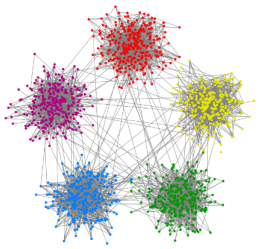
MATH420 Final Project: Community Detection

Daniel Kraft, Steven Struglia

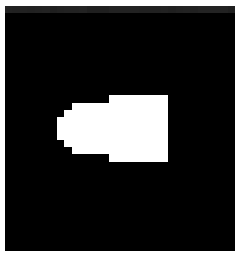
May 2020

1 Introduction

Community detection can serve a wide array of purposes in the field of data analytics and intelligence, spanning anywhere from social network clustering on the basis of being "friends" with certain individuals to clustering groups of people based off of their categorical response to a survey. The act of clustering is the partitioning of the vertices of a graph representation for a dataset into dense clusters with sparse connections between them. This idea is shown in the cluster figure below.



Our report will focus on the application of clustering algorithms applied to an image that contains two separate groups (image shown on the right), and use it to upscale the image from a 32 x 32 resolution up to an accurate 1024 x 1024 resolution while maintaining the key properties of the data. We were given a 1024 x 1024 weight matrix calculated using a combination of distances between each pixel in the image, as well as the gray-scale value of the pixel in order to calculate a weight to determine its group in the clustering.



The weight formula is defined as $W(i,j) = exp\left(-\frac{|I(i)-I(j)|}{20} - \frac{||i-j||^2}{10} \right)$

where $W(i,j)$ is defined as the weight between pixel i and pixel j in the image. When we use this formula to iterate over the entire 32 x 32 image, it results in a 1024 x 1024 symmetric weight matrix W. We then define the degree matrix D, a 1024 x 1024 diagonal matrix where D_{ii} is the number of non-zero weight connections coming out of pixel i, and $D = W \cdot 1$.

In constructing a clustering or partitioning of the groups hypothesized to exist in the graph, we first must construct an algebraic set of matrices that capture the connectivity characteristics of the graph, as defined below:

The set of edges in the graph is defined as ε

Laplacian Matrix

$$\Delta = D - W, \quad where \quad \Delta_{ij} = \begin{cases} d_i & where \quad \text{i = j} \\ -w_{ij} & where \quad (i,j) \quad in \quad \varepsilon \\ 0 & otherwise \end{cases}$$

Pseudo-Inverse Degree Matrix

$$D^{-1/2}, \quad where \quad D_{ij} = \begin{cases} 1 & where \quad \text{i = j} \\ \frac{1}{\sqrt{d_i}} & where \quad (i,j) \quad in \quad \varepsilon \\ 0 & otherwise \end{cases}$$

Normalized Weighted Laplacian Matrix

$$\widetilde{\Delta} = D^{-1/2}\Delta D^{-1/2}, \quad where \quad \widetilde{\Delta_{ij}} = \begin{cases} 1 & where \quad \text{i = j} \\ \frac{-1}{\sqrt{d_i d_j}} & where \quad (i,j) \quad in \quad \varepsilon \\ 0 & otherwise \end{cases}$$

We will use these characteristic matrices in a variety of optimization objectives in order to most appropriately partition the groups held inside of the image in order to perform a proper upscaling. We must first explore some point estimation characteristics of our graphs using two different models: The Erdős-Renyi Random Graph Model and the Stochastic Block Model.

2 Erdős-Renyi Random Graph Model

The Erdős-Renyi Random Graph Model attempts to model the existence of a graph from two parameters: n and p. The Erdős-Renyi class $G_{n,p}$ is defined as follows: Let V = set of n vertices {1, 2, ..., n} and G be the set of all graphs with vertices V. There are exactly $2^{\binom{n}{2}}$ such graphs, and p is defined under a probability mass function in the range [0,1]. Given the number of vertices and edges of a graph, we can construct a maximum likelihood estimation for p: $p_{MLE} = \frac{2m}{n(n-1)}$ where m represents the number of edges of the graph. For

the graph of our image in particular, the maximum likelihood estimator for p is 0.2768. From this estimator, we can compute the expected number of 3-cliques and 4-cliques present in the image graph, where 3-cliques are visually represented as triangles and 4-cliques are groups of four vertices where each vertex is connected to the other three vertices and not itself. Below is a table of the comparison of estimated 3-cliques and 4-cliques to the actual number of 3-cliques and 4-cliques in the image. The expected number of q-cliques given the existence of m edges in the graph is computed as follows:

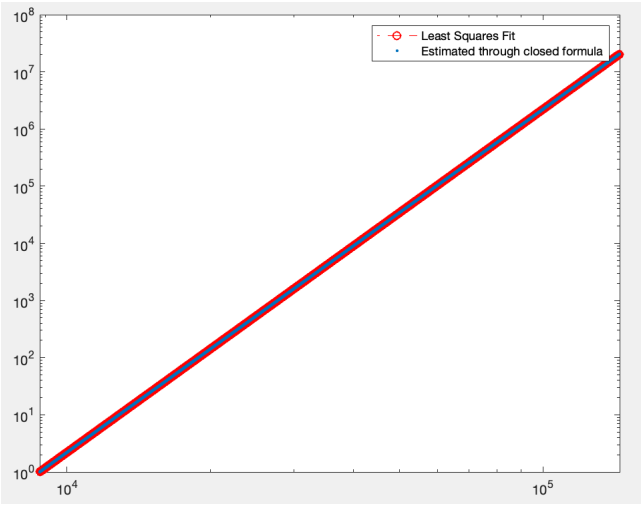
$$\binom{n}{q} \cdot \left(\frac{2m}{n(n-1)}\right)^{q(q-1)/2}$$

Estimated 3-cliques under Erdős-Renyi Model	Actual number of 3-cliques in graph
3,786,100	9,552,971
Estimated 4-cliques under Erdős-Renyi Model	Actual number of 4-cliques in graph
20,505,000	417,952,847

We then perform a log-log least squares regression on our Erdős-Renyi model by calculating the expected number of 4-cliques (X_4) for each iteration over the graph as we take the graph and reform it edge by edge. From this regression we compute the constants a_{ER} and b_{ER} under the equation $log(X_4) = a_{ER} * log(m) + b_{ER}$ where m is the number of edges at the current iteration and X_4 is computed using the appropriate maximum likelihood estimator for p on each iteration, given by the equation above. Below are the log-log plot (containing the least squares fit and estimation curves) and results of the regression.

Log-Log Plot of Erdős-Renyi Random Graph Model Estimation and Regression

$a_{ER} = 6.0$, $b_{ER} = -54.471$



3 Binary Stochastic Block Model

The Binary Stochastic Block Model assumes that there are two groups. Nodes in the same group are connected with probability a, and nodes in different groups are connected with probability b, where $a > b$. If we know the number of nodes in each group n_1, n_2, and the number of edges, which we can label m_{11}, m_{22} for intra-group edges and m_{12} for inter-group edges, we can calculate the maximum likelihood estimates of a and be to be:

$$a_{mle} = \frac{2(m_{11} + m_{22})}{n_1(n_1 - 1) + n_2(n_2 - 1)}$$

$$b_{mle} = \frac{m_{12}}{n_1 n_2}$$

However, when we are unable to know the group each node belongs to and we only have access to the edges and not the node grouping, solving the maximum likelihood estimator problem is difficult. To estimate our parameters with this information we turn to the method of moment matching. By solving the expectation of 3-cliques and 4-cliques we arrive at the estimates for a and b as:

$$a_{MM} = \frac{1}{2}(c_1 + \sqrt[3]{2c_2 - c_1^3})$$

$$b_{MM} = \frac{1}{2}(c_1 - \sqrt[3]{2c_2 - c_1^3})$$

where $c_1 = \frac{4m}{n(n-1)}$ and $c_2 = \frac{24t}{n(n-1)(n-2)}$.

We apply this method to our data and find that $a_{MM} = 0.5954, b_{MM} = 0.0417$. Using these values we can make an estimate for the total amount of 4-cliques in the graph and compare it to the actual number of 4-cliques.

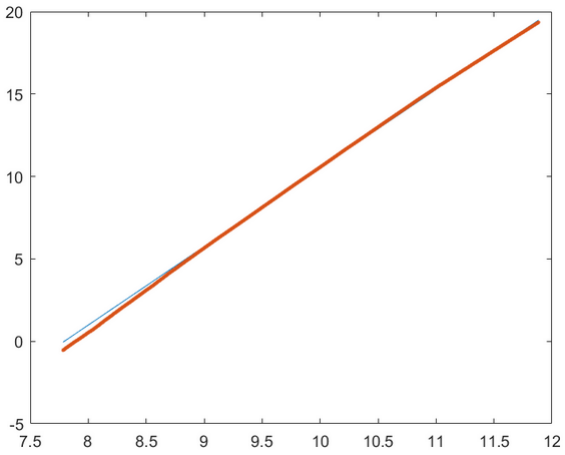
Estimated 4-cliques under Stochastic Block Model	Actual number of 4-cliques in graph
253,250,000	417,950,000

We see that the SSBM estimate is roughly half of what it actually is, compared to the Erdos-Renyi model which was off by a factor of 20, this performs relatively well.

After this, we use our ordered edge list to see how our estimate for a and b evolve as we add one edge at a time to our graph. Once this was done we did a linear fit of the data on a log-log scale. The results are plotted below:

Log-Log Plot of SSBM Random Graph Model Estimation and Regression

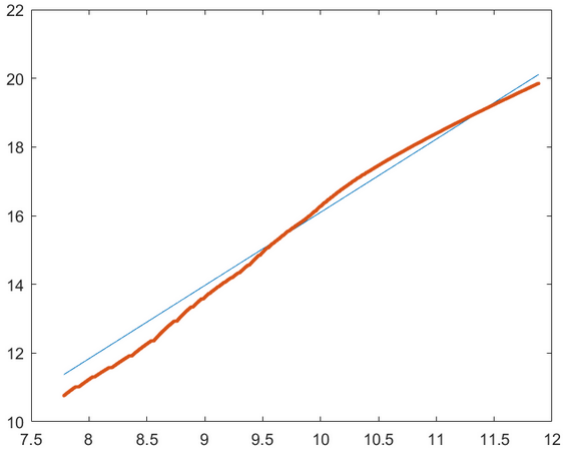
$a_{SSBM} = 4.7631$, $b_{SSBM} = -37.1215$



As we iterated through this ordered edge list we counted the amount of 4 cliques in the graph, once this was done we performed a linear fit of the data on a log-log scale. We excluded some initial points of the graph when fitting our data, as the graph does not behave linearly when there are not enough edges to from consistent 4 cliques.The results can be seen below:

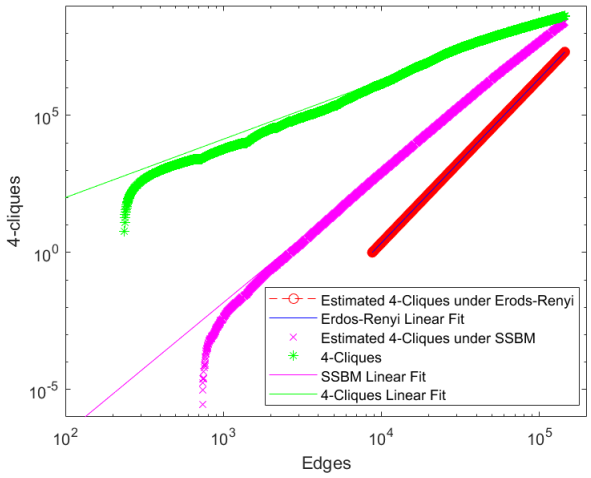
Log-Log Plot of Actual 4-Cliques in the Image Graph and Regression

$a_0 = 2.130547$, $b_0 = -5.2078$



We can now plot the Erdos-Renyi and SSBM models, and the actual number of four cliques as a function of edges with each of their respective linear fits all on the same graph:

Log-Log Overlay Plot of Both Models and Actual 4-Cliques with Regressions



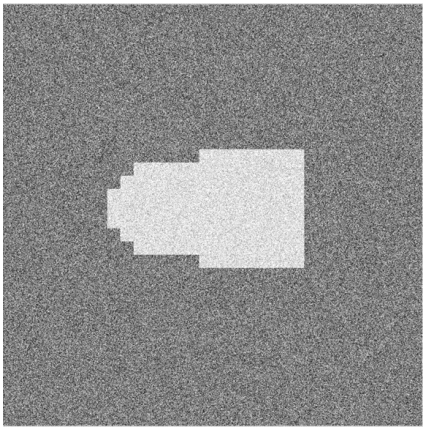
We see that the SSBM model always performs better than the Erdos-Renyi model, and it gets closer to the actual number of 4-cliques as we increase the edge size. However, it appears that soon after we stop, it would soon start overestimating the amount of 4-cliques in the graph.

4 Graph Partitioning through Spectral Algorithms

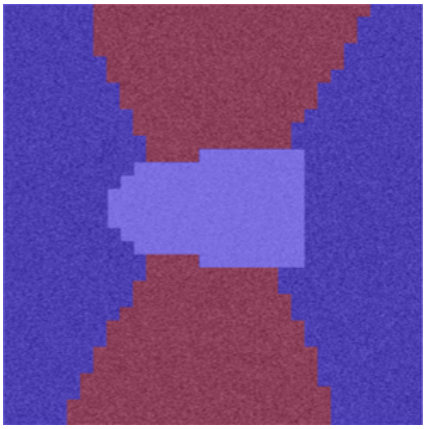
The goal of the spectral algorithms is to perform a disjoint partition into two clusters of the vertex set $V = S \cup \bar{S}$ that has the largest total weight inside each cluster while also maintaining a low cross-weight between clusters, in order to create two dense communities such that the connections between them are sparse. It has been found that using the second smallest eigenpairs for the weighted Laplacian (Δ) and normalized weighted Laplacian ($\widetilde{\Delta}$) matrices, also known as the Fiedler eigenpair, can serve as a partitioning vector over which the positive valued indices of the vector correlate to one group of vertices, while the negative and 0 values correlate to the other group of vertices. Likewise, using the raw weight matrix W, we can use the second largest eigenpair of W in order to create an analagous version of the Fiedler vector for the algorithm using W. Below, we detail the spectral algorithms using each of W, Δ, and $\widetilde{\Delta}$. These algorithms provide an approximate solution for the minimum-edge cut problem (classified as NP-Hard).

4.1 Spectral Algorithm Using Weight Matrix W

Input is the weight matrix W. First compute the second largest eigenpair (e_1,λ_1) of W such that $\lambda_1 > 0$. We then use e_1 as a partition vector to group the two underlying communities: $\Omega_1 = \{k : e_1(k) > 0\}$ and $\Omega_2 = \{k : e_1(k) \leq 0\}$. The output of this algorithm is the disjoint partition $\{\Omega_1, \Omega_2\}$, the two communities found from the weight matrix W. We take all the pixels assigned to the first community and color them blue and all the pixels assigned to the second community in red. We make the picture translucent and overlay it onto a noisy image of our original phantom image. The noisy image by itself can be seen below:



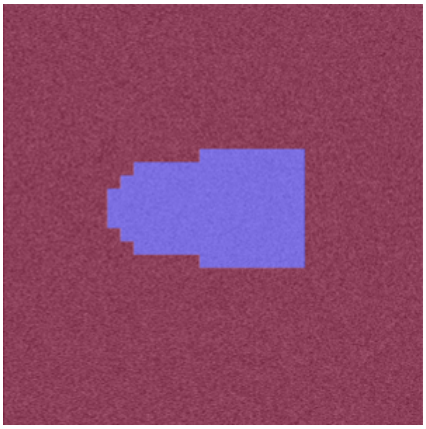
The overlayed images can be seen below.



We see that using the spectral method with the weight matrix does not perform optimally. Observe that the top and bottom sides of the image are captured by the partition but the sides are not. We attempt different spectral methods to see if we get better results.

4.2 Spectral Algorithm Using Weighted Graph Laplacian Matrix Δ

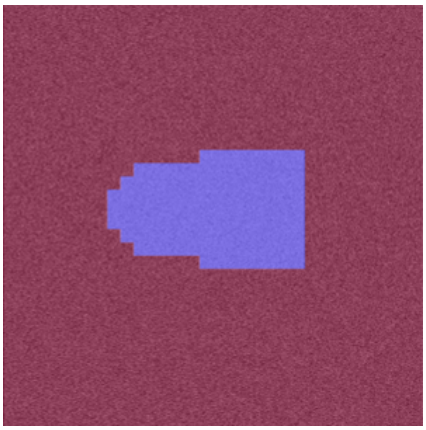
Input is the weight matrix W. First compute the weighted graph Laplacian according to its formula $D - W$ using pre-calculated D (degree) matrix. Next compute the second smallest eigenpair (e_1,λ_1) of Δ such that $\lambda_1 > 0$. We then use e_1 as a partition vector to group the two underlying communities: $\Omega_1 = \{k : e_1(k) > 0\}$ and $\Omega_2 = \{k : e_1(k) \leq 0\}$. The output of this algorithm is the disjoint partition $\{\Omega_1, \Omega_2\}$, the two communities found from the weighted graph Laplacian Δ. Just like before, the partition is overlayed onto the noisy image of our phantom.



We can see that this algorithm performs much better than the previous one. With 100% accuracy it partitions the phantom image into two groups.

4.3 Spectral Algorithm Using Symmetric Normalized Weighted Graph Laplacian Matrix $\widetilde{\Delta}$

Input is the weight matrix W. First compute the symmetric normalized weighted graph Laplacian according to its formula $D^{-1/2}\Delta D^{-1/2}$ using pre-calculated $D^{-1/2}$ (pseudo-inverse degree) and Δ (weighted graph Laplacian) matrices. Next compute the second smallest eigenpair (e_1,λ_1) of $\widetilde{\Delta}$ such that $\lambda_1 > 0$. We then use e_1 as a partition vector to group the two underlying communities: $\Omega_1 = \{k : e_1(k) > 0\}$ and $\Omega_2 = \{k : e_1(k) \leq 0\}$. The output of this algorithm is the disjoint partition $\{\Omega_1, \Omega_2\}$, the two communities found from the symmetric normalized weighted graph Laplacian $\widetilde{\Delta}$. The partition is visualized below.



Similar to the Laplacian algorithm, this algorithm performs the partition with 100% accuracy.

5 Graph Partitioning Through Semi-Definite Programs

We would like to solve the quadratic integer program:

$$min/max \qquad z^T S z$$

$$z \in -1, +1^n$$
$$z^T \cdot 1 = 0$$

However we are able to relax this program to a semi-definite one by defining $Y = zz^T$ and noting:

$$z^T S z = trace(z^T S Z) = trace(S z z^T) = trace(SY)$$

We can also note that

$$z \in -1, +1^n \iff Y_{ii} = 1$$

and

$$z^T \cdot 1 = 0 \iff Y \cdot 1 = 0$$

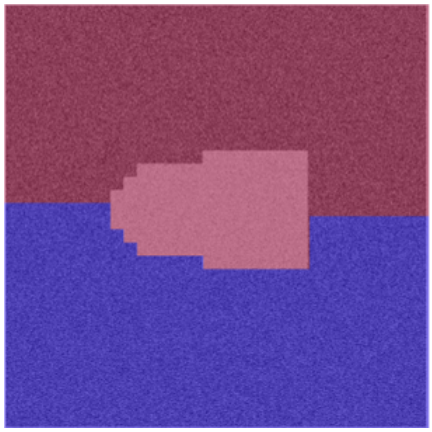
These hold as long as $Y \geq 0$ and $rank(Y) = 1$. However, we take away the rank condition and instead approximate the solution with the leading eigenvector. So we can relax the QIP into the following program:

$$min/max \qquad trace(SY)$$
$$Y = Y^T \geq 0$$
$$Y_{ii} = 1, 1 \leq i \leq n$$
$$Y \cdot 1 = 0$$

We consider three different algorithms: The weighted graph laplacian algorithm, where we approximate S with Δ as described above, the weighted normalized graph laplacian algorithm, where we approximate S with $\tilde{\Delta}$, and finally the weight matrix algorithm, where we approximate S with a weight matrix W. Once this is done we used CVX, a convex optimization solver in Matlab, to minimize the trace(SY) for the Δ algorithms, and maximise the trace(WY) for the weight matrix algorithm. Then, to satisfy the rank 1 condition, we take the eigenvector corresponding to the largest eigenvalue and define the partition as follows:

$$\Omega_1 = \{k : e_{max}(k) > 0\}, \quad \Omega_2 = \{k : e_{max}(k) \leq 0\}$$

When we applied all of our algorithms to our data we got the same partition for each, which can be seen below:



We see that the SDP's do not perfectly capture the correct partition. Observe that the program makes two partitions of roughly equal size. This is because in our constraints we restricted

$$z^T \cdot 1 = 0$$

which forces the partitions to be as close in size as possible. To counteract this in the future we could restrict the total amount of pixels we feed into the algorithm. If we can make the partitions have roughly equal space, we are confident that the SDP will achieve a more accurate partition.

6 Conclusion

	Weight Matrix	Laplacian	Normalized Laplacian
Spectral	0.1123	0	0
SDP	0.4287	0.4287	0.4287

Table 1: Partition Error

From the table above, we can see that the Spectral algorithms performed much better across the board for this specific image. Part of the reason for this discrepancy in the algorithm accuracy is the invariant of the SDP algorithms that require the two communities to have as equal volume as possible. Since this image had an unbalanced grouping of the pixels in the two communities, it lead to a fairly inaccurate detection through SDP. The Spectral algorithms matched the communities at a 100% accuracy through the use of the weighted graph Laplacian (Δ) and normalized weighted graph Laplacian ($\widetilde{\Delta}$), and a slightly lesser accuracy of 88.8% through the use of the weight matrix (W). The SDP algorithms all resulted in the same partition regardless of whether it utilized W, Δ, or $\widetilde{\Delta}$, leading to the same accuracy of 57.1%.