Approaches to Brain Parcellation using Energy Statistics and Graph Partitioning

Felix Xiao

Advisor: Professor Han Liu

Submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Engineering
Department of Operations Research and Financial Engineering
Princeton University

June 2016

Abstract

We formulate the task of brain parcellation (dividing the brain into functionally homogenous regions) as a graph partitioning problem. We devise new model-free criteria for validating parcellations based on statistical dependency between parcels using distance correlation. Based off our criteria we pose a new graph cut-type problem called Max Average Within Edge (MAWE), wherein the objective is to maximize the sum for each component, of the average weight of all edges with both endpoints in the component.

In chapter 4, we propose a family of heuristic algorithms based on a new Contractible Graph data structure for attaining good solutions for MAWE and give the results of these algorithms on a resting state fMRI scan.

In chapter 5 we explore the well-established spectral ratio-cut minimization technique and measure its performance in MAWE on the same data set.

In chapter 6 we do the same for the more recently proposed nonnegative adjacency matrix factorization method, an alternative to spectral partitioning.

In chapter 7 we show that the MAWE problem can be reduced to a special instance of generalized 0-1 fractional programming which has an mixed integer programming equivalent.

Chapter 8 presents the results of these partitioning methods on 20 (?) autism and control fMRI scans.

Acknowledgements

Contents

List of Tables

List of Figures

Chapter 1

Functional MRI Data and Brain Parcellation

Chapter 2

Energy Statistics

2.1 Energy Covariance

For some positive weight function $w : \mathbb{R}^p \times \mathbb{R}^q \mapsto [0, \infty)$ define the norm $\| \cdot \|_w : \{\gamma : \mathbb{R}^p \times \mathbb{R}^q \mapsto \mathbb{C}\} \mapsto [0, \infty)$ as

$$\|\gamma\|_w^2 = \int_{\mathbb{R}^{p+q}} |\gamma(s,t)|^2 w(s,t) ds dt$$

Definition 2.1.1 *(Distance covariance). Let X and Y be two d-dimensional random vectors with $\mathbf{E}\|X\| + \mathbf{E}\|Y\| < \infty$. Their distance covariance is*

$$\begin{aligned} \mathcal{V}^2(X,Y) &= \|\varphi_{X,Y}(s,t) - \varphi_X(s)\varphi_Y(t)\|_w^2 \\ &= \int_{\mathbb{R}^{p+q}} \frac{|\varphi_{X,Y}(s,t) - \varphi_X(s)\varphi_Y(t)|^2}{\|s\|^{1+p}\|t\|^{1+q}} ds dt \end{aligned}$$

where $w(s,t) = \dfrac{1}{\|s\|^{1+p}\|t\|^{1+q}}$.

It is clear that $\mathcal{V}^2(X,Y) = 0 \iff X \perp\!\!\!\perp Y$.

Proposition 2.1.2

$$\begin{aligned}\mathcal{V}^2(X,Y) &= \mathbf{E}[\|X-X'\|\|Y-Y'\|] + \mathbf{E}[\|X-X'\|]\mathbf{E}[\|Y-Y'\|] - 2\mathbf{E}[\|X-X'\|\|Y-Y''\|] \\ &= \mathrm{Cov}(\|X-X'\|, \|Y-Y'\|) - 2\mathrm{Cov}(\|X-X'\|, \|Y-Y''\|)\end{aligned}$$

Proof.

Definition 2.1.3 *(Distance variance).*

$$\mathcal{V}^2(X) = \mathcal{V}^2(X,X)$$

Definition 2.1.4 *(Distance correlation).*

$$\mathcal{R}^2(X,Y) = \frac{\mathcal{V}^2(X,Y)}{\mathcal{V}(X)\mathcal{V}(Y)}$$

For iid sample realizations $\{(X_i, Y_i)\}_1^n$, let $\widehat{\varphi_X}(t) = \frac{1}{n}\sum_{i=1}^n e^{it^TX_i}$ be the empirical characteristic function for X and likewise for Y. An estimate of $\mathcal{V}^2(X,Y)$ replaces the unknown characteristic functions with the empirical characteristic functions.

Proposition 2.1.5

$$\widehat{\mathcal{V}}^2(X,Y) \equiv \|\widehat{\varphi_{X,Y}}(s,t) - \widehat{\varphi_X}(s)\widehat{\varphi_Y}(t)\|_w^2 = S_1 + S_2 - 2S_3$$

where $w(s,t)$ as above and

$$\begin{aligned}S_1 &= \frac{1}{n^2}\sum_{k=1}^n\sum_{l=1}^n \|X_k - X_l\|\|Y_k - Y_l\| \\ S_2 &= \left(\frac{1}{n^2}\sum_{k=1}^n\sum_{l=1}^n \|X_k - X_l\|\right)\frac{1}{n^2}\sum_{k=1}^n\sum_{l=1}^n \|Y_k - Y_l\| \\ S_3 &= \frac{1}{n^3}\sum_{k=1}^n\sum_{l=1}^n\sum_{m=1}^n \|X_k - X_l\|\|Y_k - Y_m\|\end{aligned}$$

Alternatively, we can let $A, B \in \mathbb{R}^{n\times n}$ *such that* $A_{kl} = \|X_k - X_l\|$ *and* $B_{kl} = \|Y_k - Y_l\|$ *(*A *and* B *are symmetric elementwise nonnegative). Let* $\overline{X} = \frac{1}{n^2}\sum_{k,l=1}^{n} X_{kl}$*. Then*

$$\hat{\mathcal{V}}^2(X,Y) = \overline{A \circ B} + \overline{A} \cdot \overline{B} - \frac{2}{n}\overline{(AB)}$$

where $\circ$ *means element-wise multiplication.*

Estimates of distance variance and distance correlation are defined analogously.

Proposition 2.1.6

$$\mathcal{V}(v_1 + a_1 Q_1 X, v_2 + a_2 Q_2 Y) = \sqrt{|a_1 a_2|}\mathcal{V}(X,Y)$$

Definition 2.1.7 *(*α*-distance covariance). For* $0 < \alpha < 2$

$$\mathcal{V}_\alpha^2(X,Y) = \frac{1}{C(p,\alpha)C(q,\alpha)}\int_{\mathbb{R}^{p+q}} \frac{|\varphi_{X,Y}(s,t) - \varphi_X(s)\varphi_Y(t)|^2}{\|s\|^{\alpha+p}\|t\|^{\alpha+q}} dsdt$$

Proposition 2.1.8 *If* $\mathbf{E}[\|X\|^\alpha] + \mathbf{E}[\|Y\|^\alpha] < \infty$ *then*

$$\mathcal{V}_\alpha^2(X,Y) = \mathbf{E}[\|X-X'\|^\alpha\|Y-Y'\|^\alpha] + \mathbf{E}\|X-X'\|^\alpha\mathbf{E}\|Y-Y'\|^\alpha - 2\mathbf{E}[\|X-X'\|^\alpha\|Y-Y''\|^\alpha]$$

Corollary 2.1.9 *For* $\alpha = 2$*,* $p = q = 1$*, the distance correlation is the absolute value of Pearson's correlation coefficient.*

Chapter 3

Criteria for Evaluating Parcellations

In chapter one we discussed our graphical approach to the brain parcellation problem. We construct a weighted undirected graph where each vertex is a voxel. The graph reflects the spatial position of the voxels; it connects each vertex to the vertices representing the voxel's six cubically adjacent neighbors. The weights on these edges are sample energy distance correlation statistics between the adjacent voxels in the time series and they measure statistical dependence between the voxels. Let $G(V, E)$ denote this graph, its vertices, and its edges.

In this context, a valid k-fold partition $\mathcal{P}_k$ of the graph G is a collection of vertex subsets $(V_1, ..., V_k)$ satisfying the following:

1. $V_i \neq \varnothing$ for all $V_i \in \mathcal{P}_k$
2. $\bigcup_{i=1}^{k} V_i = V$
3. $V_i \cap V_j = \varnothing$ for all $V_i, V_j \in \mathcal{P}_k$
4. V_i is connected (i.e. for every two vertices in V_i, there is a path between them) for all $V_i \in \mathcal{P}_k$

In this chapter we will suggest various criteria for measuring the goodness-of-fit of partitions and discuss their statistical and computational advantages and drawbacks.

3.1 Within-Parcel Similarity

Voxels in the same parcel are ideally highly dependent on one another in the time series of fMRI data. As discussed in the previous chapter, distance correlation is a good measure of dependence. The distance correlation between two random vectors equals zero if and only if the two random vectors are independent, which is not true of correlation statistics such as Pearson's.

Let $\mathcal{R}(x, y)$ denote the distance correlation between two voxels x and y. Let V and W be any parcels. We will use E_V to denote the set of edges with one end in V and one end not in V, and $E_{V,W}$ the set of edges with one end in V and one in W.

Definition 3.1.1 (Within-Score)

$$\frac{1}{k} \sum_{V \in \mathcal{P}_k} \frac{1}{|V|^2} \sum_{x,y \in V} \mathcal{R}(x, y)$$

The Within-Score is non-spatial; it considers all pairs of voxels equally regardless of whether they are adjacent. As a result, it is a good measure of how much the voxels within each parcel are dependent on each other as a set. The downside of this criterion is that it is very expensive to compute. With over 300,000 voxels in an fMRI data set we would potentially have to compute tens of billions of distance correlation statistics, each of which takes time proportional to the number of samples squared.

An alternative and far less expensive criterion that measures within- parcel similarity works by counting distance correlations between adjacent pairs of voxels.

Definition 3.1.2 (Adjacent-Score)

$$\frac{1}{k}\sum_{V\in\mathcal{P}_k}\frac{1}{|E_{V,V}|}\sum_{(x,y)\in E_{V,V}}\mathcal{R}(x,y)$$

Rather than treat parcels as sets with no spatial information, the Adjacent-Score does the opposite by only considering the pairwise dependency of adjacent voxels. For sparse graphs such as ours, the number of distance correlation computations is proportional to the number of vertices.

An intermediate possibility we did not explore is to consider all pairs of voxels up to some maximum spatial distance from each other and perform a weighted averaging of sample pairwise distance correlations, with weights that depend on spatial distance.

3.2 Between-Parcel Dissimilarity

To evaluate parcellation quality, it is also useful to measure how dependent voxels belonging to different parcels are on each other. To this end we define two criterion similar to the Within-Parcel criterion; a non-spatial metric called the Between-Score and its spatial metric the Boundary-Score.

Definition 3.2.1 (Between-Score)

$$\frac{1}{\binom{k}{2}}\sum_{V,W\in\mathcal{P}_k,V\neq W}\frac{1}{|V||W|}\sum_{x\in V,y\in W}\mathcal{R}(x,y)$$

Definition 3.2.2 (Boundary-Score)

$$\frac{1}{\binom{k}{2}}\sum_{V,W\in\mathcal{P}_k,V\neq W}\frac{1}{|V||W|}\sum_{(x,y)\in E_{V,W}}\mathcal{R}(x,y)$$

Generally both of these quantities are more expensive to compute than their Within-Parcel counterparts. Boundary-Score is easy enough to compute for validation purposes, but does not convey much additional information beyond what the Adjacency-Score does, in the sense that the edges used in the computation of Adjacency-Score are the complement of the edges used in the Boundary-Score.

The ability of distance correlation to generalize to pairs of random vectors of arbitrary dimension gives us another way of computing the dependency between two parcels. The Multivariate Between-Score defined below treats parcels as random vectors and computes the distance correlation at the parcel level rather than voxel level. The result is a measure of non-spatial between-parcel similarity that is also computationally feasible. For this reason we will use Multivariate Between-Score as our primary measure of parcel dissimilarity.

Definition 3.2.3 (Multivariate Between-Score)

$$\frac{1}{\binom{k}{2}} \sum_{V,W \in \mathcal{P}_k, V \neq W} \mathcal{R}(V, W)$$

3.3 Graph Cuts

Closely related to the Boundary-Score is the notion of a graph cut from computer science. A *cut set* is the set of edges with endpoints in different parcels. The *cut weight* is the sum of weights of all edges in the cut set and can be expressed as

$$\frac{1}{2} \sum_{V \in \mathcal{P}_k} \sum_{x,y \in E_V} \mathcal{R}(x, y)$$

The *ratio cut* defined below is a weighted version of the cut weight:

$$\frac{1}{2} \sum_{V \in \mathcal{P}_k} \frac{1}{|V|} \sum_{x,y \in E_V} \mathcal{R}(x, y)$$

The subfield of graph partitioning is concerned with minimizing cut weight, ratio cut, and several other related quantities. Over the last several decades a number of highly effective approximation algorithms have been developed to find partitions of graphs that minimize these quantities. Later chapters will explore how these methods work for brain parcellation.

3.4 Balance and Jaggedness

In addition to the above distance correlation based criteria, there are two additional metrics concerned with parcel shape.

Definition 3.4.1 (Balance)

$$\frac{1}{k}\frac{1}{\max_{V\in\mathcal{P}_k}|V|}\sum_{V\in\mathcal{P}_k}|V|$$

The Balance-Score ranges from 1 (all equally sized parcels) to 0 (two parcels with one of size zero).

Definition 3.4.2 (Jaggedness)

$$\frac{1}{k}\sum_{V\in\mathcal{P}_k}\frac{|E_V|^{\frac{3}{2}}}{|V|}$$

The $\frac{3}{2}$ power makes the ratio invariant on parcel size. For instance, a $n \times n \times n$ cube of vertices would have a jaggedness of $6^{\frac{3}{2}}$ which does not depend on n.

3.5 Comparing Multiple Parcellations

Chapter 4

Local Search and Graph Growing Heuristics

We introduce several algorithms for generating brain parcellations. The algorithms in this chapter are all local search heuristics; they begin with n unconnected vertices and iteratively join adjacent ones into components until some stopping criterion is met.

For each algorithm, the resulting parcellation is presented, discussed, and evaluated according to the criteria introduced in the previous chapter.

4.1 The Add-Edge Algorithm

The first and simplest algorithm starts with an empty graph of n vertices and sequentially adds edges between adjacent voxels in order of highest sample distance correlation, until the graph has some prespecified number of connected components k. We will refer to this algorithm as "Unconstrained Add-Edge".

The Unconstrained Add-Edge algorithm produces severely imbalanced parcellations. In the 100-component graph, there was one component containing over 99.9% of all the vertices in the graph.

Our attempt to address this problem was to impose a filter on each edge considered, adding the edge only if at least one of the two following conditions are met:

1. At least one of the two components bridge by the edge is of size less than some prespecified parameter $s_{\min}$.
2. The union of the two components is of size $\leq s_{\max}$.

The restriction on adding new edges was not successful in creating balanced partitions.

4.1.1 Implementation of Unconstrained Version

A naive implementation of Unconstrained Add-Edge would re-compute the number of connected components in the graph (using linear-time bread-first or depth-first search) after each addition of an edge, resulting in a costly $O(EN)$ time complexity. A more efficient implementation takes advantage of the fact that each addition of an edge decreases the number of components in the graph by at most 1. Hence the algorithm needs only to compute the number of connected components after adding $c - k$ edges, where c is the current number of connected components of the graph, beginning at n.

Another implementation uses a binary search-type strategy and is $O((n+E) \log E)$. The idea is to "search" for the last edge to add to the graph by maintaining a range of possible last edges. In each iteration, the algorithm would add to the graph edges 1 to the midpoint of this range, compute the number of connected components, and adjust the range based on whether the number of components is higher or lower than the target K.

4.1.2 Implementation of Size-Constrained Version using Union-Find

The naive implementation must use BFS/DFS in each iteration to compute the sizes of the two components to be connected by an edge, and hence must have time complexity $O(EN)$. Fortunately, there is a way to sublinearly update information on the components of the graph, using the union-find data structure.

The core Union-Find data structure begins with an empty graph of N vertices and supports two operations. union(i, j) adds an edge between vertices i and j. root(i) returns an identifier for the component to which vertex i belongs. All vertices in the same component have the same root. We modified Union-Find to support an additional operation. component_size(i) returns the number of vertices belonging to the component containing i.

Union-Find represents each component as a rooted tree, with vertices in the graph mapping to nodes in the tree. Information about the tree is stored in two arrays of length N, parent and size, which are subject to the following invariants.

1. For each node i, parent[i] = node i's parent on the tree, unless i is a root node. If i is a root node, then parent[i] = i.

2. Nodes i and j are in the same component if and only if they are in the same tree, if and only if they share the same root node.

3. If i is a root node, then size[i] = the size of the component, or the number of nodes in the tree. If i is not a root node, then size[i] can be anything.

A baseline implementation of the three functions is

In addition to the baseline code above, there are two important optimizations:

1. Weighted union maintains information of the sizes of each component so that the root of the smaller component always becomes a child of the larger component's

Algorithm 1 Union-Find

```
function ROOT(i)
    while parent[i] $\neq$ i do
        i $\leftarrow$ parent[i]
    end while
    return i
end function

function UNION(i, j)
    parent[root(j)] $\leftarrow$ root(i)
end function

function COMPONENT_SIZE(i)
    return size[root(i)]
end function
```

root.

2. Path compression flattens the tree with each call to root. Specifically, when root is called on node i, each node traversed from i to the root has its parent set to be the root.

With these two optimizations, the time complexity of root, union, and component_size has been shown to be at least as good as $O(\log^* N)$ where $\log^*$ is the iterated logarithm, defined as the number of times the natural log must be applied to N so that it becomes less than or equal to 1.

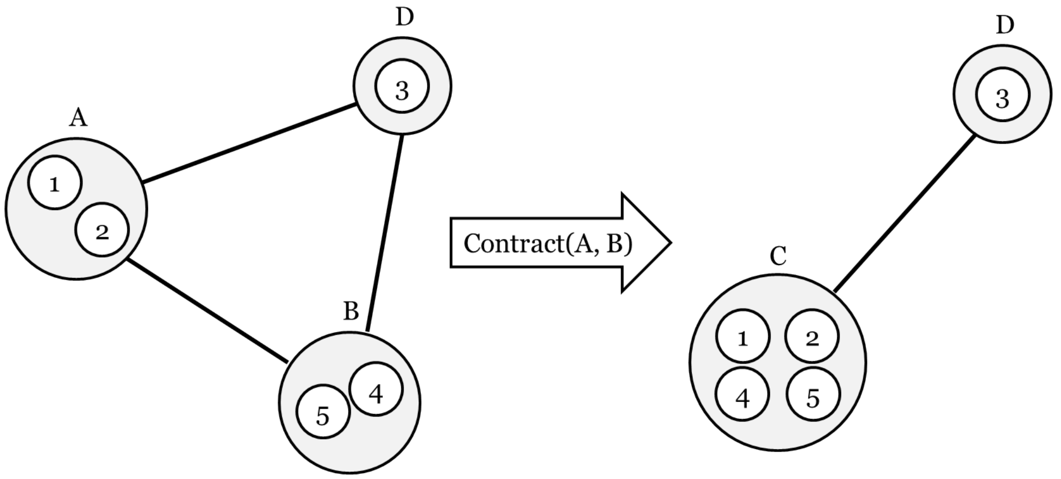
4.2 The Edge-Contraction Algorithm and Contractible Graphs

We propose a new data structure called the *Contractible Graph* (CG) for brain parcellation. The rationale behind the CG is a heuristic procedure for partitioning a graph into somewhat balanced components so as to maximize the Adjacent-Score (3.1.2).

The CG is a mapping of the vertices of the original graph to the vertices of a new graph. The vertices of the CG are called *components* and between any two components there exists exactly one weighted edge, henceforth called a *link*. The weight of a link $w_{A,B}$ between two components A and B in the CG equals the average weight of all edges in the original graph between vertices mapped to A and vertices mapped to B. If no such edges exist, the weight of the link is 0. Formally,

$$E_{A,B} = \{(i,j) \in E : i \in A, j \in B\}$$

$$w_{A,B} = \begin{cases} \frac{1}{|E_{A,B}|} \sum_{(i,j) \in E_{A,B}} w_{ij} & \text{if } |E_{A,B}| > 0 \\ 0 & \text{otherwise} \end{cases}$$



We say an edge (i,j) is *between* components A and B if i is in one of A or B and j is in the other. The *size* of a component is the number of vertices it contains. A *contraction* of a link (A,B) in a CG replaces components A and B with a new component (call it C) containing all vertices mapped to A or B, as illustrated in the figure above. Component C has one link to every other component in the CG, whose weights are the mean of the weights of the corresponding vertex edges, or 0 if no edge exists. Thus the contraction operation maintains the link-invariant property of CG. This leads to the Edge-Contraction algorithm, which begins with the original graph with all vertices as singleton components and contracts edges in a certain order until the graph has only k components in all.

Algorithm 2 Edge-Contraction

```
**Input:** Undirected positive-weighted graph $G$ and target component number $k$
Create a CG from $G$ so that every vertex maps to a unique component
**repeat**
    $\mathcal{S} \leftarrow$ smallest component(s) in the CG
    $(A, B) \leftarrow \underset{A \in \mathcal{S}}{\operatorname{argmax}}\, w(A, B)$
    Contract $(A, B)$
**until** CG has $k$ components
**Output:** Components of CG
```

Why does Edge-Contraction work better than the previous algorithms? The Edge-Contraction algorithm attempts to address two problems of the Size-Constrained Add-Edge algorithm: unbalanced parcels and poor Adjacent-Score relative to randomized graph. It solves the former problem by prioritizing edges that are connected to small components.

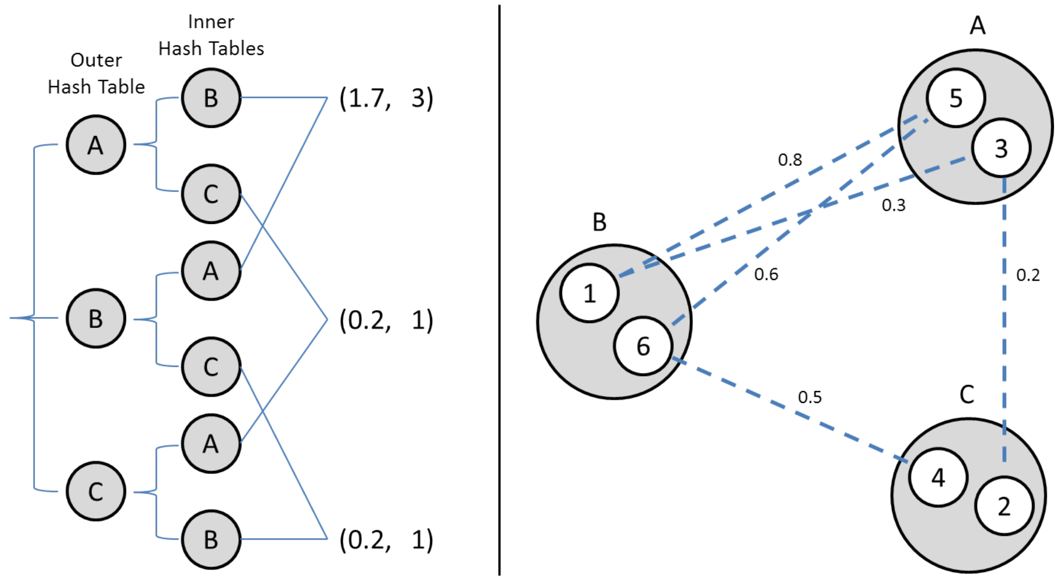
The low Adjacent-Score is likely due to the following scenario: when a vertex is added to a component, it might have multiple edges to that component. One edge might have a very high weight; this is the one that is officially "added". However, the other edges with far lower weights are implicitly added as well, lowering the average edge weights within the component.

The Edge-Contraction algorithm handles this issue by maintaining that there can be at most one edge between any two components A and B, and further that the weight on such an edge is the mean of the weights on all edges that connect a vertex in A with a vertex in B.

4.2.1 Implementation using Nested Hash Tables and Priority Queue

In a Contractible Graph, the weight of the link between two components depends on the summed weight of all edges between them, and the number of such edges.

Our implementation of the CG uses *nested hash tables*, diagrammed below. The outer hash table maps each component A to an inner hash table, which maps all components B with a positive link to A to 1) the summed weights of the edges and 2) the number of edges between A and B.



Implementing the contraction of components B and C into a new component D on this nested hash table requires the following steps. The time complexity is stated assuming no hash collisions.

1. Compute $\mathcal{X}$, the set of all components that either B or C is linked to. $O(|E_B| + |E_C|)$

2. Create a new element in the outer hash table, D, and associate it with an empty inner hash table. $O(1)$

3. For each component $X \in \mathcal{X}$,

    - Retrieve $W(X, B) + W(X, C)$, the summed weights all edges between X and B and between X and C, and $|E_{X,B}| + |E_{X,C}|$, the number of such edges. These quantities are stored explicitly as a values in the inner hash table, so this operations is $O(1)$.

- Add a new component name D to the inner hash table of X and map it to $\big(W(X,B)+W(X,C), |E_{X,B}|+|E_{X,C}|\big)$. Delete elements B and C from the inner list of X. $O(1)$
- In the D inner hash table, add component name X and map it to to the same $\big(W(X,B)+W(X,C), |E_{X,B}|+|E_{X,C}|\big)$. $O(1)$

4. Delete B and C from the outer list.

Having described the contraction step, we will next discuss how to efficiently locate the link to be contracted. In computer science, a *Maximum Priority Queue* (MaxPQ) data type is a set of well-ordered objects that supports the following operations:

- *add(obj)*: Adds an object to the set.
- *remove_maximum()*: Removes and returns an object with the largest priority in the set.

Using the heap data structure, the above two operations both run in $O(\log n)$ time.

Each component on the CG will be associated with an element of the priority queue. The priority of component A is defined as

$$\max_X \; w_{A,X} - |A|$$

Since our graph link and edge weights are all between 0 and 1, the highest priority element in the queue always has the smallest size. Therefore, if the priority queue is up-to-date with the CG, the next link to be contracted according to Edge-Contraction has an endpoint component whose priority is the highest in the queue.

However, a complication arises from the fact that a contraction can change the priorities of components neighboring the contracting components, thereby making

Table 4.1: Results of Edge-Contract for Different Component Numbers

Number of Components	Adjacency	Multi-Boundary	Jaggedness	Balance
500	0.7991	0.7578	54.65	0.327
400	0.7990	0.7723	59.20	0.343
300	0.7974	0.7921	65.20	0.297
250	0.7955	0.7991	69.51	0.356
200	0.7941	0.8101	75.47	0.357
150	0.7931	0.8225	83.88	0.418
116	0.7913	0.8389	92.14	0.439
100	0.7911	0.8488	97.63	0.383
AAL	0.7225	——	29.62	0.332

the priorities stored in the MaxPQ out-of-date. For instance, if components A and B are contracted, and there is a component C with positive links to both A and B, then the $C-A$ and $C-B$ links will be replaced by a $C-(AB)$ link with a different weight. If either $C-A$ or $C-B$ links happened to be the maximum-weighted links of C, then C's priority will be lower, and C ought to be further down the queue.

To address this issue, we could re-compute the priority of every component drawn from the MaxPQ. If the component's actual priority is not the maximum, then it is re-inserted into the queue with updated priority. Additionally, the maximum priority component may no longer exist in the CG due to contraction with another component. In this case it is simply discarded.

Without using an efficient priority queue, the linear searching method of finding the next link to contract results in a $O\big(n(n-k)\big)$ time algorithm. Using the priority queue the time complexity of Edge-Contraction is $O\big((n-k)(m+\log n)\big)$, where m is the average number of positive links a component has.

4.2.2 Results

The Edge-Contract parcellations notably outperformed the anatomical AAL parcellation in the Adjacency-Score. For further comparison, found the mean edge weight in the graph to be 0.7258, which is even slightly higher than the average adjacent within-parcel edge in AAL. This suggests that the AAL parcellation has no connection with the functional information contained in this fMRI data set. It shows on the other hand that the Edge-Contract algorithm can successfully locate regions of functional similarity.

The one apparent deficiency of Edge-Contract is the jaggedness of its parcels. Comparison of our parcellations with the AAL shows that our 116-component parcellation – the same number of components as AAL – has an average parcel surface area roughly $\left(\frac{92.14}{29.62}\right)^{\frac{2}{3}} \approx 2.11$ times that of AAL. Visually, that difference is shown in the plots below of a typical component from each parcellation.

4.3 The Generalized Edge-Contraction Algorithm

In the original Edge-Contraction algorithm, the criteria for selecting the next link to contract was to search through the set of smallest components and find the link of maximal weight. Because this criteria takes no account of the shape of the two components to be contracted, the resulting parcels tend to be very jagged.

To address this we expanded the criterion for finding the next link to contract. Rather than use only the size of the component and the weight of the link, a *Generalized Edge-Contraction* algorithm may use any piece of information stored in the Contractible Graph about a pair of components, such as the number of edges connecting two components. A *priority function* takes information of any two components in a CG and outputs a real number, the priority. For each iteration, the pair of components with the largest priority is contracted and the priorities of neighboring components with respect to the newly conjoined component are computed.

For two components A, B let $|A|$ denote size (number of vertices) of A, $E_{A,B}$ denote the set of edges between A and B, and $w_{A,B}$ the weight of the link connecting A and B. The priority function of the original Edge-Contraction algorithm is $p_0(A, B) =$

Table 4.2: Results of Generalized Edge-Contract for Various Parameter Settings

α	β	Adjacent	Jaggedness	Balance
3	1	0.7500	25.3	0.285
6	1	0.7574	31.0	0.231
10	1	0.7634	36.6	0.211
6	2	0.7565	31.0	0.199
6	4	0.7576	30.9	0.270

(a) 116 Parcels

α	β	Adjacent	Jaggedness	Balance
3	1	0.7592	23.8	0.218
6	1	0.7651	28.0	0.143
10	1	0.7705	32.0	0.221
6	2	0.7670	28.2	0.169
6	4	0.7669	28.8	0.293

(b) 300 Parcels

$w_{A,B} - |B|$.

A link (A, B) will have high priority if either component is small, if the link has a large weight, and if it has a good boundary-ratio, defined as $\frac{|E_{A,B}|}{\min(|A|,|B|)}$, which helps to minimize jaggedness. From these notions we created a family of priority functions indexed by tunable parameters α and β

$$p_1(A, B) = \frac{w_{A,B}^{\alpha}}{|A|^{\beta}} \cdot \frac{|E_{A,B}|}{\min(|A|, |B|)}$$

that modulate the balance of small size, large weight, and high boundary ratio. Table 4.2 shows the results of the 116-component and 300-component Generalized Edge-Contract parcellation when performed for various values of α and β. The adjacent scores have fallen on average from 0.79 in the vanilla edge contraact to 0.75 here. This implies a tradeoff between prioritizing edge weight, jaggedness, and balance in producing a parcellation.

With a balance score of 0.332, the AAL parcellation was better balanced than all of the parcellations obtained via this priority function, though not drastically. In terms of jaggedness, these parcellations all fell within the vicinity of AAL. In this one brain, the best parcellation arises from parameters $\alpha = 6$ and $\beta = 4$.

Parcellations of other brains using generalized edge contract can be found in chapter 8.

Chapter 5

Spectral Methods

In the previous chapter, we showed how local search heuristics produced parcels that were balanced and had high within-parcel and low between-parcel edge weights. The central idea behind such methods was to choose vertices to be in the same component if the edge connecting them has high distance correlation. Vertices were added to components one-by-one with constraints on component size, but not on component shape. As a result, one salient issue with these parcellations was lack of smoothness in the boundaries between parcels There was scant resemblence between the anatomical maps of the brain depicting smooth, rotund lobes and our jagged, web-like parcellations.

One key reason for this phenomenon are the local search heuristics' focus on maximizing *average* within-component edge weights (equivalently, minimizing *average* between-component edge weights). To get smoothness in the boundary between components, we could either impose a penalty for too many between-component edges and work that into the local search heuristics, or try minimizing over the sum of all weights on between-component edges. This chapter deals with the second approach and this family of methods is called spectral partitioning.

Spectral partitioning constitutes the second major class of techniques used to partition graphs. Rather than rely on local component-growing heuristics, spectral partitioning uses information about the entire graph at once.

Throughout this chapter, a valid partitioning $P_k = (V_1, ..., V_k)$ of the graph $G = (V, E)$ is defined in the same way as in chapter 3; i.e., it must satisfy

1. $V_i \neq \varnothing$ for all $V_i \in \mathcal{P}_k$
2. $\bigcup\limits_{i=1}^{k} V_i = V$
3. $V_i \cap V_j = \varnothing$ for all $V_i, V_j \in \mathcal{P}_k$
4. V_i is connected (i.e. for every two vertices in V_i, there is a path between them) for all $V_i \in \mathcal{P}_k$

For all edges $(i, j) \in E$, let w_{ij} denote the weight of the edge connecting vertices i and j. $S^{n\times n}$ is the set of real symmetric $n \times n$ matrices. We further define, for a given graph $G = (V, E)$, the associated

Definition 5.0.1 (Adjacency matrix) *$A \in \mathcal{S}^{n\times n}$ has entries*

$$A_{ij} = \begin{cases} w_{ij} & \textit{if } (i,j) \in E \\ 0 & \textit{otherwise} \end{cases}$$

Definition 5.0.2 (Degree matrix) *$D \in \mathcal{S}^{n\times n}$*

$$D_{ij} = \begin{cases} \sum_{k=1}^{n} A_{ik} & \textit{if } i = j \\ 0 & \textit{otherwise} \end{cases}$$

5.1 Size-Constrained MinCut and Graph Bipartitioning

Consider the case of partitioning a graph into two components, $k = 2$. For all vertices $i \in V$, let $x_i = 1$ if $i \in V_1$ and $x_1 = -1$ if $i \in V_2$. Then the sum of weights on edges between the two components is

$$\begin{aligned} C(P_2) &= \sum_{i \in V_1} \sum_{j \in V_2} A_{ij} \\ &= \sum_{i=2}^{n} \sum_{j=1}^{i-1} \frac{(x_i - x_j)^2}{4} A_{ij} \end{aligned}$$

since

$$(x_i - x_j)^2 = \begin{cases} 4 & \text{if } i, j \text{ are in different components} \\ 0 & \text{otherwise} \end{cases}$$

$C(P_2)$ can also be written in a matrix quadratic form, as

$$
\begin{aligned}
C(P_2) &= \sum_{i=2}^{n}\sum_{j=1}^{i-1} \frac{(x_i - x_j)^2}{4} A_{ij} \\
&= \frac{1}{2}\sum_{i,j=1}^{n} \frac{(x_i - x_j)^2}{4} A_{ij} \\
&= \frac{1}{2}\sum_{i,j=1}^{n} \frac{x_i^2 + x_j^2 - 2x_i x_j}{4} A_{ij} \\
&= \frac{1}{2}\sum_{i,j=1}^{n} \frac{1 - x_i x_j}{2} A_{ij} \\
&= \frac{1}{4}\sum_{i,j=1}^{n} (x_i^2 - x_i x_j) A_{ij} \\
&= \frac{1}{4}\sum_{i=1}^{n} x_i^2 \sum_{j=1}^{n} A_{ij} - \frac{1}{4}\sum_{i,j=1}^{n} x_i A_{ij} x_j \\
&= \frac{1}{4}\sum_{i=1}^{n} x_i^2 D_{ii} - \frac{1}{4} x^T A x \\
&= \frac{1}{4} x^T (D - A) x \\
&= \frac{1}{4} x^T L x
\end{aligned}
$$

where L is called the Laplacian matrix of the graph and defined as $L = D - A$. MinCut can thus be formulated as minimizing $x^T L x$ subject to $x \in \{-1, 1\}^n$.

Algorithms like Karger's can solve MinCut in polynomial time. However, MinCut in this formulation lacks constraints on the size of the partitions, and if applied to our brain parcellation problem, would result in severely inbalanced partitions. If constraints on the sizes of the components were added, the problem becomes NP-hard in the general case [Buluç et al., 2013].

An old but effective approach to bipartitioning uses the eigenvectors of the Laplacian matrix and is called spectral bipartitioning. The approach relaxes the $\{-1, 1\}$ constraint on x (and rescales x) so that it need only satisfy $\|x\| = 1$ ($\|\cdot\|$ here refering to L2 norm). It is easy to see that $\left\{x : x \in \{-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\}^n\right\} \subset \left\{x \in \mathbb{R}^n : \|x\| = 1\right\}$ The problem now becomes

$$\begin{aligned} \min_{x} \quad & x^T L x \\ \text{s.t.} \quad & \|x\| = 1 \end{aligned} \tag{5.1}$$

Using Lagrangian multipliers, it can be shown that all optimal solutions to the above must satisfy $Lx = \lambda x$ and this problem reduces to finding the smallest eigenvalues of L and their associated eigenvectors. In addition, 5.1.1 below implies that all eigenvalues are nonnegative.

Theorem 5.1.1 *Let L be a Laplacian matrix. Then $L \succeq 0$ (L is positive semidefinite)*

Proof. *Let $x \in \mathbb{R}^n$. $x^T L x = x^T D x -$*

Note that from the $C(P_2) = \sum_{i>j} \frac{(x_i - x_j)^2}{4} A_{ij} = \frac{1}{4} x^T L x$ equivalence we know that 0 and $(\frac{1}{\sqrt{n}}, ..., \frac{1}{\sqrt{n}})^T$ is a minimum eigenvalue and eigenvector to this system. For bipartitioning, the useful eigenvector is the one that corresponds to the 2nd smallest eigenvalue, which is nonzero if the graph as a whole is connected. We'll denote this eigenvalue as λ_1 and corresponding unit eigenvector as x_1. We have the following:

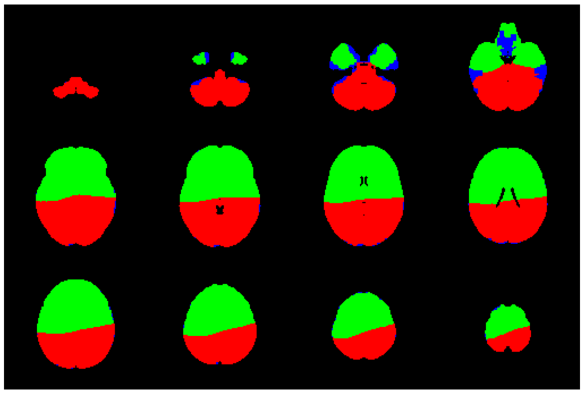
Theorem 5.1.2 *Let P_2 be any valid partition into 2 components. Then $C(P_2) \geq \lambda_1$*

Proof.

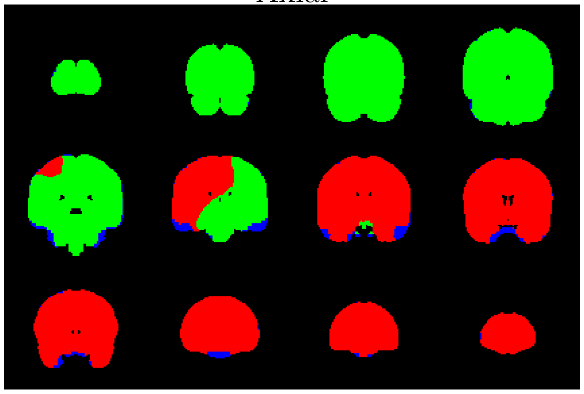
In the literature, x_1 is often refered to as the Fiedler vector, after the first mathematician who studied it in detail. From the Fiedler vector we can obtain a variety of "good" bipartitions. We can impose a size constraint $|V_1| = s$ and obtain a bipartition satisfying this by placing the vertices associated with the s largest entries of x_1 in V_1. This encompasses bipartitions of equal component size. We can also sort the entries of x_1 and find the largest difference between consecutive sorted entries. Vertices corresponding to entries sorted to the left of this split can be placed in V_1

and vertices sorted to the right in V_2. This method tends to approximate the MinCut solution.

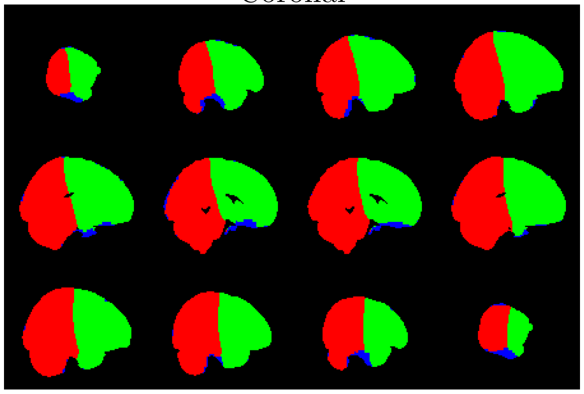
The result of spectral bipartitioning on a resting state fMRI scan is shown below. As anticipated, the boundaries of between the components are smooth.



Axial



Coronal



Sagittal

One can recursively apply this bipartitioning method to the component subgraphs to obtain k-partitions, but there is a more elegant approach involving additional eigenvectors that requires the construction of only one Laplacian matrix, which we shall discuss next.

5.2 Spectral k-partitioning

We'll begin with two definitions to set up the machinery for partitioning into k components.

Definition 5.2.1 (Assignment matrix) $X \in \{0,1\}^{n \times k}$ *has entries*

$$X_{ih} = \begin{cases} 1 & \textit{if vertex } i \in V_h \\ 0 & \textit{otherwise} \end{cases}$$

Let u_m denote a vector of m ones. An assignment matrix characterizes a valid partition only if it satisfies $Xu_k = u_n$ and $X^T u_n > 0$. The columns of X are orthogonal.

Definition 5.2.2 *[Partition matrix]* $P \in \{0,1\}^{n \times n}$ *has entries*

$$P_{ij} = \begin{cases} 1 & \textit{if vertices } i \textit{ and } j \textit{ are in the same component} \\ 0 & \textit{otherwise} \end{cases}$$

If P and X refer to the same partitioning, then $P = XX^T$.

In the k-component case, we define the weight of a partition $C(P_k)$ as the sum of weights of edges between different components (between-edges). This is equivalent to the definition below:

Definition 5.2.3 (Cut weight) *For a partition* $P_k = (V_1, ..., V_k)$*, the cut weight is* *defined as*

$$C(P_k) = \sum_{h=1}^{k} E_h$$

where E_h is the sum of the weights of all edges with one vertex in V_h and one vertex not in it.

This is equal to the sum of the weights of all edges in the graph minus the sum of the weights of all edges connecting vertices in the same component (within-edges).

In addition, if D is the degree matrix, then

$$\begin{aligned}
\mathrm{Tr}(PD) &= \sum_{i,j=1}^{n} P_{ij} D_{ij} \\
&= \sum_{i=1}^{n} P_{ii} D_{ii} \\
&= \sum_{i=1}^{n} P_{ii} \sum_{j=1}^{n} A_{ij} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}
\end{aligned}$$

is the sum of the weights of all edges in the graph. Similarly, $\mathrm{Tr}(PA)$ equals the sum of weights of all within-edges. It follows that

$$C(P_k) = \mathrm{Tr}(X^T L X)$$

The problem of minimizing this quadratic function subject to the constraint that X must be a valid assignment matrix is called (minimum) k-cut and is NP-complete for arbitrary k [Goldschmidt and Hochbaum, 1994].

Even if a polynomial time algorithm existed for minimizing $C(P_k)$, there would be no guarantee that the resulting partitions would be balanced. To address this issue, researchers have developed a similar cost objective called the *ratio-cut cost.*

Definition 5.2.4 (Ratio-cut cost) *For a given partition $P_k = (V_1, ..., V_k)$ the ratio-*

cut cost C_R is defined

$$C_R(P_k) = \sum_{h=1}^{k} \frac{E_h}{|V_h|}$$

where E_h is the sum of the weights of all edges with one vertex in V_h and one vertex not in it.

The ratio-cut cost places an implicit penalty on small components and therefore encourages balanced component sizes. Associated with ratio-cut objective there is a new decision variable called the *ratioed assignment matrix.*

Definition 5.2.5 (Ratioed Assignment Matrix) *$R \in \mathbb{R}^{n\times k}$ has entries*

$$R_{ih} = \begin{cases} \frac{1}{\sqrt{|V_h|}} & \textit{if vertex } i \in V_h \\ 0 & \textit{otherwise} \end{cases}$$

We note that R has the same form as the assignment matrix X except with columns rescaled so that the column sum is $\sqrt{|V_h|}$ for each component V_h. Similarly there is also a *ratioed partition matrix* equal to RR^T, with entries $[RR^T]_{ij} = \frac{1}{|V_h|}$ if $i, j \in V_h$ and 0 otherwise.

The ratioed assignment matrix relates to the ratio-cut cost in the same way the assignment matrix relates to cut weight; namely, if R characterizes a partition P_k then

$$C_R(P_k) = \mathrm{Tr}(R^T L R)$$

R has the additional useful property that $R^T R = I$.

In fact, for any matrix R satisfying

1. $R^T R = I$
2. $R \geq 0$ (element-wise)
3. $RR^T u_n = u_n$ where u_n is a n-dimensional vector of all ones.

there is a valid partition whose ratioed assignment matrix equals R. The third constraint ensures that all non-zero entries of R equal $\frac{1}{\sqrt{|V_h|}}$.

Minimizing $C_R(P_k)$ over the set of valid R matrices is an NP-hard combinatorial optimization problem (?). The spectral relaxation first proposed in [Chan et al., 1994] drops the second and third constraints on R and the resulting problem (shown below) has a closed-form optimal solution.

$$\begin{aligned} \min_{R\in\mathbb{R}^{n\times k}} \quad & \mathrm{Tr}(R^T L R) \\ \text{s.t.} \quad & R^T R = I \end{aligned} \tag{5.2}$$

[Fan, 1950] proved that an optimal solution $\hat{R}$ to the above has columns equaling k orthonormal eigenvectors corresponding to the k smallest eigenvalues of L: λ_1, λ_2, ..., λ_k. Furthermore, the optimal objective value, $\mathrm{Tr}(\hat{R}^T L \hat{R}) = \sum_{h=1}^{k} \lambda_h$. Analogously to $P = XX^T$, $\hat{R}$ can be thought of as n k-dimensional points where the dot products of the ith and jth rows measure the affinity of vertices i and j to be in the same component.

As [Chan et al., 1994] pointed out, recovery of the discrete assignment matrix X from the continuous assignment matrix $\hat{R}$ would be more accurate if $\hat{R}$ were un-ratioed (i.e. if each row of R had the same length), since the ratioed assignment matrix R has the problematic property that for any i, j in the same component, the dot product $R_i^T R_j$ depends on the size of that component. The un-ratioed version of $\hat{R}$ be recovered by dividing each row by its Euclidean norm. The result, $\hat{X}$, can be thought of as n points in $\mathbb{R}^k$ embedded on the surface of the unit hypersphere.

Obtaining the partition assignment matrix X, from this spherical embedding is a clustering problem. We used k-means with cosine similarity $s(x, y) = x^T y$ for this purpose. For each cluster h and its associated points matrix $H \in \mathbb{R}^{m\times k}$, the location of the cluster centroid c_h in the next iteration satisfies $\sum_{i=1}^{m} H_i = \lambda c_h$ for some positive scalar λ such that $\|c_h\| = 1$.

To summarize, [Chan et al., 1994] introduced a spectral relaxation of the graph k-partitioning to minimize the ratio cut. From the Laplacian matrix's smallest k eigenvectors we obtain a continuous approximation $\hat{R}$ to the optimal ratioed assignment matrix R. The rows of $\hat{R}$ are standardized to length 1 to get the continuous approximation $\hat{X}$ to optimal (unratioed) assignment matrix X. Following standard practice we used k-means clustering with cosine similarity to recover the component assignments from $\hat{X}$.

Chapter 6

Symmetric Nonnegative Matrix Factorization

In the previous chapter, we showed that the problem of finding the minimum ratio cut of a graph (with Laplacian matrix L, degree matrix D, and adjacency matrix A) can be formulated as minimizing

$$\mathrm{Tr}(R^T L R) \tag{6.1}$$

over the set, $\mathcal{R}$, of $n \times k$ matrices satisfying

1. $R^T R = I$
2. $R \geq 0$ (element-wise)
3. $RR^T u_n = u_n$ where u_n is a n-dimensional vector of all ones.

If the sizes of the components in the optimal ratio cut partition are perfectly balanced, which is equivalent to saying if the diagonal of the optimal ratioed assignment matrix RR^T has entries all equal to $\frac{k}{n}$, then

$$\begin{aligned}\mathrm{Tr}(R^T DR) &= \sum_{i=1}^{n} [RR^T]_{ii} D_{ii} \\ &= \sum_{i=1}^{n} \frac{k}{n} D_{ii} \\ &= \frac{k}{n} \sum_{i,j} A_{ij}\end{aligned}$$

is a constant that does not depend on R. The same is true if each vertex has the same degree $D_{ii} = d$, in which case

$$\begin{aligned}\mathrm{Tr}(R^T DR) &= \sum_{i=1}^{n} [RR^T]_{ii} D_{ii} \\ &= d \sum_{i=1}^{n} [RR^T]_{ii} \\ &= dk\end{aligned}$$

is also a constant that does not depend on R. In either case,

$$\underset{R \in \mathcal{R}}{\mathrm{argmin}}\ \mathrm{Tr}(R^T LR) = \underset{R \in \mathcal{R}}{\mathrm{argmax}}\ \mathrm{Tr}(R^T AR)$$

This equality may also hold even if neither condition is true, especially if they are approximately true.

Spectral k-partitioning drops the second and third constraints of $\mathcal{R}$ to derive a closed-form minimizer of 6.1, from which the original assignment matrix can be obtained by k-means. This chapter deals with an alternative relaxation of $\mathcal{R}$ that drops the first and third constraints.

6.1 Symmetric Nonnegative Matrix Factorization

For an $n \times m$ matrix A, a nonnegative matrix factorization (NMF) is a pair of matrices $W \in \mathbb{R}^{n\times k}$ and $H \in \mathbb{R}^{m\times k}$ that minimizes $\|A - WH^T\|_F^2$ subject to elementwise nonnegativity: $H \geq 0$ and $W \geq 0$. Here, $\|X\|_F = \sqrt{\sum_{ij} X_{ij}}$ refers to the Frobenius norm.

For $n \times n$ symmetric matrices A, a *symmetric* NMF (SymNMF) is a matrix $H \in \mathbb{R}^{n\times k}$ that minimizes $\|A - HH^T\|_F^2$, and k is an arbitrary positive integer typically much smaller than n.

The following theorem from [Ding et al., 2005] illustrates the connection between SymNMF and graph partitioning.

Theorem 6.1.1 *Let A be a $n \times n$ symmetric matrix. Then*

$$\operatorname*{argmax}_{H^TH=I, H\geq 0} \operatorname{Tr}(H^TAH) = \operatorname*{argmin}_{H^TH=I, H\geq 0} \|A - HH^T\|_F^2$$

Proof.

$$\begin{aligned}
\operatorname*{argmax}_{H^TH=I, H\geq 0} \operatorname{Tr}(H^TAH) &= \operatorname*{argmin}_{H^TH=I, H\geq 0} -2\operatorname{Tr}(H^TAH) \\
&= \operatorname*{argmin}_{H^TH=I, H\geq 0} \operatorname{Tr}(AA^T) - 2\operatorname{Tr}(H^TAH) + \|H^TH\|_F^2 \\
&= \operatorname*{argmin}_{H^TH=I, H\geq 0} \|A - HH^T\|_F^2
\end{aligned}$$

If A is the adjacency matrix, then under the equal vertex degrees condition described earlier $\operatorname*{argmax}_{H^TH=I, H\geq 0} \operatorname{Tr}(H^TAH) = \operatorname*{argmin}_{H^TH=I, H\geq 0} \operatorname{Tr}(H^TLH)$. Hence an alternative approach to the minimum ratio-cut problem is to drop the $H^TH = I$ constraint and solve the SymNMF problem:

$$\begin{aligned} \min_{H \in \mathbb{R}^{n \times k}} \quad & \|A - HH^T\|_F^2 \\ \text{s.t.} \quad & H \geq 0 \end{aligned} \tag{6.2}$$

This relaxation has two key differences from the spectral relaxation .

- There is no closed-form solution, and the optimal value is found via an optimization algorithm, described in the next section.
- The optimal assignments are recovered directly from the largest entry in each row. There is no need for k-means.

6.1.1 The Alternating Nonnegative Least Squares Algorithm

The algorithm for solving (6.2) developed by [Kuang et al., 2015] uses the same framework as existing algorithms for solving the asymmetric NMF problem: $\min_{W,H \geq 0} \|A - WH^T\|_F^2$. That framework, called *alternating nonnegative least squares* is an iterative scheme that fixes one of the matrix factors and solves for the other. Then, it fixes the factor just solved and solves for the first. In other words, these two steps are repeated until convergence:

1. $W \leftarrow \underset{W \geq 0}{\operatorname{argmin}} \|A - WH^T\|_F^2$
2. $H \leftarrow \underset{H \geq 0}{\operatorname{argmin}} \|A - WH^T\|_F^2$

One reason this method is effective in the asymmetric case is that the two subproblems are convex, so it is easy to attain the global minimum in each subproblem. Further, a number of specialized algorithms have been developed to solve these nonnegative least squares problems quickly. One in particular by [Kim and Park, 2011] works well with large, sparse A matrices and is detailed in the next subsection.

An approach for SymNMF in [Kuang et al., 2015] uses this framework by artificially creating two different factors rather than one and adding on an adjustable penalty term for the difference between the two factors:

$$\min_{W,H\geq 0} \|A - WH^T\|_F^2 + \alpha\|W - H\|_F^2 \tag{6.3}$$

where $W, H \in \mathbb{R}^{n\times k}$. As in the asymmetric case, the algorithm computes W and H iteratively by fixing one of the two factors. The difference is the introduction of the penalty term, α, which can be increased after each iteration to force the eventual convergence of W and H.

(6.3) can also be re-written in the same form as the asymmetric NMF:

$$\left\| \begin{bmatrix} W \\ \sqrt{\alpha} I_k \end{bmatrix} H^T - \begin{bmatrix} A \\ \sqrt{\alpha} W^T \end{bmatrix} \right\|_F^2 \tag{6.4}$$

with $\begin{bmatrix} W \\ \sqrt{\alpha} I_k \end{bmatrix}$ taking on the part of the fixed matrix and H the decision matrix. The ANLS algorithm for SymNMF is the following:

Algorithm 3 ANLS algorithm for SymNMF

```
1: Initialize $H$
2: **repeat**
3:     $W \leftarrow H$
4:     $H \leftarrow \operatorname{argmin}_{H\geq 0} \left\| \begin{bmatrix} W \\ \sqrt{\alpha} I_k \end{bmatrix} H^T - \begin{bmatrix} A \\ \sqrt{\alpha} W^T \end{bmatrix} \right\|_F^2$
5:     increase $\alpha$
6: **until** convergence
```

[Kuang et al., 2015] recommends multiplying *alpha* by 1.01 each iteration. The next section describes the method in [Kim and Park, 2011] for solving the embedded nonnegative least squares problem.

6.1.2 Block Pivoting Algorithm for Nonnegative Least Squares

Let us consider a simplified form of the NLS step in Algorithm 3. Let x denote a single k-dimensional row of H, and our goal is to solve

$$\min_{x \geq 0} \|Cx - b\|_2^2 \tag{6.5}$$

where $C = \begin{bmatrix} W \\ \sqrt{\alpha} I_k \end{bmatrix}$ and b is the column of $\begin{bmatrix} A \\ \sqrt{\alpha} W^T \end{bmatrix}$ with the same index as the index of row x in H. Since the rows of H share no constraints with one another, the general NLS problem in Algorithm 3 can be viewed simply as n independent cases of 6.5.

The Karush-Kuhn-Tucker necessary conditions for optimality are:

1. $y = C^T C x - C^T b$
2. $y \geq 0$
3. $x \geq 0$
4. $x_i y_i = 0$ for $i = 1, ..., k$

Since I_k is full column rank, so is C, making $C^T C$ positive definite and 6.5 strictly convex.

Let x^* be the optimal solution to 6.5. Suppose the indices of the strictly positive entries of x^* were known and denote them with F. Then all the entries of x^* and y^* can be computed. Let x_F denote the vector of entries of x indexed by F and C_F is the matrix of *columns* of C indexed by F. Let G be all the elements of $\{1, ..., k\}$ not in F. Then,

$$x_F = \underset{x_F \geq 0}{\operatorname{argmin}} \|C_F x_F - b\|_2^2 \tag{6.6a}$$

$$x_G = 0 \tag{6.6b}$$

$$y_F = 0 \tag{6.6c}$$

$$y_G = C_G^T(C_F x_F - b) \tag{6.6d}$$

(6.6a) is implied by the fact that the strictly positive entries of x must solve the unconstrained problem. (6.6b) is true by definition of F and G. (6.6c) is implied by KKT condition 4. (6.6d) is implied by KKT condition 1. If index sets F and G give the optimal solution, then x_F and y_G must satisfy KKT conditions 2 and 3 (nonnegativity). Therefore the *Block Pivoting Algorithm* finds the optimal x and y by searching for the correct F and G index sets.

Suppose the x and y computed from the current F and G do not satisfy the KKT conditions, or equivalently the set

$$V = \{j \in F : x_j < 0\} \cup \{j \in G : y_j < 0\} \tag{6.7}$$

is not empty. The algorithm updates F and G by taking a subset $\hat{V} \subseteq V$ and setting

$$F \leftarrow (F - \hat{V}) \cup (\hat{V} \cap G) \tag{6.8a}$$

$$G \leftarrow (G - \hat{V}) \cup (\hat{V} \cap F) \tag{6.8b}$$

By default, the algorithm uses $\hat{V} = V$ for speed. However, there are cases where this could result in an infinite loop, so the single pivot rule $\hat{V} = \max\{j \in V\}$ is invoked instead, despite being slower.

In the multicolumn case

$$\min_{X \geq 0} \|CX - B\|_2^2 \tag{6.9}$$

X is $k \times n$ and the algorithm needs to track, for each column of X, the F and G indices. Since in many cases including ours, $n \gg k$, so many columns of X are likely to have the same F and G. Given this fact, the block pivoting algorithm has a shortcut to solving (6.6a) and (6.6d) for each column. (6.6a) is found by solving $C_F^T C_F x_F = C_F^T b$ and the Cholesky factorization for this can be done just once for all columns with the same F indices. Additionally, the $C_F^T C_F$, $C_F^T b$, $C_G^T C_F$, and $C_G^T b$ matrices and vectors required to compute x_F and y_G can be extracted as submatrices of $C^T C$ and $C^T B$, computed once in the beginning of the algorithm. In our SymNMF case, these two matrices are much smaller than the original sparse A matrix that formed them.

$$C^T C = W^T W + \alpha I_k \tag{6.10}$$

$$C^T B = W^T A + \alpha W^T \tag{6.11}$$

Algorithm 4 Block Pivoting Algorithm for NLS

```
**Input:** $A \in \mathbb{R}^{n \times n}$, $W \in \mathbb{R}^{n \times k}$
**Output:** $X$
Compute $C^T C$ and $C^T B$ by (6.10) and (6.11)
Initialize $F_i = \varnothing$ and $G_i = \{1, ..., k\}$ for all $i = 1, ..., n$
Initialize $\alpha(\in \mathbb{R}^n) = 3$ and $\beta(\in \mathbb{R}^n) = k + 1$
repeat
    Update infeasible columns of $X$ and $Y$ by column-grouping.
    $I \leftarrow \{i : (X_{F_j}, Y_{G_j})$ is infeasible$\}$
    for all $i \in I$ do
        Compute $V_i$ by 6.7
        if $|V_i| < \beta_i$ then
            $\beta_i \leftarrow |V_i|$
            $\alpha_i \leftarrow 3$
            $\hat{V}_i \leftarrow V_i$
        else if $\alpha_i \geq 1$ then
            $\alpha_i \leftarrow \alpha_i - 1$
            $\hat{V}_i \leftarrow V_i$
        else if $\alpha_i = 0$ then
            $\hat{V}_i = \max\{j \in V_i\}$
        end if
        Update $F_i$ and $G_i$ by (6.8)
    end for
until $I = \varnothing$
```

6.2 Connected Partitions

An issue with the SymNMF approach is that the partitions may not be connected – there may exist a vertex, none of whose neighbor vertices belong to the same parcel that it does.

Definition 6.2.1 (Unweighted Adjacency Matrix) *For a graph with n vertices and edge set E, the unweighted adjacency matrix $B \in \{0,1\}^{n\times n}$ has entries*

$$B_{ij} = \begin{cases} 1 & \textit{if } (i,j) \in E \\ 0 & \textit{otherwise} \end{cases}$$

Proposition 6.2.2 *Let B be the unweighted adjacency matrix of a graph with n vertices and $X \in \{0,1\}^{n\times K}$ be an assignment matrix satisfying $Xe_K = e_n$ and $X^T e_n \geq e_K$. If the partitions of the graph defined by X are connected then*

$$(B - I)X \geq 0$$

Proof: Let b_i denote the ith row of B and X_k the kth column of X. Since the non-zero entries of b_i indicate which vertices i is neighboring and the non-zero entries of X_k indicate which vertices are in the kth partition, the dot product $b_i X_k$ equals the number of vertices neighboring i that are in the kth partition.

If the partitions of the graph are connected, then this number is at least 1 if k is the partition containing i. If k does not contain i, then $b_i X_k$ can be 0. This can be succinctly expressed as $b_i X_k \geq X_{ik}$ for all $i = 1, ..., n$ and $k = 1, ..., K$, which is equivalent to the matrix inequality above.

This fact will be used in the next section for the problem of finding a binary matrix factorization of A.

6.3 Symmetric 0-1 Matrix Factorization

The SymNMF method of graph partitioning finds continuous nonnegative matrix $H \in R^{n \times k}$ so as to minimize $\left\|A - HH^T\right\|_F^2$ and obtains the 0-1 assignment matrix X by thresholding H. In the graph partitioning case, it arguably more desirable to obtain binary X directly rather than via the continuous H.

This leads us to the *Symmetric Binary Matrix Factorization* problem, henceforth called SymBMF. Formally, for a symmetric matrix $A \in [0,1]^{n \times n}$ we want to solve

$$\begin{aligned} \text{minimize} \quad & \left\|A - XX^T\right\|_F^2 \\ \text{subject to} \quad & X \in \{0,1\}^{n \times k} \\ & Xe_k = e_n \end{aligned}$$

The constraint $X^T e_n \geq 1$ can be added to ensure no column of X contains all zeros. Additionally, the constraint $(B - I)X \geq 0$ (where B is the unweighted adjacency matrix) can be added for graph partitioning purposes to ensure all the partitions are connected (6.2.2).

It is desirable for algorithms that solve SymBMF to work well when A is sparse or incomplete. In the sparse case, when most entries of A are zero, the complexity of the algorithm ideally ought to scale with the number of non-zero entries, and not with the number of rows and columns. Similarly in the case of incomplete A there are entries of A that are unknown or that we simply don't care about approximating. This changes the objective function from a matrix norm to a summation:

$$\sum_{(i,j) \in A} (A_{ij} - x_i^T x_j)^2$$

where x_i denotes the ith row of X.

We present two methods that both scale well in sparse A and can handle incomplete A. The first is a very fast local minimizer inspired by methods from multidimensional scaling. The second is a mixed integer program that solves the problem globally, but is not as efficient.

6.3.1 An MDS-Inspired Method

The central idea behind this to begin with an random $n \times k$ binary matrix X that satisfies $Xe_k = e_n$ and iteratively edit each row of X so as to minimize $\|A - XX^T\|_F^2$ locally. Editing a row of X here means determining which column to place the 1 in. The change made in row i of X only impacts the ith row and ith column of $A - XX^T$. The column to place the 1 in that minimizes the objective locally is:

$$k^* = \underset{k=1,\dots,K}{\operatorname{argmin}} \|A_i - X_k\|^2 \tag{6.12}$$

where A_i refers to the ith column of A and X_k to the kth column of X. This is because the ith column of XX^T is X_k, where k is the parcel that vertex i has been assigned to.

In the case of incomplete matrix A, the column selection rule in 6.12 should replaced by

$$k^* = \underset{k=1,\dots,K}{\operatorname{argmin}} \sum_{j \in A_i} (A_{ij} - X_{jk})^2 \tag{6.13}$$

The squared terms in (6.12) and (6.13) can also be changed to absolute value.

The stopping criterion halts the loop before iteration i if no rows have been edited since x_i was last edited.

6.3.2 Mixed Integer Programming Method

The MIP method begins by replacing objective's squared term in the Frobenius norm with an L1 penalty. This new objective function is equivalent to the original if A is binary.

Algorithm 5 SymBMF

```
1: Initialize $X \in \{0,1\}^{n\times k}$ such that $Xe_k = e_n$
2: $i \leftarrow 1$
3: $j \leftarrow 1$
4: repeat
5:     $k \leftarrow$ index of 1 entry of $x_i$
6:     $x_{ik} \leftarrow 0$
7:     Compute $k^*$ by (6.12) or (6.13)
8:     $x_{ik^*} \leftarrow 1$
9:     if $k \neq k^*$ then                              ▷ row $i$ has been edited
10:        $j \leftarrow i$
11:    end if
12:    $i \leftarrow (i \mod n) + 1$
13: until $i = j$
```

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in A} |A_{ij} - x_i^T x_j| & \\ \text{subject to} & x_i \in \{0,1\}^k & \text{for } i = 1, ..., n \\ & e_k^T x_i = 1 & \text{for } i = 1, ..., n \end{array}$$

This problem would be a mixed integer program if it were not for the quadratic $x_i^T x_j$ in the objective. We substitute a new variable $y_{ij} = x_i^T x_j$ and find linear constraints that make this relation true for binary x_i satisfying $e_k^T x_i = 1$. Note that an equivalent definition of y_{ij} is

$$y_{ij} = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & \text{otherwise} \end{cases}$$

The following lemma provides an linearization of $x_i^T x_j$:

Lemma 6.3.1 *Let x_i and x_j both be k-dimensional binary vectors that sum to 1. Then $y_{ij} = x_i^T x_j$ is equivalent to*

$$y_{ij} \leq \min(x_i - x_j) + 1$$

$$y_{ij} \geq \max(x_i + x_j) - 1$$

Proof: Follows from the fact that $\min(x_i - x_j) + 1$ *and* $\max(x_i + x_j) - 1$ *both equal*

1 if x_i and x_j are equal and 0 if not.

Substituting y_{ij} and adding the above linear constraints gives us the following MIP equivalent of SymBMF.

$$\begin{array}{lll}
\text{minimize} & \sum_{(i,j)\in A} |A_{ij} - y_{ij}| & \\
\text{subject to} & x_i \in \{0,1\}^k & \text{for } i = 1,...,n \\
& e_k^T x_i = 1 & \text{for } i = 1,...,n \\
& \begin{cases} y_{ij} \leq x_{ik} - x_{jk} + 1 \\ y_{ij} \geq x_{ik} + x_{jk} - 1 \end{cases} & \text{for } (i,j) \in A,\ k = 1,...,K
\end{array}$$

To enforce partition connectedness we can add the constraint $(B - I)X \geq 0$ from (6.2.2) where B is the unweighted assignment matrix.

Chapter 7

Mixed Integer and Linear Programming

7.1 A Global Solution to Mean Adjacent Within Edge

We want to find a partitioning of graph $G(V, E)$ into K components so as to minimize the Adjacent-Score (3.1.2)

$$\frac{1}{K}\sum_{V\in\mathcal{P}_K}\frac{1}{|E_{V,V}|}\sum_{(i,j)\in E_{V,V}}A_{i,j}$$

Let $m = |E|$, the number of edges in the graph. Assign each edge an index in $\{1, ..., m\}$ and let a be an m-dimensional vector whose jth entry is the weight of the j-indexed edge. Since distance correlation is between 0 and 1, so are the entries of a.

For $k = 1, ..., K$, let $z_k \in \{0, 1\}^m$ be a 0-1 vector with jth entry satisfying

$$z_{jk} = \begin{cases} 1 & \text{if both endpoints of edge } j \text{ are in } V_k \\ 0 & \text{otherwise} \end{cases}$$

If $z_1, ..., z_K \in \{0,1\}^m$ describe a valid partitioning, then they must satisfy $\sum_{k=1}^{K} z_{jk} \leq 1$ for all edges j. However, the converse is not true, since this contraint still allows two edges sharing an endpoint to be within different parcels.

To prevent that from happening, we introduce the assignment matrix $X \in \{0,1\}^{n \times m}$ with entries

$$x_{i,k} = \begin{cases} 1 & \text{if vertex } i \in V_k \\ 0 & \text{otherwise} \end{cases}$$

and three constraints

$$1 + z_{jk} \geq x_{hk} + x_{ik}$$
$$z_{jk} \leq x_{hk}$$
$$z_{jk} \leq x_{ik}$$

for all $j = 1, ..., m$, $k = 1, ..., K$, where (h, i) are the two endpoints of edge j. If we constrain X to be binary then the three above constraints are equivalent to:

$$z_{jk} = \begin{cases} 1 & \text{if } x_{hk} = 1 \text{ and } x_{ik} = 1 \\ 0 & \text{otherwise} \end{cases}$$

For the X, we only need to ensure every vertex is in a parcel and every parcel has at least one vertex (or some other specified minimum):

$$\sum_{k=1}^{K} x_{ik} = 1$$

$$\sum_{i=1}^{n} x_{ik} \geq 1$$

where the equation holds for all i and the inequality for all k.

Hence the following optimization problem finds a valid partition that maximizes adjacent-score. This is an instance of generalized fractional linear programming. Let e_m denote a vector of m ones.

$$\text{maximize} \quad \frac{a^T z_1}{e_m^T z_1} + \cdots + \frac{a^T z_K}{e_m^T z_K}$$

$$\text{subject to} \quad \begin{cases} 1 + z_{jk} \geq x_{hk} + x_{ik} \\ z_{jk} \leq x_{hk} \\ z_{jk} \leq x_{ik} \end{cases} \qquad j = 1, ..., m,\ k = 1, ..., K,\ (h, i) = j$$

$$\textstyle\sum_{j=1}^{m} z_{jk} \geq 1 \qquad k = 1, ..., K$$

$$\textstyle\sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1, ..., n$$

$$\textstyle\sum_{i=1}^{n} x_{ik} \geq 1 \qquad k = 1, ..., K$$

$$X \in \{0, 1\}^{n \times K}$$

Following the result in [Li, 1994] we derive an equivalent Mixed Binary Linear Program to the above. Substitute $y_k = \dfrac{1}{e_m^T z_k}$ for each k, which amounts to introducing a new variable $y \in \mathbb{R}^K$ and non-linear constraints

$$e_m^T z_k y_k = 1$$

The key theorem in [Li, 1994] uses the fact that z_k is binary to linearize this constraint by introducing another variable w_{jk} and using linear constraints to enforce the nonlinear $w_{jk} = z_{jk} y_k$. There are four linear constraints for each w_{jk}:

1. $y_j - w_{jk} \leq 1 - z_{jk}$

2. $w_{jk} \leq y_j$

3. $w_{jk} \leq z_{jk}$

4. $w_{jk} \geq 0$

If $z_{jk} = 1$, then 1 and 2 will ensure that $w_{jk} = y_k$. If $z_{jk} = 0$, then 3 and 4 will ensure that $w_{jk} = 0$. It is important to note that this construction wouldn't work if $y_k > 1$. In our case, this occurs if and only if $z_{jk} = 0$ for all j, which has already been excluded by the $\sum_{j=1}^{m} z_{jk} \geq 1$ constraint.

Now we are ready to present the mixed integer version

$$
\begin{array}{lll}
\text{maximize} & a^T w_1 + \cdots + a^T w_K & \\
\text{subject to} & \begin{cases} 1 + z_{jk} \geq x_{hk} + x_{ik} \\ z_{jk} \leq x_{hk} \\ z_{jk} \leq x_{ik} \end{cases} & j = 1, ..., m,\ k = 1, ..., K,\ (h, i) = j \\
& \sum_{j=1}^{m} z_{jk} \geq 1 & k = 1, ..., K \\
& \sum_{k=1}^{K} x_{ik} = 1 & i = 1, ..., n \\
& \sum_{i=1}^{n} x_{ik} \geq 1 & k = 1, ..., K \\
& X \in \{0, 1\}^{n \times K} & \\
& \sum_{j=1}^{m} w_{jk} = 1 & k = 1, ..., K \\
& \begin{cases} y_j - w_{jk} \leq 1 - z_{jk} \\ w_{jk} \leq y_j \\ w_{jk} \leq z_{jk} \\ w_{jk} \geq 0 \end{cases} & j = 1, ..., m,\ k = 1, ..., K
\end{array}
$$

which can be solved globally by branch-and-bound methods. Unfortunately the size of our graph is too large for a generic MIP solver, and the largest graphs we partitioned using this method had around 400 vertices and 3000 edges, partitioned into

10 components. This is true even when the binary $\{0,1\}$ constraint was relaxed to an interval $[0,1]$ to create an LP.

7.2 An Approximate Solution Using SymBMF

A faster approximation with fewer variables to this MIP involves dropping the assignment matrix X. The problem becomes

$$\begin{array}{lll} \text{maximize} & \frac{a^T z_1}{e_m^T z_1} + \cdots + \frac{a^T z_K}{e_m^T z_K} & \\ \text{subject to} & \sum_{k=1}^{K} z_k = e_m & \\ & e_m^T z_k \geq 1 & \text{for } k = 1, ..., K \\ & z_k \in \{0,1\}^m & \text{for } k = 1, ..., K \end{array}$$

using the [Li, 1994] transformation we get

$$\begin{array}{lll} \text{maximize} & a^T w_1 + \cdots + a^T w_K & \\ \text{subject to} & \sum_{k=1}^{K} z_k = e_m & \\ & e_m^T z_k \geq 1 & \text{for } k = 1, ..., K \\ & z_k \in \{0,1\}^m & \text{for } k = 1, ..., K \\ & \sum_{j=1}^{m} w_{jk} = 1 & k = 1, ..., K \\ & \begin{cases} y_j - w_{jk} \leq 1 - z_{jk} \\ w_{jk} \leq y_j \\ w_{jk} \leq z_{jk} \\ w_{jk} \geq 0 \end{cases} & j = 1, ..., m, \; k = 1, ..., K \end{array}$$

If $K > 1$ and the graph is connected, then not every edge can be within a parcel; there must be at least one that is between two different parcels. However, the first constraint $\sum_{k=1}^{K} z_k = e_m$ forces the contrary — every edge must be assigned to exactly one parcel. Why does this constraint have to be an equality rather than a $\leq$ as in the first MIP? If it were the latter the optimal solution to this problem would be trivial: take the K edges with the largest weights in the graph and assign each to a different parcel. Do not assign any other entries of z_k 1. This optimal solution is useless for partitioning. This is a consequence of eliminating the X matrix from the first problem.

If the edges of the graph are over-assigned by z_k, how do we recover the assignment matrix X? We propose to create an approximate partition matrix (5.2.2) P from the z_k so that

$$P_{ij} = \begin{cases} 1 & \text{if } \underset{k}{\operatorname{argmax}}\, z_{ik} = \underset{k}{\operatorname{argmax}}\, z_{jk} \\ 0 & \text{otherwise} \end{cases}$$

Following this we could use either of the SymBMF methods introduced in the previous chapter to find a decomposition $P \approx XX^T$.

Chapter 8

Results

Chapter 9

Conclusion

Bibliography

[Buluç et al., 2013] Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C. (2013). Recent advances in graph partitioning. *CoRR*, abs/1311.3144.

[Chan et al., 1994] Chan, P. K., Schlag, M. D., and Zien, J. Y. (1994). Spectral k-way ratio-cut partitioning and clustering. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(9):1088–1096.

[Ding et al., 2005] Ding, C. H., He, X., and Simon, H. D. (2005). On the equivalence of nonnegative matrix factorization and spectral clustering. In *SDM*, volume 5, pages 606–610. SIAM.

[Fan, 1950] Fan, K. (1950). On a theorem of weyl concerning eigenvalues of linear transformations ii. *Proceedings of the National Academy of Sciences*, 36(1):31–35.

[Goldschmidt and Hochbaum, 1994] Goldschmidt, O. and Hochbaum, D. S. (1994). A polynomial algorithm for the k-cut problem for fixed k. *Mathematics of operations research*, 19(1):24–37.

[Kim and Park, 2011] Kim, J. and Park, H. (2011). Fast nonnegative matrix factorization: An active-set-like method and comparisons. *SIAM Journal on Scientific Computing*, 33(6):3261–3281.

[Kuang et al., 2015] Kuang, D., Yun, S., and Park, H. (2015). Symnmf: nonnegative low-rank approximation of a similarity matrix for graph clustering. *Journal of Global Optimization*, 62(3):545–574.

[Li, 1994] Li, H.-L. (1994). A global approach for general 0–1 fractional programming. *European Journal of Operational Research*, 73(3):590–596.