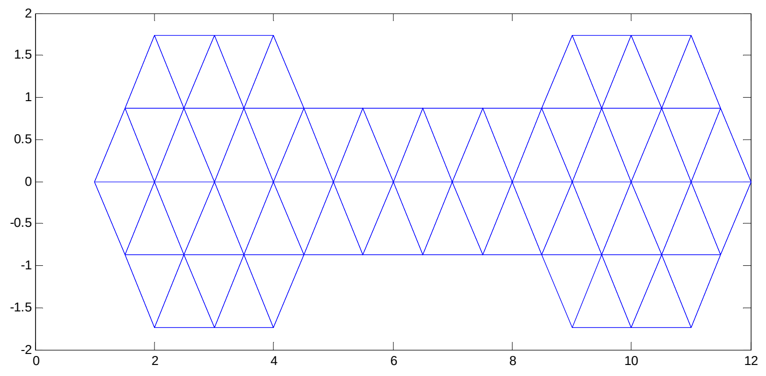
Lecture 19, Clustering Heuristics II

Monday, November 13, 2006
9:40 PM

I began class by restating the MCL algorithm, and giving an example of its performance on an interesting graph. Here's the example, with some explanation.

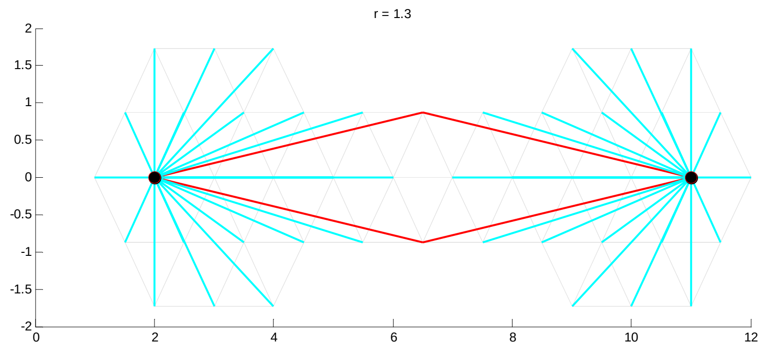
First, here's the graph I used:

```
load mclExamp
gplot(a,xy)
```

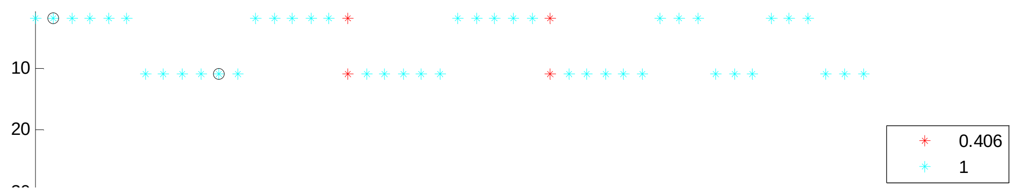


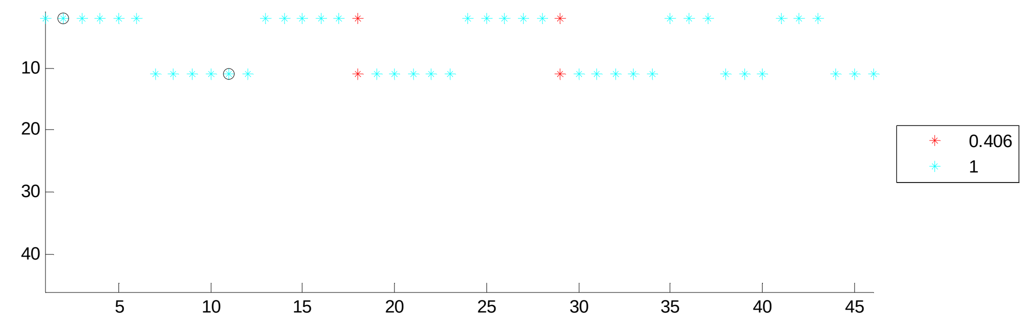
Now, I'll do a demo of the MCL algorithm on this graph, while varying the exponent r to which we raise each entry:

```
mcldemo
```



In this figure, I have drawn the graph in grey, and some other edges in blue and red. These edges correspond to the non-zero entries in the matrix produced by the MCL algorithm. The black nodes are the two nodes with non-zero diagonal entries. In the following figure, I present the matrix. But, rather than presenting it using numbers, I use white to denote zeros, and different markers for different non-zero values:

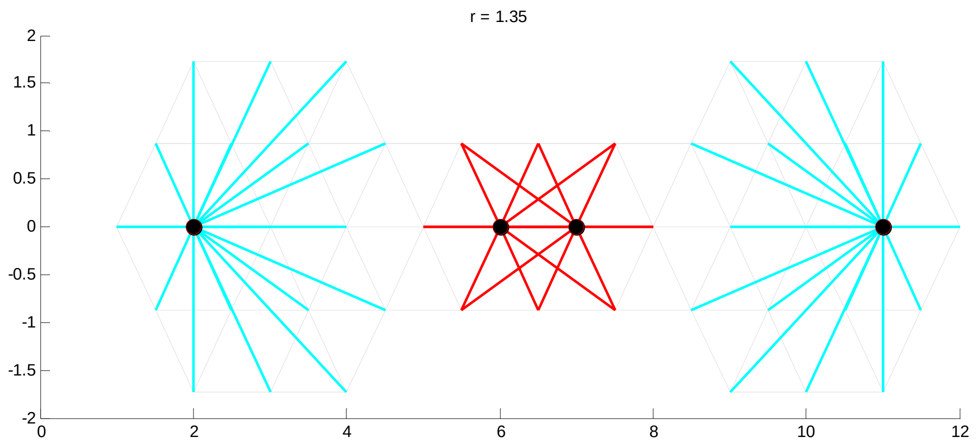


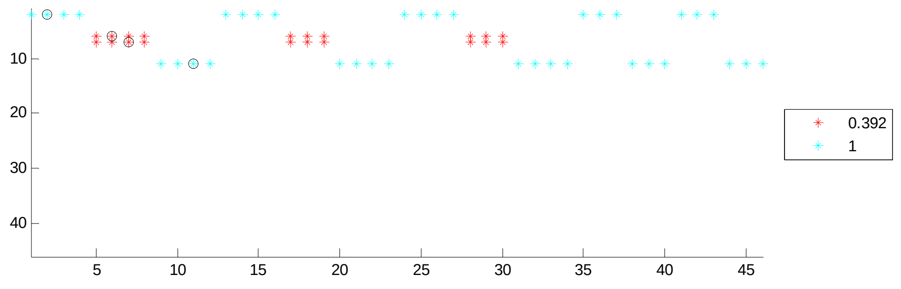


As you can see, it is a 46-by-46 matrix. You now also see that the colors of the edges above corresponded to the values in the non-zero entries. All the values were 1, except for the two nodes that appeared in both clusters, which were 0.406.

Oh, yes. I said in class that we form one cluster for each node that has a non-zero diagonal entry (circled in the picture above). We then connect it to each node that has a directed edge pointing to it (in the output matrix). The output matrix is normalized so that the sums of the weights of the edges out of each node is 1. While in class I usually make this mean that the row-sums are 1, when generating this figure I set the column-sums to 1. (I took the transpose).

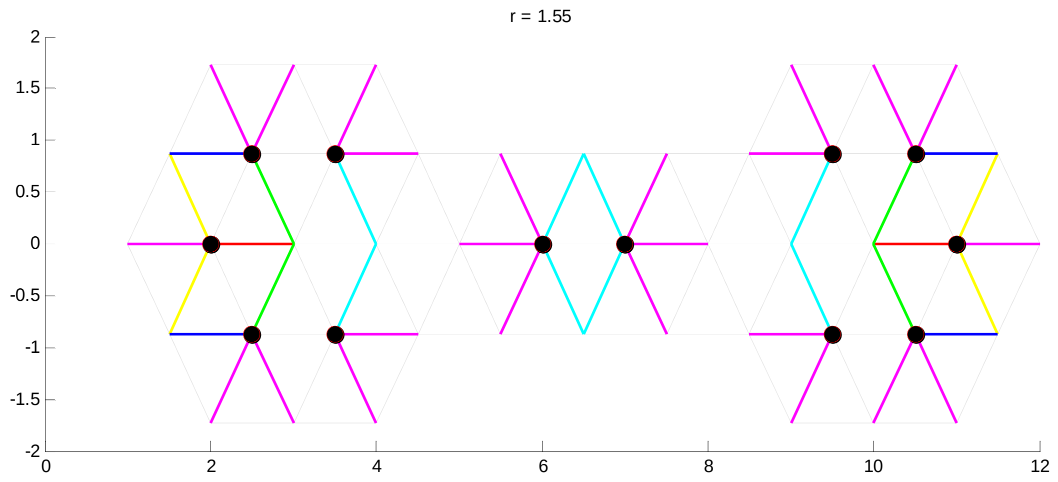
Now, let's go on with varying r.

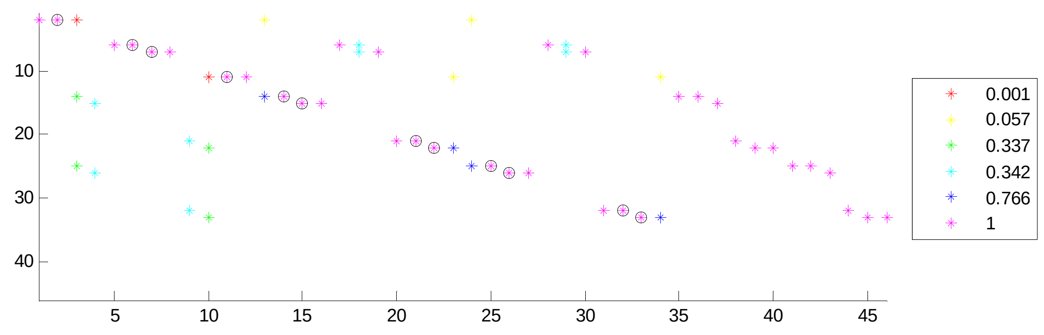




Here, we have clusters whose centers point to each other. In this case, we merge the clusters.

I go through many more values of r in the demo. But in these notes I'm skipping to the next one I want to dwell on:





Here, you can see that it is possible to get many different values in the matrix.

I then went on to prove three properties that any matrix M must have if M = M^2 and the sum of the weights on the edges coming out of each vertex is one. They were

1. $x \to y$ and $y \to z \Rightarrow x \to z$

2. $x \to x$ and $x \to y \Rightarrow y \to x$

3. $x \to y \Rightarrow \cancel{x} \to y$

I refer you to pages 57 and 58 of van Dongen's thesis for the proof.

K-Means

K-means is a popular way of clustering vectorial data. I'll briefly explain k-means, and then explain a result by Dhillon, Guan and Kulis that allows us to apply the algorithm in graphs.

Given n points $x_1, \ldots, x_n \in \mathbb{R}^d$ the k-means objective function

defines the quality of a clustering $C_1, \ldots, C_k$ to be

$$\sum_{a=1}^{k} \sum_{x_i \in C_a} \|x_i - \mu_a\|^2, \text{ where } \mu_a = \frac{1}{|C_a|} \sum_{x_i \in C_a} x_i$$

That is, μ_a is the average of points in its cluster, and the quality of the clustering

is the sum of the squares of the distances of points in a cluster to its average.

A popular algorithm for approximately optimizing this objective function (it doesn't have any provable guarantees) is the natural greedy algorithm, which repeats the following steps.

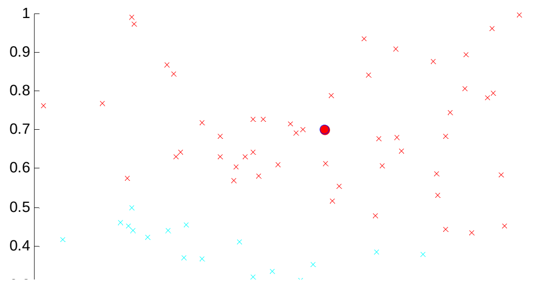
1. For each a, compute $\mu_a = \frac{1}{|C_a|} \sum_{x_i \in C_a} x_i$

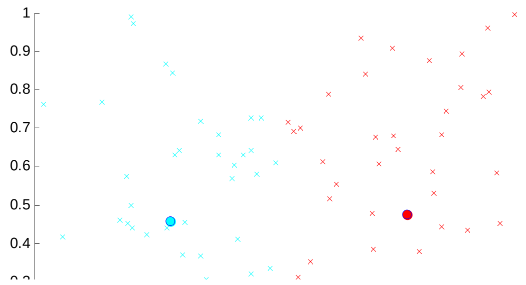
2. For each i, assign point x_i to the cluster minimizing

$$\|x_i - \mu_a\|^2$$

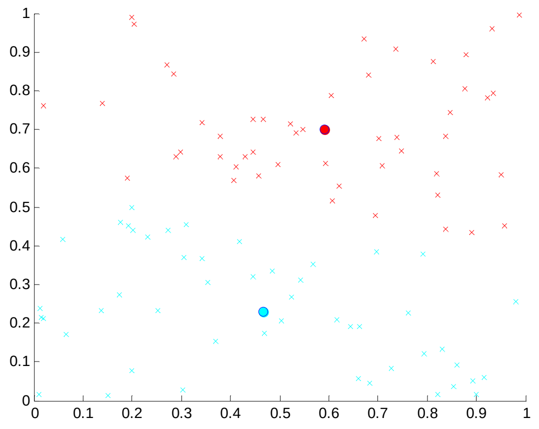
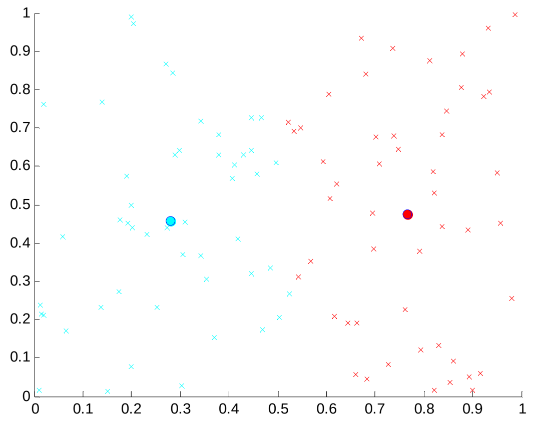
We typically initialize with either a crude clustering, or just a random clustering.
I used a random clustering in the following example. Press a key to step it.
Here is are two examples of the clustering it produced when asked for 2 clusters:

```
>> load nov7
>> kmeans(xy,2,10);
```



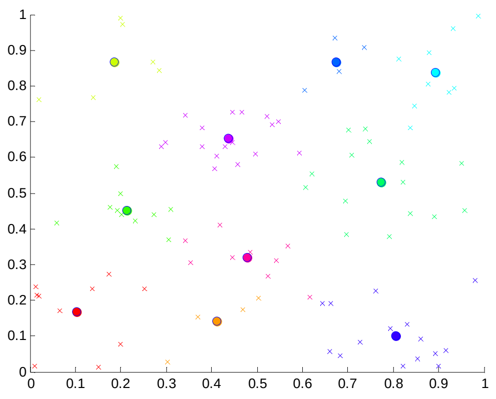
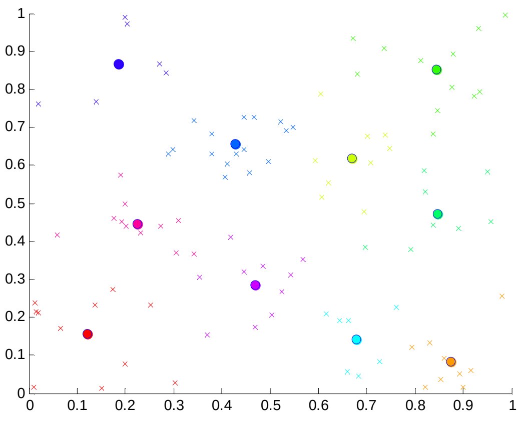


```
>> kmeans(xy,2,10);
```

And, for 10 clusters:

```
>> kmeans(xy,10,10);
```

The one formal thing we can say about this algorithm is that it converges. Informally, we can say that it is very popular in practice.

But, how are we going to apply it to graphs? The answer is to use "Kernel K-means". This means that we are going to map each vertex in a graph to a point in a vector space. This begs the question of how we are going to choose these points. The answer is that we are not going to. Rather, we are going to specify their inner products. The matrix of these inner products is called a kernel matrix, and we will denote it by K. We will never need to specify the points, as long as they exist.

That is, we just want to specify K such that

$$K_{i,j} = x_i^T x_j$$

Of course, given a particular K, it is not immediately clear whether there are vectors $x_1, \ldots, x_n$ such that $K_{i,j} = x_i^T x_j$.

It turns out that such vectors exists if and only if K is positive semi-definite. To see that the existance of these vectors implies that K is psd, note that it implies

$$K = X^T X, \text{ where } X = \begin{pmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & & | \end{pmatrix}$$

To go the other way, note that if K is psd, then its expansion in eigenvalues and eigenvectors has the form

$$K = VDV^T$$

As each entry in D is non-negative, we can set

$$X = D^{1/2} V^T, \text{ so } K = X^T X$$

So, we need to choose a psd matrix K.

We will now see that some fortunate choices of K demonstrate the equivalance between the k-means objective function and standard graph partitioning objective functions. We will consider the following two graph partitioning objective funtions:

sparsity:

$$sp(C_1, \ldots, C_k) = \sum_{a=1}^{k} \frac{|\partial(C_a)| \leftarrow \text{number of edges leaving}}{|C_a| \leftarrow \text{\# nodes in cluster } C_a}$$

conductance

$$\phi(C_1, \ldots, C_k) = \sum_{a=1}^{k} \frac{\text{vol}(\partial(C_a)) \leftarrow \Sigma \text{ of wts of edges leaving}}{\text{vol}(C_a) \leftarrow \Sigma \text{ of weighted}}$$

degree's

Conductance is usually a better measure. But, for now we'll just consider sparsity, as it is easier to deal with.

To get sparsity, let's consider the kernel matrix K =2D+A, that is the identity minus the Laplacian. First, we re-write the k-means objective function (using elementary algebra) as:

$$\sum_{a=1}^{K} \sum_{x_i \in C_a} \|x_i - \mu_a\|^2 = \sum_{a=1}^{K} \sum_{i,j \in C_a} \frac{\|x_i - x_j\|^2}{|C_a|}$$

and note

$$\|x_i - x_j\|^2 = x_i^T x_i - 2x_i^T x_j + x_j^T x_j$$

$$= K_{i,i} - 2K_i K_j + K_{jj}$$

With $K = 2D + A$, $K_{ii} = 2d_i$, $K_{i,j} = 1$ if (i,j) edge

$$\frac{1}{|C_a|} \sum_{i,j \in C_a} K_{ii} - 2K_{i,j} + K_{jj}$$

$$= \frac{1}{|C_a|}\left[(|C_a|+1) \sum_i K_{i,i} - 2 \sum_{i \neq j} K_{i,j} \right]$$

$$= \sum_i K_{i,i} + \frac{1}{|C_a|}\left[\sum_i K_{i,i} - 2 \sum_{i \neq j} K_{i,j} \right]$$

$$= 2\sum_i d_i + \frac{1}{|C_a|}\left[2\sum_i d_i - 2\left|\{ i \in C_a, j \in C_a : (i,j) \in E \}\right| \right]$$

$$= 2\sum_i d_i + \frac{1}{|C_a|}\left[2 \, |\partial(C_a)| \right]$$

$$= 2 \sum_i d_i + \frac{1}{|C_a|} \left[2 \cdot |\partial(C_a)| \right]$$

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

$$= 2 \sum_{i=1}^{n} d_i + 2 \sum_a \frac{|\partial(C_a)|}{|C_a|}$$

So, when we optimize the k-means objective function with this kernel, we also optimize the ratio cut objective function.

A warning is in order here: I got this result using a different kernel than was used in the paper of Dhillon, Guan and Kulis. I might have made a mistake. But, I'm at least close. They used K = I - D + A.

We can similarly get a k-means version of conductance. But, we need to use a weighted formulation of kernel k-means. See the paper.

Miller-Tolliver's spectral rounding

This is a paper that I am very excited about. It starts out by taking a simple spectral partitioning algorithm, and improving it by iteratively adjusting the weights on the edges using the eigenvectors computed in the previous iteration.

When producing k components, the algorithm converges with a weighted normalized Laplacian that has k zero eigenvalues. In this case, one can then read the components off immediately from the corresponding eigenvectors.

I'll give the updating rule for the case of k = 2, in which case we just use one eigenvector.

Motivation: note that the Rayleigh quotient (of a weighted graph)

$\dfrac{v^T L v}{v^T D v}$ can be written

$$\frac{\sum_{(i,j)\in E} w_{i,j}\,(v(i)-v(j))^2}{\sum_{(i,j)\in E} w_{i,j}\,(v(i)^2+v(j))^2}$$

So, let $w'_{i,j}$, the new weights, be given by

$$w'_{i,j} = w_{i,j}\,\psi\left(\frac{v(i)^2+v(j)^2}{(v(i)-v(j))^2}\right),$$

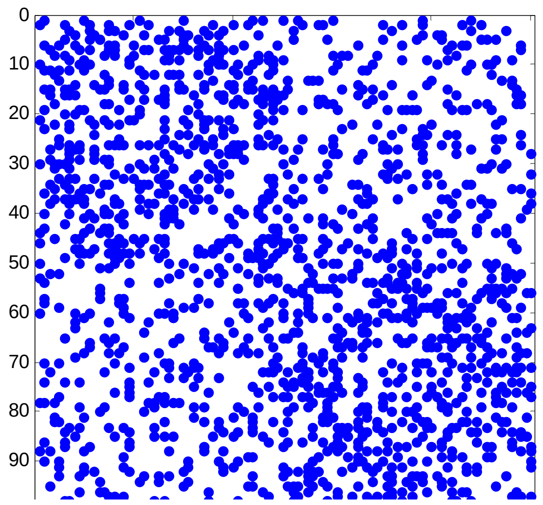
where $\psi(x) = \frac{x^2}{1+x^2}$

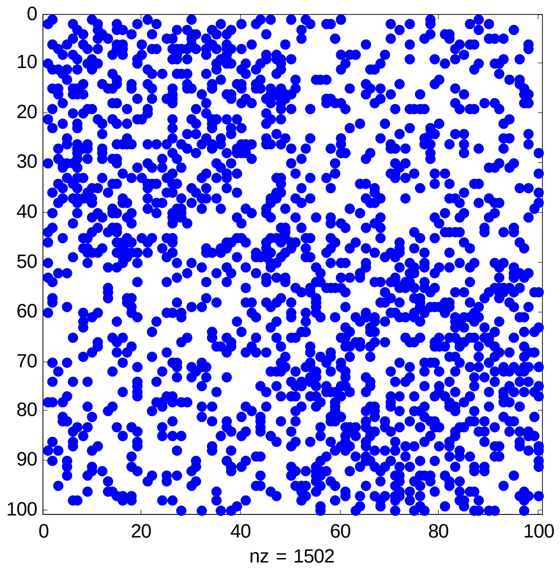
We do this with v an eigenvector ~~of~~ λ_2.

Here are some experiments on a random graph with a planted bisection. In particular, I divide 100 nodes into sets {1, …, 50} and {51, …, 100}. Within each half, I place edges with probability 0.2, and between halves with probability 0.1:

```
>> a = plantbisection(50,.2,.1);
spy(a)
```

Here is a plot of the adjacency matrix:





Let's check the quality of the partition we planed, and then of the partition generated by v_2 -- the straight spectral method:

```
>> ncutval(a,1:50)

ans =

   (1,1)         0.34968

>> f1 = crudeMillerTolliver(a,.001,1);

>> ncutval(a,find(f1 < 0))

ans =

   (1,1)         0.34742
```

So, the straight spectral method was just a little better. Now, let's run a few iterations of a crude version of Miller-Tolliver, and see that it gets even better

```
>> f = crudeMillerTolliver(a,.0001,10);
>> ncutval(a,find(f < 0))

ans =

   (1,1)         0.34683

>>
```