0. Announcements

Course Project

- Will start in early November.

Homework

- Start now to avoid clash with course project.
- Use Matlab. It is a much quicker prototyping language than C++ or C or Java.

1. Review

Previously, we have formulated the normalized cut problem as finding the solution y to the following generalized eigenvalue problem:

$$(D - W)y = \lambda D y \tag{1.1}$$

where y is a vector of length N equal to the number of pixels in the images. $W = \{w_{ij}\}$ is a size NxN sparse symmetric matrix of edge weights between pixels i and j, and D is a NxN diagonal matrix containing the row sums of W. i.e.

$$w_{ij} = w_{ij}^{texture} w_{ij}^{contour} \geq 0$$
$$D_{ii} = \sum_j w_{ij}$$

where $w_{ij}^{texture}$ and $w_{ij}^{contour}$ are formulated using methods from previous lectures. Note that D is positive definite by definition.

As an example, consider the following graph of edge weights:

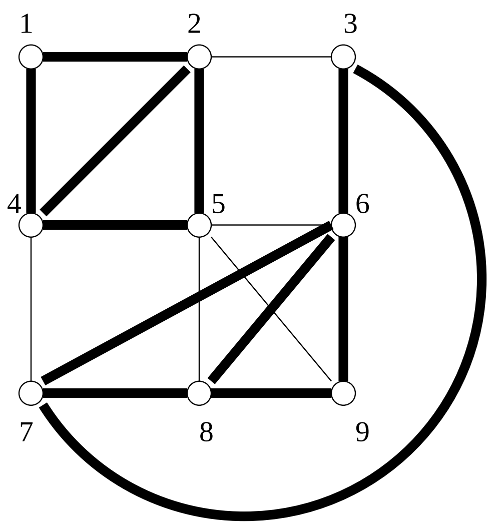


Fig. 1 Example: graph of edge weights between 9 pixels.

The strongly coupled pixel sets are $\{1, 2, 4, 5\}$ and $\{3, 6, 7, 8, 9\}$. It is easy to spot the strongly coupled pixels by looking at W. In each row, the entries with a much large weight than the rest of the row correspond to strongly coupled pixels. For example, the row in W for pixel 1 in the above graph might look like:

$$W_1 = \begin{pmatrix} 0.9 & 0.9 & 0.1 & 0.8 & 0.92 & 0.05 & 0.2 & 0.01 & 0.05 \end{pmatrix}$$

Trick 1.

Here is a trick that's used to convert generalized eigenvalue problems to standard eigenvalue problems if the matrix D is positive definite, which is true in our case. Introduce

$$z = D^{1/2} y$$

then

$$\begin{aligned} &y = D^{1/2} z \\ &(D - W) D^{-1/2} z = \lambda D^{1/2} z \\ &D^{-1/2} (D - W) D^{-1/2} z = \lambda z \end{aligned}$$

2. Using y to find a partition of the image

Let $\{y_1, y_2, \ldots, y_9\}$ be the set of solutions to equation (1.1) applied to our example above. Let $\{\lambda_1, \lambda_2, \ldots, \lambda_9\}$ be the corresponding set of eigenvalues. For this problem, all eigenvalues will be in the interval between 0 and 2. The trivial solution to the problem is a vector of all ones, and the corresponding eigenvalue is 0. So assuming that the eigenvalues are ordered in increasing order, we have

$$\lambda_1 = 0$$
$$y_1 = (1 \quad 1 \quad \ldots \quad 1)^T$$

All eigenvectors are orthogonal to each other. So

$$y_1 \cdot y_i = 0, \qquad 2 \leq i \leq 9$$

i.e.

$$\sum negative\ entries\ of\ y_i = \sum positive\ entries\ of\ y_i \tag{2.1}$$

This suggests the following approach to dividing the pixels into two groups.

2.1 Straightforward approach

Recall the original formulation for the normalized cuts problem:

$$Norm\_cut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

which is approximated by

$$\frac{y^T(D-W)y}{y^T D y}$$

Since we want the smallest normalized cut, we should pick the solution y which corresponds to the smallest non-zero eigenvalue, i.e. λ_2. The two groups will then be

$$GROUP\ A = \{i : y_2(i) > 0\}$$
$$GROUP\ B = \{j : y_2(j) < 0\}$$

This proves to work very well for images with only two groups.

Before continuing on, let us first make some observations regarding this problem.

- Eigenvectors y tend to be piecewise constant.

By piecewise constant, we mean the plot of y_i vs. i should look something like the following picture:

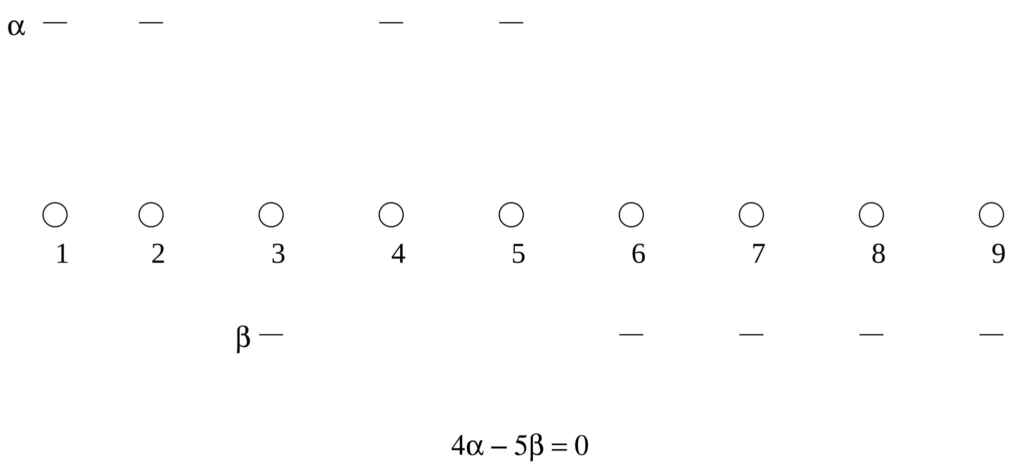


Fig. 2 Typical plot of eigenvector entries versus indices.

If the eigenvectors were not piecewise constant, then the thresholding at 0 in the above algorithm would not make sense. If the values in the eigenvector were continuous, then any thresholding value for dividing the pixels into groups would be too arbitrary.

From another point of view, if the image is not segmentable, then the eigenvectors will look like a sine wave. This is should make sense intuitively. Think of the spring mass analogy which we mentioned at the beginning of our discussion on normalized cuts. We can view the whole image as a network of spring with point masses at each pixel. The weight w_{ij} is then analogous to k_{ij}, the spring constant of the spring connecting point masses i and j. If $k_{ij} = w_{ij}$ is large, then the spring is stiff, and point masses i and j are likely to move together when the system is perturbed by external force. The motion vector is analogous to the eigenvector y_i. If the springs are stiff, the motion is continuous across neighboring pixels; if the image is unsegmentable, then y_i will resemble a sine wave rather than a piecewise constant function.

2.2 Generalization to K groups

For the generalization of our 2-group algorithm to K groups, let us first look at the other eigenvectors $y_3, y_4, \ldots, y_9$. Figure 3 shows a sample image with piecewise constant brightness values.

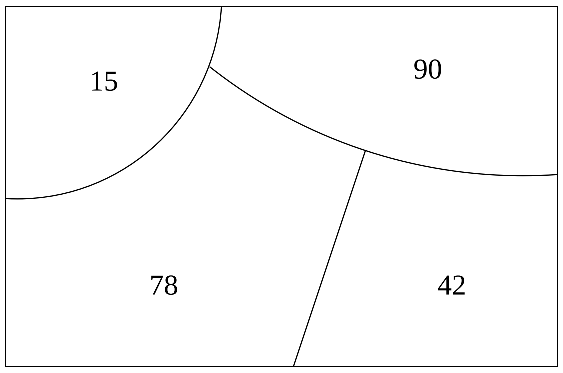
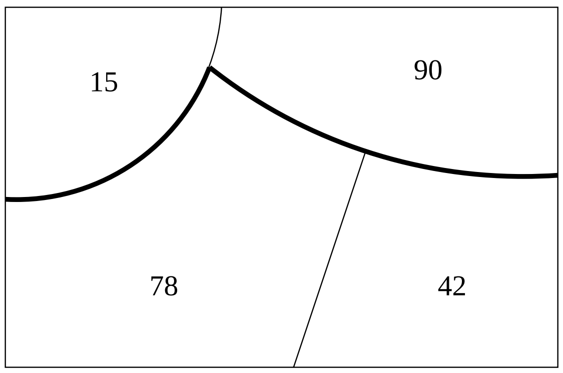
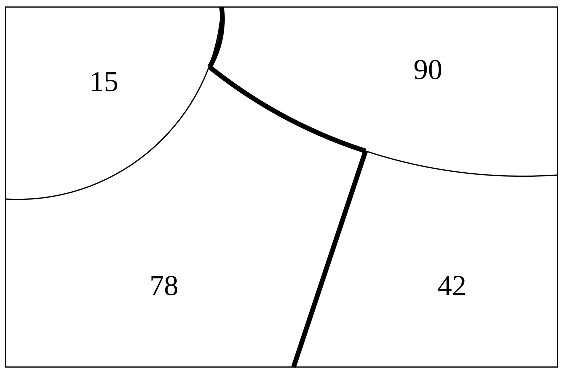


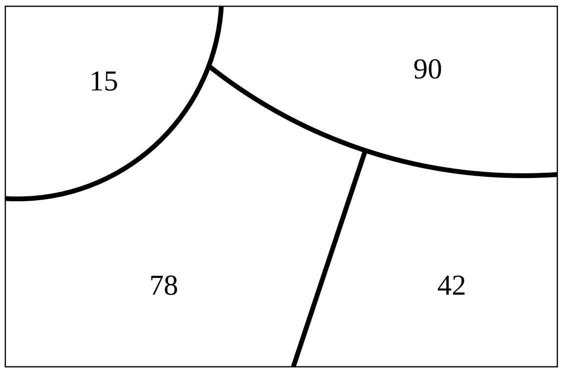
Fig. 3 Sample piecewise constant image with brightness as indicated.



Partion by y3.



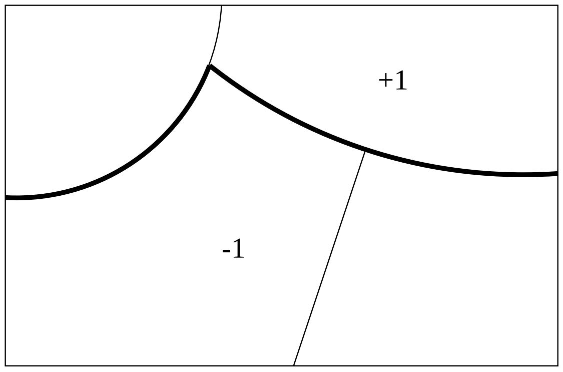
Partition by y2.



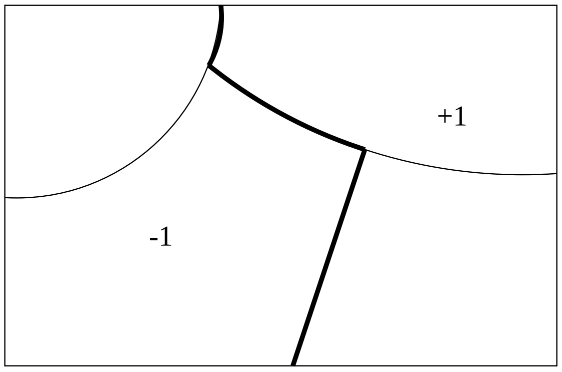
Combined results.

Fig. 4 Sample partitions of Figure 3.

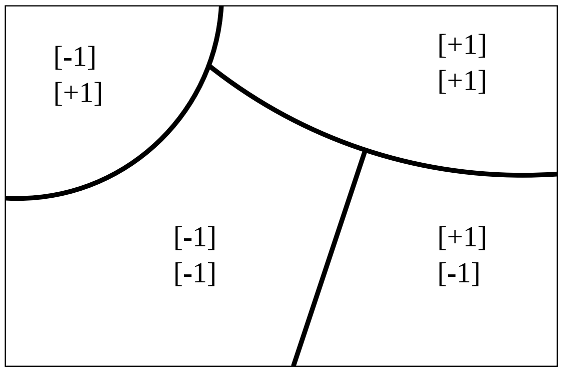
As shown in Figure 4, the eigenvector y_3 can possibly partition the image differently from y_2, but this suggests that we combine the partition from all eigenvectors to form the overall partition. Take the region with brightness value 78, for example. Since each eigenvector is piecewise constant within this group, we can form a piecewise constant vector in Z^d for each pixel, where d is the number of eigenvectors we choose to use for the partition. We can then use K-means to find clusters in Z^d.



Partion by y3.



Partition by y2.



Combined results.

Fig. 5 $f(x, y)$ of each group in image.

Algorithm 2. *Dividing an image into K groups using normalized cuts.*

- Define $f(x, y) = \{y_2(x, y), y_3(x, y), \ldots, y_d(x, y)\}$.
- Use K-means to find clusters in Z^d.

One problem remains. To date, we do not yet have a satisfactory algorithm to choose a value for K. There are two heuristics available follow in the case of normalized cuts. We can use the magnitude of the eigenvalue λ_i to decide whether to include y_i in our partition. Alternatively, one can divide the image first into 2 groups using the straightforward approach, then recursively apply the algorithm to further divide each of the two groups separately.