Monday, March 11, 2024 11:33 AM

Quick Review:

Community detection (CD)

→ Finding dense regions in a graph or network

→ Defining "dense"

* cut edge/ internal edges
* modularity
* conductance

Approaches:

→ optimize "density" directly

→ agglomerative, divise heuristics

→ greedy algos

Evaluation

→ relative to optimization metrics

→ relative to "ground truth"

Today: new approch

↳ spectral clustering

Spectral graph theory (/mining)

Basically: study and applications of eigenvalues and eigenvectors of a graphs adjacency matrix or some modified adjacency matrix

PageRank:

- spectral algo. on transition probability matrix
- $M = (D^{-1}A)^T \rightarrow Mx = \lambda x \xrightarrow{\lambda = 1} Mx = x$

Eigenvector centrality

- similar to PageRank
- $Ax = \lambda x$

Other applications:

Other applications:

- Clustering
- Partitioning
- Coarsening
- Visualization

Graph Laplacian

L = graph Laplacian

$L = D - A$ → adj. matrix

↳ diagonal degrees

Normalized Laplacian

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

↳ fixes largest eigenvalue to be 1

(like transition prob. matrix)

Overall: maps discrete → continuous space

Spectral clustering

spectral decomposition

$L = U \Lambda U^T$

Λ = diagonal matrix of 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

Fiedler value is related to the connectivity of G

Larger = more connected

smaller = less connected

Bi-clustering Algorithm

→ compute the Fiedler vector

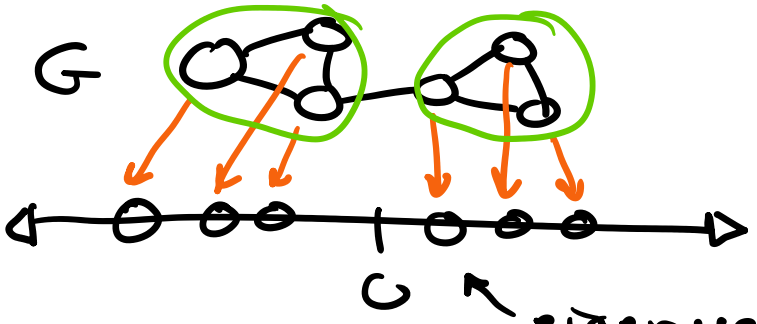
→ we get values for each vertex

→ we get values for each vertex

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

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

Think of putting each vertex along some number line



↖ eigenvector values of each vertex

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., via k-means

↳ i.e., via k-means

→ OR: recursive bisection

↓

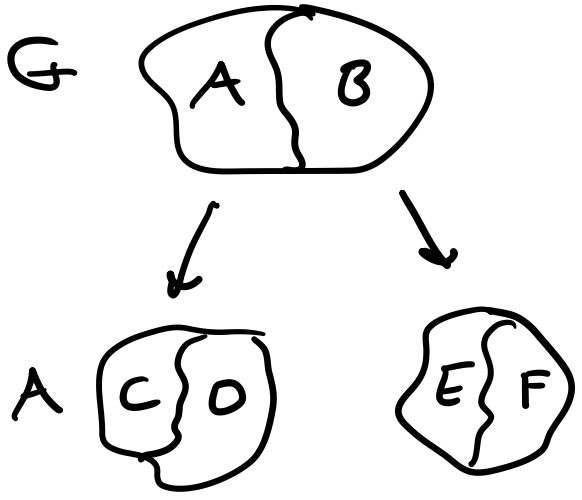
we get $G = A + B$

via Fiedler cut

↳ can then get $A = C + D$

$B = E + F$

using same approach on A, B



Downside: we are constrained by power-of-2s

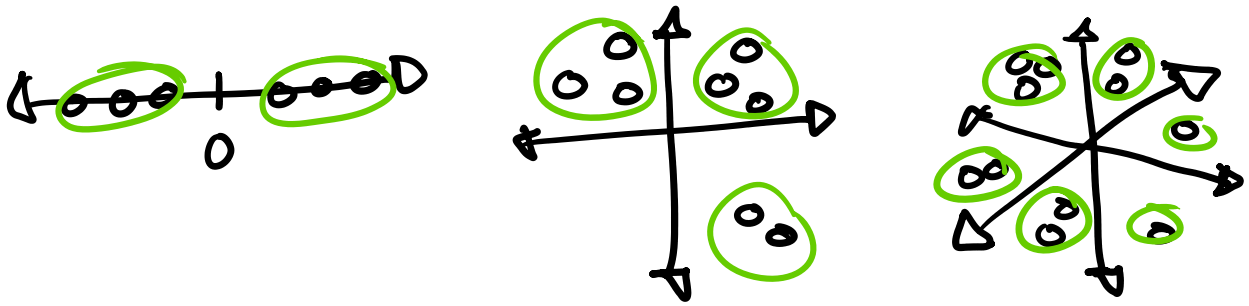
Using k-means to find k-clusters

→ Often: k eigenvectors of the largest k eigenvalues

→ Alternatively: we can use $(\log_2 k) + 1$ with geometric bisection

↳ run with 1D space

↳ we saw with 1D space for $k = 1$



Using j-eigenvectors to cluster into k clusters

→ compute the j eigenvectors

→ each vertex gets j eigenvector values

→ this gives coordinates for each vertex in j-dimensional space

→ we use those coordinates to do k-means clusters

→ output of k-means is our cluster assignments

k-means:

Unsupervised clustering algorithm to find k clusters in some

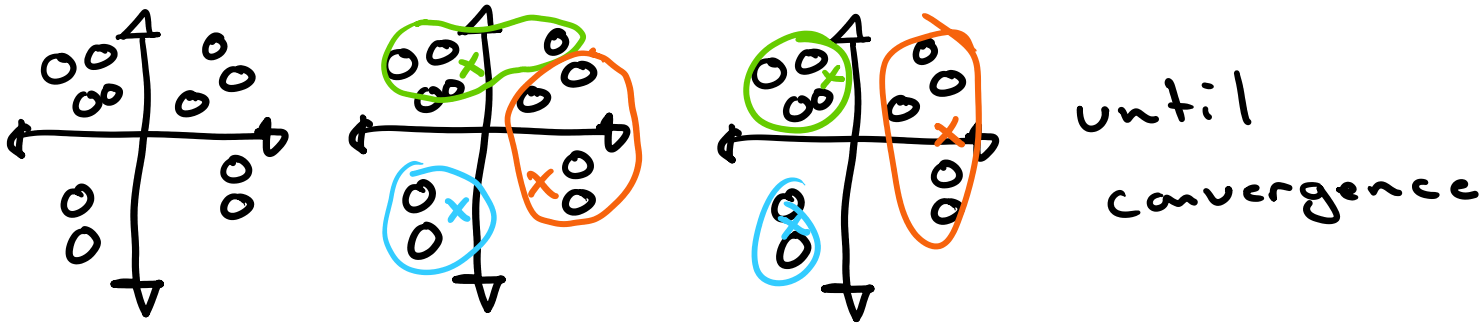
to find k clusters in same j-dimensional space

→ we initialize k points with j associated coordinates

→ each vertex joins the cluster defined by the smallest distance to one of those k points

→ each of those k points updates their coordinates to the mean coordinate values of all member vertices in the cluster

→ iterate until done



until convergence

k = 3

Spectral partitioning

I.e., balanced graph partitioning

→ Given G, we want k "equal" sized parts

→ Usually formulated as an optimization problem

* minimize edge cut (inter-part edges)
* constrain part sizes such that $|S_i| \approx |S_j| \; \forall i,j$

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

Note: many possible variants to this optimization problem

$\epsilon = 1.01$ ⇒ 1% imbalance

Why: often used in scientific computations and parallel processing

Meshes == graphs, which are used in 99.999... % of scientific problems

Edge cut = amount of communication

Imbalance = workload imbalance

Spectral partitioning

→ approach it like clustering, but constrain cluster sizes

→ Recall = "good" clusters have high density and therefore a low edge cut

↳ Intuition: we can optimize cut by finding good clusters

Q: How can we constrain cluster sizes?

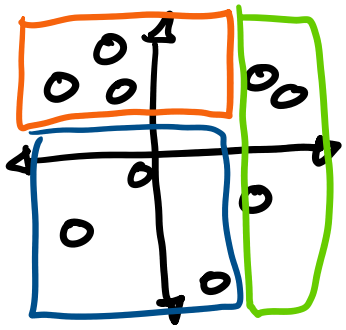
A: For bipartitioning, select the median value for the cluster assignments

↳ save for power-2 partitioning via recursive bisection

For k-way partitioning: we need to enforce balanced k-means

OR: a geometric-based approach

UK: a geometric-based approach



Visualization

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

Approaches:

Spring-model → model edges as springs and vertices as weights, and solve for some stable solution in 2D/3D space

Spectral → basically what we've been talking about, using the eigenvector values to map vertices to Euclidean (sp?) space

---

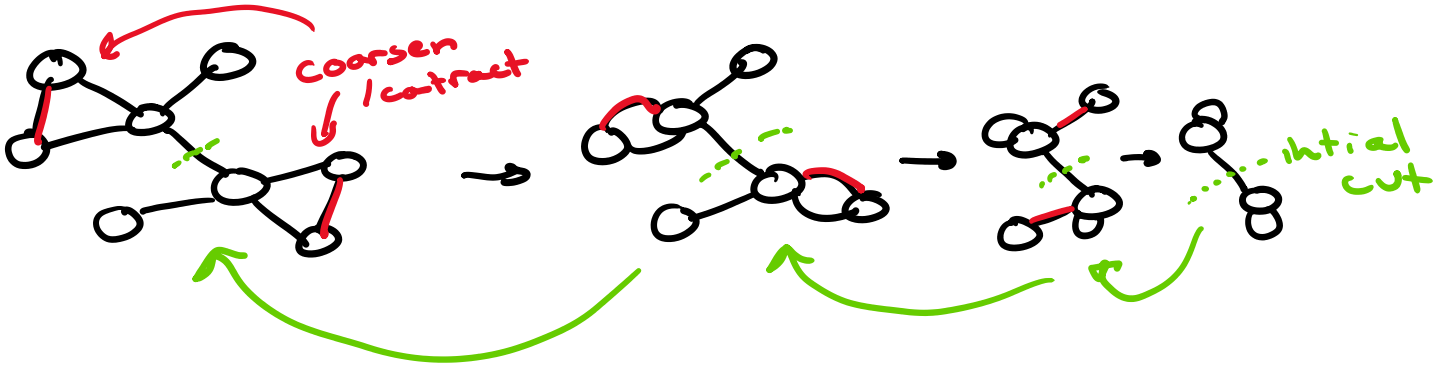
Graph Coarsening

Graph Coarsening

↳ Representing same graph as a reduced order model

Use: solving complex graph problems on a smaller graph, and then mapping that solution onto the larger original graph

(often done with partitioning)



Approaches for coarsening:

- Label propagation
- max-weight/cardinality matching
- spectral coarsening: we select the vertices with the closest eigenvector values to contract

eigenvector values to contract