

Graph Clustering

Why graph clustering is useful?

- Distance matrices are graphs → as useful as any other clustering
- Identification of communities in social networks
- Webpage clustering for better data management of web data

Outline

- Min s - t cut problem
- Min cut problem
- Multiway cut
- Minimum k -cut
- Other normalized cuts and spectral graph partitionings

Min $s-t$ cut

- Weighted graph $G(V,E)$
- An $s-t$ cut $C = (S,T)$ of a graph $G = (V, E)$ is a cut partition of V into S and T such that $s \in S$ and $t \in T$
- Cost of a cut: $\text{Cost}(C) = \sum_{e(u,v) \text{ } u \in S, v \in T} w(e)$
- **Problem:** Given G , s and t find the minimum cost $s-t$ cut

Max flow problem

- Flow network
 - Abstraction for material **flowing** through the edges
 - $G = (V, E)$ directed graph with no parallel edges
 - Two distinguished nodes: $s = \text{source}$, $t = \text{sink}$
 - $c(e) =$ capacity of edge e

Cuts

- An s–t cut is a partition (S, T) of V with $s \in S$ and $t \in T$
- capacity of a cut (S, T) is $\text{cap}(S, T) = \sum_{e \text{ out of } S} c(e)$
- Find s–t cut with the minimum capacity: this problem can be solved optimally in polynomial time by using **flow techniques**

Flows

- An s - t flow is a function that satisfies
 - For each $e \in E$ $0 \leq f(e) \leq c(e)$ [capacity]
 - For each $v \in V - \{s, t\}$:
 $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]
- The value of a flow f is:
 $v(f) = \sum_{e \text{ out of } s} f(e)$

Max flow problem

- Find **s-t** flow of maximum value

Flows and cuts

- **Flow value lemma:** Let f be any flow and let (S, T) be any $s-t$ cut. Then, the net flow sent across the cut is equal to the amount leaving s

$$\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = v(f)$$

Flows and cuts

- **Weak duality:** Let f be any flow and let (S,T) be any $s-t$ cut. Then the value of the flow is at most the capacity of the cut defined by (S,T) :

$$v(f) \leq \text{cap}(S,T)$$

Certificate of optimality

- Let f be any flow and let (S,T) be any cut. If $v(f) = \text{cap}(S,T)$ then f is a max flow and (S,T) is a min cut.
- The min-cut max-flow problems can be solved optimally in polynomial time!

Setting

- Connected, undirected graph $G=(V,E)$
- Assignment of weights to edges: $w: E \rightarrow \mathbb{R}^+$
- **Cut:** Partition of V into two sets: V' , $V-V'$. The set of edges with one end point in V' and the other in $V-V'$ define the cut
- The removal of the cut disconnects G
- **Cost of a cut:** sum of the weights of the edges that have one of their end point in V' and the other in $V-V'$

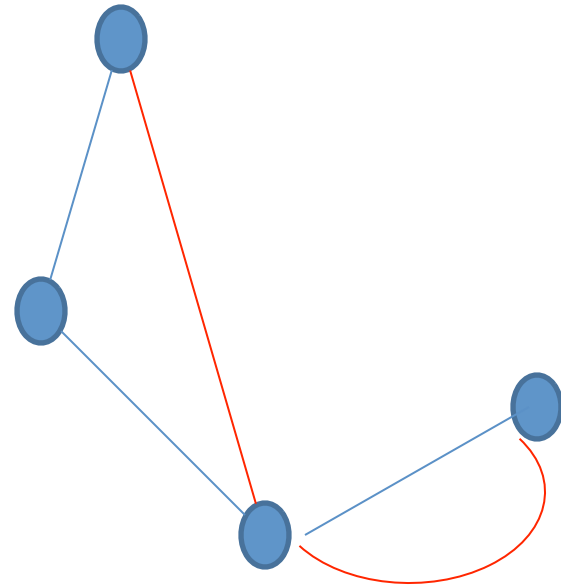
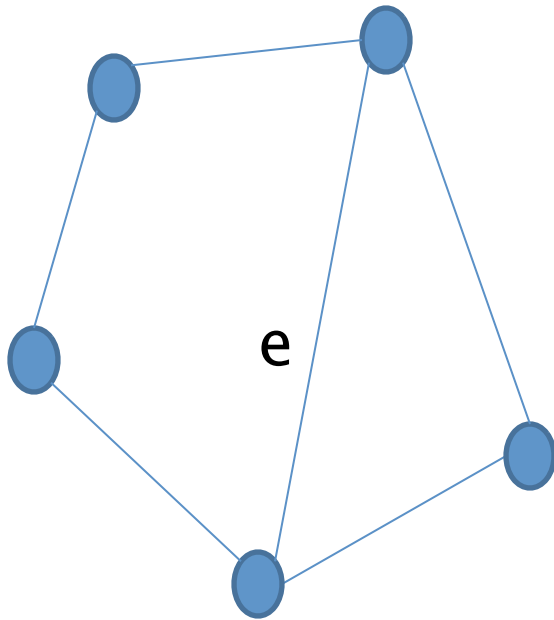
Min cut problem

- Can we solve the min-cut problem using an algorithm for s-t cut?

Randomized min-cut algorithm

- **Repeat** : pick an edge uniformly at random and merge the two vertices at its end-points
 - If as a result there are several edges between some pairs of (newly-formed) vertices retain them all
 - Edges between vertices that are merged are removed (**no self-loops**)
- **Until** only **two** vertices remain
- The set of edges between these two vertices is a cut in **G** and is output as a candidate min-cut

Example of **contraction**



Observations on the algorithm

- Every cut in the graph at any intermediate stage is a cut in the original graph

Analysis of the algorithm

- **C** the min-cut of size $k \rightarrow$ **G** has at least $kn/2$ edges
 - Why?
- E_i : the event of not picking an edge of **C** at the i -th step for $1 \leq i \leq n-2$
- **Step 1:**
 - Probability that the edge randomly chosen is in **C** is at most $2k/(kn)=2/n \rightarrow \Pr(E_1) \geq 1-2/n$
- **Step 2:**
 - If E_1 occurs, then there are at least $k(n-1)/2$ edges remaining
 - The probability of picking one from **C** is at most $2/(n-1) \rightarrow \Pr(E_2|E_1) = 1 - 2/(n-1)$
- **Step i:**
 - Number of remaining vertices: $n-i+1$
 - Number of remaining edges: $k(n-i+1)/2$ (since we never picked an edge from the cut)
 - $\Pr(E_i|\prod_{j=1\dots i-1} E_j) \geq 1 - 2/(n-i+1)$
 - Probability that no edge in **C** is ever picked: $\Pr(\prod_{i=1\dots n-2} E_i) \geq \prod_{i=1\dots n-2} (1-2/(n-i+1))=2/(n^2-n)$
- The probability of discovering a particular min-cut is larger than $2/n^2$
- Repeat the above algorithm $n^2/2$ times. The probability that a min-cut is not **found** is $(1-2/n^2)^{n^2/2} < 1/e$

Multiway cut (analogue of s-t cut)

- **Problem:** Given a set of terminals $S = \{s_1, \dots, s_k\}$ subset of V , a multiway cut is a set of edges whose removal disconnects the terminals from each other. The multiway cut problem asks for the minimum weight such set.
- The multiway cut problem is NP-hard (for $k > 2$)

Algorithm for multiway cut

- For each $i=1,\dots,k$, compute the minimum weight **isolating cut** for s_i , say C_i
- Discard the heaviest of these cuts and output the union of the rest, say C
- **Isolating cut** for s_i : The set of edges whose removal disconnects s_i from the rest of the terminals
- How can we find a minimum-weight isolating cut?
 - Can we do it with a single s-t cut computation?

Approximation result

- The previous algorithm achieves an approximation guarantee of $2 - 2/k$
- **Proof**

Minimum k -cut

- A set of edges whose removal leaves k connected components is called a k -cut. The minimum k -cut problem asks for a **minimum-weight** k -cut
- Recursively compute cuts in G (and the resulting connected components) until there are k components left
- This is a $(2 - 2/k)$ -approximation algorithm

Minimum k -cut algorithm

- Compute the Gomory-Hu tree T for G
- Output the union of the lightest $k-1$ cuts of the $n-1$ cuts associated with edges of T in G ; let C be this union
- The above algorithm is a $(2-2/k)$ -approximation algorithm

Gomory-Hu Tree

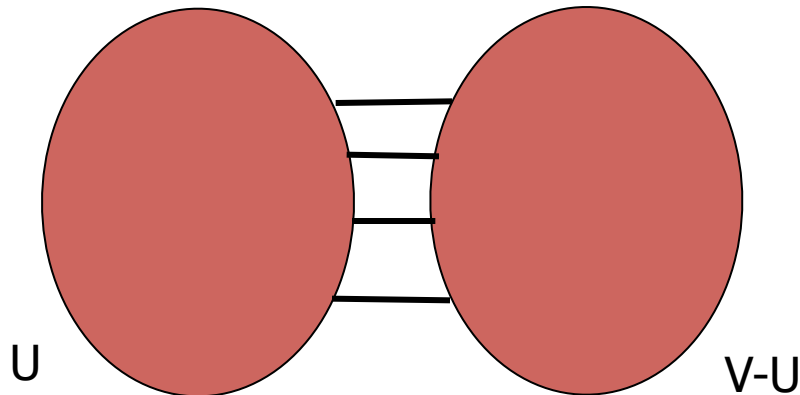
- T is a tree with vertex set V
- The edges of T need not be in E
- Let e be an edge in T ; its removal from T creates two connected components with vertex sets (S, S')
- The cut in G defined by partition (S, S') is the cut associated with e in G

Gomory–Hu tree

- Tree T is said to be the Gomory–Hu tree for G if
 - For each pair of vertices u, v in V , the weight of a minimum $u-v$ cut in G is the same as that in T
 - For each edge e in T , $w'(e)$ is the weight of the cut associated with e in G

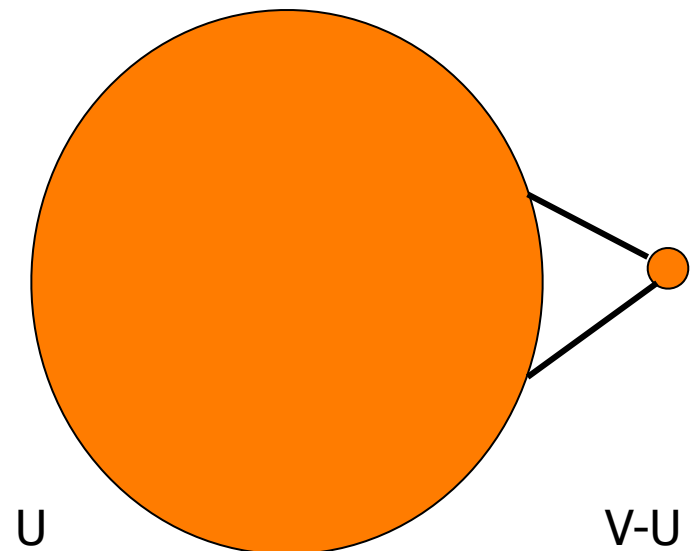
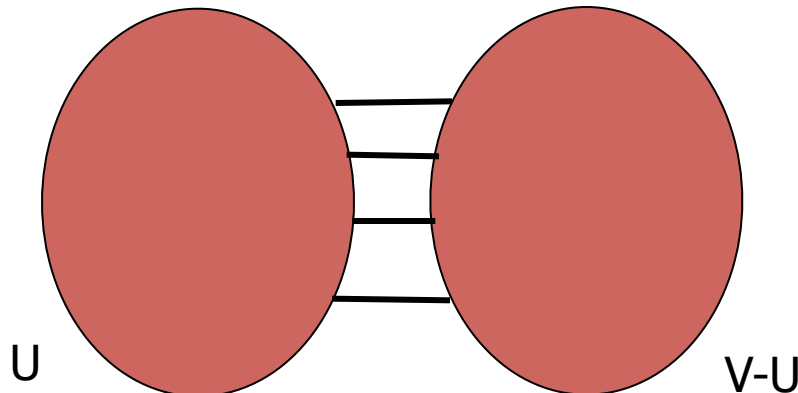
Min-cuts again

- What does it mean that a set of nodes are well or sparsely interconnected?
- **min-cut**: the min number of edges such that when removed cause the graph to become disconnected
 - small min-cut implies sparse connectivity
 - $\min_U E(U, V \setminus U) = \sum_{i \in U} \sum_{j \in V \setminus U} A[i, j]$



Measuring connectivity

- What does it mean that a set of nodes are well interconnected?
- **min-cut**: the min number of edges such that when removed cause the graph to become disconnected
 - not always a good idea!



Graph expansion

- Normalize the cut by the size of the smallest component

- **Cut ratio:**
$$\alpha = \frac{E(U, V \setminus U)}{\min\{|U|, |V \setminus U|\}}$$

- **Graph expansion:**

$$\alpha(G) = \min_U \frac{E(U, V \setminus U)}{\min\{|U|, |V \setminus U|\}}$$

- We will now see how the graph expansion relates to the eigenvalue of the adjacency matrix **A**

Spectral analysis

- The Laplacian matrix $L = D - A$ where
 - A = the adjacency matrix
 - $D = \text{diag}(d_1, d_2, \dots, d_n)$
 - d_i = degree of node i
- Therefore
 - $L(i, i) = d_i$
 - $L(i, j) = -1$, if there is an edge (i, j)

Laplacian Matrix properties

- The matrix L is **symmetric** and **positive semi-definite**
 - all eigenvalues of L are positive
- The matrix L has 0 as an eigenvalue, and corresponding eigenvector $w_1 = (1, 1, \dots, 1)$
 - $\lambda_1 = 0$ is the smallest eigenvalue

The second smallest eigenvalue

- The second smallest eigenvalue (also known as **Fiedler value**) λ_2 satisfies

$$\lambda_2 = \min_{\|x\|=1, x \perp w_1} x^T L x$$

- The vector that minimizes λ_2 is called the **Fiedler vector**. It minimizes

$$\lambda_2 = \min_{x \neq 0} \frac{\sum_{(i,j) \in E} (x_i - x_j)^2}{\sum_i x_i^2} \quad \text{where} \quad \sum_i x_i = 0$$

Spectral ordering

- The values of \mathbf{x} minimize

$$\min_{\mathbf{x} \neq 0} \frac{\sum_{(i,j) \in E} (x_i - x_j)^2}{\sum_i x_i^2} \quad \sum_i x_i = 0$$

- For weighted matrices

$$\min_{\mathbf{x} \neq 0} \frac{\sum_{(i,j)} A[i,j] (x_i - x_j)^2}{\sum_i x_i^2} \quad \sum_i x_i = 0$$

- The ordering according to the x_i values will group similar (connected) nodes together
- Physical interpretation: The stable state of springs placed on the edges of the graph

Spectral partition

- Partition the nodes according to the ordering induced by the Fiedler vector
- If $\mathbf{u} = (u_1, u_2, \dots, u_n)$ is the Fiedler vector, then split nodes according to a value s
 - **bisection**: s is the median value in \mathbf{u}
 - **ratio cut**: s is the value that minimizes α
 - **sign**: separate positive and negative values ($s=0$)
 - **gap**: separate according to the largest gap in the values of \mathbf{u}
- This works well (provably for special cases)

Fielder Value

- The value λ_2 is a good approximation of the graph expansion

$$\frac{\alpha(G)^2}{2d} \leq \lambda_2 \leq 2\alpha(G) \quad d = \text{maximum degree}$$

$$\frac{\lambda_2}{2} \leq \alpha(G) \leq \sqrt{\lambda_2(2d - \lambda_2)}$$

- If the max degree d is bounded we obtain a good approximation of the minimum expansion cut

Conductance

- The expansion does not capture the inter-cluster similarity well
 - The nodes with high degree are more important
- **Graph Conductance**

$$\phi(G) = \min_U \frac{E(U, V \setminus U)}{\min\{d(U), d(V - U)\}}$$

- weighted degrees of nodes in U

$$d(U) = \sum_{i \in U} \sum_{j \in U} A[i, j]$$

Conductance and random walks

- Consider the normalized stochastic matrix $M = D^{-1}A$
- The conductance of the Markov Chain M is

$$\phi(M) = \min_U \frac{\sum_{i \in U} \sum_{j \notin U} \pi(i) M[i, j]}{\min\{\pi(U), \pi(V \setminus U)\}}$$

- the probability that the random walk escapes set U
- The conductance of the graph is the same as that of the Markov Chain, $\varphi(G) = \varphi(M)$
- Conductance φ is related to the second eigenvalue of the matrix M

$$\frac{\phi^2}{8} \leq 1 - \mu_2 \leq \phi$$

Interpretation of conductance

- Low conductance means that there is some **bottleneck** in the graph
 - a subset of nodes not well connected with the rest of the graph.
- High conductance means that the graph is well connected

Clustering Conductance

- The conductance of a **clustering** is defined as the maximum conductance over all **clusters** in the **clustering**.
- Minimizing the conductance of clustering seems like a natural choice

A spectral algorithm

- Create matrix $M = D^{-1}A$
 - Find the second largest eigenvector v
 - Find the best ratio-cut (minimum conductance cut) with respect to v
 - Recurse on the pieces induced by the cut.
-
- The algorithm has provable guarantees

A divide and merge methodology

- **Divide** phase:
 - Recursively partition the input into two pieces until singletons are produced
 - output: a tree hierarchy
- **Merge** phase:
 - use dynamic programming to merge the leafs in order to produce a tree-respecting flat clustering

Merge phase or dynamic-programming on trees

- The **merge** phase finds the optimal clustering in the tree **T** produced by the **divide** phase
- **k**-means objective with cluster centers C_1, \dots, C_k :

$$F(\{C_1, \dots, C_k\}) = \sum_i \sum_{u \in C_i} d(u, c_i)^2$$

Dynamic programming on trees

- $OPT(C,i)$: optimal clustering for C using i clusters
- C_l, C_r the left and the right children of node C
- Dynamic-programming recurrence

$$OPT(C,i) = \begin{cases} C, & \text{when } i = 1 \\ \arg \min_{1 \leq j \leq i} F(OPT(C_l, j) \cup OPT(C_r, i - j)), & \text{otherwise} \end{cases}$$