Math 581.05
Week 8: Clustering and Graph Embeddings

Daryl DeFord

October 12, 2020

1 Introduction

Today we are talking about (and experimenting with) clustering methods on graphs. This is a setting, unlike the previous couple of days, where there actually is a fair amount of convergence between the pure and applied worlds, because we end up trying to answer similar optimization questions in both contexts. However, even though there is a great deal of similarity between the techniques, the motivating questions are quite different. Broadly, clustering or graph partitioning refers to the problem of decomposing the set of nodes of a graph into a collection of disjoint subsets. The properties that are used to select and evaluate the subsets are usually rooted in some notion of similarity between the nodes that are paired together, or some notion of difference between the nodes that are separated.

In this session, we are going to examine some of the most commonly used methods in network science for constructing graph partitions and then try to translate these methods to our census dual graphs. There are two main problems that we will encounter when trying to employ this methods for redistricting. The first is that the population values on the nodes of the graph are quite heterogenous and so the methods have to be modified to balance the populations per district, not necessarily the number of nodes. Secondly, for us, contiguity is an absolute priority, while many of the network focused methods do not necessarily require this to be the case.

Our main applications will be to generating population balanced seed plans and varying the choice of biparitioning method for the ReCom proposal. ReCom itself highlights an interesting point that is usually the case for any of these methods that bipartitioning is easier than k partitioning. For generating new seeds we will compare different ways to apply recursive bipartioning methods, as well as methods that attempt to form k districts all at once.

2 What makes a cluster?

Like many problems in network science, the choice of definition of a cluster is an attempt to generalize intuition gained from looking at examples with obvious solutions. Two main viewpoints that are studied in this area are minimizing the number of edges that have to be cut to separate the communities or maximizing the number of edges within the communities. Note that these are both formulated as optimization problems, if we view either of these perspectives as the "right" definition of community[1] then we can compare two potential clustering of a give network against each other. The first definition leads to the notion of spectral clustering while the second leads to modularity both of which are discussed in detail below.

One of the questions that must be addressed before applying either of these methods is how many clusters we are trying to find. Sometimes, we may expect to see a certain number of clusters depending on extra information that we know about the network. For example, if you look at the social network of middle school students you might expect to see several clusters per grade, separating the student body along racial or gendered lines or the karate club network, where we know which individuals joined each group after the schism. Other times, the structure of the network itself makes the answer clear. Thinking back to networks generated by an assortative SBM model shows that sometimes there is a "reasonable" answer.

[1] See this paper for a discussion of how well network models actually capture the "ground truth."

However, our census networks (viewed as abstract graphs) have very little of this structure. As an extreme version of this think about the grid graphs we have been partitioning - is there any way to define a “correct” notion of community on these graphs? Even worse, for the districting problem we are simply presented with the number of districts a quick look across the sizes of state legislatures in the country will quickly show that there not much of a global organizing principle guiding this choice and it certainly has nothing to do with the topological connections between adjacent census units. Thus, any network approach that doesn’t incorporate additional information is unlikely to succeed in this setting, the graphs are too regular and the number of communities is decided without reference to the graph itself. These are the types of issues that we are grappling with when we try to grow new seeds. Additionally, the space of contiguous, population balanced districting plans is wild and complicated. We have a great deal of recent work showing that the kinds of plans that we visualize as reasonable answers to the districting problem are just small islands separated by a sea of fractal snakes.

The preceding paragraphs may be a little depressing, so it is worth discussing why people have studied clustering at all. The first motivation is of direct interest to the social structure being represented by the network - clusters help us identify groups with similar connectivity behavior. From a dynamic perspective we can think about this in terms of (information) flows. Clustering helps us understand which groups share information easily, who is most able to control the flow of information, and whether or not there are bottlenecks in the process, much like some of our centrality measures from Monday were trying to measure. These techniques are used by applied social scientists to refine their studies and better understand the groups that they are interested in.

Another similar reason to care about networks is the notion of dimension reduction. People are complicated. This means that representing them with a single, fixed network is too rigid and sensitive to noise in the collection process. Cluster analysis can instead let us represent (a set of) fundamental units of the structure we are trying to understand without getting bogged down in the details of individuals behavior. This is a common technique in data analysis and machine learning, although the associated moral quandries are vastly underexplored.

3 Clustering and Partitions

One of the fundamental problems in network analysis (and more broadly all of data analysis) is revealing meaningful substructures from large networks. This approach is complementary to dimension reduction since our goal is to associate individual nodes that are similar in some fashion into larger clusters that represent more homogeneous components of our data. Thus, we are attempting to minimize the number of relevant data points instead of minimizing the number of dimensions associated to each point. Some techniques such as $k-$means and linkage apply to data analysis more broadly, while spectral methods and modularity rely on the algebraic and graph theoretic structure of an underlying network.

However, there are ways to move in between these interpretations. For example, geodesic difference provides a metric on the graph. Conversely, if we use a dimension reduction technique to map arbitrary data into $\mathbb{R}^n$ we can use radial connections or nearest neighbor connections to form graphs from the embedding.

3.1 Thresholding

Thresholding is a basic technique from data analysis for grouping data given with some application derived similarity metric. If no such metric is given some of the embedding techniques from dimension reduction can be employed to endow the data with a Euclidean structure. To perform clustering by thresholding an initial threshold t is selected. Then, two points i and j are then put into the same cluster if $S_{i,j} > t$. This is a simple operation to perform, but can be very sensitive to the choice of t.

In general, proportionally high thresholds lead to more meaningful groupings while low thresholds tend to identify many more relationships but without obvious interpretability. Thus, determining an application appropriate threshold value (or collection of threshold values is essential to obtaining interesting information using thresholding. One option for selecting an appropriate threshold is to compute a histogram or proportion plot of threshold values and try thresholds at gaps or steady states respectively. Alternatively, since these groupings are efficient to compute a range of threshold values can be tested and interpreted individually.

3.2 Linkage

Linkage is a hierarchical clustering method similar to thresholding. It is an iterative approach where at each step we combine the clusters that are the most similar. These iterations define a hierarchical structure, where clusters formed at earlier steps are more homogeneous than the connections made at later steps. There are several types of linkage grouping, but the basic outline of the iterative process is as follows, with the initialization placing each node into a separate cluster:

1. Determine which pair of clusters has the highest similarity value
2. Merge these identified clusters and reindex the cluster list
3. Update the similarity values to incorporate the new conjoined cluster
4. Repeat until there is only a single cluster

Obviously there are several details that must be specified before implementing this algorithm. For example, what if multiple pairs of clusters in step (1) have the same similarity score (usually one pair is chosen randomly, but this can lead to slightly perturbed results). Additionally, in step (3) there are several different methods that are used to compute the new similarity values:

- (single linkage) In single linkage the new similarity value between cluster a and cluster b is the maximum over all pairwise similarity scores between nodes in a and nodes in b.
- (complete linkage) In complete linkage the new similarity value between cluster a and cluster b is the minimum over all pairwise similarity scores between nodes in a and nodes in b.
- (average linkage) In average linkage the new similarity score is computed as the average over all pairs of nodes between the clusters.

Clearly, these different methods can have significant impacts on the final clustering results. Again, this is an efficient technique from a computational perspective and comparing the results from different versions of the method can yield interesting insights. Particularly, the distinction between results from single and complete linkage since depending on the underlying similarity distribution they can return quite different results.

3.3 k-means

The k–means method is another iterative clustering approach from data analysis that relies on embedding the nodes in $\mathbb{R}^n$. In this approach we are attempting to determine a set of canonical representatives such that each data point can be associated to a representative while minimizing the total distance between points and representatives. To begin this process, the number of clusters, k, to be obtained must be specified in advance. Usually this is guided by some intrinsic knowledge about the data, but also may be experimented with numerically. The method then initializes with a random selection of k points in $\mathbb{R}^n$.

Each node is then assigned to the cluster corresponding to the initial point that it is closest to. Each step of the algorithm then replaces the previous representative points with the centroid of each cluster from the previous step. The nodes are reassigned clusters based on distance to the newly computed centroids, and this updating continues until the clusters converge. At that point, the centroids and clusters are no longer changing, so the centroids are taken as representative elements, and the clusters are defined by this steady state.

Although for a given initial set of points this process is deterministic, the initial selection of points can lead to very differing results. Thus, in practice the algorithm is run many times on different selections of initial points, with the final results being given as the output with the smallest total distance between the points and centroids in the steady state. It is often valuable to examine the histogram of total distance values as each run of the algorithm finds a local minimum. Thus, modal data in the distance values may represent different types of clustering behavior. As the k value must be set in advance, this is important because it may suggest a may appropriate value or highlight other features of the data.

3.4 Spectral Clustering

Spectral clustering is an algebraic method that applies directly to networks. The goal of spectral clustering is to find a partition of the nodes into k sets such that the number of edges between the sets is minimized. This can be thought of as minimizing the amount of damage to the entire network if the clusters were disconnected from each other. However, solving this problem directly is NP–hard so the problem for large networks is traditionally relaxed to allow for an elegant solution in terms of eigenvalues.

We begin by describing the case where we wish to partition the network into two components. In this case we wish to find a vector v with entries of ± 1 representing the two clusters that minimizes the disconnection damage. Some simple algebraic manipulations after reformulating this expression in terms of the adjacency matrix, show that we are interesting in minimizing v^tLV, where L is the graph Laplacian introduced previously. Unfortunately, this problem is still NP–hard.

However, if we relax the restriction that the entries of v be ± 1 and instead require that $|v| = 1$ and $v \perp \mathbf{1}$ then Lagrange multipliers show that the solution is given by the eigenvector corresponding to the smallest non–zero eigenvalue of L. Then, we can use either the signs or median of the entries of this eigenvector to partition our network. Sometimes, eigenvectors of other versions of the Laplacian are used instead, including the normalized Laplacian and the random walk matrix. The results obtained from these matrices are usually equivalent to those obtained from L. To partition the nodes into k clusters, we usually use k eigenvectors corresponding to the smallest k non–zero eigenvalues and perform k–means on the associated node values.

Derive the formula for the three spectral clustering methods.

The ideas behind the three basic spectral clustering methods are that we wish to partition the network into two pieces such that if we disconnect the two components we do a minimal amount of “damage” to the network as a whole. The definition of the damage function is what distinguishes the various methods. The most naïve method uses the Cut formulation. In this case, we wish to minimize the number of edges that must be removed to disconnect the two components. Unfortunately, solving this optimization problem for the components is NP–hard, although it can be computed exhaustively for small networks. Instead, we approach a relaxed version of the problem. We begin by constructing a vector v to represent the desired partition, with entries of 1 assigned to nodes in one component and entries of -1 assigned to the other component. This gives that $v_iv_j = 1$ if and only if i and j are in the same component. We can now formulate the damage condition algebraically as

$$\text{damage}(v) = \frac{1}{2}\sum_{i,j}\frac{1}{2}(1 - v_iv_j)A_{i,j}.$$

Proceeding by algebraic manipulation, we can reduce this expression to

$$\begin{aligned} \frac{1}{2}\sum_{i,j}\frac{1}{2}(1 - v_iv_j)A_{i,j} &= \tfrac{1}{4}(\textstyle\sum_{i,j} A_{i,j} - v_iv_jA_{i,j}) \\ &= \tfrac{1}{4}\textstyle\sum_{i,j} v_i \deg(i) v_j \delta_{i,j} - v_iA_{i,j}v_j) \\ &= \tfrac{1}{4}v^TDv - v^TAv \\ &= \tfrac{1}{4}v^TLv \end{aligned}$$

Now we have reduced our problem to minimizing this bilinear form over all vectors $v \in \{\pm 1\}^n$. Unfortunately, this is still an NP–hard problem so we relax our constraints to minimize over all real vectors of norm one, where the norm condition rules out trivial minimization solutions. We further require that $v \perp \mathbf{1}$ since $\mathbf{1}$ is in the kernel of L and carries no information for our clustering. The theory of Lagrange multipliers then gives that the solution vector to our minimization problem is the eigenvector corresponding to the Fiedler value of L. We can assign nodes to components by separating the positive values and negative values.

The problem with the direct cut formulation is that it says nothing about the relative sizes of the partition. In order to rule out trivialities, such as disconnecting a pendant edge, two other types of damage metrics are frequently used in the literature. These are the RatioCut which scales the damage by the number of vertices in the subsets and the NormalizedCut which scales the damage by the number of edges in the subsets. Symbolically, these are

$$\text{RatioCut}(A, B) = \frac{1}{2}(\frac{E(A,B)}{|A|} + \frac{E(A,B)}{|B|})$$

and

$$\text{NormalizedCut}(A, B) = \frac{1}{2}(\frac{E(A,B)}{\text{vol}(A)} + \frac{E(A,B)}{\text{vol}(B)}).$$

The RatioCut derivation leads to a very similar optimization structure as the (relaxed) Cut problem. In this case we assume that there is again a vector v representing the partition, but this time the entries in v are proportional to the size of the component that the vertex is selected from: $v = \begin{cases} \sqrt{\frac{|B|}{|A|}} & i \in A \\ -\sqrt{\frac{|A|}{|B|}} & i \in B \end{cases}$. We can simply compute that $v^T L v$ again gives exactly the cut value and that $||v||^2 = v^t v$ is exactly the denominator that appears in the definition of the RatioCut. Thus, we are minimizing the expression $\frac{v^T L v}{v^T v}$ which is the Rayleigh quotient associated to L. Since $v\mathbf{1} = \mathbf{0}$ by construction this expression is minimized by the second eigenvalue of L as in the previous case. Thus, we again take the second eigenvector of L to form our partition.

If we proceed as in the standard case for the NormalizedCut, we can again imagine a partition vector v, with $v_i = \frac{1}{\text{vol}(A)}$ if $i \in A$ and $v_i = \frac{-1}{\text{vol}(B)}$ if $i \in B$. Then, we see that $v^T L v = E(A,B)(\frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)})$, while $v^T D v = \frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)}$. Thus, the normalized cut corresponding to v is the ratio of these two values. That is we wish to minimize $\frac{v^T L v}{v^T D v}$. Using the substitution $D^{-\frac{1}{2}} w = v$ this becomes $\frac{w^T D^{-\frac{1}{2}} L D^{-\frac{1}{2}} W}{w^T w}$ which is the Rayleigh quotient for the normalized Laplacian $I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Minimizing this expression is equivalent to finding the second eigenvector of this normalized Laplacian and we may partition the network with this vector as above.

Interestingly, the stochastic matrix AD^{-1} can be used to solve the NormalizedCut problem. In this case if λ, v are an eigenpair of AD^{-1} then $(1 - \lambda), v$ are an eigenpair for the normalized Laplacian. This relationship is more clear when we note that $AD^{-1} = D^{-\frac{1}{2}}(I - D^{-\frac{1}{2}} L D^{-\frac{1}{2}}) D^{\frac{1}{2}}$. The NormalizedCut value of a partition Q can be realized in the random walk setting as $\text{NormalizedCut}(Q, \overline{Q}) = P(Q|\overline{Q}) + P(\overline{Q}|Q)$ which provides some intuition for the relationship. Thus, finding the second largest eigenvalue of AD^{-1} and its corresponding eigenvector also gives a solution of the NormalizedCut problem on a network.

3.5 Modularity

Modularity is a measure of the proportion of connectivity with clusters to connectivity between clusters. Given a partition of the network into clusters, the modularity is defined as the difference between the number of edges that occur within each cluster and the number of edges that would be expected if the edges were distributed randomly between the nodes while keeping the same degree distribution. Other variants of the random edge distribution are possible, and can be adjusted based on the application. In general, the probability that two arbitrary nodes in the network are connected in a random network preserving degree distribution is approximately $\frac{\deg(i) \cdot \deg(j)}{2m}$ for large networks, where m is the total number of edges in the network.

This is an example of a null model analysis with the configuration model providing the null model. Although there are many similarities between modularity and spectral clustering, the focus on density is a distinguishing feature because spectral clustering does not account (except incidentally) for the interconnectivity within the clusters it discovers.

This can be formulated and computed linear algebraically, by forming the modularity matrix B from the adjacency matrix A as $b_{i,j} = a_{i,j} - \frac{\deg(i) \cdot \deg(j)}{2m}$. Then, we form a matrix C representing the clusters whose entries are $c_{i,j} = \begin{cases} 1 & \text{if node } i \text{ is in cluster } j \\ 0 & \text{otherwise} \end{cases}$. This allows us to compute the modularity value of the partition as $\frac{1}{2m}\text{trace}(C^t B C)$. Recasting this problem as an optimization problem as in spectral clustering allows us to attempt to determine the appropriate clusters for the network. In this case we are maximizing $s^t B s$ and so the eigenvector corresponding to the leading eigenvalue is usually used to construct the initial clusters. Several algorithms such as simulated annealing, have been applied to give solutions to this optimization problem. This can also be recast as a spectral problem in terms of the modularity matrix B. A complete derivation of modularity for scalar characteristics can be found in the section describing assortativity above.

How does modularity use a null model to determine communities in the network?

The modularity of a network is a function that maps a partition of the network to a value measuring the proportion of connectivity that occurs within the clusters compared to the edges between clusters. Given a partition of the network into clusters, the modularity is computed as the difference between the number of edges that lie within the observed clusters to the number of edges that would occur if the edges had been distributed uniformly with the same degree distribution. This is usually normalized by the maximal value obtainable from the configuration model.

This is an example of a null model because the you are comparing the observed network to a theoretical model, of a network with same degree distribution and randomly placed edges, in order to determine the significance of the observed data. This particular model is formed by splitting each edge in half and reattaching the edge ends at random. Subtracting off the amount of clustering that appears in the null model leaves behind the extra (or deficient) amount of clustering that appears in the observed network.

The definition of modularity is also frequently extended to cover scalar partitions where each cluster is assigned a relative value. In this case the modularity can be interpreted as a covariance of the partition labels. Specifically, the notion of assortativity is a scalar modularity with the partitions defined by the degrees of the nodes.

3.6 Edge Deletion

If we are trying to decompose a network into connected components, one natural methodology is to think about ways to delete edges until we have reached the desired number of pieces. Many choices of edge methods are available and indeed, we have talked about one (spanning tree cutting) extensively. There are also various max flow or min cut algorithms that can be used to find similar types of partitions, discussed in the final section below.

Another common method uses a centrality measure on edges to define the partitions. Given a function that represents the importance of each edge, we can remove the most important edge at each step and then recalculate, restricting to connected components as necessary. Edge betweeness centrality, where you measure the proportion of shortest paths between all pairs that use a given edge is the standard choice for this type of partitioning. As discussed on Monday, given that the underlying assumption of betweeness is at the deterministic end of the "knowledge-of-all-paths" spectrum, it is also interesting to compare to something like random walk betweeness centrality.

3.7 Hierarchical methods

So far, most of our methods have focused on a top down approach to clustering, where we start with the entire graph as a community and try to figure out how to slice it into pieces. Instead, we could start with each node in its own cluster and think about devising methods that merge clusters together until we have reached a desired number. One approach is to use an edge centrality score as above but this time at every step, merge the two clusters that are joined by the most significant edge. What does this say about the interpretations of the edges/scores in these two versions?

One advantage to this type of approach is that it provides a hierarchy of clusters, sorted by their relevance under our choice of merging procedure. These are usually visualized with a dendrogram, which shows which nodes/clusters were merged at each step. This type of multiresolution clustering offers additional information about how strongly the nodes are coupled and can be used to help make informed decisions about which the appropriate number of clusters, in situations where we aren't told in advance how many clusters to use.

3.8 Belief Propogation

Keeping with our electoral theme, we can't leave out methods where the nodes vote on their assignments at every step. Beginning with an initial assignment of nodes to districts, we proceed iteratively. At each step, each node changes their assignment to the assignment held by the majority[2] of their neighbors. This process continues until no nodes change assignment (or until we get bored).

[2] or plurality or RCV-STV etc. ☺

4 Graph Embeddings

In our discussion of null models we looked at a couple of examples of geometric graphs that were constructed by associating a (distribution over) graph(s) to a point cloud. Today we are also interested in the opposite problem of representing the nodes of a graph as a point cloud, first using the spectral properties of some associated matrices and then some Euclidean embeddings where the distance measure between the points in space reflects a natural notion of distance on the graph.

4.1 Spectral Embeddings

One common technique is to use either the adjacency matrix or the Laplacian matrix associated to the graph in order to embed the points in the plane. Specifically, we compute the eigenvectors of the matrix, sorted by the magnitude of the corresponding eigenvalues and select the k vectors corresponding to the first k values. We then use the rows to embed our nodes - the ith node corresponds to the ith row. Usually we select $k = 2$ or $k = 3$ for ease of visualization. We will discuss more of the properties of these embeddings tomorrow when we talk about network dynamics.

4.2 Dimension Reduction

Dimension reduction techniques broadly fall under the heading of data analysis. They are used for many different purposes in all parts of applied mathematics. The key idea is to reduce in some fashion data that is presented is high dimensional space to a natural embedding in a smaller dimensional space that preserves as much of the original structure as possible. Mitigating the curse of dimensionality, providing useful visualizations, both for exploratory and explanatory purposes, and cleaning noisy data are all parts of dimension reduction techniques and methods. Many of these techniques and their results are intrinsically related to problems of clustering. We also note that standard network clustering techniques can be applied to data sets by interpreting the data as a network, perhaps by using thresholding or k nearest neighbors to form a representative graph for the data.

4.2.1 Multidimensional Scaling

Multidimensional scaling (MDS) is a technique for embedding data into a Euclidean space so that the Euclidean distance (or another metric on $\mathbb{R}^n$) can be used as a "good" approximation to a given metric on a dataset. That is, given a dataset $X = \{x_i\}$ and a metric[3] $d : X \times X \to \mathbb{R}$ we wish to form an embedding $\varphi : X \to \mathbb{R}^k$ for some k together with some metric d' on $\mathbb{R}^k$ so as to minimize $\sum_{i,j} |d'(\varphi(x_i), \varphi(x_j)) - d(x_i, x_j)|$ or some similar variant. Usually we begin with a dissimilarity matrix $D_{i,j} = d(x_i, x_j)$ and state (and solve) the problem linear algebraically by forming a derived matrix that can represent the new inner products.

The choice of k obviously influences the embedding process a great deal. For visualization of course, two or three dimensions are required, or alternatively there may be some intrinsic dimensionality to the data that is known in advance. For example, we may know or expect certain dimensions of our data are highly correlated and subtract these superfluous quantities from k. Alternatively, if our data stems from a Euclidean metric originally then the points can be embedded exactly using linear algebra and factoring a symmetric positive definite matrix. This method works for all data that comes from a metric and has the additional property of revealing the smallest dimension that provides an exact embedding as the rank of the derived matrix.

When the original measure is not a true metric it is not always possible to solve this problem exactly. One possible approach to correct this defect is to try to convert d into a metric (or at least get it closer), such as is done with cosine dissimilarity measure. Regardless associated to any such embedding is a measure of stress that determines the effectiveness of an embedding at capturing the original information. These stress measures are frequently statistical and dependent on the particular data and can be used to search for the proper number of dimensions for an embedding.

Most of the standard software systems for data analysis (Matlab, R, Python, etc.) incorporate methods for doing MDS. These are usually computed iteratively from a random embedding, by adjusting the positions of

[3] Need not be an actual metric, but the closer it is to satisfying all of the metric properties the better results obtained from it will be.

the points slightly at each stage to minmize the stress function measuring the normalized difference between the embedded distances and the original dissimilarity matrix entries. Due to the random nature of the initialization it is possible that the software can return a local minimum instead of a global minimum, so it is recommended to perform the operation multiple times and select the embedding that minimizes the stress function.

Show that if D is a distance matrix giving distances between points in $\mathbb{R}^n$ the MDS will recover the coordinates of the points up to a rigid motion.

Let D be a matrix whose entries represent differences between k points in $\mathbb{R}^n$. That is $D_{i,j}^2 = \mathrm{d}(x_i, x_j)$. Since the points are assumed to be Euclidean already, we can realize the distance as an inner product: $D_{i,j}^2 = ||x_i - x - j|| = \langle x_i - x_j, x_i - x_j \rangle$. If X is a $n \times k$ matrix whose columns are the entries of the x_i then the matrix $A = XX^T$ has entries $A_{i,j} = \langle x_i, x_j \rangle$. Thus, if we can construct the matrix A from D we can recover X from the spectral decomposition of A since it is symmetric and positive definite by construction.

Looking entrywise we see that $D_{i,j}^2 = \langle x_i, x_i \rangle - 2\langle x_i, x_j \rangle + \langle x_j, x_j \rangle = A_{i,i} - 2A_{i,j} + A_{j,j}$. Considering the entries of A as k^2 variables we obtain a system of equations that can be solved exactly to obtain A. Since we can obtain X from A, this gives us at least a translate of the original vectors up to a rotation since the Euclidean distance is translation and rotationally invariant. In practice however, these problems are usually solved with iterative approaches for a fixed dimension, using a stress measure as an objective function.

Another approach is to form the matrix $M_{i,j} = \frac{D_{1,i}^2 + D_{1,j} - D_{i,j}^2}{2}$. This is also a symmetric matrix whose spectral decomposition gives rise to another realization of X as above. In this case we are shifting the first element in X to the origin and then the entries of M represent the differences from x_0 to each other point in the data set. The rank of M captures the minimal dimension such that the distances can be realized exactly in Euclidean space with the standard metric.

4.2.2 Principle Component Analysis

Principle component analysis (PCA) is a technique from linear algebra for projecting a dataset of vectors onto a lower dimensional subspace in such a way as to capture most of the variance of the original data. The idea is that we can use the spectral decomposition of the covariance matrix of a data set (derived from the $D_{i,j}$ above) to suggest the most efficient vectors to project our information onto by selecting the eigenvectors corresponding to the largest eigenvalues. Specifically, we compute the sample covariance matrix $\frac{1}{n-1}(X - \hat{X})(X - \hat{X})^T$ and use its orthogonal decomposition.

In practice, parts of this decomposition may be done by SVD both for speed and numerical stability. The total variance is preserved under the diagonalization, but the new random variables are clearly uncorrelated in the change of variables. Thus, the eigenvalues, sorted descending by magnitude, describe decreasingly small proportions of the total variance explained by projecting onto the first k eigenvectors. Note that these projections are orthogonal since the covariance matrix is symmetric. This means that the projections can be computed independently for the most meaningful k eigenvectors to achieve the representation in $\mathbb{R}^k$ that explains the most variance possible.

5 Applications to Redistricting

So what does this all have to do with redistricting? Well, many of the initial ensemble analyses of districting plans used various network theoretic techniques to build their plans. And even now that we are mostly thinking about doing Markov chain sampling, it is useful to have a collection of diverse seed plans to compare as different starting points for the chains, to evaluate mixing. Some slides discussing this material are here.

Our exercises are going to be focused on partitioning a 10×10 grid into 10 contiguous pieces. We will start with the standard network measures and then add population values and additional constraints to our clustering. As many of you are discovering, generating a good starting plan for your ensemble can be quite difficult. We are going to work on ways to generate good new plans and clean up plans that aren't necessarily as good.

5.1 Recursive Methods

Given a bipartioning method we have two ways, each with two sub-options to create a new set of plans. They are:

1. Recursively applying the methods to generate full k partitions from scratch. Given a graph G, each method below can be applied to generate a contiguous component with population bounded by ideal population $\pm\delta\%$ whose complement is also connected. We can continue to apply the method to the complement to eventually partition the entire graph. Alternatively, we could instead use a "binary" recursive method where we attempt to split by pairs

2. ReCom also provides a way to generate a new plan from a starting plan (if we don't have one, we could always take $k-1$ individual nodes and then group the rest together to start with). At every step we merge a pair of districts and repartition them but we could also merge several districts or several pairs of districts and then k partition them instead.

Each of the following methods has the property that any permissible partition occurs with positive probability – sadly not uniform. In addition, the simple walk version does have the property that if it is possible to go from partition a to b it is also possible to go back. There are plenty of variants of these methods, the ones listed below are just those that are currently implemented. One interesting variant for many of these methods would be to weight the edges by demographic or municipal similarity instead of independently uniformly.

6 Methods for generating partitions

- Inputs:
  - Graph to partition into two pieces
  - Target population
  - δ – allowable population deviation
- Ways to generate:
  - **Tree Sampling**
    - Randomly weight the edges of G in $[0, 1]$
    - Use Kruskal's algorithm to generate a random spanning tree.
    - Snip it somewhere balanced
  - **Min–Cut**
    - Choose a random start and end node
    - Weight the edges exponentially proportional to min distance from either the source or sink (currently $10^{\min -3}$)
    - Compute the min cut
    - Repeat until population balanced.
    - **Variant:**
      - Randomly delete edges until graph disconnects
      - If a deleted edge would disconnect but not population balance add it back to the graph
      - Repeat until partitioned
  - **Hierarchical/agglomerative**
    - Each node starts in own cluster
    - Select an edge between clusters
      - Merge clusters (contract edge in graph) if population allows and doesn't disconnect the complement

    - If population doesn't allow, delete edge in graph
    - If merging would disconnect the graph, merge the smallest population component that would be disconnected into the nodes.
  - Repeat until only 2 clusters
- **Flood Fill**
  - Add a random node to the district
  - Select a random neighbor of the district
  - If adding it to the district doesn't disconnect the complement and doesn't raise the population too high add it to the district
  - If it would disconnect the complement, merge the smallest population connected component into the neighbor
  - If the population would be too large, remove the edge from the graph
  - Repeat until the district has the appropriate population.
  - **Variant:** Select all neighbors of the district – add until they run out and then recompute neighbors
- **Path Fill**
  - Choose a node to start the district
  - Choose another arbitrary node in the graph
  - Compute a shortest path between the district and the new node
  - Add the entire path between the nodes if it doesn't disconnect the complement or add too much to the population
  - Repeat until the district population is large enough
  - **Variants:**
    - If the path would disconnect the complement, merge it into a node
    - Contract the edges each time a path is added
    - Only permit paths to be added if their total sum is less than the target population
  - **Graph Algorithms:**
    - **Spectral Clustering**
    - Randomize edge weights until population is balanced
    - Pick your favorite Laplacian and use spectral embedding
    -
    - **Modularity**
    - Use municipalities as scalar measure along with random edge weights
    - Alternatively, use similar population as scalar measure
    - **Centrality**
    - Use a function of edge/node centrality to determine weights for all the methods above.