Today: new approach for community detection aka "clustering"

↳ spectral methods

Spectral graph theory (/mining)

Basically: study and applications of eigenvectors and eigenvalues of a graph's adjacency matrix or some modified form of the adjacency matrix

PageRank: consider transition probability matrix $M = (D^{-1}A)^T$

↳ $Mx = \lambda x \rightarrow \lambda = 1$

Eigenvector Centrality:

Eigenvector Centrality:

Similar idea to PageRank

↳ $Ax = \lambda x$

Other applications:

- Clustering (or C.D.)
- Partitioning
- Coarsening
- Visualization

Graph Laplacian

L = graph laplacian

$L = D - A$ ← adj matrix

↑ diagonal degree

Normalized Laplacian

$L_N = I - D^{1/2} A D^{1/2}$

↳ fixes the largest eigenvalue

↳ fixes the largest eigenvalue to be 1 (like with PR)

Overall: spectral methods

map discrete → continuous space

(keep this in mind)

Spectral clustering

spectral decomposition

$L = U \Lambda U^T$

Λ = diagonal matrix eigenvalues

U = associated eigenvectors

↳ since L is symmetric, all of Λ are real, positive, and non-increasing

Fiedler vector/value

→ first non-null eigenvalue/vector of our Laplacian

of our Laplacian

Fiedler value is related to the connectivity of G

Larger = more connected

smaller = less connected

Bi-clustering Algorithm

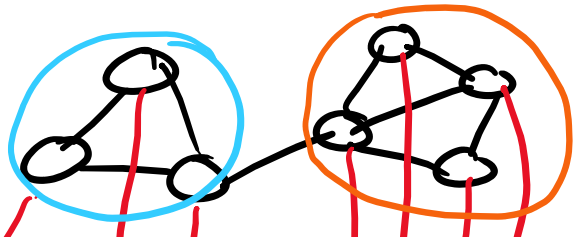
→ compute our Fiedler vector

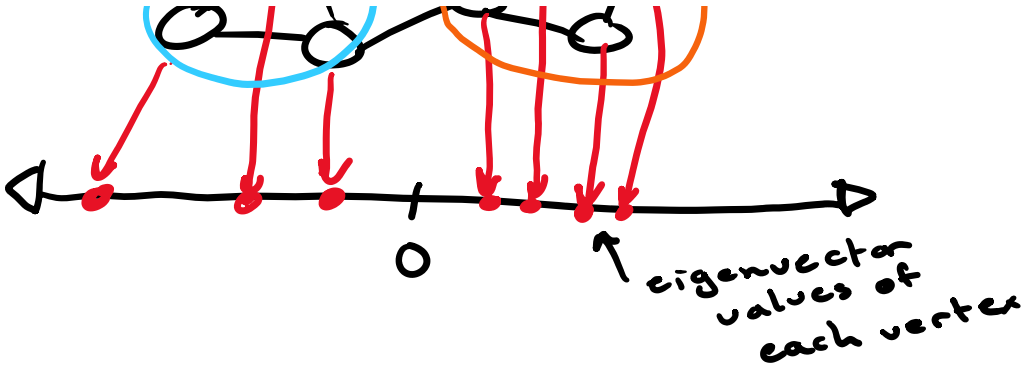
→ we get eigenvector values for each vertex

→ if value[v] ≥ 0 → cluster A
if value[v] < 0 → cluster B

Intuitively: we are mapping each vertex from discrete topological space to 1D continuous space

→ Think of placing each vertex along a number line





What about some arbitrary k number of clusters?

→ we can consider clustering each vertex's eigenvector values for some k in some j-dimensional space

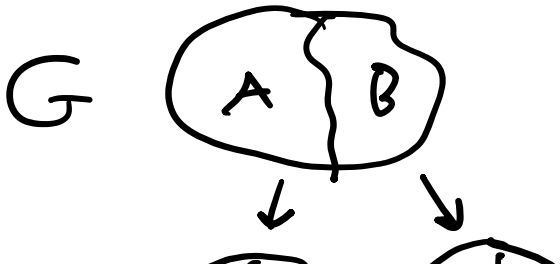
↳ i.e., we'll run k-means using the first j eigenvectors

OR: recursive bisection

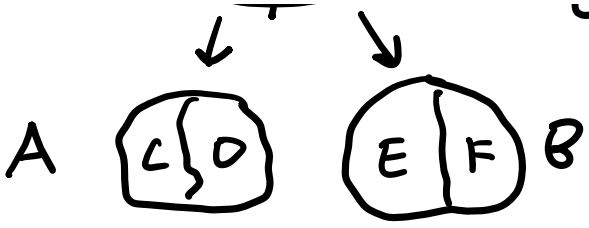
↓

we get A+B = G via Fiedler cut

→ can then get A = C + D
B = E + F
using same approach



Downside: we are constrained by



constrained by

$k = 2^m$

Using k-means to find k-clusters

→ Often: j eigenvectors of the largest j eigenvalues

where $j \sim k$

usually at least $j = \log_2(k)$

To use j-eigenvectors to cluster into k clusters

→ compute our j eigenvectors

→ each vertex gets j eigenvalues

→ this gives us coordinates for the vertex in j-dimensional space

→ we use those coordinates with a k-means algorithm

→ Output of k-means is our cluster assignments

cluster assignments

k-means:

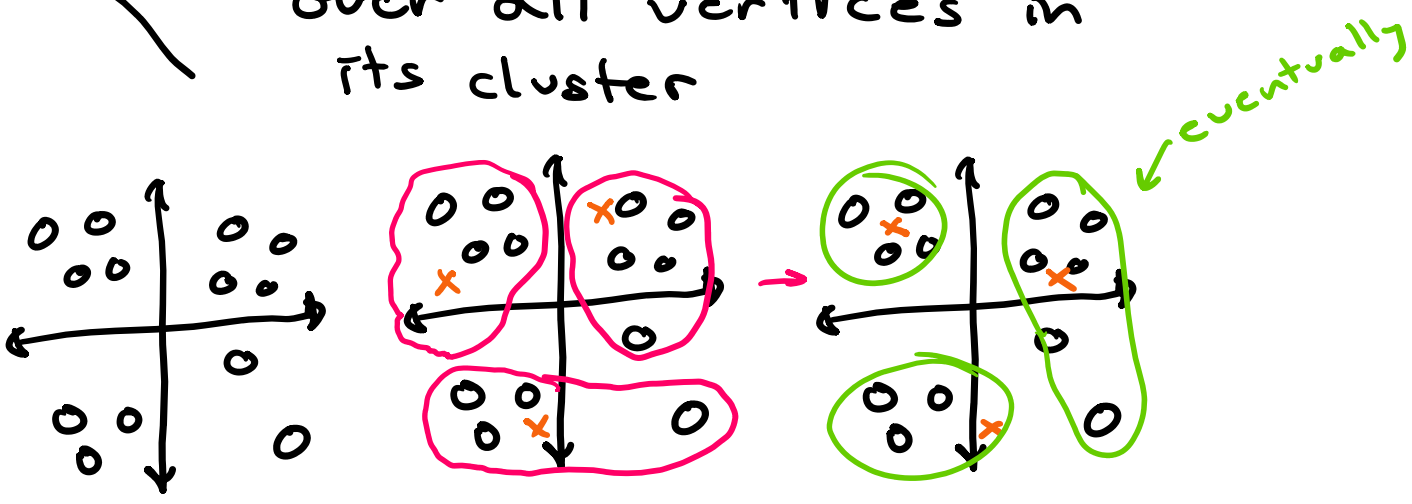
unsupervised clustering algorithm to find k cluster in some n-dimensional space

→ initialize k points randomly with n coordinates

iterate:

→ each vertex joins the closest cluster based on distance to one of the k points

→ each point updates their coordinates to the average over all vertices in its cluster



Spectral Partitioning

I.e., balanced graph partitioning

→ given G, find k equal(ish) sized clusters (or parts)

→ Usually formulated as an optimization problem

* minimize edge cut (inter-part edges)

* constrain part sizes

s.t. $|S_i| \approx |S_j| \quad \forall i,j$

$|S_i| \leq \frac{|V(h)|}{k} \epsilon$ ← imbalance tolerance

$\epsilon = 1.01$

↳ 1% imbalance

Note: many possible variants to this problem

Why: often used in HPC ⅓ scientific computing applications

Meshes = graphs, which are used in ~99% of scientific computation problems

Spectral partitioning

→ approach as with basic clustering, but constrain part sizes (cluster)

Intuition: a cut of clusters gives us a decent partitioning

Q: How to constrain cluster sizes?

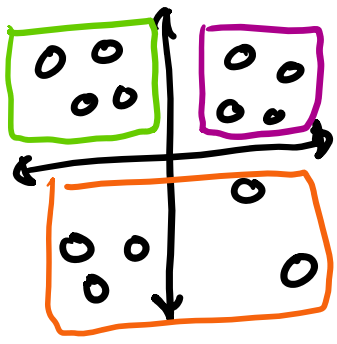
A: Consider median for cutoff for bi-clustering (bipartitioning)

↳ for any $k = 2^m$

For k-way partitioning where $k \neq 2^m$

↳ balanced k-means

OR: geometric-based approach



← geometric cut (many possible ways)

(graph) Visualization

(Graph) Visualization

Generally: drawing a graph in 2D/3D or some more complex multi-level space

Approches:

Spring-model → model edges as springs and vertices as weights, then solve for a stable solution

Spectral → basically what we've been doing, using eigenvectors to map vertices to 2D/3D visualization space

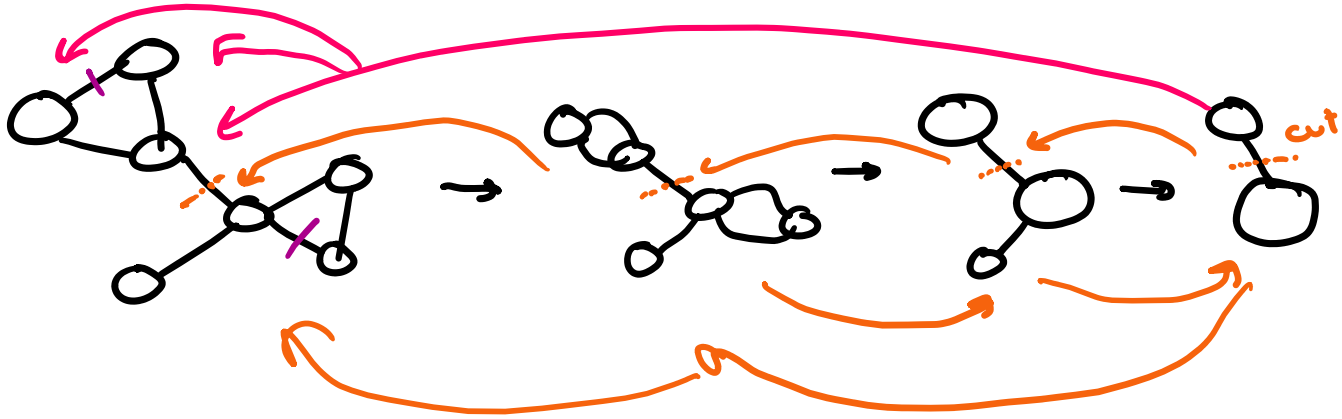
---

Graph Coarsening

↳ Representing some graph as a reduced-order model (think: edge-contraction)

Use: Solving complex problems at

Use: solving complex problems at a smaller scale, then extrapolating and refining the solution to the original input graph



Approaches for coarsening:

→ Label propagation (HW3)
(CD. algos)

→ max-weight/cardinality matching

→ spectral coarsening:
we select pairs or sets of vertices with similar eigenvector values