Proximity-based Clustering
Clustering with no distance information

- What if one wants to cluster objects where only similarity relationships are given?

Consider the following visualization of relationships between 9 objects:

- Nodes are the objects
- Edges are pairwise relationships
- Not embeddable in Euclidean space
- Not even a metric space! 😞

So how can we proceed with clustering??
Clustering with no distance information

• Say $k = 2$ (i.e., partition the objects into two clusters), what would be a reasonable answer?

Since edges indicate similarity, want to find a cut that minimizes crossings

Which of the three partitions is most preferable? Why?
Clustering with no distance information

- Say $k = 2$ (i.e., partition the objects in two clusters), what would be a reasonable answer?

Want a cut which minimizes crossings, but also keep cluster/partition sizes large.
Clustering by finding “balanced” cut

Let the two partitions be \( P \) and \( P' \), then we can minimize the following

\[
\min_{P, P'} \frac{\text{cut}(P, P')}{\text{vol}(P)} + \frac{\text{cut}(P', P)}{\text{vol}(P')}
\]

‘cut’ is the number of edges across a partition
‘vol’ is the number of edges within a partition

In general, for \( k \) partitions the optimization generalizes to

\[
\min_{P_1, \ldots, P_k} \sum_i \frac{\text{cut}(P_i, \bar{P}_i)}{\text{vol}(P_i)}
\]

[Shi and Malik ’00]
Clustering by finding “balanced” cut

Let the two partitions be $P$ and $P'$, then we can minimize the following

$$\min_{P, P'} \frac{\text{cut}(P, P')}{\text{vol}(P)} + \frac{\text{cut}(P', P)}{\text{vol}(P')}$$

‘cut’ is the number of edges across the partition

So how can we minimize above?
Let’s simplify it further...

$$\frac{\text{cut}(P, P')}{\text{vol}(P)} = \frac{(1_p)^T L (1_p)}{\text{vol}(P)}$$

$$= \left( \frac{1_p}{\sqrt{\text{vol}(P)}} \right)^T L \left( \frac{1_p}{\sqrt{\text{vol}(P)}} \right)$$

$1_p = \text{indicator vector on } P$
$L = \text{graph Laplacian}$
Detour: The (graph) Laplacian

Given an (unweighted) directed graph \( G = (V, E) \)

Consider the incidence matrix \( C \) representation of the graph \( G \)

Define Graph Laplacian \( L \) as... \( L := C^T C \)

For each edge in the graph:
- +1 on source vertex
- -1 on the destination vertex
The graph Laplacian

Hence, \( L = C^T C = \begin{pmatrix} e_1 & e_2 & \ldots & e_m \end{pmatrix} \begin{pmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_m^T \end{pmatrix} = \sum_k e_k e_k^T \)

Say \( e_k \) is an edge \((i,j)\), then

\[ e_k = \begin{pmatrix} \ldots \\ 1 \\ \ldots \\ -1 \\ \ldots \end{pmatrix} \]

\[ e_k e_k^T = \begin{pmatrix} \ldots & i & \ldots \\ i & 1 & \ldots \\ \ldots & \ldots & \ldots \end{pmatrix} \]

\[ L = D - W \]

- \( D \) degree matrix (diagonal)
- \( W \) weight matrix

\( L = D - W \)

- \( \text{diagonals always positive} \)
- \( \text{off-diagonals always negative} \)
But why is $L=D-W$ called a Laplacian?

Let’s consider the Laplace operator from calculus...

For a function $f : \mathbb{R}^d \to \mathbb{R}$, Laplace $\Delta$ of $f$ is defined as:

$$\Delta f := \text{divergence of the gradient of } f$$

$$= \nabla \cdot \nabla f$$

$$= \left(\begin{array}{c}
\frac{\partial}{\partial x_1} \\
\frac{\partial}{\partial x_2} \\
\vdots \\
\frac{\partial}{\partial x_d}
\end{array}\right) \cdot \left(\begin{array}{c}
\frac{\partial}{\partial x_1} \\
\frac{\partial}{\partial x_2} \\
\vdots \\
\frac{\partial}{\partial x_d}
\end{array}\right) f$$

$$= \sum_i \frac{\partial^2 f}{\partial x_i^2}$$

$$= \text{Trace of the Hessian of } f$$

$$\approx \text{(mean) curvature}$$
Consider a discretization of $\mathbb{R}^d$, i.e., a regular lattice graph.

The (graph) Laplacian of this graph

Each row/col of $L$ looks as:

\[
\begin{bmatrix}
2d & -1 & -1 & -1 & -1 & 0 & 0 & 0 & \ldots \\
-1 & 2d & -1 & -1 & -1 & 0 & 0 & 0 & \ldots \\
-1 & -1 & 2d & -1 & -1 & 0 & 0 & 0 & \ldots \\
-1 & -1 & -1 & 2d & -1 & 0 & 0 & 0 & \ldots \\
-1 & -1 & -1 & -1 & 2d & 0 & 0 & 0 & \ldots \\
0 & 0 & 0 & 0 & 0 & 2d & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 2d & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 2d & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 2d \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{bmatrix}
\]

For better understanding, consider each coordinate direction:

\[
\begin{bmatrix}
\ldots & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & \ldots \\
\end{bmatrix}
\]

This acts like (discretized version of) the (negative) second derivative!!
Graph Laplacian of Regular Lattice

Each coordinate looks like
\[
\begin{bmatrix}
\ldots & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & \ldots
\end{bmatrix}
\]

This acts like (discretized version of) the (negative) second derivative!!

Consider the finite difference method for derivatives...

- (forward) difference: \( f' = \frac{f(x+h) - f(x)}{h} \)
- (backward) difference: \( f' = \frac{f(x) - f(x-h)}{h} \)

So the second order (central) difference:
\[
f'' = \frac{f(x+h) - f(x)}{h^2} - \frac{f(x) - f(x-h)}{h^2} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}
\]

That is, -2 on self, +1 on neighbors
The **graph Laplacian** captures the **second order information** about a function (on vertices), it can quantify how ‘wiggly’ a (vertex) function is.

**Applications:**

- Quantify the (average) **rate of change** of a function (on vertices)
- One can try to **minimize the curvature** to derive ‘flatter’ representations
- Can be used as a **regularizer** to penalize the complexity of a function
- Can be used for **clustering**!!
- ...
Let the two partitions be $P$ and $P'$, then we can minimize the following:

$$\min_{P, P'} \frac{\text{cut}(P, P')}{\text{vol}(P)} + \frac{\text{cut}(P', P)}{\text{vol}(P')}$$

'cut' is the number of edges across the partition.

So how can we minimize above? Let's simplify it further...

$$\frac{\text{cut}(P, P')}{\text{vol}(P)} = \frac{(\mathbf{1}_P)^\top L (\mathbf{1}_P)}{\text{vol}(P)}$$

$$= \left( \frac{\mathbf{1}_P}{\sqrt{\text{vol}(P)}} \right)^\top L \left( \frac{\mathbf{1}_P}{\sqrt{\text{vol}(P)}} \right)$$

$\mathbf{1}_P$ = indicator vector on $P$

$L$ = graph Laplacian
So the optimization can be re-written as:

\[
\min_{P,P'} \frac{\text{cut}(P, P')}{\text{vol}(P)} + \frac{\text{cut}(P', P)}{\text{vol}(P')}
\]

\[
\min_{f_1,f_2} \sum_{i=1}^{2} f_i^T L f_i
\]

s.t. \( f_i^T D f_i = 1 \) \quad \( f_i^T f_j = 0 \)

all entries of \( f_i \) are equal

Since we are minimizing a quadratic form subject to orthogonality constraints, we can approximate the solution via a generalized eigenvalue system!

Generalized eigensystem... \( A x = \lambda D x \) Since spectral decomposition is used to determine \( f_i \) clusters, this methodology is called spectral clustering
Spectral Clustering: the Algorithm

**Input:** $S$: $n \times n$ similarity matrix (on $n$ datapoints), $k$: # of clusters

- Compute the degree matrix $D$ and adjacency matrix $W$ from the *weighted* graph induced by $S$
  
  $d_i = \sum_j s_{ij}, \ w_{ij} = s_{ij}$

- Compute the graph Laplacian $L = D - W$

- Compute the *bottom $k$* eigenvectors $u_1, \ldots, u_k$ of the generalized eigensystem: $Lu = \lambda Du$

- Let $U$ be the $n \times k$ matrix containing vectors $u_1, \ldots, u_k$ as columns

- Let $y_i$ be the $i^{th}$ row of $U$; it corresponds to the $k$ dimensional representation of the datapoint $x_i$

- Cluster points $y_1, \ldots, y_n$ into $k$ clusters via a centroid-based alg. like $k$-means

**Output:** the partition of $n$ datapoints returned by $k$-means as the clustering
Spectral Clustering: the Geometry

- The eigenvectors are an approximation to the $f$ partition ‘indicator’ vectors in the normalized cut problem.

Data in original space, similar points can be located *anywhere* in the original space.

Data is *easy* to cluster in the new transformation.
What if similarity information is unavailable?

If distance information is available, one can usually compute similarity as

\[ e^{-\text{dist}^2 / \sigma^2} \]
Spectral Clustering in Action

(a) K-means

(b) Spectral Clustering
Spectral Clustering in Action
Spectral Clustering in Action

K-means

Spectral Clustering
Spectral Clustering in Action