Graphs and Networks Lecture 21

Spectral Partitioning and Clustering

Daniel A. Spielman November 12, 2013

21.1 Overview

21.2 A useful identity

For real numbers $x_1, \ldots, x_n$ and $\mu = (1/n)\sum_i x_i$,

$$\sum_{i,j}(x_i - x_j)^2 = 2n\sum_i (x_i - \mu)^2.$$

21.3 Spectral Relaxation of Modularity

I recall Newman's modularity measure for a partition $P = \{C_1, \ldots, C_k\}$ of a graph G:

$$Q = \sum_{j=1}^{k}\Big(e(C_j) - \mathbf{E}\left[e(C_j)\right]\Big),$$

where $e(C_j)$ is the number of edges internal to cluster C_j and $\mathbf{E}\left[e(C_j)\right]$ is the expected number of such edges under a randomly re-wired model. Recall that

$$\mathbf{E}\left[e(C_j)\right] = \frac{d(C_j)^2}{4m},$$

where m is the number of edges in the graph and $d(C_j)$ is the sum of the degrees of the vertices in C_j.

We will now develop a way of expressing this with matrices and vectors. To begin, we could let $\boldsymbol{y}_j$ be the characteristic vector of set C_j. Then,

$$e(C_j) = (1/2)\sum_{a,b \in C_j} A_{a,b} = (1/2)\sum_{a,b:\boldsymbol{y}_j(a)=\boldsymbol{y}_j(b)=1} A_{a,b} = (1/2)\boldsymbol{y}_j^T A \boldsymbol{y}_j.$$

where A is the adjacency matrix Similarly,

$$d(C_j) = \boldsymbol{d}^T\boldsymbol{y}_j,$$

where $\boldsymbol{d}$ is the vector of degrees of vertices. This gives

$$\mathbf{E}\left[e(C_j)\right] = \frac{1}{4m}(\boldsymbol{d}^T\boldsymbol{y}_j)^2 = \frac{1}{4m}(\boldsymbol{y}_j^T\boldsymbol{d})(\boldsymbol{d}^T\boldsymbol{y}_j) = \frac{1}{4m}\boldsymbol{y}_j^T(\boldsymbol{d}\boldsymbol{d}^T)\boldsymbol{y}_j,$$

where we observe that $\boldsymbol{d}\boldsymbol{d}^T$ is a matrix.

So, we can now write the problem of maximizing modularity as that of maximizing

$$\sum_j \boldsymbol{y}_j^T\left(\frac{A}{2} - \frac{\boldsymbol{d}\boldsymbol{d}^T}{4m}\right)\boldsymbol{y}_j, \tag{21.1}$$

over vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_k$ in $\{0,1\}^n$ that have disjoint support and sum to the all 1s vector. This really doesn't simplify the problem.

We can achieve some simplification if we want a partition into just two parts: we will get it down to one vector. We let $\boldsymbol{x}$ be this vector, and set it to 1 for $a \in C_1$ and -1 for $a \in C_2$. Our instinct is to measure

$$\boldsymbol{x}^T\left(\frac{A}{2} - \frac{\boldsymbol{d}\boldsymbol{d}^T}{4m}\right)\boldsymbol{x}, \tag{21.2}$$

and this works!

To see why, observe that $(\boldsymbol{x}(a)\boldsymbol{x}(b)+1)/2$ equals 1 when a and b are in the same cluster, and 0 when they are in different clusters. So,

$$\begin{aligned}
(21.1) &= \sum_{a \equiv_P b} \frac{1}{2}A_{a,b} - \frac{1}{4m}\boldsymbol{d}_a\boldsymbol{d}_b \\
&= \sum_{a,b} \frac{1}{2}A_{a,b}(\boldsymbol{x}(a)\boldsymbol{x}(b)+1)/2 - \frac{1}{4m}\boldsymbol{d}_a\boldsymbol{d}_b(\boldsymbol{x}(a)\boldsymbol{x}(b)+1)/2 \\
&= \frac{1}{4}\boldsymbol{x}^T A\boldsymbol{x} + \frac{1}{4}\mathbf{1}^T\boldsymbol{A}\mathbf{1} - \frac{1}{8m}\boldsymbol{x}^T(\boldsymbol{d}\boldsymbol{d}^T)\boldsymbol{x} - \frac{1}{8m}\mathbf{1}^T(\boldsymbol{d}\boldsymbol{d}^T)\mathbf{1}.
\end{aligned}$$

As $\mathbf{1}^T\boldsymbol{A}\mathbf{1} = 2m$ and $\mathbf{1}\boldsymbol{d} = 2m$, this simplifies to

$$\frac{1}{4}\boldsymbol{x}^T\left(\boldsymbol{A} - (1/2m)(\boldsymbol{d}\boldsymbol{d}^T)\right)\boldsymbol{x}. \tag{21.3}$$

Also, maximizing this over $\boldsymbol{x} \in \{\pm 1\}^n$ is still a hard problem.

21.4 Relaxation of Modularity

One of the great tricks of optimization is to attempt approximate the maximimum of (21.3) by relaxing it. In this case, we do this by dropping the condition that $\boldsymbol{x} \in \{\pm 1\}^n$, and replace it with the condition that $\|\boldsymbol{x}\|^2 = n$. We then obtain the problem of maximizing

$$\boldsymbol{x}^T\left(\boldsymbol{A} - (1/2m)(\boldsymbol{d}\boldsymbol{d}^T)\right)\boldsymbol{x}$$

over vectors $\boldsymbol{x}$ of a given norm. It turns out that the maximum will be achieved by the eigenvector of largest eigenvalue of the matrix

$$\boldsymbol{A} - (1/2m)(\boldsymbol{d}\boldsymbol{d}^T).$$

For a proof of this, I recommend either looking in Newman's book or page 6 of Lecture 1 from my course on spectral graph theory.

There is then an obvious way to turn the eigenvector into two sets: we can take a prefix of the values in the vector that gives the lowest modularity score. While this is a useful heuristic, we unfortunately do not have any guarantees for how well it performs. We can, however, obtain guarantees about the analogous procedure for minimizing conductance.

21.5 Relaxing Conductance and Sparsest Cut

For this lecture, I re-define the conductance of a cluster to be

$$\Phi(C) \overset{\text{def}}{=} m \frac{|\partial(C)|}{d(C)d(V-C)}.$$

We will obtain a spectral approximation of this.

Actually, I will consider a simpler computation for the *sparsity* of a cut, which is defined to be

$$\mathsf{sp}(C) \overset{\text{def}}{=} \frac{|\partial(C)|}{|C|\,|V-C|}.$$

If $\boldsymbol{x}$ is the characteristic vector of C, then

$$|C|\,|V-C| = \sum_{a,b} (\boldsymbol{x}(a) - \boldsymbol{x}(b))^2.$$

And,

$$|\partial(C)| = \sum_{(a,b)\in E} (\boldsymbol{x}(a) - \boldsymbol{x}(b))^2 = \boldsymbol{x}^T \boldsymbol{L} \boldsymbol{x},$$

where $\boldsymbol{L}$ is the Laplacian matrix of the graph.

So, we want to minimize

$$\frac{\sum_{(a,b)\in E} (\boldsymbol{x}(a) - \boldsymbol{x}(b))^2}{\sum_{a,b} (\boldsymbol{x}(a) - \boldsymbol{x}(b))^2} \tag{21.4}$$

over vectors $\boldsymbol{x} \in \{0,1\}^n$. This is hard, so we will relax the problem by letting $\boldsymbol{x}$ range over all real vectors.

To figure out which vector achieves the minimum, we simplify the denominator of this expression. Using the identity I gave at the start of the lecture, we see that

$$\sum_{a,b} (\boldsymbol{x}(a) - \boldsymbol{x}(b))^2 = 2n \sum_{a} (\boldsymbol{x}(a) - \mu)^2,$$

where μ is the average of the $\boldsymbol{x}(a)$. Note that neither the numerator nor the denominator of our ratio is changed by shifting $\boldsymbol{x}$. That is, we can force the average to be zero. We thereby learn that the minimum of (21.4) is

$$\min_{\boldsymbol{x} \perp \mathbf{1}} \frac{\boldsymbol{x}^T \boldsymbol{L} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}.$$

The vector that minimizes this is precisely the eigenvector of the second-smallest eigenvalue of $\boldsymbol{L}$. Recall that the smallest eigenvalue is 0 and has eigenvalue $\mathbf{1}$, which doesn't tell us much.

If we wanted to do the same for minimizing conductance, we would take the left-eigenvector of second-largest eigenvalue of the walk matrix:

$$\boldsymbol{A}\boldsymbol{D}^{-1}.$$

As usual, we would partition the vertices by performing a sweep along this vector.

In this case, one of the great theorems of spectral graph theory, known as Cheeger's inequality, provides a guarantee on the conductance of the partition returned: it tells us that it is within a quadratic factor of the optimum. That is, up to constants, its conductance is no worse than the square root of the conductance of the optimal partition.

21.6 Clustering with many parts

To find a partition into many parts, it would be natural to use many eigenvectors. For the walk matrix, this works reasonably well. People have been doing it for a long time **(give references)**, Cheeger's inequality has recently been extended to prove that is works **(give more references)**. Let me explain one way to do this.

To begin, let $\boldsymbol{x}_2, \ldots, \boldsymbol{x}_k$ be the 2nd through kth (left) eigenvectors of the walk matrix. Recall that the first eigenvector is the constant vector, which does not tell us much. These eigenvectors give us $(k-1)$ coordinates for each vertex. For vertex a, create the vector $\boldsymbol{v}_a = (\boldsymbol{x}_2(a), \ldots, \boldsymbol{x}_k(a))$. When we were partitioning into just 2 parts, this was just a point on the real line. Now, we have n points in a $(k-1)$ dimensional space.

So, if we want to divide them into clusters, we could try to use some algorithm for clustering points in space. This actually works reasonably well.

21.7 K-Means

Before we get too into how one should cluster the vertices of a graph, lets take a moment to consider the seemingly easier problem of clustering vectors in $\mathbb{R}^d$. Lets call the vectors $x_1, \ldots, x_n$. One of the most popular measures of the quality of a partition of these vectors into clusters $C_1, \ldots, C_k$ is the k-means objective function. It is

$$\sum_{a=1}^{k} \frac{1}{|C_a|} \sum_{i,j \in C_a} \|x_i - x_j\|^2 . \tag{21.5}$$

This expression is simplified by setting μ_a to be the average of the points in cluster C_a:

$$\mu_a = \frac{1}{|C_a|} \sum_{i \in C_a} x_i .$$

We then have that (21.5) equals

$$\sum_{a=1}^{k} \sum_{i \in C_a} \|x_i - \mu_a\|^2 . \tag{21.6}$$

That is, we sum the square of the distance of each point to the center of its cluster.

While it is NP-hard to find the clusters that minimize this objective function (even for $k = 2$), there is a very popular heuristic called the k-means algorithm (introduced by Lloyd [Llo82]) for approximately minimizing the objective function. Before I tell you the algorithm, I'd like to complain that many people don't make the distincition between the objective function and the algorithm, which is just careless.

Lloyd's consists of alternating steps in which one computes the cluster-averages, $\mu_1, \dots, \mu_k$, and then shifts each point to the cluster with the closest center. That is, we alternate the steps

1. For each $1 \leq a \leq k$, set $\mu_a = (1/|C_a|) \sum_{i \in C_a} x_i$.
2. For each $1 \leq i \leq n$, put i in the cluster a for which $\|\mu_a - x_i\|$ is lowest.

One can show that each of these steps will decrease the objective function. I didn't say how to start. Typically, one will choose k random data points and make them the cluster centers. A better initialization is given by choosing the k points with probability inversely proportional the the square of their distance from the previous points (k-means++ [AV07]).

One typically runs this algorithm until it stops making any changes. Then, one usually runs it again and again with different random starts. It is not very consistent. But, it is easy to implement, so people like to use it.

21.8 Drawing Graphs using Eigenvectors

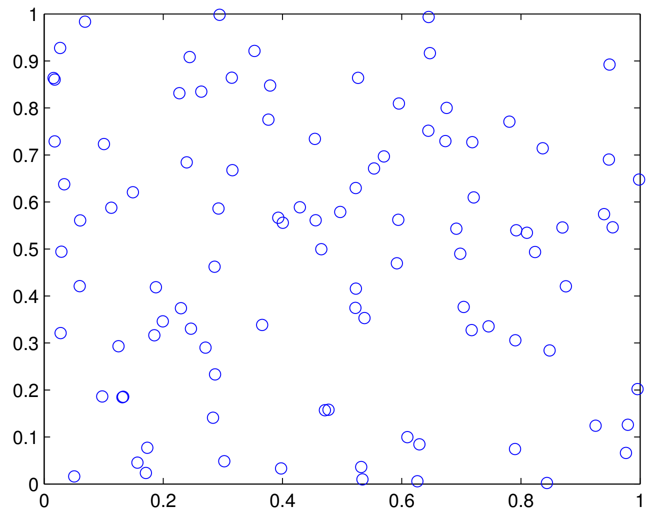
It turns out that if you want to draw a graph, a good easy way to do it is to take the two vectors $\boldsymbol{v}_2$ and $\boldsymbol{v}_3$, and locate vertex i as position

$$x_i = (\boldsymbol{v}_2(i), \boldsymbol{v}_3(i)) .$$

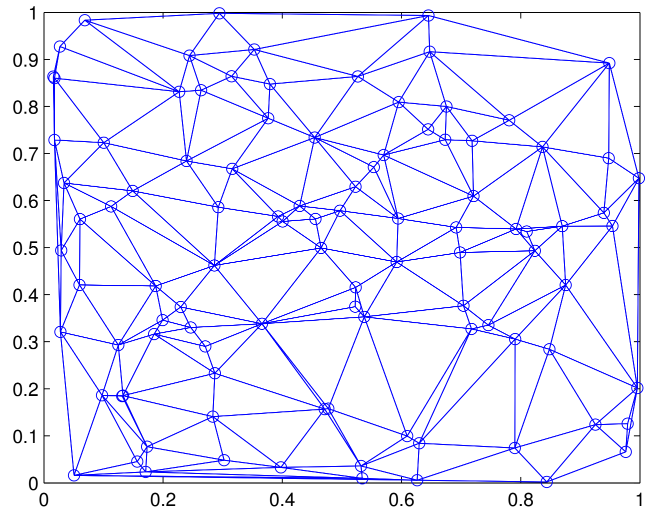
To convince you of this, let me show you the pictures this gives of some simple graphs. To draw the pictures, I will represent the edges as straight lines connecting the vertices.

To create my initial graph, I will choose 100 random points in the plane. I will then create a graph on them by taking their Delaunay triangulation.

```
>> [a,xy] = delGraph(100);
>> plot(xy(:,1),xy(:,2),'o')
```

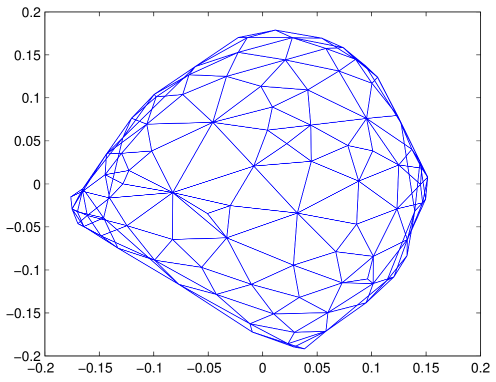


```
>> hold on
>> gplot(a,xy)
```



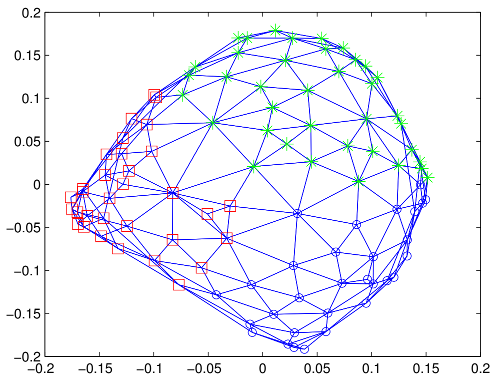
Now, I'll draw a picture of the graph using the first two non-trivial eigenvectors to obtain coordinates.

```
>> lap = diag(sum(a)) - a;
>> di = diag(1./sum(a));
>> [V,D] = eig(di*lap);
>> [val,ord] = sort(diag(D));
>> W = V(:,ord(2:3));
>> figure(2)
>> gplot(a,W)
```



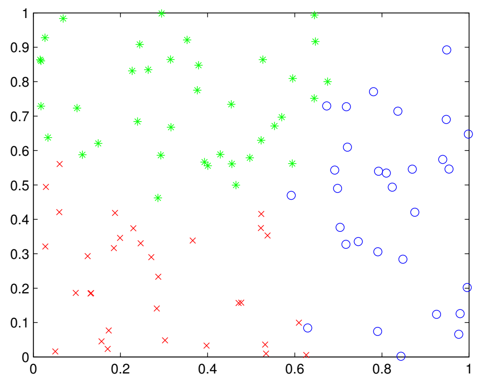
I'd say that this gives a pretty good picture. Moreover, it is clear that if we run k-means on these coordinates, we will get a reasonable clustering of the vertices. Let's try it out. We'll create three clusters. I'll first plot them over the spectral picture, and then in original space.

```
>> ide = kmeans(W,3);
>> figure(2)
>> hold on
>> plot(W(ide==1,1),W(ide==1,2),'o')
>> plot(W(ide==2,1),W(ide==2,2),'rs','MarkerSize',10)
>> plot(W(ide==3,1),W(ide==3,2),'g*','MarkerSize',10)
```



And, here it is in the original space.

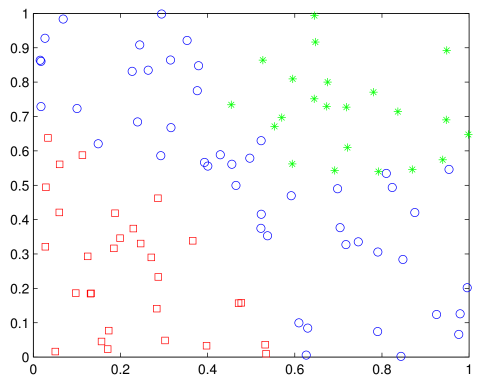
```
>> figure(1)
>> plot(xy(ide==1,1),xy(ide==1,2),'o')
>> hold on
>> plot(xy(ide==2,1),xy(ide==2,2),'rx')
>> plot(xy(ide==3,1),xy(ide==3,2),'g*')
```



Of course, we could use a more direct method to cluster points in the plane. I am not advocating using this method for that problem (although there are reasons to do something like this). Rather, I'm just trying to do an example in which it is visually clear that we are getting a reasonable answer.

Of course, I should compare this with using k-means on the adjacency matrix directly. Here is the result, plotted in the xy space.

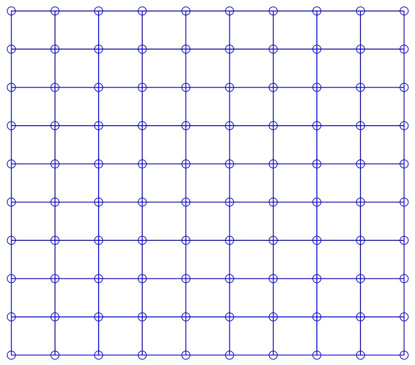
```
>> idx = kmeans(a,3);
>> figure(1)
>> clf
>> plot(xy(idx==1,1),xy(idx==1,2),'o')
>> hold on
>> plot(xy(idx==2,1),xy(idx==2,2),'rs')
>> plot(xy(idx==3,1),xy(idx==3,2),'g*')
```



It's not so bad, but I don't think it is as good as the spectral clustering.

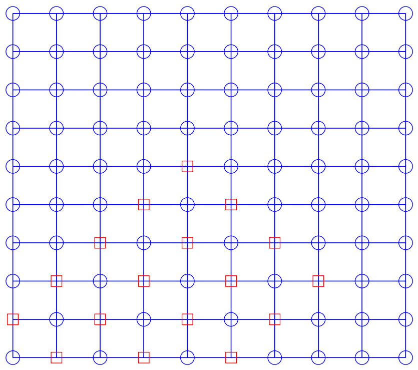
Now, let's see an example where using k-means directly does very poorly: on the grid graph. First, here's an image of this graph.

```
>> [a,jnk,xy] = grid2(10,10);
>> figure(1)
>> clf
>> plot(xy(:,1),xy(:,2),'o')
>> hold on
>> gplot(a,xy)
>> axis off
```



Let's cluster it with k-means.

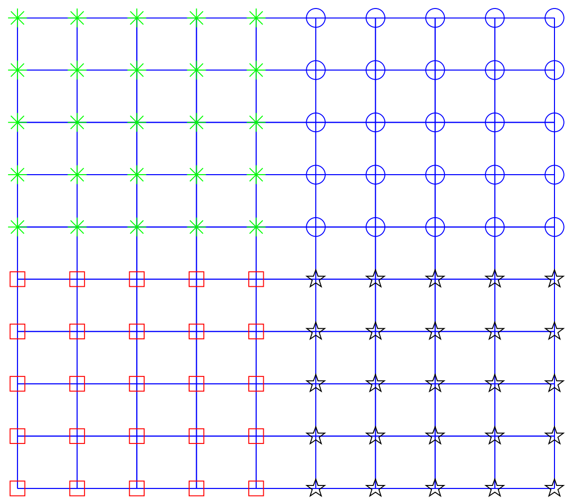
```
>> idx = kmeans(a,2);
>> clf
>> gplot(a,xy)
>> hold on; axis off
>> plot(xy(idx==1,1),xy(idx==1,2),'o','MarkerSize',10)
>> plot(xy(idx==2,1),xy(idx==2,2),'rs','MarkerSize',10)
```



I told you that k-means can do bad things on bipartite graphs.

Spectral partitioning, using k-means on the eigenvectors, gives almost perfect results for this graph. For overkill, I'll partition it into 4 pieces.

```
>> lap = diag(sum(a)) - a;
>> di = diag(1./sum(a));
>> [V,D] = eig(di*lap);
>> [val,ord] = sort(diag(D));
>> W = V(:,ord(2:4));
>> ide = kmeans(W,4);
>> clf
>> gplot(a,xy); hold on; axis of
>> plot(xy(ide==1,1),xy(ide==1,2
>> plot(xy(ide==2,1),xy(ide==2,2
>> plot(xy(ide==3,1),xy(ide==3,2
>> plot(xy(ide==4,1),xy(ide==4,2
```



21.9 Intrinsic Measures of Quality

We still need a way to measure the quality of a clustering. One way to start is to use a purely graph-theoretic measure.

Shi and Malik [SM00] advocate for the normalized cut measure:

$$\sum_a \frac{|\partial(C_a)|}{d(C_a)},$$

where we recall that $d(C_a)$ is the sum of the degrees of the vertices in C_a. In the case of two clusters, this has the advantage of exactly coinciding with a measure of conductance:

$$\begin{aligned}\frac{|\partial(C_0)|}{d(C_0)} + \frac{|\partial(C_1)|}{d(C_1)} &= \frac{|\partial(C_0)|}{d(C_0)} + \frac{|\partial(C_0)|}{d(C_1)} \\ &= (d(C_0) + d(C_1))\,\frac{|\partial(C_0)|}{d(C_0)d(C_1)} \\ &= (d(V))\,\frac{|\partial(C_0)|}{d(C_0)d(V-C_0)}.\end{aligned}$$

When I defined conductance I usually put the minimum in the denominator. But, it is common to take the product instead. I note that Shi and Malik [SM00] introduced their spectral clustering algorithm as a relaxation of the problem of minimizing the normalized cut objective function.

This is a variation of the k-way ratio cut measure introduced by Chan, Schlag and Zien [CSZ94]:

$$r(C_1,\ldots,C_k) \stackrel{\text{def}}{=} \sum_a \frac{|\partial(C_a)|}{|C_a|}.$$

Chan, Schlag and Zien [CSZ94] also derive their spectral clustering algorithm as a relaxation of this optimization problem.

In fact, Dhillon, Guan and Kulis [DGK04] have proved that there is a set of vectors that are naturally associated with a graph so that one minimizes the above quantity by optimizing the k-means objective function on those vertices. We get the vectors from the signed edge-vertex adjacency matrix (from Lecture 12):[1]

$$\boldsymbol{U}((a,b),c) = \begin{cases} 1 & \text{if } a = c \\ -1 & \text{if } b = c \\ 0 & \text{otherwise.} \end{cases}$$

This came up because the laplacian of an unweighted graph is given by

$$\boldsymbol{L} = \boldsymbol{U}^T \boldsymbol{U}.$$

We take the vector corresponding to vertex i to be the ith column of U.

The following result is proved by Dhillon, Guan and Kulis [DGK04].

Theorem 21.9.1. *For each vertex i, let x_i be the ith column of $\boldsymbol{U}$. The clustering $C_1, \ldots, C_k$ on these vectors that minimizes the k-means objective function is also the clustering that minimizes*

$$r(C_1, \ldots, C_k).$$

Proof. We first note that

$$x_i^T x_j = \begin{cases} -1 & \text{if } (i,j) \in E \\ d_i & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

So, for $i \neq j$,

$$\|x_i - x_j\|^2 = d_i + d_j - 2\mathbf{1}_{(i,j)\in E}.$$

In the following, I let $E(C_a)$ denote the set of edges between vertices in C_a, and recall that

$$d(C_a) = 2\,|E(C_a)| + |\partial(C_a)|\,.$$

For a cluster C_a,

$$\begin{aligned} \sum_{(i,j)\in C_a} \|x_i - x_j\|^2 &= (|C_a| - 1)\sum_i d_i + 2\,|E(C_a)| \\ &= |C_a| \sum_{i\in C_a} d_i - |\partial(C_a)|\,. \end{aligned}$$

So, the k-means objective function of a clustering $C_1, \ldots, C_k$ is

$$\begin{aligned} \sum_a \frac{1}{|C_a|} \sum_{(i,j)\in C_a} \|x_i - x_j\|^2 &= \sum_a \left(\sum_{i\in C_a} d_i + \frac{|\partial(C_a)|}{d(C_a)} \right) \\ &= 2m + r(C_1, \ldots, C_k). \end{aligned}$$

□

[1] In class, I thought that this was the unsigned version, but I was wrong.

The only problem with this result is that it is fragile. When one exatly optimizes the k-means objective function, one minimizes the k-way ratio cut score. But, if one merely approximately optimizes the k-means objective function then one can be very far from the optimum of the k-way ratio cut objective function.

References

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Jan 2007.

[CSZ94] P.K Chan, M.D.F Schlag, and J.Y Zien. Spectral k-way ratio-cut partitioning and clustering. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on DOI - 10.1109/43.310898*, 13(9):1088–1096, 1994.

[DGK04] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, Aug 2004.

[Llo82] S Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129 – 137, 1982.

[SM00] J. B. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.