge$;
   3.5. G.addEdge($bridge$);
   3.6. it++;
   3.7. **if** $maxIter < it$: **break**;
4. **return** $bestClustering$, $bestBridge$;

BridgeCuttingRecursive:

1. Input: Gs subgraphs till now, k number of clusters, $maxIter$ maximum number of iterations,
   $betweennessHeuristic$ whether you want to use it, $parallel$ whether you want it parallelized
2. **if** $k == size(Gs)$: return Gs;
3. **if** Gs has only 1 graph G with more than 1 connected components $COMPs$:
   3.1. **return** BridgeCuttingRecursive($COMPs$, k, $maxIter$, $betweennessHeuristic$, $parallel$);
4. $bridges$, $removed$ = [], []
5. **while** Gs is not empty:
   5.2. **if** $size(Gs) > 1$: G = largest graph **in** Gs;
   5.3. **else**: G = Gs[0];
   5.4. $tempBridges$ = G.bridges();
   5.5. **if** $tempBridges$ is empty: Gs.remove(G); $removed$.append(G); **continue**;
   5.6. **for** $bridge$ **in** $tempBridges$: add $bridge$ to $bridges$ only if both its nodes have degrees > 1;
   5.7. **if** $bridges$ is empty: **continue**;
   5.8. **else**: Gs.remove(G); $removed$.append(G);
6. is Gs is empty: print("No useful bridges"); **return Null**;
7. Gs.extend(removed); Gs.remove(G); # not needed in the list
8. **if** $betweennessHeuristic == True$ and $size(Gs) == 0$:
   8.1. randomly choose 20% of nodes as sources and 20% as targets (src, trg) nodes from G;
   8.2. $betweennessCentrality$ = edgeBetweennessCentrality(G, src, trg);
   8.3. $bridgesCentrality$ = $betweennessCentrality$.subset($bridges$);
   8.4. $centralBridges$ = choose $max(50, k)$ largest bridges from $bridgesCentrality$;
9. **if** $betweennessHeuristic == True$: $bridges = bridges \cap centralBridges$;
10. **if** $parallel == True$:
    10.1. $bridgeChunks$ = split $bridges$ into chunks for every thread;
    10.2. $workersClusters$ = multithread.map(bridgeCut(G, $bridgeChunks$, $maxIter$));
    10.3. $bestBridge$ = reduce($argmax$ { min { $size(workersClusters)$ } });
11. **else**: $bestBridge$ = bridgeCut(G, $bridges$, $maxIter$);
12. $bestSubgraphs$ = []
13. G.removeEdge($bestBridge$)
14. $bestSubgraphs$ = G.connectedComponents()
15. Gs.extend($bestSubgraphs$)
16. G.addEdge($bestBridge$)
17. $return$ BridgeCuttingResursive(Gs, k, $maxIter$, $betweennessHeuristic$, $parallel$)

Experimentation and Results

Part 1

Below are the plots of every algorithm used for every graph of Part 1 for different values of k. This experiment is useful to understand the behaviours of each algorithm on different types of graph and set our expectations for Part 2 graphs, which are bigger and more complex.

As we can cleary see, *Normalized Spectral* by Ng et al. performs much worse than every other algorithm, followed by *Shi & Malik's Normalized*, which also performs bad compared to the others. Their performance becomes even worse as k increases. In contrast, *Unnormalized Spectral* and *Bridge Cutting* perform very well and seems to not be affected by the increase of k. The winner in every test was Bridge Cutting except for the case of ca-AstroPh with k=60, as there weren't any good bridges to cut.

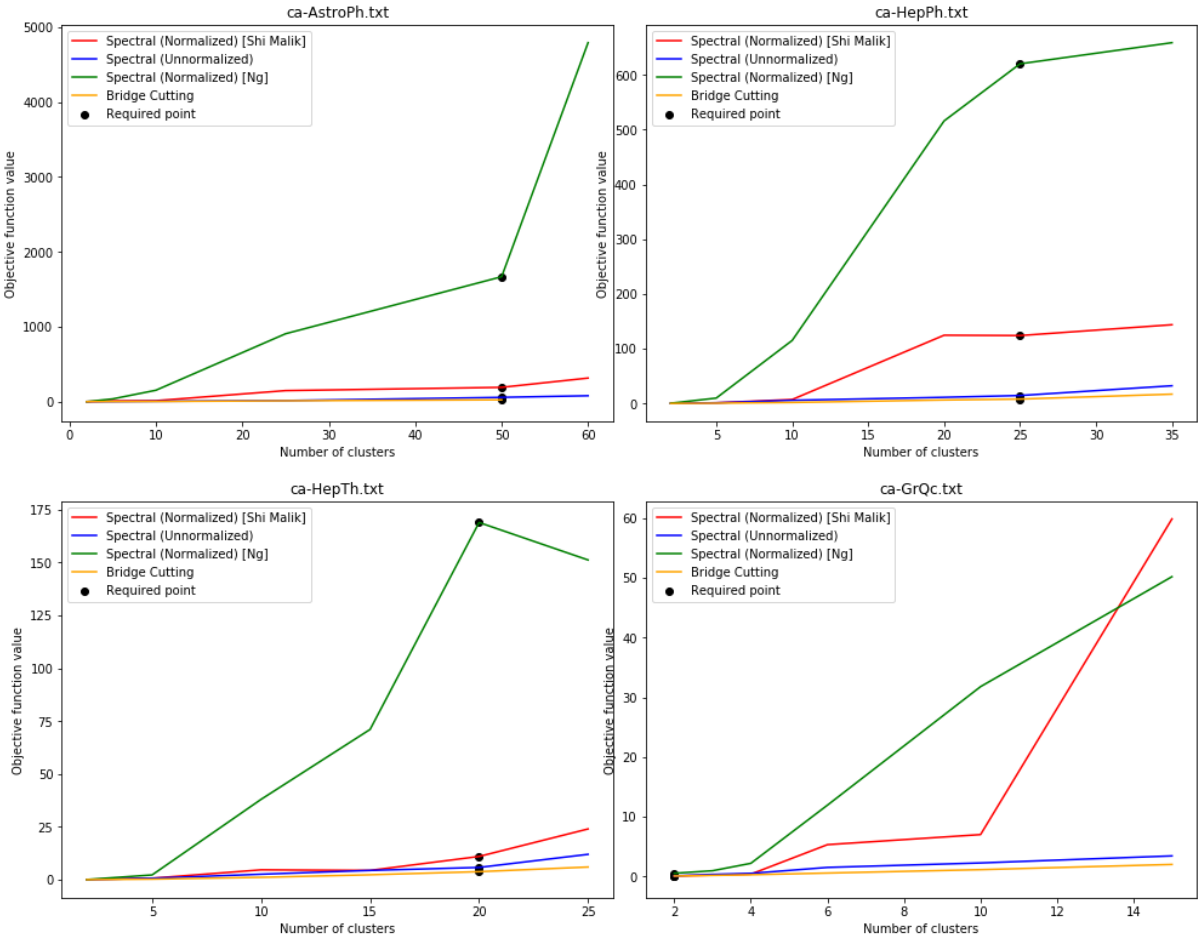
These results were to be expected as discussed previously. The normalized versions of spectral clustering doesn't work well with our metric, as it promotes balance (or at least higher minimum cluster size) at the expense of much more cuts, and our previous theoretical evaluation of what we can expect from it for dense graphs is empirically justified in the tests.

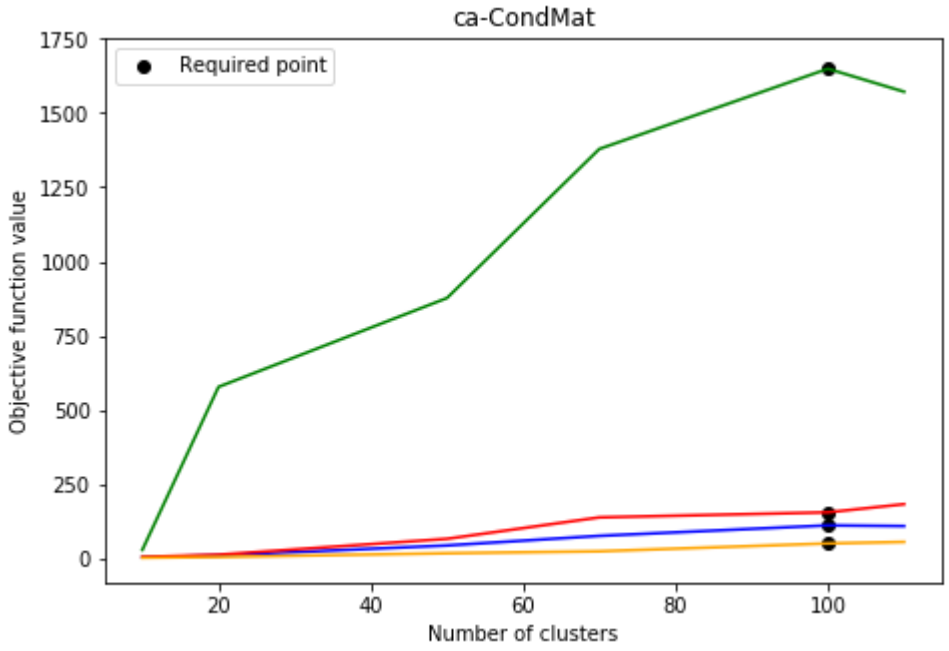
To showcase what happends with some examples, check the plots and then the tables below.

The tables showcase the performance of Normalized Spectral (Ng et al), Unnormalized Spectral and Bridge Cutting on ca-GrQc and ca-HepTh graphs for 3 different values of k. Every value in the cells is: $score = \frac{NumberOfCuts}{MinimumClusterSize}$. As you can clearly see, Normalized tries to up the balance by creating an immense amount of cuts, whereas Unnormalized and Bridge Cutting don't really care about the balance as long as the cuts are kept low. Interestingly, Bridge Cutting not only minimizes cuts (by design), but creates sets with higher minimum cluster size than Unnormalized (not saying more balanced because we don't know it's it's really the case, but if we accept the definition of balance of the metric, then they are more balanced indeed). It seems like the balance heuristics of our approach perform quite well. Unnormalized disregards balance completely and performs more cuts than bridge cutting obviously, so it scores lower, but still it performs quite well, especially compared to the normalized versions.

ca-GrQc	Normalized (Ng et al.)	Unnormalized	Bridge Cutting
k=4	2.1875 = 35/16	0.5 = 6/12	0.2727 = 3/11
k=8	21.2093 = 912/43	2 = 16/8	0.875 = 7/8
k=12	36.1395 = 1554/43	2.5 = 20/8	1.375 = 11/8

ca-HepTh	Normalized (Ng et al.)	Unnormalized	Bridge Cutting
k=4	40 = 800/20	2.6 = 13/5	1.125 = 9/8
k=8	67.875 = 1086/16	4.4 = 22/5	2.3334 = 14/6
k=12	189.0695	5.8 = 29/5	3.8 = 19/8





Part 2 / Competition

Using the knowledge we drew from Part 1 we could use our algorithm in a better way and save time. We expected Bridge Cutting to perform the best in every graph except maybe for *roadNet-CA* which is not dense at all, especially compared to its size, but the results had some interesting twists. In the table you can see the results for every graph and for every method. Bold values are the best for each graph and italic values are the second best.

- Bridge Cutting is the best on *ca-GrQc* as we saw previously and the best on *ca-Oregon-1* and *web-NotreDame* as well, which was expected as they are very dense.
- Interestingly, despite its dense nature, *soc-Epinions1* gave better results with Shi & Malik's Normalized by even beating Bridge Cutting - that is due to the fact that Bridge Cutting's performed minimum cuts, 8, but the smallest cluster had a size of 2, which is very small, whereas Normalized performed 11 cuts with the smallest cluster being of size 7.
- Unfortunately, Unnormalized on *web-NotreDame* never finished running. Ananth Mahadevan and Abhilash Jain had the same problem using the same eigendecomposition function, which may be to blame as it's not guaranteed to converge.
- *roadNet-CA* was too large to be run on more tests and still be on time, but we're happy with Normalized's performance.

As for the competition, we are in the Top 10 for every competition graph and for some of them in the 1st place.

.	Unnormalized	Normalized (Ng et al)	Normalized (Shi & Malik)	Bridge Cutting
ca-GrQc	0.0834	0.52	*0.075*	**0.0625**
ca-Oregon-1	15.5	*5.3228*	14.3	**0.0667**
soc-Epinions1	*4*	23.125	**1.5714**	*4*
web-NotreDame	never finished	*0.5804*	0.5842	**0.0379**
roadNet-CA	-	**0.3355**	-	-

Conclusions

Graph clustering with constraints on the number of cuts and the balance is a very challenging problem. We analyzed the graphs we performed the tests on to get a glimpse of their density and structure, and dived into the theory of minimum cuts to understand the nature of our objective function / metric. Spectral clustering is a versatile way to cluster graphs as it performs the clustering on the spectral embedding of the graph which not only captures the underlying structure of the graph, but can be created with different methods that lead the algorithm to behave in different ways. We found that our metric is connected to the RatioCut problem to which the Spectral Clustering with Unnormalized Laplacian is a relaxation, due to the fact that its balance constraint is based on cluster cardinality. This can't be said about Spectral Clustering with Normalized Laplacians (both by Ng et al and Shi & Malik) as it's a relaxation of the NCut or the MinMaxCut problem which relates balance to the volume of the cluster, that is the sum of its nodes' degrees, or the weight/adjacency matrix. In addition, we developed an straightforward algorithm that operates directly on the graph, called Bridge Cutting, which uses heuristics to minimize the cuts by only cutting useful bridges while simultaneously motivates balance as defined by the metric. Our experiments empirically justified our prior claims of the inability of Normalized Clustering to minimize our metric on dense graphs, as it tries to balance the clusters at the expense of an immense amount of cuts. On the densest of graphs, generally, the clear winner was Bridge Cutting, as it performs the minimum amount of cuts and is as balanced as Unnormalized Spectral Clustering, which performs more cuts. The Normalized versions of Spectral Clustering, especially Ng et al's, fails miserably on dense graph for high values of k, but on more sparse graphs the perform very well, even better than the others.

Future Work

For future work we would like to improve the performance of Bridge Cutting either from an algorithmic approach or as an implementation. Specifically, we would like to try out other python libraries that are written in C/C++, like *Graph-Tool* and *igraph*, which are extremely faster than NetworkX, but with fewer functions in their APIs, a better multithreaded implementation that removes a lot of thread overhead, test the effectiveness of betweenness centrality as a heuristic, and find a good way of approximating it, consider and test other centrality measures and optimize the code. We would also like test the same problem setting with different metrics and see how the algorithms behave. Furthermore, we want to find the mathematical connection between the performance of a spectral algorithm with regards to a metric to the density of the graph, if such thing can be derived.

References

[1] J.M. Kleinberg, E. Tardos, Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields, Journal of the ACM 49 (5) (2002) 14–23.

[2] M.E.J. Newman, Fast algorithm for detecting community structure in networks, Physical Review E 69 (6) (2004) 066133.

[3] R.M. Karp, Reducibility among combinatorial problems, in: Proceedings of a Symposium on the Complexity of Computer Computations, IBM, Plenum, NY, USA, 1972.

[4] I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo, The maximum clique problem, in: D.-Z. Du, P.M. Pardalos (Eds.), in: Handbook of Combinatorial Optimization, vol. Supplement Volume A, Kluwer Academic Publishers, Boston, MA, USA, 1999, pp. 1–74

[5] J. Šíma, S.E. Schaeffer, On the NP-completeness of some graph cluster measures, in: J. Wiedermann, G. Tel, J. Pokorný, M. Bieliková, J. Štuller (Eds.), Proceedings of the Thirty-second International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM, in: Lecture Notes in Computer Science, vol. 3831, SpringerVerlag GmbH, Berlin, Heidelberg, Germany, 2006.

[6] Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, V. Vinay, Clustering in large graphs and matrices, Machine Learning 56 (2004) 9–33.

[7] E. Hartuv, R. Shamir, A clustering algorithm based on graph connectivity, Information Processing Letters 76 (4–6) (2000) 175–181

[8] J. Edachery, A. Sen, F.J. Brandenburg, Graph clustering using distance-k cliques, in: Proceedings of the Seventh International Symposium on Graph Drawing, in: Lecture Notes in Computer Science, vol. 1731, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 1999.

[9] M.R. Garey, D.S. Johnson, L.J. Stockmeyer, Some simplified NP-complete graph problems, Theoretical Computer Science 1 (3) (1976) 237–267.

[10] T.N. Bui, F.T. Leighton, S. Chaudhuri, M. Sipser, Graph bisection algorithms with good average case behavior, Combinatorica 7 (2) (1987) 171–191

[11] R.M. MacGregor, On partitioning a graph: A theoretical and empirical study, Ph.D. Thesis, University of California, Berkeley, CA, USA, 1978.

[12] K. Soumyanath, J.S. Deogun, On bisection width of partial k-trees, Congressus Numerantium 74 (1990) 45–51.

[13] M.E.J. Newman, M. Girvan, Mixing patterns and community structure in networks, in: R. Pastor-Satorras, M. Rubi, A. Díaz Guilera (Eds.), Statistical Mechanics of Complex Networks: Proceedings of the XVIII Sitges Conference on Statistical Mechanics, in: Lecture Notes in Physics, vol. 625, SpringerVerlag GmbH, Berlin, Germany, 2003.

[14] F.R.K. Chung, Spectral Graph Theory, American Mathematical Society, Providence, RI, USA, 1997.

[15] D. Vukadinovic, P. Huang, T. Erlebach, On the spectrum ´ and structure of Internet topology graphs, in: H. Unger, T. Böhme, A.R. Mikler (Eds.), Proceedings of Second International Workshop on Innovative Internet Computing Systems, in: Lecture Notes in Computer Science, vol. 2346, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2002.

[16] D. Cvetkovic, Signless laplacians and line graphs, Bulletin, ´ Classe des Sciences Mathématiques et Naturelles, Sciences mathématiques Académie Serbe des Sciences et des Arts CXXXI (30) (2005) 85–92

[17] Ulrike von Luxburg, Max Planck Institute for Biological Cybernetics, Technical Report No. TR-149, A Tutorial on Spectral Clustering Updated Version, 2007.

[18] Yan, Donghui, Ling Huang, and Michael I. Jordan. "Fast approximate spectral clustering." Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2009.

[19] Girvan M, Newman ME (2002) Community structure in social and biological networks. Proc Natl Acad Sci USA 99(12):7821–7826

[20] Hagen, L. and Kahng, A. (1992). New spectral methods for ratio cut partitioning and clustering. IEEE Trans. Computer-Aided Design, 11(9), 1074 – 1085.

[21] Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8), 888 – 905.

[22] S. E. Schaeffer, Laboratory for Theoretical Computer Science, Helsinki University of Technology TKK, Graph clustering, Survey, Computer Science Review I (2007), 27-64

[23] Ding, C., He, X., Zha, H., Gu, M., and Simon, H. (2001). A min-max cut algorithm for graph partitioning and data clustering. In Proceedings of the first IEEE International Conference on Data Mining (ICDM) (pp. 107 – 114). Washington, DC, USA: IEEE Computer Society