University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 1: Introduction to expanders

October 5, 2021

Instructor: Thatchaphol Saranurak

Scribe: Aditya Anand

1 Expanders: an introduction

Expanders are central objects in TCS and Math with a lot of applications. In this course, we will see how expanders are so powerful in the area of efficient graph algorithms. Expanders find a lot of applications in graph algorithms - many problems can be solved faster on expanders. We will see *expander decomposition* - the result that allows us to think of any graph as a collection of expanders connected by a small fraction of edges. This will allow us to transfer our study of expanders to general graphs - this nice structure can often be helpful in designing algorithms.

There are other areas where expanders appear a lot: coding theory, pseudo-randomness, PCP (probabilistic checkable proofs). Before we look at a formal definition of expanders, let us try to understand what an expander is "morally" - what are the properties, and what do expander graphs look like.

2 Expanders - sparse but well-connected!

Very simply put, an expander is a **sparse complete graph**! This sounds contradictory - but let us try to understand the context of this statement. What we mean to say is that expanders behave like complete graphs with respect to many properties, but they can be really sparse. Here are some views of expanders: each view captures "connectivity" in its own sense.

- **Cut view**: Robust against deletions
- **Flow view**: Allow all-to-all routing
- **Probabilistic view**: Random walks mix rapidly
- **Algebraic view**: Large spectral gap

Each of the above is useful, but it turns out that they are equivalent! They all characterize the same thing, expanders. In the first half of the course, we will study each of these views and understand how they are all equivalent in charactetrizing expanders. Along the way we will learn a lot of concepts like the flow-cut gap, metric embeddings, Laplacian of graphs, spectral gaps, random

Table 1: Characteristics of a *good* algorithm

Correct					
Fast					
Deterministic					
Parallel					
Distributed					
Dynamic					
Low space					
Simple to implement					
Easy to understand					
What else ???					

walks, etc. Some of the algorithmic applications which we will look at along the way include Multi-commodity flow, multicut, sparsest cut, vertex and edge connectivity. In the second half of the course, we will try to develop *good* algorithms based on expanders. For instance, we will look at graph sparsification from cut, flow and spectral perspectives. We will develop fast near-linear time algorithms for solving many graph problems based on our study of expanders using the key tool of *expander decomposition*, which says that every graph looks like a collection of expanders, with a small fraction of edges between the expanders.

What is a *good* algorithm though? What is the goal of an algorithm designer? Table 1 tries to list some of the properties that a *good* algorithm might hope to achieve. Discuss graph algorithms what you have learned before and try to rate them in terms of the parameters in Table 1.

We will look at some state of the art algorithms for solving fundamental graph problems in the second half of the course. The tentative goal in the second half of the course will include:

- $(1+\epsilon)$-approximate max flow in **undirected** graphs in near-linear time. [All exact max flow algorithms are still much slower.]
- $(1+\epsilon)$-approximate shortest path in **undirected** graphs in near-linear work, polylog depth. [Dijkstra's is not parallel]
- other things along the way... sparsifiers, oblivious routing, dynamic algorithms, etc.

3 Background

I will assume some background knowledge. You can always ask question but I might sometimes tell you where you can look up about these topics instead.

- Basic algorithms and data structures (sorting, binary search trees, graph searching, shortest path, max flow, etc).
- Basic graph theory

- Basic probability and concentration bounds (Markov, Chernoff)
- Linear programming and duality

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 2: High Conductance = Robust Against Deletions

September 2, 2021

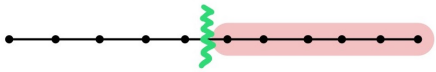
Instructor: Thatchaphol Saranurak

Scribe: Jingyi Gao

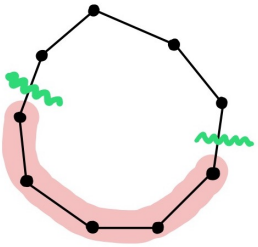
1 Motivate the Definition: Robust Against Deletions

Given a connected graph G, we could describe G is well-connected in the following sense: when we delete d edges in G, the size of the largest component can be reduced by at most $O(d)$ edges. In other words, delete a small set of edges only disconnect small parts of graph.
More formally, let $D \subseteq E$ be a set of d edges. Let $G' = G \setminus D$. Let C be the union of all small components in G'. Then, C contains at most $O(d)$ edges.
Some non-examples are as follows:

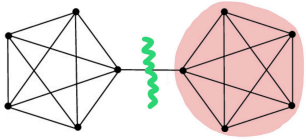
1. paths, where we can delete one edge and disconnect a large chain of vertices,



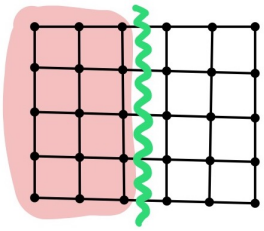
2. cycles, which is similar to a path but we can delete two edges and disconnect a large portion of the cycle,



3. dumbbells, where we can delete one edge and disconnect a large portion of the graph,

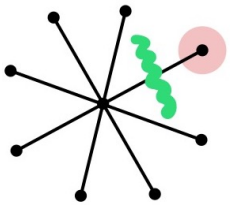


4. grids, where we delete $O(\sqrt{n})$ edges and disconnect $O(n)$ vertices.

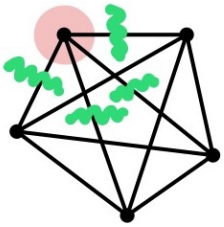


On the other hand, some examples are as follows:

1. stars, where deleting one edge always corresponds to disconnect one vertex from the graph,



2. complete graph, where disconnecting a set A of vertices requires deleting all edges that are connected to vertices outside of A.



An **expander** should satisfies the property above, namely be robust against edge deletions. This motivates the definition of conductance.

2 Formal definition of conductance

We first formalize some notations and the definition of volume

Notation 2.1 (Edges between A and B)**.** Let $G = (V, E)$ be a undirected unweighted graph. $\forall A, B \subseteq V$, $E_G(A, B) = \{(u, v) \in E \mid u \in A, b \in B\}$.

Notation 2.2 (Cut edges)**.** Let $G = (V, E)$ be a undirected unweighted graph. $\forall S \subseteq V$, denote the set of cut edges as $\partial_G S = E_G(S, V \setminus S)$.

Definition 2.3 (Volume)**.** Let $G = (V, E)$ be a undirected unweighted graph. For any $S \subseteq V$, the volume of S is $\mathrm{vol}_G(S) = \sum_{u \in S} \deg(u)$.

Observation 2.4. *Given the definition of volume, some observations are listed below:*

1. $|E(S,V)| \leq \text{vol}(S) \leq 2|E(S,V)|$, *namely the volume counts the number of edges incident to S up to a factor of 2. This is because there are two types of edges that will be counted towards* $\text{vol}(S)$*: (a) edges with two endpoints within S and (b) edges with only one endpoint within S. Edges of type (a) will be counted twice and edges of type (b) will be counted once. Hence,* $\text{vol}(S)$ *is bounded between the number of edges attached to S and twice of that number.*

2. $\text{vol}(S) + \text{vol}(V \setminus S) = \text{vol}(V) = 2|E|$.

Definition 2.5 (Conductance)**.** Let $G = (V, E)$ be a undirected unweighted graph. **Conductance of set/cut** is defined as

$$\Phi_G(S) = \frac{|\partial S|}{\min\{\text{vol}(S), \text{vol}(V \setminus S)\}}$$

Conductance of graph G is defined as

$$\Phi(G) = \min_{S: 0 < vol(S) < vol(V)} \Phi_G(S).$$

Intuitively, conductance describes overall the graph is sparse or not. The smaller the conductance, the sparser the graph.

Remark 2.6. Computing conductance is NP-hard.

Observation 2.7. 1. $\forall S \subset V, \Phi_G(S) \in [0,1]$. *So* $\Phi_G \in [0,1]$.

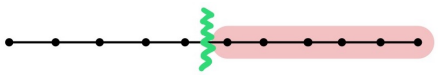
2. $\Phi(G) = 0 \iff$ *G is not connected (with the assumption that each vertex in the graph has a self loop.)*

We say

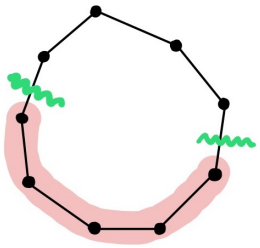
- S is a $\phi-$**sparse cut** $\iff \Phi_G(S) < \phi$ (i.e. its conductance is smaller than ϕ)
- G is a $\phi-$**expander** $\iff \Phi(G) \geq \phi$ (i.e. G has conductance at least ϕ)
- namely, G is $\phi-$expander $\iff$ G has no $\phi-$sparse cut
- larger $\phi \iff$ more well-connected.

Exercise 2.8. To internalize the definition, we look at the following examples:

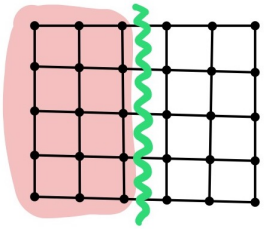
- paths, as mentioned at the beginning, if we pick the right subhalf (the red portion) as S, then $\text{vol}(S) = \text{vol}(V \setminus S) = O(n)$, and $|\partial S| = 1$. Hence $\Phi(G) = O\left(\frac{1}{n}\right)$.



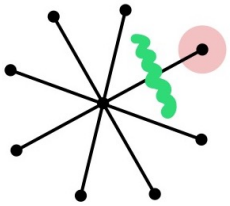
- cycles, which is similar to a path. If we pick the red portion as S, then $\text{vol}(S) = \text{vol}(V \setminus S) = O(n)$, and $|\partial S| = 2$. Therefore $\Phi(G) = O\left(\frac{1}{n}\right)$.



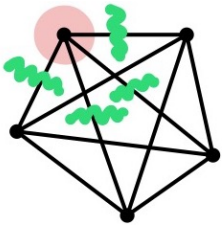
- grids, where we delete $O(\sqrt{n})$ edges and disconnect $O(n)$ vertices. Therefore $\Phi(G) = O\left(\frac{1}{\sqrt{n}}\right)$.



- stars, where disconnect one vertex from the graph always corresponds to delete the edge that attached to that vertex, whence $\Phi(G) = 1$.



- complete graph, where disconnecting a set A of vertices requires deleting all edges that are connected to vertices outside of A, whence $\Phi(G) = \Omega(1)$.



2.1 High Conductance = Robust Against Edge Deletions

We will see how the definition conductance captures the notion of "robustness against edge deletions."

Theorem 2.9. *A connected graph $G = (V, E)$ has conductance at least $\Omega(\phi) \iff \forall D \subseteq E$, the total volume of small components $G \setminus D$ is at most $O\left(\frac{|D|}{\phi}\right)$, where we say a connected component C in $G \setminus D$ is small if $\mathrm{vol}_G(C) \leq \frac{\mathrm{vol}_G(V)}{2}$.*

Proof of Theorem. " $\Longleftarrow$ " we prove the contrapositive of the statement, namely: If $\Phi(G) < \phi$, then $\exists D \subseteq E$, s.t. the total volume of small components of $G \setminus D$ is greater than $\frac{|D|}{\phi}$.
According to the definition of conductance, $\exists S \subseteq V$, s.t. $\frac{|\partial S|}{\mathrm{vol}(S)} < \phi$, i.e. $\frac{|\partial S|}{\phi} < \mathrm{vol}(S) \leq \frac{\mathrm{vol}(V)}{2}$ and the last inequality is a consequence of $\mathrm{vol}(S)$ is the minimum between $\mathrm{vol}(S)$ and $\mathrm{vol}(V \setminus S)$.
We simply pick $D = \partial S$. Clearly S is a small component of $G \setminus D$ and $\mathrm{vol}(S) > \frac{|\partial S|}{\phi} = \frac{|D|}{\phi}$.
" $\Longrightarrow$ " we again prove the contrapositive, namely: If $\exists D$ where the total volume of small components of $G \setminus D$ is greater than $|D|/\phi$, then $\Phi(G) < 3\phi$.
We break into cases:

Case 1:

The normal situation is that after we delete a batch of edges, there will be a big component in the graph. Say $\exists C$ which is not a small component of $G \setminus D$, i.e. $\mathrm{vol}(C) > \frac{\mathrm{vol}(V)}{2}$.
Let S be the union of all small components, we have $\frac{|D|}{\phi} < \mathrm{vol}(S) < \mathrm{vol}(V) - \mathrm{vol}(C) < \frac{\mathrm{vol}(V)}{2}$. We know that $\partial S \subseteq D$ since when we delete D from G, S gets disconnected. Then,

$$\Phi(G) \leq \Phi_G(S) = \frac{|\partial S|}{\min\{\mathrm{vol}(S), \mathrm{vol}(V \setminus S)\}} \leq \frac{|D|}{\mathrm{vol}(S)} < \phi.$$

Case 2:

When all connected components in $G \setminus D$ are small components of G, this is harder to tackle.

Claim 2.10. *There exists S a union of small components in $G \setminus D$ s.t. $\frac{\mathrm{vol}(V)}{3} \leq \mathrm{vol}(S) \leq \frac{2\mathrm{vol}(V)}{3}$.*

Proof of Claim. Prove by contradiction. Denote all small components in $G \setminus D$ as $S_1, \cdots, S_p$. Since all components in $G \setminus D$ are small components, $\mathrm{vol}(S_1 \cup \cdots \cup S_p) = \mathrm{vol}(V)$. If for all unions S of small components of $G \setminus D$, either $\mathrm{vol}(S) < \frac{\mathrm{vol}(V)}{3}$ or $\mathrm{vol}(S) > \frac{2\mathrm{vol}(V)}{3}$. Let S be union of some S_is such that $\mathrm{vol}(S) \leq \frac{\mathrm{vol}(V)}{3}$, and $\forall S'$ being union of some S_is such that $\mathrm{vol}(S') \leq \frac{\mathrm{vol}(V)}{3}, \mathrm{vol}(S') \leq \mathrm{vol}(S)$. Then for all j s.t. $S_j \notin S$, $\mathrm{vol}(S_j) > \frac{\mathrm{vol}(V)}{3}$ in order that $\mathrm{vol}(S \cup S_j) > \frac{2\mathrm{vol}(V)}{3}$ as assumption. However, $\frac{\mathrm{vol}(V)}{2} \geq \mathrm{vol}(S_j) > \frac{\mathrm{vol}(V)}{3}$, which contradict with the assumption. □

Since $\partial S \subseteq D, |D| \leq \phi \mathrm{vol}(V)$,

$$\Phi(G) \leq \Phi_G(S) = \frac{|\partial S|}{\min\{\mathrm{vol}(S), \mathrm{vol}(V \setminus S)\}} \leq \frac{\phi \mathrm{vol}(V)}{\mathrm{vol}(V)/3} < 3\phi$$

□

2.2 Application: Checking Connectivity under Failures in Sub-linear time

- Let $G = (V, E)$ be an undirected graph with n vertices and m edges.
- We can preprocess G in $O(m)$. So that, given any $s, t \in V$, return if s and t are connected in $O(1)$ time.
  - How: using BFS to compute all the connected components in linear time.
- What if there are failures? Consider the following dynamic settings
  - Imagine a computer/road network.
  - Some edges might fail because of ... earthquake maybe.
  - We want to know if s and t are still connected, without recomputing from scratch.

Theorem 2.11. *Suppose that $G = (V, E)$ is a ϕ-expander. Without any preprocessing on G, for any $D \subseteq E$, we can check if $s, t \in V$ are connected in $G \setminus D$ in $O\left(\frac{|D|}{\phi}\right)$ time.*

Proof of Theorem. The proof follows from Theorem 2.9. WLOG, assume that $|D| \leq \frac{\phi \text{vol}(V)}{100}$, otherwise it will just trivially take $O(m)$ time which is the same as conducting BFS for the whole graph. Notice that there is a unique large component C_{large} in $G \setminus D$. Then, consider the algorithm below:

Algorithm:

- Start a BFS from s in the graph $G \setminus D$. There are two cases:
  - Once we have explored more than $10|D|/\phi$ volume, then just stop and conclude that s is in C_{large}.
  - Otherwise, we identify that is s is in a component C_s where $\text{vol}_G(C_s) < 10|D|/\phi$.
  - This takes $O(|D|/\phi)$ time, because BFS takes time proportional to the volume explored.
- Do the same for t.

In this way, we can identify which connected components containing s and t respectively in $O(|D|/\phi)$ time. □

2.3 Basic Property: Expanders have Small Diameter

Definition 2.12 (Diameter)**.** The *diameter* of a graph $G = (V, E)$ is $\max_{u,v \in V} \text{dist}(u, v)$.

Proposition 2.13. *The diameter of a ϕ-expander with n vertices and m edges is $O(\log(m)/\phi)$.*

Proof of Proposition. The proof uses ball-growing argument, which is also a useful tool for other proofs in the lecture.
Let $B(s, r) = \{u \mid \text{dist}(s, u) \leq r\}$ be a ball of radius r around s, r_s be the threshold radius such that $\text{vol}(B(s, r_s - 1)) \leq \text{vol}(V)/2$ yet $\text{vol}(B(s, r_s)) > \text{vol}(V)/2$. Let r_t be defined similarly for t. Notice that $B(s, r_s) \cap B(t, r_t) \neq \emptyset$ since both balls occupied more than half of the edges in G. Hence $\text{dist}(s, t) \leq r_s + r_t$.
We will show that $r_s, r_t \leq O(\log(m)/\phi)$.

Since all cut edges in $\partial(B(s,r-1))$ are at the boundry of $B(s,r-1)$, all paths starting from s within $B(s,r-1)$ given access to set $\partial(B(s,r-1))$ can have increment in length at most by 1, and therefore the length of all path in $B(s,r-1)\cup\partial(B(s,r-1))$ have length $\leq r$. Hence,

$$\mathrm{vol}(B(s,r)) \geq \mathrm{vol}(B(s,r-1)) + |\partial B(s,r-1)|.$$

According to the definition of G being a $\phi-$expander, $\forall r \leq r_s$

$$\frac{|\partial B(s,r-1)|}{\min\{\mathrm{vol}(B(s,r-1)), \mathrm{vol}(V \setminus B(s,r-1))\}} \geq \phi.$$

Since $\mathrm{vol}(B(s,r-1)) \leq \mathrm{vol}(B(s,r_s-1)) \leq \frac{\mathrm{vol}(V)}{2}$, $\min\{\mathrm{vol}(B(s,r-1)), \mathrm{vol}(V\setminus B(s,r-1))\} = \mathrm{vol}(B(s,r-1))$,

$$|\partial B(s,r-1)| \geq \phi \mathrm{vol}(B(s,r-1)).$$

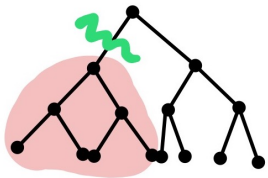
Since $\mathrm{vol}(B(s,0))$ is $\deg s \geq 1$, $\forall r \leq r_s$

$$\mathrm{vol}(B(s,r)) \geq (1+\phi)\mathrm{vol}(B(s,r-1)) \geq (1+\phi)^r.$$

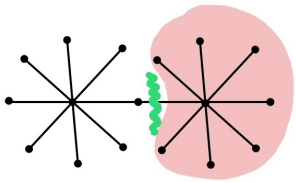
Since $e^{\phi r_s} \approx (1+\phi)^{r_s} \leq \mathrm{vol}(B(s,r_s)) \leq \mathrm{vol}(V) = 2m$, we have $r_s = O(\log(m)/\phi)$. The same holds for r_t. □

Remark 2.14. Conversely, not all low diameter graphs have high conductance. For example,

- Binary trees, where the diameter is $O(\log n)$, but the conductance is $O\left(\frac{1}{n}\right)$,



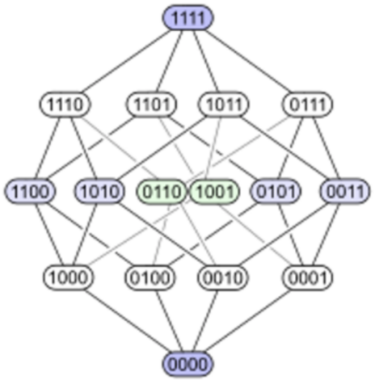
- stars connected by a short path, where the diameter is 4 but the conductance is $O\left(\frac{1}{n}\right)$.



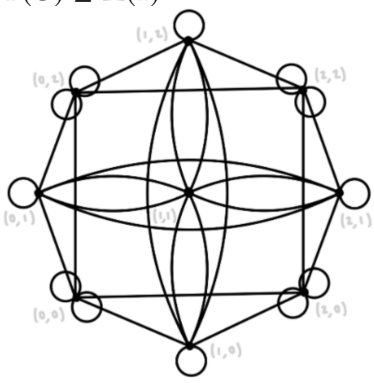
3 Non-trivial Examples: Small Degree Expanders

Intuitively, graphs containing large degree vertices can be more well-connected, whence having higher conductance. For example, the fact that complete graphs and stars have high conductance are not very surprising.
However, the existence of graphs with small maximum degree but still have large conductance is nontrivial. Namely, sparse graphs can still be very well-connected. Some examples are listed as below:

- Deterministic examples:
  - Hypercube:
    - $V = \{0,1\}^{\log n}$ and $E = \{(u,v) \mid u = v \text{ except that one entry}\}$.
    - $\Phi(G) \geq \Theta(1/\log n)$.

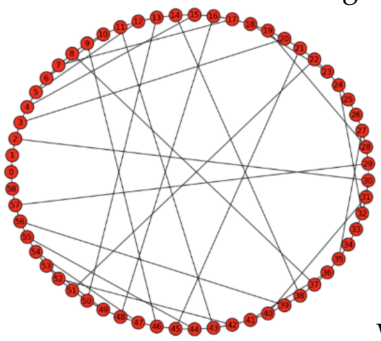


  - Margulis–Gabber–Galil
    - $V = (\mathbb{Z}/n\mathbb{Z}) \times (\mathbb{Z}/n\mathbb{Z})$
    - Each $(x,y) \in V$ has 8 neighbors: $(x \pm 2y, y), (x \pm (2y+1), y), (x, y \pm 2x), (x, y \pm (2x+1))$
    - $\Phi(G) \geq \Omega(1)$



when $n = 3$ [1]

  - Prime expander
    - $V = \mathbb{F}_p = \{0, \ldots, p-1\}$
    - $x \in V$ is connected to 3 neighbors: $x+1$, $x-1$ and x^{-1} (for $x \neq 0$)



when $p = 59$ [2]

[1] https://web.stanford.edu/~styopa/pdfs/expanders.pdf
[2] https://lucatrevisan.github.io/teaching/expanders2016/lecture16.pdf

- Randomized examples:
  - Erdos-Renyi graph $G_{n,p}$ when $p = \Omega(\log(n)/n)$:
    * With high probability, G has maximum degree $O(\log n)$ and$\Phi(G) \geq \Omega(1)$.
    * This means that almost all graphs are actually expanders.
  - Union of random $O(1)$ matchings
    * G has maximum degree $O(1)$
    * $\Phi(G) \geq \Omega(1)$ with high probability[3]

Note 3.1. "Canonical expanders" can be depicted as: well-connected, yet with low degree. (But of course, in general graphs with high conductance can also have various other appearances.)

4 Vertex Expansion

Conductance describes the ratio between #(cut edges) and #(edges disconnected by the cut). We can also talk about the analogue of it for vertices.

Definition 4.1 (Vertex-expansion)**.** Let (L, S, R) be a vertex cut, i.e. L, S, R partition V and $E(L, R) = \emptyset$. **Vertex-expansion of a cut** (L, S, R) is

$$h_G(S) = \frac{|S|}{\min\{|L \cup S|, |R \cup S|\}} \in [0, 1]$$

Vertex-expansion of a graph G is

$$h(G) = \min_{(L,S,R)} h(L, S, R) \in [0, 1]$$

- We say that
  - S is a ϕ**-vertex-sparse cut** iff $h_G(S) < \phi$
  - G is a ϕ**-vertex-expander** iff $h(G) \geq \phi$

Exercise 4.2. (vertex expansion and conductance are not necessarily the same, but could be under certain conditions) Show that

1. If a graph G has maximum degree $O(1)$, then vertex expansion and conductance has within a constant factor $h(G) = \Theta(\Phi(G))$.

   Proof. First of all, notice that all edge cuts corresponds to a vertex cut by simply letting the endpoints of cut edges to be the vertex cut. Since $\deg G = O(1)$, there is some constant $M, s.t. \deg G \leq M$. Let the vertex cut (L, S, R) be such that $h(G) = h(L, S, R)$. If $\Phi(G) = \frac{|\partial A|}{\text{vol}(A)}$, then $\frac{|\partial A|}{M} \leq |S| \leq 2|\partial A|$, which shows that $|S| = \Theta(|\partial A|)$. Since the degree of G is bounded by a constant, this means that for any subgraphs of G, the total number of edges is also bounded by the number of vertices up to a constant factor, namely $\min\{|L \cup S|, |R \cup S|\} = \Theta(\text{vol}(A))$. Hence, $h(G) = \Theta(\Phi(G))$. □

[3] https://lucatrevisan.github.io/teaching/expanders2016/lecture19.pdf

2. There is a graph G where $\Phi(G) \geq \Omega(1)$ but $h(G) \leq O(1/n)$.

   Proof. A star is an example where we need to delete the same amount of edges in order to disconnect that amount of vertices, whence $\Phi(star) = 1$. On the other hand we only need to delete the central vertex of the star to disconnect all vertices in the graph, therefore $h(star) = \frac{1}{n}$. □

3. There is a graph G where $h(G) \geq \Omega(1)$ but $\Phi(G) \leq O(1/n)$.

   Proof. Consider a graph contains 2 same cliques with a perfect matching between them. To disconnect one vertex from the graph, we need to delete the whole clique that this vertex stays in plus the vertex that matches with it. Therefore, $h(G) \approx \frac{1}{2} = \Omega(1)$. On the other hand, deleting the perfect matching let the whole graph break into two parts with same volume, i.e. $\Phi(G) = \frac{n}{n^2} = \Theta(\frac{1}{n})$. □

Exercise 4.3 (Robustness of vertex-expander against vertex deletion)**.** Suppose that $G = (V, E)$ is a ϕ-vertex-expander. Without any preprocessing on G, for any $D \subseteq V$, we can check if $s, t \in V$ are connected in $G \setminus D$ in $O((|D|/\phi)^2)$ time.

Proof. The proof should be an analogue of the proof of Theorem 2.11.

Claim 4.4. *$h(G) \geq \Omega(\phi) \iff \forall D \subseteq V$, the total number of vertices of small components of $G \setminus D$ is at most $O\left(\frac{|D|}{\phi}\right)$, where we say a connected components C is small in $G \setminus D$ is the number of vertices in C is less than or equal to $\frac{|V|}{2}$.*

Proof of Claim. Suffices to show the contrapositive. We first show that if $h(G) < \phi \implies \exists D \subseteq V$, s.t. the total number of vertices of small components of $G \setminus D > O\left(\frac{|D|}{\phi}\right)$. Since $h(G) < \phi, \exists (L, S, R)$ a vertex cut of V s.t. $\frac{|S|}{|L \cup S|} < \phi \implies \frac{|S|}{\phi} < |L \cup S| = |L| + |S| \implies O\left(\frac{|S|}{\phi}\right) = \frac{(1-\phi)|S|}{\phi} < |L|$. Let $S = D$, then since $\min\{|L \cup S|, |R \cup S|\} = |L \cup S|, |L| \leq \frac{|V|}{2}$, and hence L is a small component of $G \setminus S$. We next show that if $\exists D \subseteq V$, s.t. the total number of vertices of small components of $G \setminus D > O\left(\frac{|D|}{\phi}\right) \implies h(G) < 3\phi$. We break into cases:

- Case 1. $\exists C \subseteq G \setminus D$ s.t. C is not a small component in $G \setminus D$. Let A denote the union of all small components of $G \setminus D$, then $\frac{|D|}{\phi} < |A| \leq \frac{|V|}{2}$. Then

  $$h_G(D) = \frac{|D|}{\min\{|L \cup D|, |R \cup D|\}} < \frac{|D|}{|A|} < \phi$$

  Hence $h(G) \leq h_G(D) < \phi$.

- Case 2. when all elements in $G \setminus D$ are small components. Make a similar argument as in the claim in the proof of Theorem 2.11, we know that there exists A a union of small components in $G \setminus D$, s.t. $\frac{|V|}{3} \leq |A| \leq \frac{2|V|}{3}$. We also know that $|D| < \phi|V|$. Then

  $$h_G(D) = \frac{|D|}{\min\{|L \cup D|, |R \cup D|\}} \leq \frac{|D|}{|A|} < \frac{\phi|V|}{\frac{|V|}{3}} = 3\phi$$

□

Knowing that for a $\phi-$vertex expander, all small components in $G \setminus D$ will have total size at most $O\left(\frac{|D|}{\phi}\right)$, we did the similar algorithm as before: we start BFS from s in $G \setminus D$, and once we have explored more than $10\frac{|D|}{\phi}$ vertices, stop and conclude that s is in C_{large}. Otherwise, we identify that s is in a small component C_s. This takes $O\left(\left(\frac{|D|}{\phi}\right)^2\right)$ time since the time complexity of BFS is proportional to the number of edges, which is bounded by the square of number of vertices. We do the same for t, whence the total time is $O\left(\left(\frac{|D|}{\phi}\right)^2\right)$. □

Exercise 4.5. (vertex-expander has small diameter) Show that a ϕ-vertex-expander has diameter $O(\log(n)/\phi)$.

Proof. We will use the ball-growing argument again. Let $B(s,r) = \{u | dist(s,u) \leq r\}$. Let r_s be a threshold s.t. $|B(s, r_s - 1)| \leq \frac{|V|}{2}$, yet $|B(s, r_s)| > \frac{|V|}{2}$. Define the analogue for t.
We know that $B(s, r_s) \cap B(t, r_t) \neq \emptyset$ since they all contains more than $\frac{|V|}{2}$ vertices and by PHP there must be at least one vertex falls into both balls. Hence, $dist(s,t) \leq r_s + r_t$.
We next show that $r_s, r_t < O\left(\frac{\log(n)}{\phi}\right)$. Indeed, since $|B(s, r_s)| \geq |B(s, r_s - 1)| + |S_r|$ where S_r is the vertex cut that separate $B(s,r)$ from G. Since G is a $\phi-$vertex expander, and $\forall r \leq r_s - 1, |B(s,r)| \leq \frac{|V|}{2}$, $h_G(S_r) = \frac{|S_r|}{\min\{|L \cup S_r|, |R \cup S_r|\}} = \frac{|S_r|}{|B(s,r)|} \geq \phi$ i.e. $|S_r| \geq \phi |B(s,r)|$. Hence, $|B(s, r_s)| \geq (1+\phi)|B(s, r_s - 1)| \implies n = |V| \geq |B(s, r_s)| \geq (1+\phi)^{r_s} \approx e^{\phi r_s} \implies r_s \leq \frac{\log n}{\phi}$. Similar for r_t. □

5 Edge Expansion with General Demand

- The following notion generalizes conductance.
- Let $\boldsymbol{d} : V \to \mathbb{R}_{\geq 0}$ be a *demand function* of vertices.
- A **$\boldsymbol{d}$-expansion of a cut** S where $0 < \boldsymbol{d}(S) < \boldsymbol{d}(V)$ is

$$\Phi_{G,\boldsymbol{d}}(S) = \frac{\partial S}{\min\{\boldsymbol{d}(S), \boldsymbol{d}(V \setminus S)\}}$$

  where $\boldsymbol{d}(S) = \sum_{u \in S} \boldsymbol{d}(u)$.
- A **$\boldsymbol{d}$-expansion of a graph** G is

$$\Phi(G, \boldsymbol{d}) = \min_{S : 0 < \boldsymbol{d}(S) < \boldsymbol{d}(V)} \Phi_{G,\boldsymbol{d}}(S)$$

- Conductance is the same as $\boldsymbol{d}$-expansion when $\boldsymbol{d}(u) = \deg(u)$ for all u.

Exercise 5.1. (a graph with big expansion and large diameter: expansion and conductance can be different) In the literature, people often consider $\boldsymbol{d}^{unit}$-expansion where $\boldsymbol{d}^{unit}(u) = 1$ for all $u \in V$ and call this "*expansion*". This measures the ratio between #(cut edges) and #(vertices disconnected by the cut). Show that even if $\Phi(G, \boldsymbol{d}^{unit}) \geq \Omega(1)$, then diameter of G can be large.

Remark 5.2. This is unrelated to the solution, but notice that this value $\Phi(G, \boldsymbol{d}^{unit})$ is not bounded by 1 but can be as big as the max degree. For example, in a complete graph, $\Phi(G, \boldsymbol{d}^{unit}) = \frac{|S| \times |V \setminus S|}{\min\{|S|, |V \setminus S|\}} = \Omega(n)$.

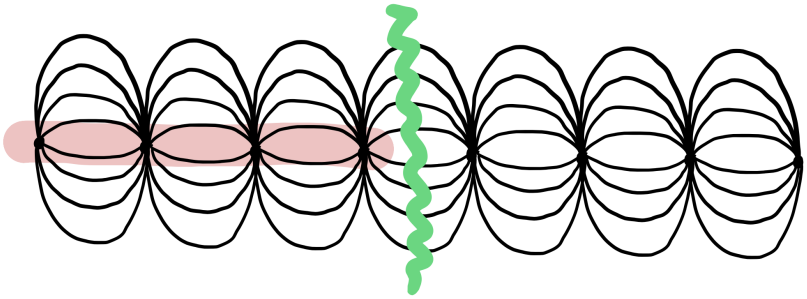


Figure 1: G_1

Solution. Consider the graph above, where $\Phi(G_1, \boldsymbol{d}^{unit}) = 2 \geq \Omega(1)$, yet the diameter is 7 which is pretty large.
Generalize to simple graph, consider changing one vertex into layer of n vertices, and connecting each layers with the layer next to it using a complete bipartite graph as below

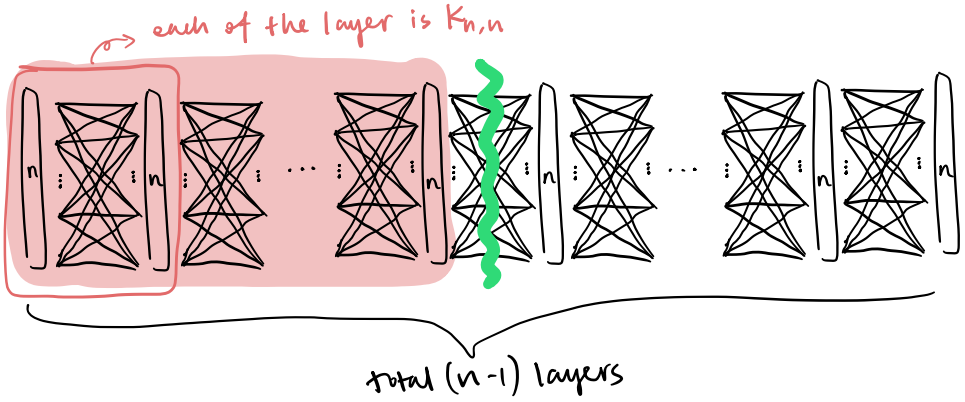


Figure 2: G_2

In this case, $\Phi(G_2, \boldsymbol{d}^{unit}) = \frac{n^2}{n^2/2} = 2 \geq \Omega(1)$, but the diameter is $O(n)$. □

Exercise 5.3. (a graph with big d-expansion but small conductance) Let $T \subseteq V$ be a subset called terminal. Let $\boldsymbol{d}(u) = \deg(u)$ if $u \in T$ otherwise $\boldsymbol{d}(u) = 0$. Give an example of a connected graph G where $\Phi(G, \boldsymbol{d}) = \Omega(1)$ but $\Phi(G)$ is very small.
Hint: Start with a graph $G_0 = (T, E_0)$ where $\Phi(G_0) = \Omega(1)$. Let $G = (V, E)$ be obtained from G_0 by subdividing each edge into a path of length n. Show that $\Phi(G, \boldsymbol{d}) = \Omega(1)$.

Solution. 1. Consider the connected graph G consist of a clique connected with a path, where the clique is T. Then, the conductance of the graph is very small since it's basically the con-

ductance of the path. However, $\Phi(G, \boldsymbol{d})$ is large since it evaluates how well-connected the subset T is, in this case the clique.

2. A more nontrivial example is as suggested in the hint. We first show the conductance of G is small. Indeed, $\Phi(G) \leq O\left(\frac{1}{n}\right)$ since we could pick any random edge in G_0, which turns into a path of length n in G, and for that path in G, we delete two edges that are attached to the original vertices in G_0, and this will make the whole chunk of path get disconnected from the graph.
   We next show that $\Phi(G, d) \geq \Omega(\Phi(G_0))$. Indeed, given a cut $S \subseteq V$ in G, we can construct a corresponding cut S_0 of G_0 by contracting G back to G_0 and take the contraction S_0 of S in G_0. Hence, $|\partial_G(S)| \geq |\partial_{G_0}(S_0)|$. Since d is defined to be $\deg(u)$ on T and 0 otherwise, $\min\{\mathrm{vol}_G(S), \mathrm{vol}_G(V \setminus S)\} = \min\{\mathrm{vol}_G(T \cap S), \mathrm{vol}_G(V \setminus S \cap T)\} = \min\{\mathrm{vol}_{G0}(S_0), \mathrm{vol}_{G_0}(T \setminus S_0)\}$. Then

$$\Phi_{(G,\boldsymbol{d})}(S) = \frac{|\partial_G(S)|}{\min\{\mathrm{vol}_G(S), \mathrm{vol}_G(V \setminus S)\}} \geq \frac{|\partial_{G_0}(S_0)|}{\min\{\mathrm{vol}_{G0}(S_0), \mathrm{vol}_{G_0}(T \setminus S_0)\}} = \Phi_{G_0}(S_0) \geq \Omega(1)$$

   Hence $\Phi(G, \boldsymbol{d}) = \Omega(1)$.

□

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 3: Approximating Conductance

September 7, 2021

Instructor: Thatchaphol Saranurak

Scribe: Gary Hoppenworth

1. Overview

Conductance is an important property of graphs that measures how well-connected they are. Expanders are graphs with "large" conductance. (Typically, graphs on n vertices are considered good expanders if they have conductance greater than $1/\text{polylog}\, n$.) Unfortunately, computing conductance exactly is NP-hard in general graphs.

However, there is an efficient $O(\log n)$-approximation of conductance which allows us to check the expansion of a graph (up to some extent). This approximation relies on two ingredients: 1) Bourgain's metric embedding into ℓ_1-metrics and 2) connections between ℓ_1-metrics and cut-metrics. These two tools are used to obtain an approximation via linear programming relaxation. In this lecture, we introduce the necessary background for the above two ingredients and then prove the LP relaxation approximation.

2. Metrics

A metric space (or metric) is a mathematical notion that captures distance. Metric spaces are composed of a set X of points and a distance function d that lets us measure the distance between points.

2.1. Definition. Let X be a set and let d be a function $d : X \times X \mapsto \mathbb{R}_{\geq 0}$. We say that (X, d) is a **metric** if the following hold:

1. $d(x, x) = 0$ for all $x \in X$ [1]
2. $d(x, y) = d(y, x)$ for all $x, y \in X$
3. $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in X$

The function d is called the **distance function**.

[1] In mathematics, this condition is stated as $d(x, y) = 0$ if and only if $x = y$. What we call metric in computer science is usually referred to as semi-metric in math.

2.1. Examples of Metrics

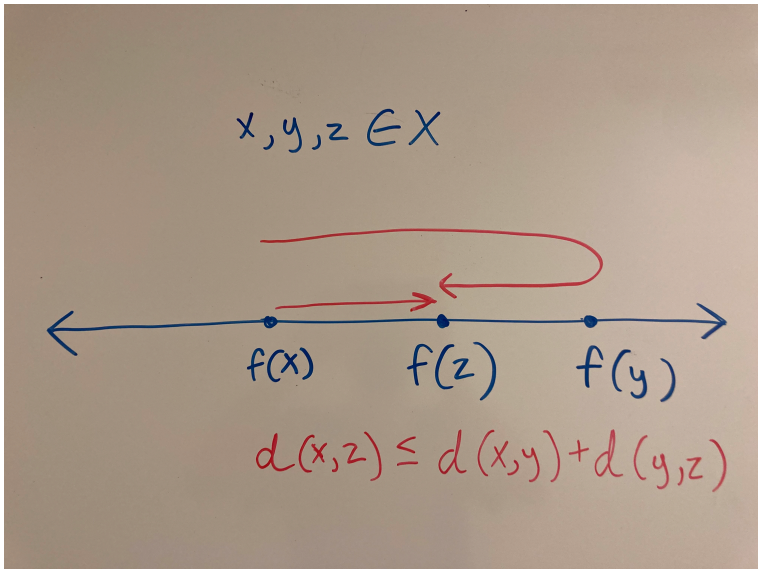
Line Metric: Given a set X and a function $f : X \mapsto \mathbb{R}$, define the distance function as $d(x, y) = |f(x) - f(y)|$.

Proof that this is a metric:

1. $d(x, x) = |f(x) - f(x)| = 0$ for all $x \in X$
2. $d(x, y) = |f(x) - f(y)| = |f(y) - f(x)| = d(y, x)$ for all $x, y \in X$.
3. $d(x, z) = |f(x)-f(z)| = |f(x)-f(y)+f(y)-f(z)| \leq |f(x)-f(y)|+|f(y)-f(z)| = d(x, y)+d(y, z)$ for all $x, y, z \in X$

□

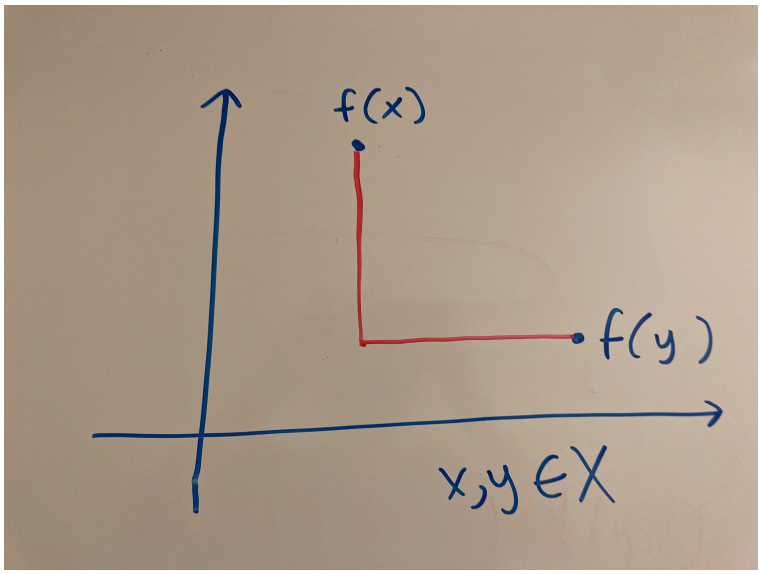
Below is a visualization of a line metric and the proof of statement 3.



ℓ_1-Metric: Given a set X and a function $f : X \mapsto \mathbb{R}^m$, define the distance function as $d(x, y) = \|f(x) - f(y)\|_1 = \sum_i |f_i(x) - f_i(y)|$. (Note when $m = 1$ this is the line metric.)

Proof that this is a metric: The ℓ_1-metric is really a sum of line metrics. This proof is essentially the same as the previous proof. □

Below is a visualization of distances in an ℓ_1-metric.



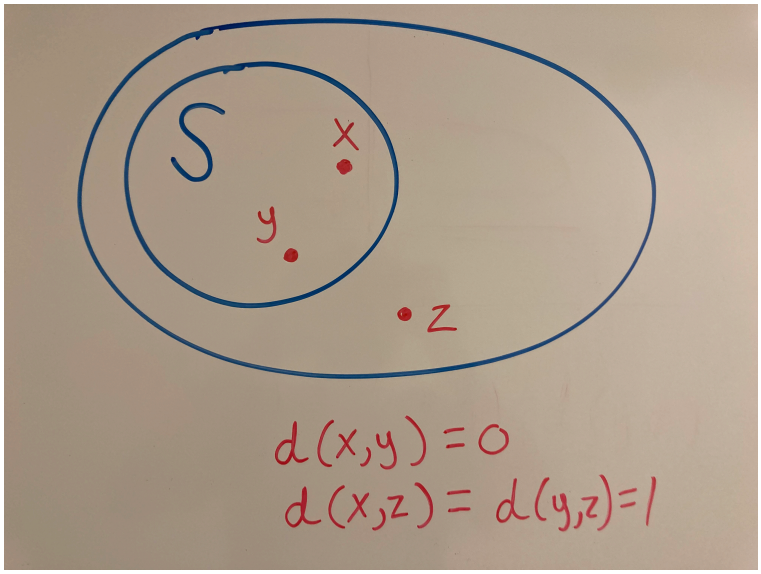
Cut Metric: Given a set X and a set $S \subseteq X$, let $d(x, y) = 0$ if $x, y \in S$ or $x, y \in X \setminus S$. Else, $d(x, y) = 1$.

Proof that this is a metric:

1. $d(x, x) = 0$, since x cannot belong to both S and $X \setminus S$.
2. $d(x, y) = d(y, x)$ since the definition of d does not depend on the order of x and y.
3. If $d(x, z) = 0$, then the inequality immediately holds. Else $d(x, z) = 1$. Assume without loss of generality that $x \in S$ and $z \in X \setminus S$. If $y \in S$, then $d(y, z) = 1$. Otherwise, $y \in X \setminus S$ and $d(x, y) = 1$. Either way, $d(x, y) + d(y, z) \geq 1 = d(x, z)$.

□

Below is a visualization of distances in the cut metric for a set S.



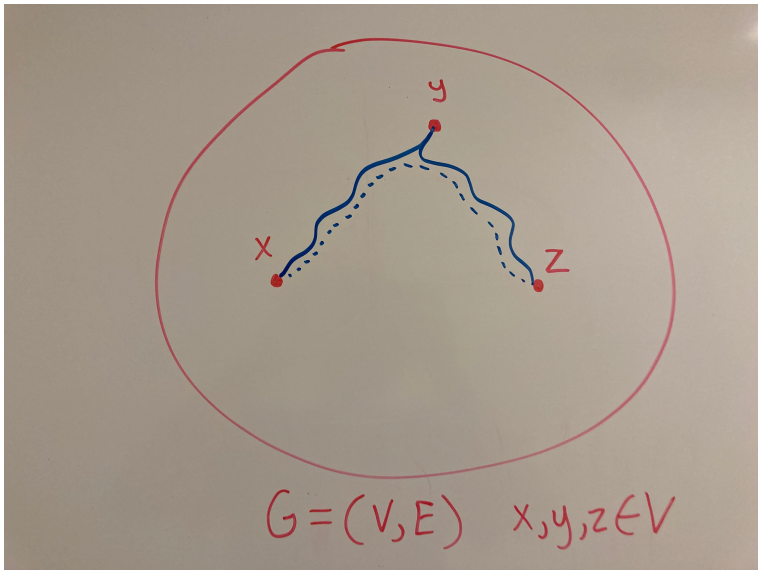
Shortest Path Metric: Given a set X and an undirected graph $G = (X, E)$, define the distance function as $d(x, y) = d_G(x, y)$, where $d_G(x, y)$ is the length of the shortest (x, y)-path in G.

Proof that this is a metric:

1. $d(x, x) = \text{dist}_G(x, x) = 0$. (The shortest path from a vertex to itself has length zero.)
2. $d(x, y) = \text{dist}_G(x, y) = \text{dist}_G(y, x) = d(y, x)$. (A shortest (x, y)-path has the same length as a shortest (y, x)-path in any undirected graph.)
3. $d(x, z) = \text{dist}_G(x, z) \leq \text{dist}_G(x, y) + \text{dist}_G(y, z) = d(x, y) + d(y, z)$. The inequality follows from the fact that a (x, y)-shortest path can be combined with a (y, z)-shortest path to obtain a (x, z)-path.

□

Below is a visualization of the proof of the triangle inequality (statement 3) for undirected graphs.



3. Metric Embeddings

3.1. Motivation

Computing the distance in some metrics is more expensive than in other metrics. This can be seen by comparing the costs of computing distances in the shortest path metric and the ℓ_1-metric.

Consider a shortest path metric of a graph G on n vertices and m edges. Computing $d(x, y) = \text{dist}_G(x, y)$ where $x, y \in V(G)$ takes $\Omega(m)$ time using Dijkstra's algorithm. By precomputing all distances you can answer queries in $O(1)$ time, but this requires $\Omega(n^2)$ space. These computations can be expensive for large graphs.

On the other hand, for an ℓ_1-metric with dimension k, computing $d(x, y) = \|f(x) - f(y)\|$, where $x, y \in \mathbb{R}^k$, takes $O(k)$ time by simply reading x and y and performing $O(k)$ operations. This computation is cheap for small k.

Because we can compute distances in a low dimension ℓ_1-metric quickly, it would be advantageous to represent any metric (X, d) as an ℓ_1-metric with low dimension. This would allow us to measure distance in the (X, d) metric (perhaps within some distortion factor) using cheaper computations. We call this representation an embedding and define it formally in the next section.

3.2. Formal Definition

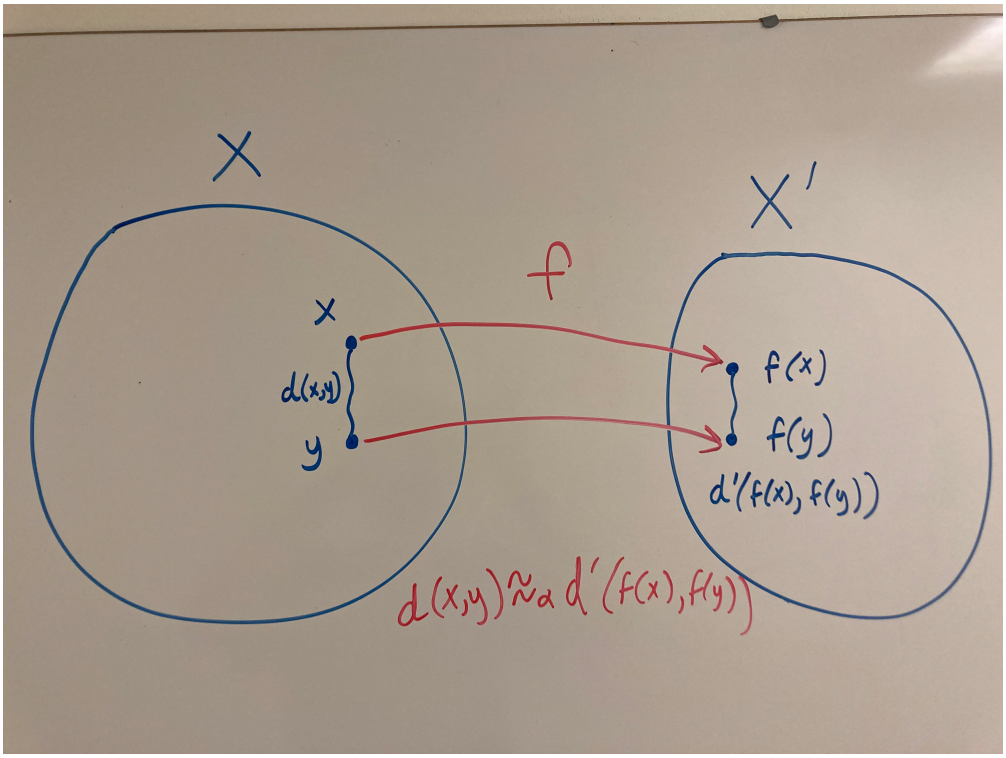
3.1. Definition (Metric Embeddings)**.** Let (X, d) and (X', d') be two metrics. Let $f : X \mapsto X'$ be a mapping such that for any $x, y \in X$

$$d'(f(x), f(y)) \leq d(x, y) \leq \alpha \cdot d'(f(x), f(y))$$

Then f is a **metric embedding**. We say that f **embeds** (X, d) into (X', d') with **distortion** α. This can also be written as

$$(X, d) \overset{\alpha}{\hookrightarrow} (X', d')$$

Below is a visualization of the notion of a metric embedding.



3.2. *Remark.* When $\alpha = 1$, we say that f has **no distortion**, or is **distance-preserving**, or is **isometric**.

3.3. Definition (ℓ_p^k-metrics)**.** We say that a metric (X, d) is a ℓ_p^k-metric ($p \geq 1$) if and only if there is an embedding f that embeds (X, d) into (X', d') with no distortion, and (X', d') is an ℓ_p-metric with dimension k.

3.4. Example. A cut metric (X, d_S) is a line metric (and therefore it is an ℓ_1^1-metric).

Proof: Define the function $1_S : X \mapsto \{0, 1\}$ so that $1_S(x) = 1$ if and only if $x \in S$. Let $d'(x, y) = |1_S(x) - 1_S(y)|$ for all $x, y \in X$. Note that (X, d') is a line metric. It follows from the definition of the cut metric that $d_S(x, y) = |1_S(x) - 1_S(y)| = d'(x, y)$ for all $x, y \in X$. Thus we have an embedding of (X, d_S) to (X, d') with distortion $\alpha = 1$. □

We now describe the two main ingredients that we use to approximate conductance. The first ingredient will show that any metric can be embedded in a ℓ_1^p metric for "small" p with low distortion. The second ingredient will show that any ℓ_1^p metric can be expressed as a conic combination of cut-metrics. Thus cut-metrics are in some sense universal: they capture every metric in some (approximate) sense.

3.3. Ingredient 1: Bourgain's Metric Embedding

A surprising fact is that every n-point metric can be embedded into an ℓ_1-metric of dimension $O(\log^2 n)$ with distortion $O(\log n)$. This is powerful because the ℓ_1-metric we are embedding into has both small dimension and small distortion. The formal statement of this fact is as follows.

3.5. Theorem (Bourgain)**.** *Given an n-point metric (X, d), there is an embedding $f : X \to \mathbb{R}^{O(\log^2 n)}$ such that for any $x, y \in X$,*

$$\|f(x) - f(y)\|_1 \leq d(x, y) \leq O(\log n) \cdot \|f(x) - f(y)\|_1$$

Moreover, given d, in $\tilde{O}(n^2)$ time, the mapping f can be computed correctly with probability at least $1 - 1/n$. If (X, d) is a shortest path metric of an m-edge graph, then the running time is $\tilde{O}(m)$ time. (Here $\tilde{O}(\cdot)$ is used to hide polylog *factors.)*

Note that we may ensure that for all $x \in X$, the i-th coordinate $f_i(x) = d(x, y)$ for some $y \in X$, so the coordinate $f_i(x)$ is not *too* big. The original embedding to an ℓ_1-metric was by Bourgain (1985)[2], but the ℓ_1-metric had more dimensions. The bound of $O(\log^2 n)$ dimensions is due to Linial, London, and Rabinovich (1995)[3]. There are many other examples of embedding metrics into more friendly metrics, such as the Johnson-Lindenstrauss dimension reduction and Bartal's tree embedding. In fact, there are entire courses on metric embeddings[4].

We can use Bourgain's embedding to approximate the shortest path distances in a graph G efficiently. For instance, given a graph G on n vertices, Bourgain's theorem allows us to embed the shortest path metric of G into a $\ell_1^{O(\log^2 n)}$-metric with $O(\log n)$ distortion. Then we can label each vertex of G with $O(\log^2 n)$ coordinates so that given vertices $u, v \in V(G)$, we can obtain a $O(\log n)$-approximation of $\text{dist}_G(u, v)$ in $O(\log^2 n)$ time by reading the labels of u and v and

[2] J. Bourgain. On Lipschitz embeddings of finite metric spaces in Hilbert space. Israel J. of Math. 1985.

[3] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. Combinatorica, 15(2):215–245, 1995. (Preliminary version in 35th FOCS, 1994).

[4] https://home.ttic.edu/ harry/teaching/pdf/lecture1.pdf

performing $O(\log^2 n)$ operations compute the corresponding distance in the $\ell_1^{O(\log^2 n)}$-metric. The total space used is $O(n \log^2 n)$, which is sublinear in the number of edges. However, we will be using Bourgain's embedding in a very different way.

4. Ingredient 2: ℓ_1-Metrics are in the Cut Cone

We saw earlier that a cut metric is a very simple ℓ_1-metric. Now we will see the other direction: every ℓ_1-metric is a combination of cut metrics. To formalize this, we will need some additional notation.

4.1. Observation. *For any n-point metric (X, d), we may think of d as a vector $d \in \mathbb{R}^{\binom{n}{2}}$. Each coordinate in this vector corresponds to a pair of points $x, y \in X$ and has value $d(x, y)$.*

We will refer to the vector $d \in \mathbb{R}^{\binom{n}{2}}$ as an n-point metric, since it encodes everything about (X, d).

4.2. Fact (Closure under addition)**.** *If d_1 and d_2 are n-point metrics, then $d_1 + d_2$ is an n-point metric.*

Proof. We must verify that the three requirements of a metric space hold for the $d_1 + d_2$ metric space, which is defined as $(d_1 + d_2)(x, y) = d_1(x, y) + d_2(x, y)$.

1. $(d_1 + d_2)(x, x) = d_1(x, x) + d_2(x, x) = 0$, where the final equality follows since d_1 and d_2 are metric spaces.
2. $(d_1 + d_2)(x, y) = d_1(x, y) + d_2(x, y) = d_1(y, x) + d_2(y, x) = (d_1 + d_2)(y, x)$, where the final equality follows since d_1 and d_2 are metric spaces.
3. $(d_1 + d_2)(x, z) = d_1(x, z) + d_2(x, z) \leq d_1(x, y) + d_1(y, z) + d_2(x, y) + d_2(y, z) = (d_1 + d_2)(x, y) + (d_1 + d_2)(x, z)$. The inequality follows from the fact that d_1 and d_2 are metric spaces.

□

4.3. Fact (Closure under scalar multiplication)**.** *If d is an n-point metric, then $\alpha \cdot d$ is a metric.*

Proof. We must verify that the three requirements of a metric space hold for the $\alpha \cdot d$ metric space, which is defined as $(\alpha \cdot d)(x, y) = \alpha \cdot d(x, y)$.

$(\alpha \cdot d)(x, y) = \alpha \cdot d(x, y) = 0$, since d is a metric.

$(\alpha \cdot d)(x, y) = \alpha \cdot d(x, y) = \alpha \cdot d(y, x) = (\alpha \cdot d)(y, x)$, since d is a metric.

$(\alpha \cdot d)(x, z) = \alpha \cdot d(x, z) \leq \alpha(d(x, y) + d(y, z)) = (\alpha \cdot d)(x, y) + (\alpha \cdot d)(y, z)$. (The inequality follows from the fact that d is a metric.) □

To summarize, non-negative linear combinations of metrics are metrics.

4.1. The Cone of Cut Metrics

4.4. Notation. Let $\text{CUTCONE}_n = \{d \in \mathbb{R}^{\binom{n}{2}} \mid d = \sum_{S \subseteq V} \alpha_S \cdot d_S$ where $\alpha_S \geq 0$ and d_S is a cut metric$\}$

CUTCONE_n contains all nonnegative combinations of all cut metrics (i.e. it is a *convex cone* of cut metrics). By Fact 4.2 and Fact 4.3, any $d \in \text{CUTCONE}_n$ is a metric.

4.2. Key Insight: $[0,1]$-Line Metrics are in the Cut Cone

4.5. Definition. We define a $[0,1]$-line metric to be a line metric with distance function $d(x,y) = |f(x) - f(y)|$, where f is a function $f : X \mapsto [0,1]$. That is, f maps only to points in the interval $[0,1]$ on the line.

4.6. Lemma. *Let (X,d) be a $[0,1]$-line metric where $d(x,y) = |f(x) - f(y)|$ for some $f : X \mapsto [0,1]$. Pick a threshold $t \in [0,1]$ uniformly at random. Then define*

$$S_t = \{x \mid f(x) \leq t\}$$

Then for every $u, v \in X$,

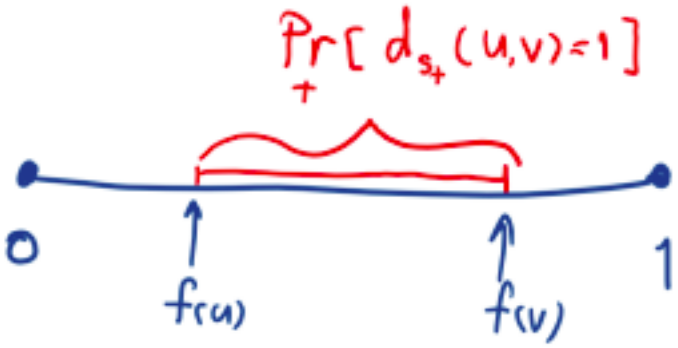
$$\mathbb{E}_t[d_{S_t}(u,v)] = d(u,v).$$

That is, $d = \mathbb{E}_t[d_{S_t}]$ is simply a distribution of cut metrics.

Proof. Note that d_{S_t} is the distance function for the cut metric defined by cut $S_t \subseteq X$. Consider $u, v \in X$, and assume without loss of generality that $f(u) \leq f(v)$.

$$\begin{aligned}
\mathbb{E}_t[d_{S_t}(u,v)] &= \Pr_t[d_{S_t}(u,v) = 1] && \text{by the definition of expectation} \\
&= \Pr_t[|S_t \cap \{u,v\}| = 1] && \text{by the definition of the cut metric} \\
&= \Pr_t[f(u) \leq t \leq f(v)] && \text{by the definition of } S_t \\
&= |f(v) - f(u)| && \text{since } t \text{ is sampled uniformly from } [0,1] \text{ (see figure)} \\
&= d(u,v)
\end{aligned}$$

□



If d is an $n-$point metric, we can rephrase this result in an equivalent way. Write $X = \{x_1, x_2, \ldots, x_n\}$, where $f(x_i) \leq f(x_{i+1})$ (i.e. so the points are ordered lowest to highest). Then for $1 \leq i < n$, define $S'_i = \{x_1, \ldots, x_i\}$, and let $\alpha_i = f(x_{i+1}) - f(x_i)$.

4.7. Observation. *Let d' be the metric*

$$d' = \sum_{i=1}^{n-1} \alpha_i d_{S'_i}$$

then $d' \in \text{CUTCONE}_n$.

Proof. Observe that for $1 \leq i < n$, we have that $d_{S'_i}$ is a cut metric and $\alpha_i > 0$. Then d' is a nonnegative linear combination of cut metrics and therefore is in CUTCONE_n. □

Example: Suppose our set X was defined as $X = \{x_1, x_2, x_3, x_4\}$ as below.

Then we would define $S'_1 = \{x_1\}$, $S'_2 = \{x_1, x_2\}$, and $S'_3 = \{x_1, x_2, x_3\}$. Additionally, $\alpha_1 = 0.7$, $\alpha_2 = 0.1$, and $\alpha_3 = 0.2$. Then the metric d' would equal $d' = 0.7d_{S'_1} + 0.1d_{S'_2} + 0.2d_{S'_3}$. We will prove that metric d' as defined in Observation 4.7 is identical to the original $[0, 1]$-line metric d.

4.8. Claim. $d = d' = \sum_{i=1}^{n-1} \alpha_i d_{S'_i}$

Proof. Observe that for any choice of $t \in [0, 1]$, we have that $S_t = S'_i$ for some i. Specifically, $S_t = S'_i$ if and only if $f(x_i) \leq t < f(x_{i+1})$. Then we have that:

$$\begin{aligned} d &= \mathbb{E}_t[d_{S_t}] \\ &= \sum_{i=1}^{n-1} \Pr_t[S_t = S'_i] d_{S'_i} \\ &= \sum_{i=1}^{n-1} \Pr_t[f(x_i) \leq t < f(x_{i+1})] d_{S'_i} \\ &= \sum_{i=1}^{n-1} \alpha_i d_{S'_i} \\ &= d' \end{aligned}$$

□

Observe that $S'_i \subset S'_{i+1}$ for $1 \leq i < n$. Then following corollary is immediate.

4.9. Corollary. *Any $[0, 1]$-line metric d of n points is a nonnegative combination of $n - 1$ **nested** cut metrics. In particular, $d \in \text{CUTCONE}_n$.*

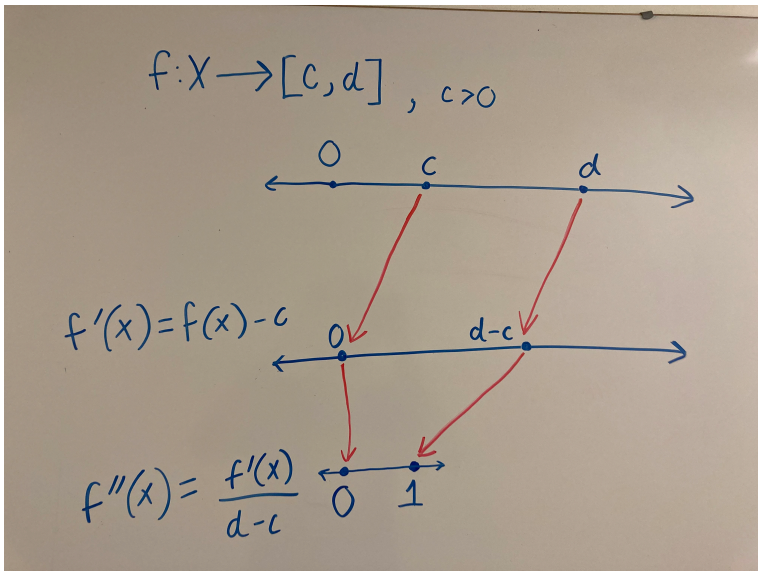
4.3. Generalizing to any Line Metric

Now we extend this result to arbitrary $n-$point line metrics. Consider an $n-$point line metric (X, d) where $d(x, y) = |f(x) - f(y)|$ for some $f : X \mapsto \mathbb{R}$. Without loss of generality, we can assume that $\min_x f(x) = 0$. This is because if $\min_x f(x) = c > 0$, then we may let $f'(x) = f(x) - c$ for all x (translating f), so that $d(x, y) = |f(x) - f(y)| = |f'(x) - f'(y)|$.

It follows that $d(x, y) = |f(x) - f(y)|$ for some function $f : X \mapsto [0, \alpha]$, where $\alpha > 0$. Now if we define $f'(x) = 1/\alpha \cdot f(x)$ for all x (rescaling f), then observe that f' is a function from X to $[0, 1]$.

Then the line metric d' defined as $d'(x,y) = |f'(x) - f'(y)|$ is a $[0,1]$-line metric. Furthermore, $d = \alpha \cdot d'$, so d is a nonnegative combination of a $[0,1]$-line metrics.

This translation and rescaling can be visualized in the figure below.



Note that since metric d' is a nonnegative combination of $n-1$ nested cut metrics by Corollary 4.9, it follows that metric d is a nonnegative combination of $n-1$ cut metrics as well.

4.10. Corollary. *Any line metric d on n points is a nonnegative combination of $n-1$ nested cut metrics. In particular, $d \in \text{CUTCONE}_n$.*

4.4. Generalizing to any ℓ_1-Metric

Consider any ℓ_1^k-metric (X,d) where $d(x,y) = \|f(x) - f(y)\|_1$ for some $f : X \mapsto \mathbb{R}^k$.

4.11. Observation. *d is a sum of k line metrics.*

Proof. Let $d_i(x,y) = |f_i(x) - f_i(y)|$, where $f_i(x)$ is the ith coordinate of $f(x)$. Note that (X,d_i) is a line metric. Furthermore, by the the definition of an ℓ_1^k-metric, we have that

$$d(x,y) = \|f(x) - f(y)\|_1 = \sum_{i=1}^{k} |f_i(x) - f_i(y)| = \sum_{i=1}^{k} d_i(x,y)$$

Then d is a nonnegative combination of the k line metrics $d_1, \ldots, d_k$. □

Since d is a nonnegative combination of k line metrics, which are in turn each a nonnegative combination of $n-1$ cut metrics by Corollary 4.10, it follows that d is a nonnegative combination of $k(n-1)$ cut metrics.

4.12. Corollary. *Any ℓ_1^k-metric d of n points is a nonnegative combination of $k(n-1)$ cut metrics. In particular, $d \in \text{CUTCONE}_n$.*

Given an ℓ_1-metric and its mapping f, we can efficiently compute its equivalent nonnegative combination of cut metrics specified in Corollary 4.12.

4.13. Notation. Given a mapping f where $d(x,y) = \left\|f(x) - f(y)\right\|_1$, define $\mathcal{S}_f$ as a collection of cuts $S \subseteq X$ such that $d = \sum_{S \in \mathcal{S}_f} \alpha_S \cdot d_S$ where $\alpha_S > 0$ and d_S is a cut metric.

4.14. Corollary. *Given a mapping f of an n-point ℓ_1^k-metric $d(x,y) = \left\|f(x) - f(y)\right\|_1$, we can compute S_f and $\{\alpha_S\}_{S \in \mathcal{S}}$ in polynomial time.*

Proof. In $O(kn)$ time we can obtain the k line metrics $d_1, \ldots, d_k$ of Observation 4.11. Now in $O(kn)$ time we can rescale and translate each of these k line metrics $d_1, \ldots, d_k$ to obtain k $[0,1]$-line metrics $d_1^{01}, \ldots, d_k^{01}$ as in the proof of Corollary 4.10. Finally, for each $[0,1]$-line metric d_i^{01}, we can obtain the corresponding collection of cuts by replicating the procedure in Corollary 4.9. This requires us to sort the n points according to function f_i and obtain $O(n)$ cuts of size $O(n)$ each in $O(n^2)$ time. To compute the coefficients for the collection of cuts it suffices to compute $O(n)$ distances between points, which can be done in $O(n)$ time. Then overall this requires $O(kn^2)$ time. To obtain our final collection $\mathcal{S}_f$ we simply need to undo the scaling we performed to obtain $[0,1]$-line metrics earlier by changing the cut coefficients appropriately for each $[0,1]$-line metric. This can be done in $O(kn)$ time. Then we can compute $\mathcal{S}_f$ and $\{\alpha_S\}_{S \in \mathcal{S}_f}$ in $O(kn^2)$ time. □

4.15. *Remark.* Notice that the bottleneck is actually the size of the output - we can actually compute $\mathcal{S}_f$ in $O(k \cdot n \log n)$ time if the nested cuts in $\mathcal{S}_f$ are represented implicitly.

5. Approximating Conductance via LP Relaxation

5.1. Overview

Our goal is to approximate the conductance $\Phi(G)$ of a graph G. To accomplish this we will first define another notion of expansion $\Phi(G,H)$, which generalizes the notion of conductance (this will make notation easier). Then we will define a linear program (LP) relaxation of $\Phi(G,H)$ denoted by $\mathrm{LR}(G,H)$ such that $\mathrm{LR}(G,H) \leq \Phi(G,H)$. Then $\mathrm{LR}(G,H)$ can be computed in polynomial time since it can be modeled by a LP. We will show that $\mathrm{LR}(G,H)$ is not too small using the ingredients we have seen above.

5.2. A General Notion of Edge Expansion: H-expansion $\Phi(G,H)$

Let $G = (V, E_G, w_G)$ be a weighted undirected graph, and let $H = (V, E_H, w_H)$ be a weighted undirected graph with the same set of vertices as G.

5.1. Definition. The H-expansion of a cut S is defined as

$$\Phi_{G,H}(S) = \frac{w_G(\partial_G S)}{w_H(\partial_H S)}$$

where $w_G(E') = \sum_{e \in E'} w_G(e)$ and $w_H(E') = \sum_{e \in E'} w_H(e)$.

5.2. Definition. The H-expansion of a graph G is defined as [5]

$$\Phi(G,H) = \min_{S: w_H(\partial_H S)\neq 0} \Phi_{G,H}(S)$$

Observe that the H-expansion of a graph G minimizes the H-expansion over all cuts S where this value is well-defined. The lemma below shows that $\Phi(G,H)$ generalizes the $\mathbf{d}$-expansion $\Phi(G,\mathbf{d})$ of a graph.

5.3. Lemma. *Given $G=(V,E)$, let H be a complete weighted graph with $w_H(u,v) = \frac{\boldsymbol{d}(u)\boldsymbol{d}(v)}{\boldsymbol{d}(V)}$ for each u,v. We have $\Phi_{G,H}(S) = \Theta(\Phi_{G,\boldsymbol{d}}(S))$ for all $S \subseteq V$ and so $\Phi(G,H) = \Theta(\Phi(G,\boldsymbol{d}))$.*

Proof. This follows because for any S where $\mathbf{d}(S) \leq \mathbf{d}(V)/2$, we have

$$w_H(\partial_H S) = \sum_{u\in S}\sum_{v\notin S} \frac{\mathbf{d}(u)\mathbf{d}(v)}{\mathbf{d}(V)} = \sum_{u\in S} \mathbf{d}(u)\frac{\mathbf{d}(V\setminus S)}{\mathbf{d}(V)} = \Theta(\mathbf{d}(S))$$

since $\mathbf{d}(V\setminus S) \geq \frac{1}{2}\mathbf{d}(V)$.

It follows that

$$\Phi_{G,H}(S) = \frac{w_G(\partial_G S)}{w_H(\partial_H S)} = \frac{w_G(\partial_G S)}{\Theta(\mathbf{d}(S))} = \Theta(\Phi_{G,\mathbf{d}}(S)).$$

□

Recall that conductance $\Phi(G) = \Phi(G,\mathbf{d})$, where $\mathbf{d}(u) = \deg(u)$. It immediately follows from Lemma 5.3 that we can choose a graph H so that $\Phi(G,H) = \Theta(\Phi(G))$.

5.4. Corollary. *Given $G=(V,E)$, let H be a complete weighted graph where $w_H(u,v) = \frac{\deg(u)\deg(v)}{\mathrm{vol}(V)}$ for each u,v. Then $\Phi(G,H) = \Theta(\Phi(G))$.*

5.3. LP Relaxation of $\Phi(G,H)$

We start with the following observation:

5.5. Observation.

$$w_G(\partial_G S) = \sum_{u,v:|\{u,v\}\cap S|=1} w_G(u,v) = \sum_{u,v} w_G(u,v)\cdot d_S(u,v)$$

where $d_S : V\times V \mapsto \{0,1\}$ is a cut metric defined by $S\subset V$, and the sum $\sum_{u,v}$ is summing over unordered *pairs of vertices.*

Using this observation, we may rewrite $\Phi(G,H)$ as

$$\Phi(G,H) = \min_{S: w_H(\partial_H S)\neq 0} \frac{w_G(\partial_G S)}{w_H(\partial_H S)} = \min_{d_S:\text{cut metric over } V} \frac{\sum_{u,v} w_G(u,v)\cdot d_S(u,v)}{\sum_{u,v} w_H(u,v)\cdot d_S(u,v)}$$

Then it follows that we may consider computing $\Phi(G,H)$ as an optimization problem minimizing over all cut metrics. Now the crucial step by Leighton and Rao that allows us to obtain a LP approximation is to instead minimize over *all metrics* instead of all cut metrics. We now define this new measure, which we denote $\mathrm{LR}(G,H)$.

[5] $\Phi(G,H)$ is also called the "non-uniform sparsity of G with respect to H".

5.6. Notation.

$$\mathrm{LR}(G,H) = \min_{d:\text{metric over } V} \frac{\sum_{u,v} w_G(u,v)\cdot d(u,v)}{\sum_{u,v} w_H(u,v)\cdot d(u,v)}$$

Now we claim that $\mathrm{LR}(G,H)$ can be modeled as a linear program (LP).

5.7. Claim. $\mathrm{LR}(G,H)$ *can be modeled as a linear program (LP).*

Proof. First, note that by normalizing, we can assume that $\sum_{u,v} w_H(u,v)\cdot d(u,v) = 1$. More concretely, if there exists a metric d' that minimizes $LR(G,H)$, then there exists a metric d'' that can be obtained by scaling d' that minimizes $LR(G,H)$ and further $\sum_{u,v} w_H(u,v)\cdot d''(u,v) = 1$. Minimizing $\sum_{u,v} w_G(u,v)\cdot d(u,v)$ over all metrics d where $\sum_{u,v} w_H(u,v)\cdot d(u,v) = 1$ is captured exactly by the following LP:

$$\begin{aligned}
\text{minimize} \quad & \sum_{u,v} w_G(u,v)\cdot d(u,v) \\
\text{subject to} \quad & \sum_{u,v} w_H(u,v)\cdot d(u,v) = 1 \\
& d(u,v) \le d(u,w) + d(w,v) \qquad \forall u,v,w \\
& d(u,v) \ge 0 \qquad \forall \{u,v\} \in \tbinom{V}{2}
\end{aligned} \tag{1}$$

This follows from the fact that any collection of $\binom{V}{2}$ values $d(u,v)$ that satisfy the above conditions correspond to a $|V|$-point metric (by definition), and so the value of the solution to this LP is exactly $\mathrm{LR}(G,H)$. □

The following is immediate.

5.8. Lemma. *Let λ^* be the optimal value of the above LP. We have $\lambda^* = \mathrm{LR}(G,H)$ and we can compute it in polynomial time.*

Now we need to ensure that $\mathrm{LR}(G,H)$ is a good approximation of $\Phi(G,H)$. It is immediate that $\mathrm{LR}(G,H) \le \Phi(G,H)$ because we consider all metrics including all cut metrics. We will now show that $\mathrm{LR}(G,H)$ cannot be too much smaller than $\Phi(G,H)$.

5.4. Showing that LR(G,H) is a Good Relaxation

We will prove that $\mathrm{LR}(G,H) \ge \frac{1}{\Theta(\log n)}\Phi(G,H)$. In other words, we will show that relaxing cut metrics to all metrics may reduce our desired value by at most a $O(\log n)$ factor. Why might we expect this to be true? At a high level, our ingredients say the following:

1. **Bourgain's embedding:** Any metric (X,d) can be "captured" by an ℓ_1-metric with just $O(\log n)$ distortion.
2. **ℓ_1-metrics are in the cut cone:** any ℓ_1-metric is just a combination of cut metrics.

Taken together, these two facts basically suggest that any metric is captured by a combination of cut metrics within a $O(\log n)$ factor. We will now formally prove our desired inequality.

5.9. Lemma. $\mathrm{LR}(G,H) \ge \frac{1}{\Theta(\log n)}\Phi(G,H)$

Proof. Let d^{OPT} be the optimal metric from the linear program 1. Then we may write $\mathrm{LR}(G,H)$ as

$$\mathrm{LR}(G,H) = \frac{\sum_{u,v} w_G(u,v)\cdot d^{\mathrm{OPT}}(u,v)}{\sum_{u,v} w_H(u,v)\cdot d^{\mathrm{OPT}}(u,v)}$$

Then by applying Bourgain's to d^{OPT}, we get an embedding $f : V \mapsto \mathbb{R}^{O(\log^2 n)}$ where for all $u,v \in V$,

$$\left\|f(u)-f(v)\right\|_1 \le d^{\mathrm{OPT}}(u,v) \le O(\log n)\cdot\left\|f(u)-f(v)\right\|_1$$

Then by applying Corollary 4.14 to f, we have

$$\left\|f(u)-f(v)\right\|_1 = \sum_{S\in\mathcal{S}_f} \alpha_S d_S(u,v)$$

where $\mathcal{S}_f$ is a collection of $O(n\log^2 n)$ cuts. So we have

$$\begin{aligned}
\mathrm{LR}(G,H) &\ge \frac{1}{\Theta(\log n)}\cdot\frac{\sum_{u,v} w_G(u,v)\sum_{S\in\mathcal{S}_f}\alpha_S d_S(u,v)}{\sum_{u,v} w_H(u,v)\sum_{S\in\mathcal{S}_f}\alpha_S d_S(u,v)}\\
&= \frac{1}{\Theta(\log n)}\cdot\frac{\sum_{S\in\mathcal{S}_f}\alpha_S(\sum_{u,v} w_G(u,v)d_S(u,v))}{\sum_{S\in\mathcal{S}_f}\alpha_S(\sum_{u,v} w_H(u,v)d_S(u,v))}\\
&\ge \frac{1}{\Theta(\log n)}\cdot\min_{S\in\mathcal{S}_f}\frac{\sum_{u,v} w_G(u,v)d_S(u,v)}{\sum_{u,v} w_H(u,v)d_S(u,v)} \qquad \text{as } \frac{\sum_{i=1}^k a_i}{\sum_{i=1}^k b_i} \ge \min_i \frac{a_i}{b_i}\\
&= \frac{1}{\Theta(\log n)}\cdot\min_{S\in\mathcal{S}_f}\frac{w_G(\partial_G S)}{w_H(\partial_H S)}\\
&= \frac{1}{\Theta(\log n)}\cdot\min_{S\in\mathcal{S}_f}\Phi_{G,H}(S)\\
&\ge \frac{1}{\Theta(\log n)}\cdot\Phi(G,H)
\end{aligned}$$

Therefore we can conclude

$$\mathrm{LR}(G,H) \le \Phi(G,H) \le O(\log n)\cdot\mathrm{LR}(G,H)$$

□

The following theorem is immediate.

5.10. Theorem. *There is a randomized algorithm that $O(\log n)$-approximates $\Phi(G,H)$ in polynomial time. The same holds for $\Phi(G)$.*

5.11. *Remark.* In fact, the only slow step in the above algorithm is to obtain d^{OPT}. All other steps take near linear time.

5.12. Exercise. Given an embedding f of the optimal metric d^{OPT} of the LP, show how to compute a cut S such that $\Phi_{G,H}(S) \le O(\log n)\cdot\Phi(G,H)$ in $\tilde{O}(|E(G)|+|E(H)|)$ time. In the case where we want to compute conductance, the running time is just $\tilde{O}(|E(G)|)$.

6. Limitation and Outlook

This approach gives a $O(\log n)$ approximation to $\Phi(G, H)$, which is tight (you will prove this in the homework). We can obtain better approximations via smaller distortion metric embeddings. Bourgain's embedding is very general in that it can embed any metric into an ℓ_1-metric. However, we can obtain better embeddings by considering a more specific set of metrics, like the set of *negative-type* metrics.

6.1. Definition. A *negative-type* metric (X, d) is such that there exists $f : X \mapsto \mathbb{R}^k$ such that $d(x, y) = \|f(x) - f(y)\|_2^2$.

Any n-point negative-type metric can be embedded into an ℓ_1-metric with distortion $O(\sqrt{\log n} \log \log n)$ [6]. This embedding is similar to Bourgain's, but with better distortion on a smaller class of metrics. A relaxation of $\Phi(G, H)$ to all negative-type metrics can be computed using semidefinite programming (SDP). Following the same argument as the one in this lecture (but more complicated), we get an $O(\sqrt{\log n} \log \log n)$-approximation algorithm for $\Phi(G, H)$. This topic is further exmplored in a survey by Naor on connections between expansion and functional analysis[7].

This approach works for a general notion of edge expansion in undirected graphs. Metric embeddings have also been studied in the context of vertex expansion[8]. Later, we will see a more robust framework based on the "cut-matching game" which works for both **vertex expansion** and **directed graphs**.

7. Exercises

The shortest path metric is universal in the following sense:

7.1. Exercise. Show that any (finite) metric (X, d) is a shortest path metric of a weighted graph $G = (X, E, w)$ where $w(u, v) = d(u, v)$.

7.2. Exercise (More examples of metrics)**.** Convince yourself that the following are metrics:

1. **(ℓ_p metric):** $X = \mathbb{R}^m$ and $d(x, y) = \|x - y\|_p = (\sum_i (x_i - y_i)^p)^{1/p}$.
   - For $p = 2$, $\|x - y\|_2 = \sqrt{\sum_i (x_i - y_i)^2}$ is just a Euclidean distance.
   - For $p = \infty$, $\|x - y\|_\infty = \max_i \{|x_i - y_i|\}$
2. **(uniform metric):** Let X be any set and $d(x, y) = 1$ for all $x \neq y$.
3. **(1-2 metric):** Let X be any set and $d(x, y) \in \{1, 2\}$ for all $x \neq y$.
4. **(edit distance):** Let $X = \{0, 1\}^*$ be a set of strings. Let $d(x, y)$ be the edit distance between x and y.

The earlier discussion about ℓ_1-metrics and cut metrics gives the following characterization:

7.3. Exercise. Let $L1_n$ contain all n-point ℓ_1-metrics. Prove that $L1_n = \text{CUTCONE}_n$.

[6] Sanjeev Arora, James Lee, and Assaf Naor. Euclidean distortion and the sparsest cut

[7] Assaf Naor. L1 embeddings of the heisenberg group and fast estimation of graph isoperimetry, 2010.

[8] https://homes.cs.washington.edu/ jrl/papers/pdf/fhl-sicomp.pdf

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 4: Bourgain's Theorem

September 9, 2021

Instructor: Thatchaphol Saranurak

Scribe: Cheng-Hao Fu

1. Briefly Motivate and State the Theorem

In general, an arbitrary distance metric (X, d) can be difficult to work with directly. As seen in the algorithm for computing conductance, it is far easier to work with ℓ_1 metric. In this pursuit, finding embeddings of (X, d) into adequately small-dimension $\mathbb{R}^n$ quickly is a gold-standard of computation.

Bourgain's Theorem gives us a guarantee that we can find a low distortion embedding into low-dimension Euclidean space:

1.1. Theorem (Bourgain)**.** *Given an n-point metric (X, d), there is an embedding $f : X \to \mathbb{R}^{O(\log^2 n)}$ such that, for any $x, y \in X$,*

$$\|f(x) - f(y)\|_1 \leq d(x, y) \leq O(\log n) \cdot \|f(x) - f(y)\|_1.$$

Moreover, given d, in $\tilde{O}(n^2)$ time, the mapping f can be computed correctly with probability at least $1-1/n$. If (X, d) is a shortest path metric of an m-edge graph, then the running time is $\tilde{O}(m)$ time.

Note that our proof will show an embedding f where

$$\Omega(\log n) \cdot d(x, y) \leq \|f(x) - f(y)\|_1 \leq O(\log^2 n) \cdot d(x, y).$$

And after scaling $\bar{f} = \frac{f}{(\log n)^2}$, $\bar{f}$ satisfies the desired inequality in the theorem.

The usefulness of this theorem is clear. For example, in the case of graphs with n vertices, the theorem implies the existence of an assignment of $O(\log^2 n)$ numbers (representing the dimension of Euclidean space) to each node that allows distance calculation on the graph to be done by comparing the labels element-wise with a small amount of distortion. This is much better than the $O(n^2)$ guarantee that Dijkstra's gives on calculating Single Source Shortest Path.

2. The Algorithm

The algorithm itself is quite simple. We first define a notion of the distance of any point to a particular set of points $A \subseteq X$ as follows:

2.1. Definition. The distance between a point x and a set of points A with respect to the distance metric is:

$$d(x, A) := \min_{a \in A} d(x, a).$$

We then define A_{ij} for $i = 1, \ldots, i_{\max} = 10000 \log n$ and for $j = 1, \ldots, j_{\max} = \lceil \log n \rceil$ by letting $A_{ij} \subseteq X$ include each element of X with probability $1/2^j$. We sample all A_{ij} independently.

We then define our embedding as follows:

$$f : X \to \mathbb{R}^{i_{\max} j_{\max}} \text{ where } f_{ij}(x) = d(x, A_{ij})$$

We note that every element in the space now has an associated vector of labels that is of length $O(\log^2 n)$, representing its location in $\mathbb{R}^{i_{\max} j_{\max}}$.

It is then clear that if this embedding achieves the desired distortion, along with the runtime, we will be done.

3. Proving Runtime

3.1. Claim. *Given d, the embedding f can be computed in $\tilde{O}(n^2)$ time. If (X, d) is a shortest path metric defined by an m-graph, then f can be computed in $\tilde{O}(m)$ time.*

Proof of Claim. First of all, it takes $\tilde{O}(n)$ time to sample all A_{ij}. To compute each entry of $f(x)$

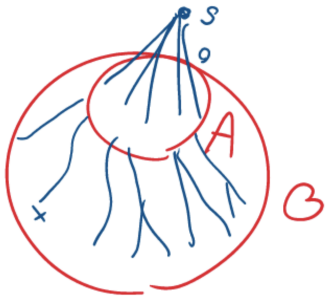
for all $x \in X$, when (X, d) is a shortest path metric, instead of fixing a point in X and compute the distance between itself and all possible A_{ij}, we switch gear by fixing a A_{ij} and computing its distance between all $x \in X$. To do this, for a fixed A_{ij}, we assign a source s that connects to each point in A_{ij} with distance 0, and running Dijkstra's (which takes $\tilde{O}(m)$ time) to compute the shortest path between any point x and s, which is exactly the distance between x and A_{ij}. Since there are $O(\log n^2)$ A_{ij}s that we have to do this with, computing the embedding will take $\tilde{O}(m)$ time, as desired.

When (X, d) is a general metric, the key observation is that we can treat any general metric as the shortest path metric on the weighted complete graph $G_d(X, E, \kappa)$ where $\forall \{x, y\} \in E, \kappa(\{x, y\}) = d(x, y)$. In this way, we could use the similar strategy as we described above to compute f in $\tilde{O}(n^2)$ time since G_d has $O(n^2)$ edges. It remains to show that d is indeed the shortest path metric on G_d. $\forall \{x, y\} \subseteq V$, on one hand, $\text{dist}_{G_d}(x, y) \leq d(x, y)$ since there exists the path $< x, y >$ with weight $d(x, y)$ between x and y; on the other hand, $\text{dist}_{G_d}(x, y) \geq d(x, y)$ since d satisfies

triangle inequality whence by an inductive argument we can see that $\forall P \in \mathcal{P}_{x,y}, d(P) \geq d(x,y)$, and therefore the distance of the shortest $(x,y)-$path is also greater than $d(x,y)$.

□

For an illustration of the process:



4. Analysis of Algorithm

Now, the goal is to prove $\Omega(\log n) \cdot d(x,y) \leq \|f(x) - f(y)\|_1 \leq O(\log^2 n) \cdot d(x,y)$. We begin with the upper bound.

4.1. Upper Bound

4.1. Claim. $\|f(x) - f(y)\|_1 \leq O(\log^2 n) \cdot d(x,y)$.

Proof of Claim. We start by breaking down the definition of $\|f(x) - f(y)\|_1$. It is simply the sum of the magnitudes of the element-wise differences of $f(x)$ and $f(y)$. We can then use the definition of f to arrive at:

$$\|f(x) - f(y)\|_1 = \sum_{ij} |f_{ij}(x) - f_{ij}(y)| = \sum_{ij} |d(x, A_{ij}) - d(y, A_{ij})|$$

We know that there are only $O(\log^2 n)$ elements in the final summation, which is exactly the multiplier of $d(x,y)$ in the desired statement. This tells us that proving $|d(x, A_{ij}) - d(y, A_{ij})| \leq d(x,y)$ is sufficient for all i, j.

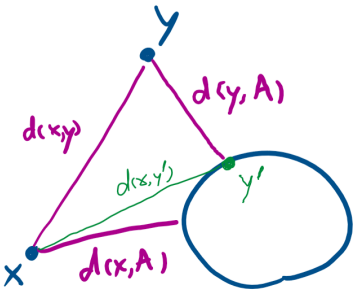
Without loss of generality, assume that $d(x, A_{ij}) \geq d(y, A_{ij})$. It suffices to prove that $d(x, A_{ij}) \leq d(x,y) + d(y, A_{ij})$

Let y' be such that $d(y, A_{ij}) = d(y, y')$.

By definition, we have that $d(x, A_{ij}) \leq d(x, y')$. Then, we get $d(x, y') \leq d(x,y) + d(y, y') = d(x,y) + d(y, A_{ij})$ by triangle inequality. Note that this actually works for any distance metric (X, d) with $A \subseteq X$.

□

Visually:



4.2. Lower Bound

4.2. Claim. $\|f(x) - f(y)\|_1 \geq \Omega(\log n) \cdot d(x, y)$.

Proof of Claim. We begin by defining a couple of useful objects.

4.3. Definition. For any point s and radius r, define $B(s, r) = \{z | d(z, s) \leq r\}$ to be the ball radius r around s. Similarly, $B^o(s, r) = \{z | d(z, s) < r\}$ will be referred to as the open ball around s with radius r.

We then fix $x, y \in X$, and define a couple more things.

Let $r_0 = 0$. For $j = 1, \ldots, j_{\max} = \lceil \log n \rceil$, r_j be the minimum r where both $|B(x, r)|, |B(y, r)| \geq 2^j$.

Define $\Delta_j = r_j - r_{j-1}$. Think of this as the amount that you have to grow the ball in order for the ball to encompass the next magnitude in size. Notice that, by telescoping, $\sum_{j=1}^{j_{\max}} \Delta_j = r_{j_{\max}}$.

We start off by assuming $r_{j_{\max}} < d(x, y)/4$ for now. Combinatorially, this means that the radius that we are considering is significantly less than the distance between them. This, of course, implies that $B(x, r_{j_{\max}})$ and $B(y, r_{j_{\max}})$ are disjoint. Note that there is no reason to expect that this assumption will hold. We will relax the assumption later in the proof.

Without loss of generality, assume that $|B(x, r_j)| \geq |B(y, r_j)|$. This means that y was the point that required r_j to be as large as it was, in fact, we can say then that $|B(y, r_j)| = 2^j$. If it is the case that multiple distances that are the same cause this assumption to be false, we can perturb each of the distances by a small constant ϵ, which will not cause any significant distortion, and give us the equality we are looking for. Thus, we can say that, $|B^o(y, r_j)| < 2^j$. Also, trivially $|B(x, r_{j-1})| \geq 2^{j-1}$.

4.4. Definition. We say a set A_{ij} is $good$ if $A_{ij} \cap B(x, r_{j-1}) \neq \emptyset$ and $A_{ij} \cap B(y, r_{j-1}) = \emptyset$

We use the term $good$ because it will help us definitively add to the minimum distance between the two points. In fact, we can observe that it will add at minimum Δ_j to $\|f(x) - f(y)\|_1$. This is because $d(y, A_{ij}) \geq r_j$ but $d(x, A_{ij}) \leq r_{j-1}$.

This is extremely powerful. To see why, let us see what happens when all A_{ij} are good. Then we would have

$$\|f(x) - f(y)\| = \sum_{ij} |d(x, A_{ij}) - d(y, A_{ij})| \geq \sum_{ij} \Delta_j = i_{\max} \cdot r_{j_{\max}} = \Omega(\log n) \cdot d(x, y)$$

The first equality comes from the definition of f. The second inequality comes from the justification provided above, and the rest is by definition (Noting that our assumption has $r_{j_{\max}} \approx d(x, y)/4$). This is exactly the result we are looking for, but unfortunately not all A_{ij} are good. But it should be clear that since we are aiming for $\Omega(\log n)$, a constant fraction of A_{ij} being good for each j will be sufficient for the bound. This is what we will next attempt to prove.

4.5. Proposition. *For every i, j, A_{ij} is good with probability at least $1/100$.*

Proof of Proposition. First, we calculate a lower bound the probability that $A_{ij} \cap B^o(y, r_j) = \emptyset$. Because of the way that we picked all of the points in A_{ij}, we have that the probability that every element in the open ball is avoided by A_{ij} is:

$$(1 - 1/2^j)^{|B^o(y, r_j)|} > (1 - 1/2^j)^{2^j} \geq 1/10.$$

The second inequality can be verified via graphing.

Similarly, $A_{ij} \cap B(x, r_{j-1}) = \emptyset$ occurs with probability:

$$(1 - 1/2^j)^{|B(x, r_{j-1})|} \leq (1 - 1/2^j)^{2^{j-1}} \leq 9/10.$$

Thus, $A_{ij} \cap B(x, r_{j-1}) \neq \emptyset$ occurs also with probability at least $1/10$. Because of our assumption that these two balls have an empty intersection, these two events are independent, and thus we can multiply their probabilities together to get a lower bound on the probability that A_{ij} is good:

$$\Pr[A_{ij} \cap B(x, r_{j-1}) \neq \emptyset \text{ and } A_{ij} \cap B^o(y, r_j) = \emptyset] \geq 1/10 \times 1/10 = 1/100$$

□

From the above, for each j, the expected total number of good sets A_{ij} over all i is at least $i_{\max}/100$.

However, before we proceed with the final part of the proof, we need to show that this occurs with high enough probability to get the high probability argument that we also want. This motivates the need for the following claim:

4.6. Claim. *For each j, the total number of good A_{ij} over all i is at least $i_{\max}/200$ with probability at least $1 - 1/n^{10}$.*

Proof of Claim. We will simplify this to the question of proving an upper bound on the probability of $i_{max}/200$ or fewer successes in i_{max} trials of Bernoulli random variables with success probability $1/100$. Let us define the random variables $X_1, .., X_{i_{max}}$ to represent the trials, and $Y = \sum_i X_i$.

Note that with Chernoff Bound, we have:

$$\Pr[Y \leq i_{max}(1/100 - 1/200)] \leq e^{-(1/8)(1/100)i_{max}}$$

Plugging in i_{max} with $10000 \log n$:

$$\Pr[Y \leq i_{max}/200] \leq e^{-(1/8)(1/100)10000 \log n} \leq 1/n^{10}$$

□

This gives us the following corollary:

4.7. Corollary. *For probability at least* $1 - 1/n^{10}$, $\|f(x) - f(y)\| \geq \frac{i_{\max}}{200} \cdot r_{j_{\max}} = \Omega(\log n) \cdot d(x, y)$.

Proof. This follows pretty much exactly the same as the argument used at the top of page 5. The only difference is that we can only sum over the good sets, which makes us lose a constant factor, which is still good enough for us.

$$\|f(x) - f(y)\| \geq \sum_{ij:A_{ij} \text{ is good}} |d(x, A_{ij}) - d(y, A_{ij})| \geq \sum_{ij:A_{ij} \text{ is good}} \Delta_j \geq \frac{i_{\max}}{200} \cdot r_{j_{\max}}$$

The last inequality holds with the probability desired, and so we are done. □

This corollary gives us the tools to prove the lower bound, however, we have to keep in mind that we are still operating under the assumption that $r_{j_{\max}} < d(x, y)/4$. In order to lift this assumption, we redefine $r'_j = \min\{r_j, d(x, y)/4\}$ and work with r'_j instead. By noticing that r'_j is a lower bound on r_j, the above analysis can still go through.

Now that we have addressed this point, we can finish the proof:

Since each distance $d(x, y)$ fails in being mapped to within the bounds with $1/n^{10}$ probability, we can use union bound to see that over all pairs, the probability of failure is less than or equal to:

$$\binom{n}{2}/n^{10} \leq 1/n^8$$

This allows us to say: For probability at least $1 - 1/n^8$, $\|f(x) - f(y)\| = \Omega(\log n) \cdot d(x, y)$ for all $x, y \in X$, which proves the original claim. □

Thus completes the proof of Bourgain's Theorem.

5. Examples

To motivate the way we have built the framework for the theorem, we will begin with a very basic attempt at embedding a distance into a ℓ_1 metric, and justifying why our approach needs to be as complicated as it is. This will be primarily done by testing ideas on shortest path metrics on various types of graphs. In addition to the discussion section, a large part of this section will be supported by the lecture notes of Luca Trevisan of UC-Berkeley[1].

5.1. Cycles

A natural precursor to the method illustrated in Bourgain's Theorem picks a point uniformly at random, and takes the embedding as labeling any point a with some constant factor times the distance between a and that point.

$$f(v) = k \times d(v, a)$$

Distances are then, naturally, the differences between these labels. The fact that this is a metric follows trivially. This is effective on cycles. We start by proving the following fact:

5.1. Claim. *Let a be a random node. $E[|d(x,a) - d(y,a)|] = \Theta(d(x,y))$.*

Proof of Claim. By the same argument used in the proof for 4.1, it is clear that $E[|d(x,a) - d(y,a)|] = O(d(x,y))$. Thus, we just have to justify that the expectation is not too small.

Consider the expected value of the distance when a is on the path between x and y. It is more or less uniformly distributed between 0 and $d(x,y)$ (depending on parity), with the low when the random point is the midpoint, and the high being when the point is one of the endpoints. This occurs with probability approximately equal to $d(x,y)/n$

Thus:

$$E[|d(x,a) - d(y,a)|] \approx d(x,y)^2/(2n) + \alpha$$

Where α represents the contribution in the other case. We also notice that the embedded distance is ranged from $d(x,y)$ to 0, with the high occurring whenever the paths representing x to a and y to a overlap, and the low happening whenever the random point is exactly the same distance to both x, y. This means that α contributes roughly (slightly more than) $d(x,y)/2$ on average per point. This clearly tells us that $E[|d(x,a) - d(y,a)|] = \Omega(d(x,y))$.

Thus finishes the proof. □

In much the same fashion as classic Bourgain's, we can then use a logarithmic number of f as described above, to get the desired probability for the accuracy of our bound.

[1] https://lucatrevisan.github.io/teaching/expanders2016/lecture10.pdf

5.1.1. Not Quite Enough

As is clear upon reading the proofs, this method is much easier. However, we will next see that it is not an effective way to embed an arbitrary metric.

Suppose that we have a metric that has distances between any two points be either 1 or 2. It is 2 if and only if z is one of the endpoints. All other distances are 1.

Looking at $E[|d(r,z) - d(r,v)|]$, reveals:

$$E[|d(r,z) - d(r,v)|]1 + 2/n$$

.
This means that the distortion has the potential to be linear, which is a problem. There are many steps between this simple argument with the proof discussed above, but hopefully this gives sufficient information to inform a little bit of the background of Bourgain's.

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 5: Approximate Max-flow Min-cut Theorem

September 14, 2021

Instructor: Thatchaphol Saranurak

Scribe: Yaowei Long

1. Overview and Preliminaries

We are going to prove a generalization of the max-flow min-cut theorem. Roughly speaking, for two undirected weighted graph G and H, we will show that the H-expansion of G and the maximum H-concurrent flow on G are approximately the same. In details, the gap between them is bounded by a $O(\log n)$ factor, and this factor is tight. Moreover, we will learn two partial orders based on cuts and concurrent flows respectively. The approximate max-flow min-cut theorem implies that these two orders are essentially the same.

The proof of the approximate max-flow min-cut theorem is based on the relationship between the H-expansion of G and its cut-metrics-to-all-metrics relaxation (Leighton-Rao relaxation).

1.1. Definition (H-expansion $\Phi(G,H)$)**.** Let G, H be two undirected weighted graphs on the same vertex set with edge weight functions ω_G, ω_H. The H-expansion of G is defined as

$$\Phi(G,H) = \min_S \frac{w_G(\partial_G S)}{w_H(\partial_H S)} = \min_{d_S:\text{cut metric over } V} \frac{\sum_{u,v} w_G(u,v)\cdot d_S(u,v)}{\sum_{u,v} w_H(u,v)\cdot d_S(u,v)}.$$

The latter equation follows the property of the cut metric, i.e. $\omega(\partial S) = \sum_{u,v} \omega(u,v)\cdot d_S(u,v)$.

1.2. Definition (LR relaxation)**.** Let G, H be graphs as in definition 1.1, the Leighton-Rao relaxation of $\Phi(G,H)$ is defined as

$$\text{LR}(G,H) = \min_{d:\text{metric over } V} \frac{\sum_{u,v} w_G(u,v)\cdot d(u,v)}{\sum_{u,v} w_H(u,v)\cdot d(u,v)}$$

$\text{LR}(G,H)$ is also equal to the optimal solution of the following LP.

$$\begin{aligned} &\text{(LR)} \\ &\min \sum_{u,v} w_G(u,v)\cdot d(u,v) \end{aligned} \tag{1}$$

$$\text{s.t.} \sum_{u,v} w_H(u,v) \cdot d(u,v) = 1$$
$$d(u,v) \le d(u,w) + d(w,v) \qquad \forall u,v,w$$
$$d(u,v) \ge 0 \qquad \forall \{u,v\} \in \binom{V}{2}$$

1.3. Theorem. *Let G, H be undirected weighted graphs.*

$$\mathrm{LR}(G,H) \le \Phi(G,H) \le O(\log n) \cdot \mathrm{LR}(G,H)$$

The detailed proof of theorem 1.3 is shown in lecture 2-1.

2. Concurrent Flow

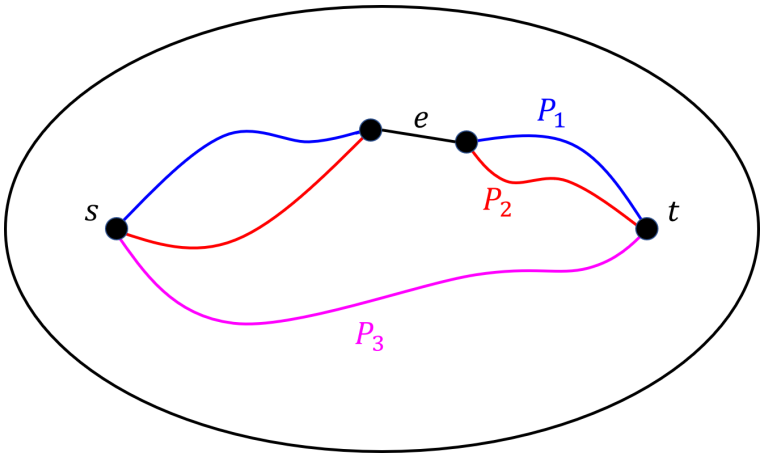
Concurrent flows are also called multi-commodity flows, which will route on an undirected weighted graph different types of commodities each of which has independent source, sink and demands.

The single commodity flow is the special one with just one type of commodity and single source and sink.

2.1. Definition ((s,t)-flows (single commodity flows))**.** Consider a graph $G = (V, E, \kappa)$ with edge capacity $\kappa : E \to \mathbb{R}_{\ge 0}$. Let s, t be the source and sink and let $\mathcal{P}_{st}$ be the set of all (s,t)-paths in G. An (s,t)**-flow** f is simply a function that assigns non-negative values to paths in $\mathcal{P}_{st}$. That is, $f : \mathcal{P}_{st} \to \mathbb{R}_{\ge 0}$.

- The **value** of f is $\|f\| = \sum_P f(P)$, i.e. total amount of flow sent through (s,t)-paths.
- The **congestion** of flow f on edge $e = (u,v)$ is the total amount of flow on e relative to the capacity of e:
$$\mathsf{cong}_f(e) = \frac{\sum_{P \ni e} f(P)}{\kappa(e)}.$$
- The **congestion of the flow** is the maximum congestion over all edges:
$$\mathsf{cong}_f = \max_{e \in E} \mathsf{cong}_f(e).$$

See figure 1 for an example.



1. ábra. In this example, let s and t denote the source and the sink respectively. There are three path P_1, P_2, P_3 connecting s and t. Let's say the (s,t)-flow f is defined by $f(P_1) = 10, f(P_2) = 3, f(P_3) = 2$. The value of f is $\|f\| = f(P_1) + f(P_2) + f(P_3) = 15$ and the congestion on edge e is $\mathsf{cong}_f(e) = (f(P_1) + f(P_2))/\kappa(e) = \frac{13}{\kappa(e)}$.

Actually, the definition of single commodity flows is equivalent to the classical definition of single-source and single-sink flows. In the classical definition, an (s,t)-flow $f : E \to \mathbb{R}_{\geq 0}$ assigns value on **edges** (not paths) such that for every node $u \neq s,t$, the total flow coming in to u equals the total flow going out of u (i.e. $\sum_v f(v,u) = \sum_v f(u,v)$). However, given any (s,t)-flow in the sense of this definition, we can always decompose it into a collection of (s,t)-paths and cycles. Now, by removing all cycles that does not contribute anything to the value of the flow. We obtain precisely the flow in the definition we are working with here.

2.2. Definition (Concurrent flows (multi-commodity flows))**.** Consider two graphs $G = (V, E_G, \kappa_G)$ and $H = (V, E_H, \kappa_H)$ with nonnegative edge capacities. An H**-concurrent flow** $\mathcal{F}$ is a collection of (s,t)-flows of value $\kappa_H(s,t)$ for all $(s,t) \in E(H)$:

$$\mathcal{F} = \{f_{(s,t)} \mid \|f_{(s,t)}\| = \kappa_H(s,t)\}_{(s,t)\in E(H)}.$$

$\mathcal{F}$ is also called a **flow-embedding** and we say $\mathcal{F}$ **embeds** H **into** G.

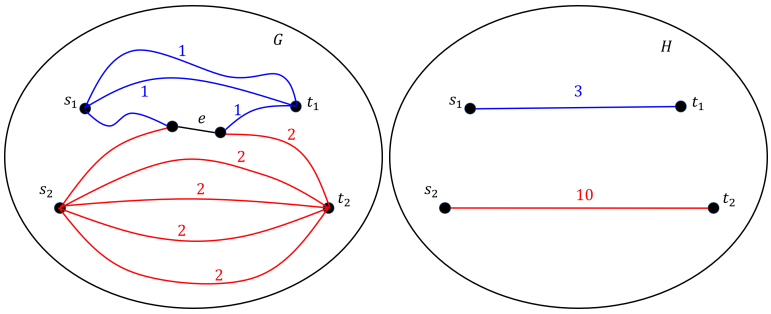
The **congestion of multi-commodity flow** $\mathcal{F}$ **on** e is

$$\mathsf{cong}_{\mathcal{F}}(e) = \sum_{(s,t)\in E(H)} \mathsf{cong}_{f_{(s,t)}}(e).$$

and **the congestion of** $\mathcal{F}$ is

$$\mathsf{cong}_{\mathcal{F}} = \max_{e\in E} \mathsf{cong}_{\mathcal{F}}(e).$$

If congestion $\mathsf{cong}_{\mathcal{F}} \leq 1$, we say that $\mathcal{F}$ has **no congestion** or $\mathcal{F}$ is **feasible**.



2. ábra. In this example, let $\mathcal{F}$ be the H-concurrent flow on G. We can see that $\mathcal{F}$ embeds blue edge (s_1, t_1) in H into three blue paths in G and embeds red edge (s_2, t_2) in H into five red paths in G. The congestion of edge e in G is $\mathsf{cong}_{\mathcal{F}}(e) = 1/\kappa(e) + 2/\kappa(e) = 3/\kappa(e)$.

Intuitively, H is the **demand graph** of $\mathcal{F}$ because each edge in H represents a unique type of commodity with its own source, sink and demand. Note that concurrent flows are not the same as the multiple-source multiple-sink flows, where the latter essentially has only one type of commodity but multiple sources and sinks.

3. Partial Order based on Concurrent Flow

3.1. Definition (Partial order based on concurrent flow)**.** Consider two undirected weighted graphs G and H.

We define $H \preccurlyeq^{\text{flow}} G$ if there is a feasible H-concurrent flow $\mathcal{F}$ in G. In this case, we say that **the demand** H **is routable in** G or H **is embeddable into** G. Notice that this defines a partial order between graphs.

We also say that **the demand** H **is routable in** G **with congestion** q or H **is embeddable into** G **with congestion** q if $\mathcal{F}$ has congestion q.

3.2. Fact. *For any graph G and G',*

- *let $G'' = G + G'$ be such that $\kappa_{G''}(u, v) = \kappa_G(u, v) + \kappa_{G'}(u, v)$ for all u, v.*
- *let $G' = \alpha \cdot G$ be such that $\kappa_{G'}(u, v) = \alpha \cdot \kappa_G(u, v)$ for all u, v.*

From the definition, H is embeddable into G with congestion q iff $H \preccurlyeq^{\text{flow}} q \cdot G$.
The partial order also has following properties.

- $G \preccurlyeq^{\text{flow}} G$
- *If $G_1 \preccurlyeq^{\text{flow}} G_2$ and $G_2 \preccurlyeq^{\text{flow}} G_3$, then $G_1 \preccurlyeq^{\text{flow}} G_3$.*
- $G_1 \preccurlyeq^{\text{flow}} G_2$ *iff* $G_1 + H \preccurlyeq^{\text{cut}} G_2 + H$.
- $G_1 \preccurlyeq^{\text{flow}} \alpha \cdot G_2$ *iff* $\frac{1}{\alpha} G_1 \preccurlyeq^{\text{flow}} G_2$.

4. The Maximum Concurrent Flow Problem

4.1. Definition (**Maximum concurrent flow** problem)**.** Given a graph G and a demand graph H, we need to compute a feasible $\tau \cdot H$-concurrent flow in G with maximum τ. We denote the optimal value τ^* by $\mathsf{mcf}(G, H) = \max\{\tau \mid \tau \cdot H \preccurlyeq^{\text{flow}} G\}$. In particular, we have $H \preccurlyeq^{\text{flow}} G \iff \mathsf{mcf}(G, H) \geq 1$.

Note that this problem is a generalization of the single-source single-sink maximum flow problem. Let H_{st} be the graph containing only one edge (s, t) with capacity 1. $\mathsf{mcf}(G, H)$ is exactly the value of maximum (s, t)-flow and $\Phi(G, H)$ is the value of minimum (s, t)-cut. The classical max-flow min-cut theorem states the following.

4.2. Theorem (Max-flow min-cut theorem)**.** *For any graph G where $s, t \in V(G)$, we have* $\mathsf{mcf}(G, H_{st}) = \Phi(G, H_{st})$.

5. An Approximate Max-flow Min-cut theorem

5.1. Theorem (Approximate Maxflow Mincut Theorem)**.** *Given undirected graphs G and H,*

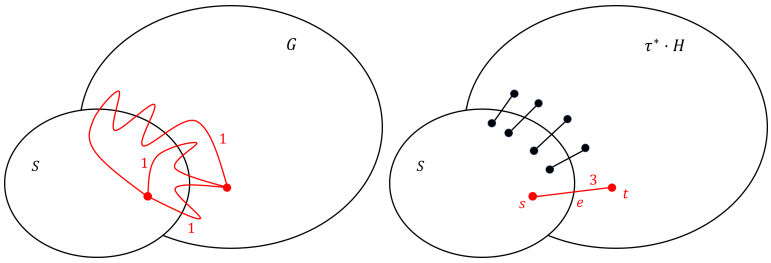
$$\mathsf{mcf}(G, H) \leq \Phi(G, H) \leq O(\log n) \cdot \mathsf{mcf}(G, H).$$

proof of one direction of theorem 5.1, $\mathsf{mcf}(G, H) \leq \Phi(G, H)$.

- Let $\tau^* = \mathsf{mcf}(G, H)$ be the optimal value of the concurrent flow. Let $\mathcal{F}$ be the multi-commodity flow that embeds $\tau^* H$ to G.
- Consider any cut $S \subset V$.
- For each edge $(s, t) \in \partial_H S$, $\mathcal{F}$ sends at least $\tau^* \cdot \kappa_H(s, t)$ unit of flow crossing the cut S.
- So the total unit of flow crossing the cut S is $\tau^* \cdot \kappa_H(\partial_H S)$.
- But the total capacity of the cut is $\kappa_G(\partial_G S)$. So

  $$\tau^* \cdot \kappa_H(\partial_H S) \leq \kappa_G(\partial_G S).$$

  Formally, let $\mathcal{F} := \{f_{s,t}\}_{(s,t)\in E(H)}$ be the feasible flow that embeds $\tau^* \cdot H$ into G. $\mathsf{cong}_{\mathcal{F}} \leq 1$ implies that $\forall e \in E(G), \sum_{(s,t)\in H} \sum_{P \ni e} f_{s,t}(P) \leq \kappa_G(e)$. Since $LHS = \sum_{(s,t)\in\partial_H S} \sum_{P\ni e} f_{s,t}(P) \leq \sum_{(s,t)\in H} \sum_{P\ni e} f_{s,t}(P) \leq \kappa_G(e) = RHS$, summing over all edges in $\partial_G S$ for both sides, we have $LHS \geq \sum_{(s,t)\in\partial_H S} \sum_{P\cap\partial_G S\neq\emptyset} f_{s,t}(P) = \sum_{(s,t)\in\partial_H S} \sum_{P\in\mathcal{P}_{s,t}} f_{s,t}(P) = \sum_{(s,t)\in\partial_H S} \tau^* \cdot \kappa_H(s, t) = \tau^* \cdot \kappa_H(\partial_H S)$; $RHS = \kappa_G(\partial_G S)$. $\implies \tau^* \cdot \kappa_H(\partial_H S) \leq \kappa_G(\partial_G S)$.
  See figure 3 for an intuitive proof.



3. ábra. Let the left figure represent G and the right figure represent $\tau^* \cdot H$. The capacity of the cut S in $\tau^* \cdot H$ is $\tau^* \cdot \kappa_H(\partial_H S)$, represented by five edges in the right figure. For each edge in $\tau^* \cdot H$ crossing the cut, it is embedded into several paths in G and each path will cross the cut in G at least on time. Take the red edge e with $\kappa_H(e) = 3$ as an example. It is embedded into three paths each of which has flow 1, which means that at least 3 units of capacity of the cut in G are owned by e. It turns out that $\tau^* \cdot \kappa_H(\partial_H S) \leq \kappa_G(\partial_G S)$.

- Therefore, we have

$$\mathsf{mcf}(G, H) = \tau^* \leq \min_S \frac{\kappa_G(\partial_G S)}{\kappa_H(\partial_H S)} = \Phi(G, H)$$

□

We prove another direction of theorem 5.1 (i.e. $\Phi(G, H) \leq O(\log n) \cdot \mathsf{mcf}(G, H)$) via LP duality. In fact, we will show that the optimal value of concurrent flow $\mathsf{mcf}(G, H)$ is exactly the same as the optimal value from the Leighton-Rao relaxation $\mathrm{LR}(G, H)$ of $\Phi(G, H)$ (see LP 1), as lemma 5.2 says. By lemma 5.2 and theorem 1.3, we immediately get

$$\Phi(G, H) \leq O(\log n) \cdot \mathrm{LR}(G, H) = (\log n) \cdot \mathsf{mcf}(G, H).$$

5.2. Lemma (Key lemma)**.** $\mathsf{mcf}(G, H) = \mathrm{LR}(G, H)$.

proof of one direction of theorem 5.1, $\Phi(G, H) \leq O(\log n) \cdot \mathsf{mcf}(G, H)$. Because $\mathsf{mcf}(G, H) = \mathrm{LR}(G, H)$ by lemma 5.2 and $\Phi(G, H) \leq O(\log n) \cdot \mathrm{LR}(G, H)$ by theorem 1.3, we immediately get $\Phi(G, H) \leq O(\log n) \cdot \mathsf{mcf}(G, H)$.

□

6. Proof of Lemma 5.2

We start by modeling the maximum concurrent flow problem as an LP.

- The problem is fully specified given **parameters** $\{C_{uv}\}_{(u,v)\in\binom{V}{2}}$ and $\{D_{uv}\}_{(u,v)\in\binom{V}{2}}$ defined as follows.
  - Let $C_{uv} = \kappa_G(u, v)$ be the capacity of edge (u, v) in G. If (u, v) is not in G, then $C_{uv} = 0$.
  - Let $D_{uv} = \kappa_H(u, v)$ be the demand of from u to v defined by H. If (u, v) is not in H, then $D_{uv} = 0$.

- Recall definition 4.1 of the maximum concurrent flow problem and definition 2.2 of concurrent flows. The **variables** of this LP are the scaling factor τ and flow assignments $f(P)$ to all (u, v)-paths for all u, v.

$$
\begin{aligned}
& \text{(Primal } \mathsf{P}_{LP}\text{)} \\
\max \quad & \tau \\
\text{s.t.} \quad & D_{uv} \cdot \tau - \sum_{P \in \mathcal{P}_{u,v}} f(P) \leq 0 && \forall u, v \\
& \sum_{P \ni (u,v)} f(P) \leq C_{uv} && \forall u, v \\
& f(P) \geq 0 && \forall P \in \mathcal{P}_{uv} \text{ over all } u, v
\end{aligned}
$$

We let $y(u, v)$ be variables in the dual corresponding to the first type of constraints in the primal LP and $x(u, v)$ be variables corresponding to the second type of constraints in the primal LP. We immediately get the dual LP as follows.

$$
\begin{aligned}
& \text{(Dual } \mathsf{D}_{LP}\text{)} \\
\min \quad & \sum_{u,v} C_{uv} \cdot x(u, v) \\
\text{s.t.} \quad & \sum_{u,v} D_{uv} \cdot y(u, v) \geq 1 \\
& -y(u, v) + \sum_{(u',v') \in P} x(u', v') \geq 0 && \forall P \in \mathcal{P}_{uv} \text{ over all } u, v \\
& x(u, v), y(u, v) \geq 0 && \forall u, v
\end{aligned}
$$

6.1. *Remark.* Deriving dual LPs is a very important skill for researching in TCS. To interpret the problem that D_{LP} is solving, we can treat $x(u, v)$ as "the length of edge uv", C_{uv} as "the capacity or width of the edge uv", and $y(u, v)$ as the "distance between u and v" since it plays the role of a lower bound of the length of paths between u and v. The objective function is trying to minimize the "total volume" of the graph, yet still guarantee the distances between vertices are big enough.

By definition, $\mathsf{mcf}(G, H)$ is exactly the optimal value of P_{LP}, and by strong duality, we have $\text{OPT}(\mathsf{P}_{LP}) = \text{OPT}(\mathsf{D}_{LP})$, which means $\mathsf{mcf}(G, H) = \text{OPT}(\mathsf{D}_{LP})$. In what follows, we will show that $\text{OPT}(\mathsf{D}_{LP}) = \text{OPT}(\text{LR})$, where LR is LP 1 in definition 1.2 (see below for an equivalent version).

$$
\begin{aligned}
& \text{(LR)} \\
& \min \sum_{u,v} C_{uv} \cdot d(u, v) \\
& \text{s.t.} \sum_{u,v} D_{uv} \cdot d(u, v) \geq 1 \\
& \quad d(u, v) \leq d(u, w) + d(w, v) && \forall u, v, w
\end{aligned}
$$

$$d(u,v) \geq 0 \qquad \forall u, v$$

By lemma 6.3 and definition 1.2, we immediately have

$$\mathsf{mcf}(G,H) = \mathrm{OPT}(\mathsf{D}_{LP}) = \mathrm{OPT}(\mathrm{LR}) = \mathrm{LR}(G,H).$$

6.2. Fact. *Let* $\mathrm{dist}_x(u,v)$ *be the distance metric defined by* x*. We have* $y(u,v) \leq \mathrm{dist}_x(u,v) \leq x(u,v)$ *for all* u, v

6.3. Lemma. $\mathrm{OPT}(\mathrm{LR}) = \mathrm{OPT}(\mathsf{D}_{LP})$.

Proof of $\mathrm{OPT}(\mathrm{LR}) \leq \mathrm{OPT}(\mathsf{D}_{LP})$.

Let (x, y) be an optimal solution for D_{LP}. Let $d(u,v) = \mathrm{dist}_x(u,v)$ for all u, v. We have $d(u,v)$ also a feasible solution for LR because

- $\sum_{u,v} D_{uv} \cdot d(u,v) \geq \sum_{u,v} D_{uv} \cdot y(u,v) \geq 1$ and
- the triangle inequality is trivially satisfied as d is a metric.

Furthermore,

$$\mathrm{OPT}(\mathrm{LR}) \leq \sum_{u,v} C_{uv} \cdot d(u,v) \leq \sum_{u,v} C_{uv} \cdot x(u,v) = \mathrm{OPT}(\mathsf{D}_{LP})$$

□

Proof of $\mathrm{OPT}(\mathrm{LR}) \geq \mathrm{OPT}(\mathsf{D}_{LP})$.

For convenience, we write $d(P) = \sum_{(u',v') \in P} d(u', v')$, and the constraint in D_{LP} can be rewritten as $x(P) \geq y(u,v)\ \forall P \in \mathcal{P}_{uv}$.

Obviously, for any function $d : V \times V \to \mathbb{R}_{\geq 0}$, if $d(u,v) \leq d(u,w) + d(w,v)$ for all u, v, w (i.e. d satisfies the triangle inequality), then for all u, v and $P \in \mathcal{P}_{uv}$, we have $d(u,v) \leq d(P)$.

Let d be an optimal solution for LR. Let $x(u,v) = y(u,v) = d(u,v)$ for all u, v. We have

- $\sum_{u,v} D_{uv} \cdot y(u,v) = \sum_{u,v} D_{uv} \cdot d(u,v) \geq 1$ and
- $y(u,v) = d(u,v) \leq d(P) = x(P)$ for all u, v and $P \in \mathcal{P}_{uv}$.

Furthermore,

$$\mathrm{OPT}(\mathsf{D}_{LP}) \leq \sum_{u,v} C_{uv} \cdot x(u,v) = \sum_{u,v} C_{uv} \cdot d(u,v) = \mathrm{OPT}(\mathrm{LR})$$

□

7. The Flow-Cut Gap

Actually, the factor $O(\log n)$ in the approximate max-flow min-cut theorem is tight. This is called the **flow-cut gap**.

7.1. Theorem. *There exist* G *and* H *where* $\mathsf{mcf}(G,H) \leq \frac{1}{\Omega(\log n)} \cdot \Phi(G,H)$.

Sketch of proof. The proof is from problem 5 in homework 1.

Let K be a complete graph with n nodes where each edge has weight $1/n$. Let G be an $\Omega(1)$-expander with constant maximum degree. We have $\Phi(G, K) \geq \Omega(1)$.

Because G is a graph with bounded degree, there will be at least $\Omega(n^2)$ pairs (u, v) of nodes where $\text{dist}_G(u, v) \geq \Omega(\log n)$. Let $\mathcal{F}$ be a concurrent flow that embeds K into G and define the volume of $\mathcal{F}$ as $\sum_{P \in \mathcal{F}} f(P) \cdot |P|$, where $f(P)$ is the flow value along the path P and $|P|$ is the number of edges in the path. The volume of $\mathcal{F}$ will be at least $\Omega(n \log n)$. Therefore, $\mathsf{mcf}(G, K) \leq O(1/\log n)$.

□

8. Partial Order based on Cut

In section 3, we define partial order based on concurrent flow. In this section, we define another partial order based on cut. We will see that these two partial orders are essentially equivalent.

8.1. Definition (Partial order based on cut)**.** Consider two undirected weighted graphs $G = (V, E_G, \kappa_G)$ and $H = (V, E_H, \kappa_H)$.

We define $H \preccurlyeq^{\text{cut}} G$ if for all $S \subset V$, we have $\kappa_H(\partial_H S) \leq \kappa_G(\partial_G S)$. In this case, we say that H **is cut-wise at most** G.

8.2. Fact. $H \preccurlyeq^{\text{cut}} G \iff \Phi(G, H) \geq 1$.

For any graph G and G'

- *let $G'' = G + G'$ be such that $\kappa_{G''}(u, v) = \kappa_G(u, v) + \kappa_{G'}(u, v)$ for all u, v and*
- *let $G' = \alpha \cdot G$ be such that $\kappa_{G'}(u, v) = \alpha \cdot \kappa_G(u, v)$.*

Similar to the partial order based on concurrent flow, the partial order based on cut also has following properties.

- *If $G_1 \preccurlyeq^{\text{cut}} G_2$ and $G_2 \preccurlyeq^{\text{cut}} G_3$, then $G_1 \preccurlyeq^{\text{cut}} G_3$.*
- *$G_1 \preccurlyeq^{\text{cut}} G_2$ iff $G_1 + H \preccurlyeq^{\text{cut}} G_2 + H$.*
- *$G_1 \preccurlyeq^{\text{cut}} \alpha \cdot G_2$ iff $\frac{1}{\alpha} G_1 \preccurlyeq^{\text{cut}} G_2$.*

We have seen two ways to compare graphs $\preccurlyeq^{\text{cut}}$ and $\preccurlyeq^{\text{flow}}$. The two relations are basically the same because of the approximate max-flow min-cut theorem. Roughly speaking, for two graphs G and H,

- If there exists a sparse cut in G w.r.t. H, then we cannot route the H-demand.
- If there is no sparse cut in G w.r.t. H, then we can route the H-demand with small congestion.

See lemma 8.3 for a formal description.

8.3. Lemma. *Consider two weighted undirected graphs G and H.*

- *If $H \preccurlyeq^{\text{flow}} G$, then $H \preccurlyeq^{\text{cut}} G$.*

- *If* $H \preccurlyeq^{\text{cut}} G$*, then* $H \preccurlyeq^{\text{flow}} O(\log n) \cdot G$.

Proof.

- Recall the approximate max-flow min-cut theorem.

$$\mathsf{mcf}(G, H) \leq \Phi(G, H) \leq O(\log n) \cdot \mathsf{mcf}(G, H).$$

- Recall that $H \preccurlyeq^{\text{flow}} G \iff \mathsf{mcf}(G, H) \geq 1$ and $H \preccurlyeq^{\text{cut}} G \iff \Phi(G, H) \geq 1$.
- If $H \preccurlyeq^{\text{flow}} G$, then $\Phi(G, H) \geq \mathsf{mcf}(G, H) \geq 1$ and so $H \preccurlyeq^{\text{cut}} G$.
- If $H \preccurlyeq^{\text{cut}} G$, then $\Phi(G, H) \geq 1$. So $\mathsf{mcf}(G, H) \geq \frac{1}{O(\log n)}$. So $H \preccurlyeq^{\text{flow}} O(\log n) \cdot G$.

□

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 7: Cheeger's inequality

September 21, 2021

Instructor: Thatchaphol Saranurak

Scribe: Jingyi Gao

1. Recap

So far, we've seen several characterization of expanders. Intuitively, expanders are graphs that are robust against any adversarial deletions. From the perspective of cut, when we compare all graphs with the same degree profile, the size of every cut of expander is almost maximum among them. On the other hand, regarding flows, for all graphs with a degree profile no greater than the expander, they are embeddable into expander with small congestion.
All of the three characterization of expander are combinatorial. Today, we are going to introduce an algebraic way to characterize expanders based on **eigenvalue**s. This invokes an area called **spectral graph theory.** Roughly speaking, in this area, we associate a graph G with some matrix M and relate eigenvalues of M to combinatorial properties of G.
A very nice resource is the textbook by Spielman [1].

2. Review of Linear Algebra

We start by reviewing some linear algebra facts.

2.1. Theorem. *Let $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ be a symmetric real matrix. Then there exists $\lambda_1, ..., \lambda_n \in \mathbb{R}$ and orthonormal vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^n$ such that*

- *For all i, $\boldsymbol{M}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$.*
- *In fact, $\boldsymbol{M} = \sum_i \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^\top$*

We call $\lambda_1, ..., \lambda_n$ the eigenvalues *of $\boldsymbol{M}$ and each $\boldsymbol{v}_i$ is called the* eigenvector *corresponding to λ_i.*

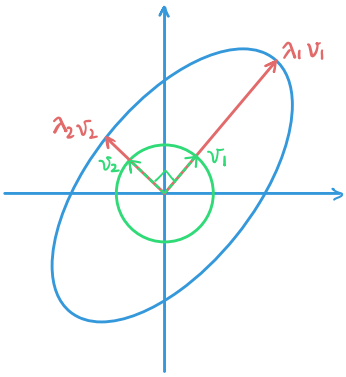
2.2. *Remark.*

- A useful way to think of the relation between $\boldsymbol{M}$ and its eigenvectors and eigenvalues is that any real symmetric matrix $\boldsymbol{M}$ corresponds to an ellipsoid. Consider the image of $\boldsymbol{M}C$, where $C = \{x \in \mathbb{R}^n : |x| = 1\}$. Let $x \in C$ and we can write x in terms of basis

[1] http://cs-www.cs.yale.edu/homes/spielman/sagt/sagt.pdf

$\{v_1, \cdots, v_n\}$ (an orthonormal basis consists of all eigenvectors of $\boldsymbol{M}$,) $x = x_1 v_1 + \cdots + x_n v_n$, where $\sum_{i=1}^n x_i^2 = 1$. Then $y = \boldsymbol{M}x = \sum_{i=1}^n x_i \lambda_i v_i$. This is to say that the component of the image point $y = \boldsymbol{M}x$ has components $y_i = x_i \lambda_i$ in terms of the basis v_i. Hence the image of $\boldsymbol{M}x$ is on the ellipsoid $\left\{ \sum_{i=1}^n y_i v_i : \sum_{i=1}^n \left(\frac{y_i}{\lambda_i}\right)^2 = 1 \right\} = \boldsymbol{M}C$ with axes along the vectors $v_i, i = 1, \cdots, n$ with length of the axes being $\lambda_i, i = 1, \cdots, n$, respectively.

- In short, any $\boldsymbol{M} \in \mathbb{SR}^{n \times n}$ corresponds to an ellipsoid, which is the image of unit circle under mapping $\boldsymbol{M}$. Each eigenvector v_i is the $i-$th axis of the ellipsoid and each eigenvalue λ_i describes how much we stretch the unit circle along the $i-$th axis. A picture for $n = 2$ is as below.



2.3. Theorem (Variational Characterization of Eigenvalues). *Let $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ be a symmetric real matrix with eigenvalues $\lambda_1 \leq \ldots \leq \lambda_n$ and eigenvectors $v_1, \ldots, v_n$. Then*

- $\lambda_1 = \min_{x \neq 0} \frac{x^\top \boldsymbol{M} x}{x^\top x}$
- $\lambda_n = \max_{x \neq 0} \frac{x^\top \boldsymbol{M} x}{x^\top x}$
- $\lambda_2 = \min_{x \perp v_1, x \neq 0} \frac{x^\top \boldsymbol{M} x}{x^\top x} = \min_{2\text{-}dim\, \mathcal{V}} \max_{x \in \mathcal{V} \setminus \{0\}} \frac{x^\top \boldsymbol{M} x}{x^\top x}$
- *In general, we have*

$$\lambda_k = \min_{k\text{-}dim\, \mathcal{V}} \max_{x \in \mathcal{V} - \{0\}} \frac{\boldsymbol{x}^\top \boldsymbol{M} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} = \min_{x \perp \text{span}\{v_1, \ldots v_{k-1}\}, x \neq 0} \frac{\boldsymbol{x}^\top \boldsymbol{M} \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}}$$

We call $R_M(\boldsymbol{x}) = \frac{x^\top \boldsymbol{M} x}{x^\top x}$ the Rayleigh quotient of x w.r.t. $\boldsymbol{M}$.

3. The Laplacian of Graphs

Given an undirected graph $G = (V, E, w)$ with non-negative weights $w : E \to \mathbb{R}_{\geq 0}$, the *adjacency matrix* $\boldsymbol{A}$ is such that $\boldsymbol{A}_{u,v} = w_{u,v}$ for all $(u, v) \in E$. Note that since the graph is undirected, the matrix is symmetric. The *degree matrix* $\boldsymbol{D}$ is a diagonal matrix such that $\boldsymbol{D}_{uu} = \boldsymbol{d}(u) = \sum_v w_{uv}$ for all $u \in V$, where $\boldsymbol{d}(u) = \deg_G(u)$ is the short hand for degree, and $\boldsymbol{d}(S) = \text{vol}_G(S)$ is the volume of

the vertex set S.
The **Laplacian matrix** is defined as the difference of these two matrices: $\boldsymbol{L}_G = \boldsymbol{D} - \boldsymbol{A}$.[2]

3.1. Fact. *$\boldsymbol{L}_G$ is a real symmetric matrix. Therefore, all the eigenvalues of $\boldsymbol{L}_G$ are real.*[3]

Let $\boldsymbol{L}_{(u,v)}$ be the Laplacian of the subgraph of G that contains only one unweighted edge (u, v). Then in this case, since

$$D = \begin{array}{c} \begin{array}{ccccccccccc} & \cdots & & u & & \cdots & & v & & \cdots & \end{array} \\ \begin{pmatrix} 0 & & & & & \cdots & & & & & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & & & & & \cdots & & & & & 0 \end{pmatrix} \end{array} \begin{array}{l} \vdots \\ \\ u \\ \\ \vdots \\ \\ v \\ \\ \vdots \end{array}$$

$$A = \begin{array}{c} \begin{array}{ccccccccccc} & \cdots & & u & & \cdots & & v & & \cdots & \end{array} \\ \begin{pmatrix} 0 & & & & & \cdots & & & & & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & & & & & \cdots & & & & & 0 \end{pmatrix} \end{array} \begin{array}{l} \vdots \\ \\ u \\ \\ \vdots \\ \\ v \\ \\ \vdots \end{array}$$

[2]Note that L_G does not change when we add self-loops.

[3]In fact, it is quite easy to see that L_G is positive semidefinite (this can be show by endowing the graph an arbitrary direction, then $\boldsymbol{L}_G$ is the incidence matrix multiply by its transpose) and hence all its eigenvalues are non-negative.

the Laplacian of a single-edge subgraph

$$\boldsymbol{L}_{(u,v)} = \boldsymbol{D} - \boldsymbol{A} = \begin{array}{c} \begin{matrix} & \cdots & & u & & \cdots & & v & & \cdots & \end{matrix} \\ \begin{pmatrix} 0 & & & & & \cdots & & & & & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & -1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & & & & & & & & & & \vdots \\ 0 & & & & & \cdots & & & & & 0 \end{pmatrix} \end{array} \begin{matrix} \vdots \\ \\ u \\ \\ \vdots \\ \\ v \\ \\ \vdots \end{matrix}$$

which looks very simple.

By decomposing the whole graph into edges, the Laplacian of the graph $\boldsymbol{L}_G$ is the sum of these simple matrices:

3.2. Proposition. $\boldsymbol{L}_G = \sum_{e \in E} w_e \boldsymbol{L}_e$.

Since the Laplacian is a linear combination of Laplacian of a single-edge subgraph, $\boldsymbol{L}_{(u,v)}$ is important and we should take a closer look. Observe that for $\boldsymbol{L}_{(u,v)}$, we have $\forall \boldsymbol{x} \in \mathbb{R}^V$

$$\boldsymbol{x}^\top \boldsymbol{L}_{(u,v)} \boldsymbol{x} = x_u^2 - 2x_u x_v + x_v^2 = (x_u - x_u)^2$$

whence

$$\mathbf{1}_S^\top \boldsymbol{L}_{(u,v)} \mathbf{1}_S = \begin{cases} 1 & \text{if } |\{u,v\} \cap S| = 1 \text{ (i.e. } \{u,v\} \text{ is cut)} \\ 0 & \text{otherwise} \end{cases}$$

Since $\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x} = \sum_{e \in E} w_e (\boldsymbol{x}^\top \boldsymbol{L}_e \boldsymbol{x})$, we can conclude that

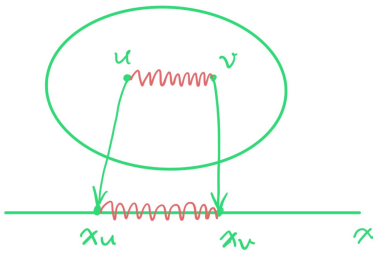
3.3. Proposition. *$\forall \boldsymbol{x} \in \mathbb{R}^V$, we have*

$$\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x} = \sum_{(u,v) \in E} w_{uv} (x_u - x_v)^2.$$

In particular, if $x = \mathbf{1}_S$, we have

$$\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x} = w_G(\partial_G S).$$

3.4. *Note.* A intuitive interpretation of this proposition is that we can think of in the original graph, all edges (u,v) are springs with spring constant $w_{(u,v)}$. We use $\boldsymbol{x}$ to keep track of each edge (u,v), in the way that $(x_u - x_v)^2$ scaled by the "spring constant," namely the edge weight, is the energy of the spring corresponds to (u,v). $\boldsymbol{x}^T \boldsymbol{L}_G \boldsymbol{x}$ stands for the total spring energy in the graph G.



3.5. Proposition. *Let $\lambda_1(\boldsymbol{L}_G) \leq \cdots \leq \lambda_n(\boldsymbol{L}_G)$ be the eigenvalues of $\boldsymbol{L}_G$. We have*

1. *$\lambda_1(\boldsymbol{L}_G) = 0$ with a corresponding eigenvector $\boldsymbol{v}_1 = \mathbf{1}$.*
2. *$\lambda_k(\boldsymbol{L}_G) = 0$ if and only if G contains at least k connected components.*

Bizonyítás.

1. By definition,
$$\lambda_1(\boldsymbol{L}_G) = \min_{\boldsymbol{x}\neq\mathbf{0}} R_{\boldsymbol{L}_G}(\boldsymbol{x}) = \min_{\boldsymbol{x}\neq\mathbf{0}} \frac{\sum_{(u,v)\in E} w_{uv}(x_u - x_v)^2}{\boldsymbol{x}^\top \boldsymbol{x}}.$$
Since $(x_u - x_v)^2 \geq 0$, we have $\lambda_1(\boldsymbol{L}_G) \geq 0$. On the other hand, if we let $\boldsymbol{x} = \mathbf{1}$, then $R_{\boldsymbol{L}_G}(\mathbf{1}) = 0$. Therefore, as the minimum over all possible $\mathbb{R}_{\boldsymbol{L}_G}(x)$, $\lambda_1(\boldsymbol{L}_G) \leq R_{L_G}(\mathbf{1}) = 0$. Hence $\lambda_1(\boldsymbol{L}_G) = 0$. To find its corresponding eigenvector, as previously mentioned on page 3, the Laplacian of the subgraph with a single unweighted edge has exactly 2 non-zero rows containing exactly one 1 and one -1 each. Hence when we multiply this matrix by an all one vector, we will get zero vector. Therefore, $\boldsymbol{L}_G\mathbf{1} = \sum_{e\in E} w_e \boldsymbol{L}_e\mathbf{1} = \mathbf{0}$. Hence the eigenvector corresponding to the eigenvalue 0 is $\boldsymbol{v}_1 = \mathbf{1}$.

   Another way to find $\boldsymbol{v}_1 = \mathbf{1}$ is by eyeballing $\boldsymbol{L}_G\mathbf{1} = (\boldsymbol{D} - \boldsymbol{A})\mathbf{1} = \boldsymbol{D}\mathbf{1} - \boldsymbol{A}\mathbf{1}$. The i^{th} row of $\boldsymbol{D}\mathbf{1}$ is the degree of u, and the i^{th} row of $\boldsymbol{A}\mathbf{1}$ is the sum of the weight of all edges adjacent to u, which is also the degree of u. Therefore for any arbitrary i, the i^{th} row of $(\boldsymbol{D} - \boldsymbol{A})\mathbf{1}$ is 0, whence $(\boldsymbol{D} - \boldsymbol{A})\mathbf{1} = \mathbf{0}$.

2. For $\lambda_k(\boldsymbol{L}_G)$, recall that
$$\lambda_k(\boldsymbol{L}_G) = \min_{k\text{-dim }\mathcal{V}} \max_{\boldsymbol{x}\in\mathcal{V}-\{\mathbf{0}\}} R_{L_G}(\boldsymbol{x})$$
   " $\Longleftarrow$ " Suppose there are k connected components $C_1, \ldots, C_k$.
   Let $\mathcal{V} = \text{span}\{1_{C_1}, \ldots 1_{C_k}\}$. Note that $\dim(\mathcal{V}) = k$. For any $\boldsymbol{x} \in \mathcal{V}$ and for each component C_i, entries of $\boldsymbol{x}$ correspond to vertices in C_i are the same constant since by definition of $\mathcal{V}$, we can represent any $\boldsymbol{x} \in \mathcal{V}$ by using basis $\boldsymbol{x} = \sum_{i=1}^{k} \mathbf{1}_{C_i}$. Therefore $\sum_{(u,v)\in E(C_i)} w_{uv}(x_u - x_v)^2 = 0$. Since there's no edges between each connected components $R_{\boldsymbol{L}_G}(\boldsymbol{x}) = \sum_{i=1}^{k} \sum_{(u,v)\in E(C_i)} w_{uv}(x_u - x_v)^2 = 0$, and $\lambda_k(\boldsymbol{L}_G) \leq 0$. Combining with the fact that $\lambda_k(\boldsymbol{L}_G)$ is non-negative, it is 0
   " $\Longrightarrow$ " Suppose $\lambda_k(\boldsymbol{L}_G) = 0$.
   Let $\mathcal{V}$ be a k-dimensional space where $\max_{\boldsymbol{x}\in\mathcal{V}-\{\mathbf{0}\}} R_{\boldsymbol{L}_G}(\boldsymbol{x}) = 0$. For any $\boldsymbol{x} \in \mathcal{V}$ and for each component C_i, entries of $\boldsymbol{x}$ in C_i must be constant. Otherwise, $R_{\boldsymbol{L}_G}(\boldsymbol{x}) > 0$. Therefore $\mathcal{V} \subseteq \text{span}\{1_{C_1}, \ldots 1_{C_z}\}$ where z counts the connected components in G. So $k = \dim(\mathcal{V}) \leq z$, which in English means that the number of connected components of G is at least k.

□

3.6. *Remark.* Basically, $\lambda_1(\boldsymbol{L}_G)$ is not informative at all since it is always zero. But $\lambda_2(\boldsymbol{L}_G) > 0$ iff G is connected.

In other words, if $\lambda_2 = 0$, then the graph is not connected. A intuitive question to ask is that what if λ_2 is very closed to 0, would that means that the graph is very closed to be disconnected? And the answer is yes. We'll see the proof in the following section.

4. The Normalized Laplacian of Graphs

4.1. Definition. Excluding isolated vertices(namely $\boldsymbol{D}$ is of full rank,) the **normalized Laplacian** matrix is

$$\boldsymbol{N}_G = \boldsymbol{D}^{-1/2}\boldsymbol{L}_G\boldsymbol{D}^{-1/2} = I - \boldsymbol{D}^{-1/2}\boldsymbol{A}_G\boldsymbol{D}^{-1/2}.$$

Basically a scaled Laplacian.

4.2. Exercise. We have

1. $\lambda_1(\boldsymbol{N}_G) = 0$ with a corresponding eigenvector $\boldsymbol{v}_1 = \boldsymbol{d}^{1/2}$, where $\boldsymbol{d} = (\deg(u))_{u\in V(G)}$.
2. $\lambda_k(\boldsymbol{N}_G) = 0$ if and only if G contains at least k connected component.

4.3. Lemma.

$$\lambda_2(\boldsymbol{N}_G) = \min_{\boldsymbol{x}\perp\boldsymbol{d}} \frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}.$$

Bizonyítás.

Firstly, by definition

$$\lambda_2(\boldsymbol{N}_G) = \min_{\boldsymbol{v}\perp\boldsymbol{d}^{1/2}} \frac{\boldsymbol{v}^\top \boldsymbol{N}_G \boldsymbol{v}}{\boldsymbol{v}^\top \boldsymbol{v}}$$

since the first eigenvector of $\boldsymbol{N}$ is $\boldsymbol{d}^{1/2}$. Now, define $\boldsymbol{x} = \boldsymbol{D}^{-1/2}\boldsymbol{v}$, so $\boldsymbol{v} = \boldsymbol{D}^{1/2}\boldsymbol{x}$. We have $\frac{\boldsymbol{v}^\top \boldsymbol{N}_G \boldsymbol{v}}{\boldsymbol{v}^\top \boldsymbol{v}} = \frac{\boldsymbol{v}^\top \boldsymbol{D}^{-1/2}\boldsymbol{L}_G\boldsymbol{D}^{-1/2}\boldsymbol{v}}{\boldsymbol{v}^\top \boldsymbol{v}} = \frac{\boldsymbol{x}\boldsymbol{L}_G\boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}$. Also, $\boldsymbol{v}\perp\boldsymbol{d}^{1/2}$ iff $\boldsymbol{x}\perp\boldsymbol{d}$. Because $0 = \boldsymbol{v}^\top\boldsymbol{d}^{1/2} = \boldsymbol{x}^\top\boldsymbol{D}^{1/2}\boldsymbol{d}^{1/2} = \boldsymbol{x}^\top\boldsymbol{d}$. Since the mapping $\boldsymbol{v} \mapsto \boldsymbol{D}^{-1/2}\boldsymbol{v} = \boldsymbol{x}$ is a bijection, we have $\lambda_2(\boldsymbol{N}_G) = \min_{\boldsymbol{x}\perp\boldsymbol{d}} \frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}$ as desired. □

4.4. *Note.* The above lemma is the reason why we look at $\boldsymbol{N}_G$-the second eigenvalue of normalized Laplacian is closely related to the conductance since they are minimizing the same objective but over different domains. Indeed, we can see the similarity between $\lambda_2(\boldsymbol{N}_G)$ and $\Phi(G)$:

$$\lambda_2(\boldsymbol{N}_G) = \min_{\boldsymbol{x}\perp\boldsymbol{d}} \frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}$$

and

$$\Phi(G) = \min_{\boldsymbol{x}=\mathbf{1}_S : 0<\boldsymbol{d}(S)\le\boldsymbol{d}(V)/2} \frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}$$

where $\boldsymbol{x} = \mathbf{1}_S$, $\boldsymbol{x}^\top\boldsymbol{L}_G\boldsymbol{x} = w_G(\partial_G S)$ and $\boldsymbol{x}^\top\boldsymbol{D}\boldsymbol{x} = \boldsymbol{d}(S)$.

5. Cheeger's Inequality: Statement

It turns out that $\lambda_2(\boldsymbol{N}_G)$ does not only tell us if G is connected or not. It actually measures how much G is well-connected.

5.1. Theorem (Cheeger's Inequality)**.** *We have*

$$\frac{\lambda_2(\boldsymbol{N}_G)}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G)}.$$

Furthermore, there is an algorithm that find a cut S where $\Phi_G(S) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G)} \leq 2\sqrt{\Phi(G)}$.

5.2. *Note.*

1. What this means - G **is expander if and only if the second eigenvalue of $\boldsymbol{N}_G$ is big.**
2. So when $\Phi(G) = \Omega(1)$, this gives $O(1)$-approximation algorithm for $\Phi(G)$.
3. But the approximation can be very bad when $\Phi(G)$ is small. For example, when $\Phi(G) = O(\frac{1}{n})$, the algorithm return a cut where $\Phi_G(S) \leq O\left(\sqrt{\frac{1}{n}}\right)$ which can be much larger than $O(\frac{1}{n})$.

5.3. Question. *Is there a $O(1)$-approximation algorithm $\Phi(G)$ for any graph G? If so, this would refute the small-set expansion conjecture.*

- Best known approximation: $O(\sqrt{\log n})$ via ARV.
  - We saw $O(\log n)$-approx algorithm (which is tight because of the flow-cut gap).
- The Cheeger's Inequality is tight.
  - the second inequality: for cycle of size n, $\Phi(G) = \Theta(1/n)$ and $\lambda_2(\boldsymbol{N}_G) = O(1/n^2)$
  - the first inequality: for d-dimensional hypercube on $n = 2^d$ nodes, $\Phi(G) = 1/d$ and $\lambda_2(\boldsymbol{N}_G) = 2/d$
  - See proofs: https://lucatrevisan.github.io/teaching/expanders2016/lecture15.pdf

6. Easy Direction: Sparse Cut implies Small Eigenvalue

6.1. Lemma. $\lambda_2(\boldsymbol{N}_G)/2 \leq \Phi(G)$

Bizonyítás. Consider any $S \subset V$ where wlog $0 < \boldsymbol{d}(S) \leq \boldsymbol{d}(V)/2$. Define $\boldsymbol{y}$ based on S,

$$\boldsymbol{y} = \mathbf{1}_S - \sigma\mathbf{1}$$

where $\sigma = \boldsymbol{d}(S)/\boldsymbol{d}(V)$ is chosen so that $\boldsymbol{y} \perp \boldsymbol{d}$. We have

$$\lambda_2(\boldsymbol{N}_G) \leq \frac{\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y}}.$$

Note that $\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y} = w_G(\partial_G S)$, since $\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y} = \sum_{(u,v)\in E} w_{uv}(y_u - y_v)^2$ is translation invariant whence is the same as $\mathbf{1}_S \boldsymbol{L}_G \mathbf{1}_S = w_G(\partial_G S)$.
Next, we compute $\boldsymbol{y}^\top \boldsymbol{D}\boldsymbol{y}$

$$\begin{aligned}
\boldsymbol{y}^\top \boldsymbol{D}\boldsymbol{y} &= \sum_{u\in S} \boldsymbol{d}(u)(1-\sigma)^2 + \sum_{u\notin S} \boldsymbol{d}(u)\sigma^2 \\
&= \boldsymbol{d}(S)(1-\sigma)^2 + \boldsymbol{d}(V\setminus S)\sigma^2 \\
&= \boldsymbol{d}(S) - 2\sigma \boldsymbol{d}(S) + \sigma^2 \boldsymbol{d}(V) \\
&= \boldsymbol{d}(S)(1-\sigma) \\
&= \boldsymbol{d}(S)\frac{\boldsymbol{d}(V\setminus S)}{\boldsymbol{d}(V)} \geq \frac{\boldsymbol{d}(S)}{2}
\end{aligned}$$

So we have

$$\lambda_2(\boldsymbol{N}_G) \leq 2 \cdot \frac{w_G(\partial_G S)}{\boldsymbol{d}(S)}$$

for all S where $0 < \boldsymbol{d}(S) \leq \boldsymbol{d}(V)/2$. Therefore,

$$\lambda_2(\boldsymbol{N}_G) \leq 2 \cdot \Phi(G).$$

□

7. Harder direction: Small Eigenvalue implies Sparse Cut

7.1. Theorem. *Given any $x \perp \boldsymbol{d}$, we can find a cut S in G where*

$$\Phi_G(S) \leq \sqrt{2\frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}}$$

7.2. Corollary. $\Phi(G) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G)}$.

Sketch of proof of Theorem 7.1. We have $\lambda_2(\boldsymbol{N}_G) = \min_{x\perp \boldsymbol{d}} \frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}$.

- Remark that it will be necessary that the theorem works for any $x \perp \boldsymbol{d}$
  - because we will show how to find $x \perp \boldsymbol{d}$ where $\frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}$ is *almost* minimum in the next lecture.
- Given a vector $\boldsymbol{x}$, there are three main steps for constructing S.
  1. Firstly, we define a *centered* vector $\boldsymbol{y}$ where $\frac{\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{D}\boldsymbol{y}} \leq \frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}$,
  2. then, we further define *non-negative* vector $\boldsymbol{z}$ where $\frac{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}} \leq \frac{\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{D}\boldsymbol{y}}$
  3. $\boldsymbol{z}$ will have a nice enough structure such that it determines a *threshold cut* S where $\Phi_G(S) \leq \sqrt{2\frac{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}}}$

□

We proceed to elaborate on the three steps in the following sections.

7.1. Centering

Without loss of generality assume that

$$\boldsymbol{x}_1 \leq \boldsymbol{x}_2 \leq \cdots \leq \boldsymbol{x}_n.$$

7.3. Definition. Say y is *centered w.r.t.* $\boldsymbol{d}$ if

$$\sum_{u:y_u<0} \boldsymbol{d}(u) \leq \boldsymbol{d}(V)/2 \quad \text{and} \quad \sum_{u:y_u>0} \boldsymbol{d}(u) \leq \boldsymbol{d}(V)/2$$

Namely the total volume of both vertices with strictly less then zero value in $\boldsymbol{y}$ and vertices with strictly greater then zero value in $\boldsymbol{y}$ are less than a half.

Proof of the first step of Theorem 7.1. Given an arbitrary vector $\boldsymbol{x}$, we can make it into a centered vector by translation.

Let j be the minimum index where $\sum_{u=1}^{j} \boldsymbol{d}(u) \geq \boldsymbol{d}(V)/2$. Set

$$\boldsymbol{y} = \boldsymbol{x} - x_j \mathbf{1}$$

So we have $y_j = 0$, $\sum_{u:y_u<0} \boldsymbol{d}(u) \leq \boldsymbol{d}(V)/2$ and $\sum_{u:y_u>0} \boldsymbol{d}(u) \leq \boldsymbol{d}(V)/2$. Namely, $\boldsymbol{y}$ is centered w.r.t. $\boldsymbol{d}$. Also observe that

$$\frac{\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y}} \leq \frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D} \boldsymbol{x}}$$

because (a) the numerator is not changing at all since $\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y} = (\boldsymbol{x} - x_j \mathbf{1})^\top \boldsymbol{L}_G (\boldsymbol{x} - x_j \mathbf{1}) = \boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}$, as $\mathbf{1} \in \ker(\boldsymbol{L}_G)$;
(b) and the denominator can only possibly be larger, i.e. $\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y} \geq \boldsymbol{x}^\top \boldsymbol{D} \boldsymbol{x}$. To see this, let $\boldsymbol{v}_s = \boldsymbol{x} + s\mathbf{1}$, where s is a translating parameter. The minimum of $\boldsymbol{v}_s^\top \boldsymbol{D} \boldsymbol{v}_s$ is achieved at s for which $\boldsymbol{v}_s^\top \boldsymbol{d} = 0$. This is because if we take the derivative of this value $\frac{\partial(\boldsymbol{v}_s^\top \boldsymbol{D} \boldsymbol{v}_s)}{\partial s} = \mathbf{1}^\top \boldsymbol{D} \boldsymbol{v}_s + \boldsymbol{v}_s^\top \boldsymbol{D} \mathbf{1} = 2\boldsymbol{d}^\top \boldsymbol{v}_s$, we know that when $\boldsymbol{d}^\top \boldsymbol{v}_s$ reaches 0 the value reaches its minimum. But notice that $\boldsymbol{x}$ is orthogonal to $\boldsymbol{d}$, where we have $\boldsymbol{x}^\top \boldsymbol{d} = 0$ and hence we must have $\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y} \geq \boldsymbol{x}^\top \boldsymbol{D} \boldsymbol{x}$.

□

7.2. Non-negative

We next proceed to show the second step of the proof. Given the vector $\boldsymbol{y}$ above, we decompose it into two vectors, one comprising the positive entries and the other comprising the negative entries.

$$\boldsymbol{y} = \boldsymbol{y}^+ - \boldsymbol{y}^-$$

where

$$y_j^+ = \begin{cases} y_j & y_j > 0 \\ 0 & \text{otherwise} \end{cases} \text{ and } y_j^- = \begin{cases} -y_j & y_j < 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that $\boldsymbol{y}^+$ and $\boldsymbol{y}^-$ has several neat properties

- have disjoint non-zero entries,

- are non-negative, and
- are small w.r.t. $\boldsymbol{d}$. More precisely, $\sum_{u:y_u^+>0} \boldsymbol{d}(u) \leq \boldsymbol{d}(V)/2$ and $\sum_{u:y_u^->0} \boldsymbol{d}(u) \leq \boldsymbol{d}(V)/2$.

Suppose the following claim is true.

7.4. Claim.

$$\min\{\frac{(\boldsymbol{y}^+)^\top \boldsymbol{L}_G \boldsymbol{y}^+}{(\boldsymbol{y}^+)^\top \boldsymbol{D} \boldsymbol{y}^+}, \frac{(\boldsymbol{y}^-)^\top \boldsymbol{L}_G \boldsymbol{y}^-}{(\boldsymbol{y}^-)^\top \boldsymbol{D} \boldsymbol{y}^-}\} \leq \frac{\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y}}$$

Then we can simply take one of $\boldsymbol{y}^+$ and $\boldsymbol{y}^-$ that makes the inequality holds to be $\boldsymbol{z}$ where

$$\frac{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D} \boldsymbol{z}} \leq \frac{\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y}}$$

with $\boldsymbol{z}$ being non-negative and $\sum_{u:z_u>0} \boldsymbol{d}(u) \leq \boldsymbol{d}(V)/2$. It remains to prove the claim.

Proof of claim 7.4. We first show $\forall uv \in E, ((y_u^+ - y_v^+) - (y_u^- - y_v^-))^2 \geq \left((y_u^+ - y_v^+)^2 + (y_u^- - y_v^-)^2\right)$.
We will prove this by considering the contribution of every edge. Consider any edge (u, v). If both u and v such that $y_u, y_v \geq 0$ or $y_u, y_v \leq 0$, then (u, v) contributes the exact same to both sides of inequality.
Otherwise, if $y_u > 0 > y_v$, then (u, v) contributes $(y_u^+ + y_v^-)^2$ to the left, but $(y_u^+)^2 + (y_v^-)^2$ to the right, where we know the inequality $(y_u^+ + y_v^-)^2 \geq (y_u^+)^2 + (y_v^-)^2$ always holds.
Therefore, we have

$$\begin{aligned}
\frac{\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y}}{\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y}} &= \frac{\sum_{uv\in E} w_{uv}(y_u - y_v)^2}{\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y}} \\
&\quad \text{by decomposing and rearranging} \\
&= \frac{\sum_{uv\in E} w_{uv} \left((y_u^+ - y_v^+) - (y_u^- - y_v^-)\right)^2}{(\boldsymbol{y}^+)^\top \boldsymbol{D} \boldsymbol{y}^+ + (\boldsymbol{y}^-)^\top \boldsymbol{D} \boldsymbol{y}^- - 2((\boldsymbol{y}^-)^\top \boldsymbol{D} \boldsymbol{y}^+)} \\
&\quad \text{according to what we've just shown} \\
&\geq \frac{\sum_{uv\in E} w_{uv} \left((y_u^+ - y_v^+)^2 + (y_u^- - y_v^-)^2\right)}{(\boldsymbol{y}^+)^\top \boldsymbol{D} \boldsymbol{y}^+ + (\boldsymbol{y}^-)^\top \boldsymbol{D} \boldsymbol{y}^-} \\
&\quad \text{since } \forall a_1, a_2, b_1, b_2, \frac{a_1 + a_2}{b_1 + b_2} \geq \min\left(\frac{a_1}{b_1}, \frac{a_2}{b_2}\right) \\
&\geq \min\left\{\frac{\sum_{uv\in E} w_{uv}(y_u^+ - y_v^+)^2}{(\boldsymbol{y}^+)^\top \boldsymbol{D} \boldsymbol{y}^+}, \frac{\sum_{uv\in E} w_{uv}(y_u^- - y_v^-)^2}{(\boldsymbol{y}^-)^\top \boldsymbol{D} \boldsymbol{y}^-}\right\} \\
&= \min\left\{\frac{(\boldsymbol{y}^+)^\top \boldsymbol{L}_G \boldsymbol{y}^+}{(\boldsymbol{y}^+)^\top \boldsymbol{D} \boldsymbol{y}^+}, \frac{(\boldsymbol{y}^-)^\top \boldsymbol{L}_G \boldsymbol{y}^-}{(\boldsymbol{y}^-)^\top \boldsymbol{D} \boldsymbol{y}^-}\right\}
\end{aligned}$$

□

7.3. Threshold Cuts

It suffices to show the following lemma,

7.5. Lemma. *Given a non-negative $\boldsymbol{z}$ as above, there is a number τ where $S_\tau = \{u \mid \boldsymbol{z}_u \geq \tau\}$ such that*

$$\Phi_G(S_\tau) \leq \sqrt{2\frac{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}}}$$

7.6. *Note.* Visually, when we set all entries of $\boldsymbol{y}$ on a real axis, if $\boldsymbol{z} = \boldsymbol{y}^-$, then the cut S_τ would be a cut includes all vertices correspond to points including everything from the left most on the axis all the way through the smallest u such that $\boldsymbol{z}_u \geq \tau$. This is because when we defined $\boldsymbol{y}^-$ we flipped the sign of each entry contained in this part in $\boldsymbol{y}$;
Conversely, when $\boldsymbol{z} = \boldsymbol{y}^+$, then the cut S_τ would includes all vertices correspond to points including everything from the right most on the axis all the way through the smallest u such that $\boldsymbol{z}_u \geq \tau$.

To show the Lemma, w.l.o.g we may assume $\max_u z_u = 1$ since scaling does not change the ratio $\frac{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}}$. We are going to use a probabilistic argument, where we first let $\tau > 0$ be sampled such that τ^2 **is uniformly distributed in** $[0, 1]$. Namely, $\Pr[\tau \leq \alpha] = \Pr[\tau^2 \leq \alpha^2] = \alpha^2$ for all $\alpha \in [0, 1]$.[4] Let $S_\tau = \{u \mid \boldsymbol{z}_u \geq \tau\}$, then the key claim is that

7.7. Claim.

$$\frac{\mathbb{E}_\tau w_G(\partial_G S_\tau)}{\mathbb{E}_\tau \boldsymbol{d}(S_\tau)} \leq \sqrt{2\frac{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}}}$$

Given the claim, even though

$$\mathbb{E}_\tau \left(\frac{w_G(\partial_G S_\tau)}{\boldsymbol{d}(S_\tau)} \right) \neq \frac{\mathbb{E}_\tau w_G(\partial_G S_\tau)}{\mathbb{E}_\tau \boldsymbol{d}(S_\tau)},$$

we are done since there must exist a τ s.t.

$$\min_\tau \frac{w_G(\partial_G S_\tau)}{\boldsymbol{d}(S_\tau)} \leq \frac{\mathbb{E}_\tau w_G(\partial_G S_\tau)}{\mathbb{E}_\tau \boldsymbol{d}(S_\tau)}$$

since $\frac{\sum_{i=1}^k a_i}{\sum_{i=1}^k b_i} \geq \min_i \frac{a_i}{b_i}$. Since we have $\boldsymbol{d}(S_\tau) \leq \sum_{u:z_u>0} \boldsymbol{d}(u) \leq \boldsymbol{d}(V)/2$,

$$\Phi_G(S_\tau) = \frac{w_G(\partial_G S_\tau)}{\boldsymbol{d}(S_\tau)}$$

which implies

$$\Phi_G(S_\tau) \leq \sqrt{2\frac{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}}}$$

as desired.

[4]For the existential argument, we consider the probability density function $f(x) = 2x$ because $\int_{x=0}^{\alpha} 2x = x^2$.

Proof of claim 7.7. To show $\frac{\mathbb{E}_\tau w_G(\partial_G S_\tau)}{\mathbb{E}_\tau \boldsymbol{d}(S_\tau)} \leq \sqrt{2\frac{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}}{\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}}}$, for the denominator,

$$\begin{aligned}\mathbb{E}_\tau \boldsymbol{d}(S_\tau) &= \sum_u \boldsymbol{d}(u) \Pr[u \in S_\tau] \\ &= \sum_u \boldsymbol{d}(u) \Pr[\tau \leq z_u] \\ &= \sum_u \boldsymbol{d}(u) z_u^2 = \boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}\end{aligned}$$

For the numerator,

$$\mathbb{E}_\tau w_G(\partial_G S_\tau) = \sum_{uv} w_{uv} \Pr[\{u, v\} \text{ is cut}]$$

since τ is the exact value that separate the vertices within the cut and vertices that are not in the cut

$$= \sum_{uv} w_{uv} \Pr[z_u < \tau \leq z_v] \qquad \text{assuming } z_v > z_u$$

this probability is exactly $\Pr[\tau \leq z_v] - \Pr[\tau \leq z_u]$ i.e.

$$= \sum_{uv} w_{uv}(z_v^2 - z_u^2)$$

$$= \sum_{uv} w_{uv}(z_v - z_u)(z_v + z_y)$$

By Cauchy-Schwarz, we have

$$\begin{aligned}\sum_{uv} w_{uv}(z_v - z_u)(z_v + z_y) &\leq \sqrt{\sum_{uv} w_{uv}(z_v - z_u)^2} \cdot \sqrt{\sum_{uv} w_{uv}(z_v + z_u)^2} \\ &\leq \sqrt{\boldsymbol{z}^\top \boldsymbol{L}_G \boldsymbol{z}} \cdot \sqrt{2\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}}\end{aligned}$$

since

$$\begin{aligned}\sum_{uv} w_{uv}(z_v + z_u)^2 &\leq \sum_{uv} w_{uv}(2z_v^2 + 2z_u^2) \\ &= \sum_u 2\deg(u) \cdot z_u^2 \\ &= \sum_u 2\boldsymbol{d}(u) \cdot z_u^2 = 2\boldsymbol{z}^\top \boldsymbol{D}\boldsymbol{z}.\end{aligned}$$

[5] □

8. Conclusion: Fiedler's Algorithm

The proof above is in fact algorithmic - we can find a sparse cut S where $\Phi_G(S) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G)} \leq 2\sqrt{\Phi(G)}$. Specifically, the algorithm is as follows:

[5] Note that it is this last step where we need that $\boldsymbol{d}(u) \geq \deg(u)$. So it would not work if we want to prove a generalized Cheeger's inequality for $\Phi(G, \boldsymbol{d})$ for arbitrary vector $\boldsymbol{d}$

1. Compute the second eigenvector $\boldsymbol{v}_2$ of $\boldsymbol{N}_G$. We have $\boldsymbol{v}_2 \perp \boldsymbol{d}^{1/2}$ and $\frac{\boldsymbol{v}_2^\top \boldsymbol{N}_G \boldsymbol{v}_2}{\boldsymbol{v}_2^\top \boldsymbol{v}_2} = \lambda_2(\boldsymbol{N}_G)$.
2. Compute $\boldsymbol{x} = \boldsymbol{D}^{-1/2}\boldsymbol{v}_2$. We have $x \perp \boldsymbol{d}$ and $\frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}} = \lambda_2(\boldsymbol{N}_G)$.
3. Sort indices such that $x_1 \leq \cdots \leq x_n$.
4. Define $S_i' = \{1, \ldots, i\}$ for all $i < n$, where $\forall j \in [i]$, j represents the vertex in G corresponding to entry x_j.
5. Return S^* with minimum $\Phi_G(S_i')$ among all $i \in \{1, \ldots, n-1\}$.

8.1. Lemma (Correctness)**.** *Consider the cut S_τ from the proof above. We have $(S_\tau, V \setminus S_\tau) = (S_i', V \setminus S_i')$ for some i. So $\Phi_G(S^*) \leq \Phi_G(S_\tau) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G)}$.*

8.2. Exercise. If instead of $\boldsymbol{v}_2$, we are given a vector $\boldsymbol{v} \perp \boldsymbol{d}^{1/2}$ where $\frac{\boldsymbol{v}^\top \boldsymbol{N}_G \boldsymbol{v}}{\boldsymbol{v}^\top \boldsymbol{v}} \leq \lambda_2(\boldsymbol{N}_G) + \epsilon$. We would have $\Phi_G(S^*) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G) + 2\epsilon}$

Solution. Let $\boldsymbol{x} = \boldsymbol{D}^{-1/2}\boldsymbol{v}$, then $\boldsymbol{x} \perp \boldsymbol{d}$. Theorem 7.1 shows that as long as $\boldsymbol{x} \perp \boldsymbol{d}$, the S^* found by Fiedler's algorithm satisfies $\Phi_G(S^*) \leq \sqrt{2\frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}}}$. Since $\frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}} = \frac{\boldsymbol{v}^\top \boldsymbol{N}_G \boldsymbol{v}}{\boldsymbol{v}^\top \boldsymbol{v}} \leq \lambda_2(\boldsymbol{N}_G) + \epsilon, \implies \Phi_G(S^*) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G) + 2\epsilon}$. □

8.3. Lemma (Time)**.** *Given the second eigenvector $\boldsymbol{v}_2$, we can compute S^* in $O(m + n \log n)$ time.*

- Modulo the time for computing $\boldsymbol{v}_2$, the algorithm is very fast and gives a good approximation when $\Phi(G)$ is big.
- In the next lecture, we will show how to obtain the vector which is almost as good as $\boldsymbol{v}_2$ in $\tilde{O}(m/\Phi(G))$ or even $\tilde{O}(m)$ time.

8.4. Question (Possible project)**.** *I think that using ideas from Section 7 of https://arxiv.org/pdf/1910.07950.pdf, we should be able to improve the above algorithm/analysis in two ways*

1. *The analysis based on "median cut" will be more intuitive.*
2. *The algorithm will only need to "query" only $O(\log^2 n)$ cuts of the form S_i' instead of $n-1$ cuts.*

The analysis in the paper is for PageRank but it can possibly work for Cheeger's cut too.

9. Extensions of Cheeger's Inequality

- Extension to Directed graphs [6]
- Improvement when vertex expansion is high[7]
- What about quantitative implications of λ_k for $k > 2$?
  - $\lambda_k(\boldsymbol{N}_G) = 0$ iff G has at least k connected components
  - What if $\lambda_k(\boldsymbol{N}_G) \approx 0$? We can also say that G can be clustered into k parts by deleting few edges.

[6] https://core.ac.uk/download/pdf/206608976.pdf
[7] https://arxiv.org/pdf/1504.00686.pdf

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 8: λ_n and The Power Method

September 22, 2021

Instructor: Thatchaphol Saranurak

Scribe: Peter Ly

We will mainly talk about two things for this lecture:

1. $\lambda_n(\mathbf{N})$ for finding max cuts (similar to $\lambda_2(\mathbf{N})$ for finding sparse cuts)
2. The power method: a fast algorithm for finding approximate eigenvectors.

1 $\lambda_n(\mathbf{N})$ vs Max Cut

We saw from Cheeger's Inequality that $\lambda_2(\mathbf{N}_G)$ corresponds to sparsest cut of the graph G. That is, $\lambda_2(\mathbf{N}_G)$ is small if and only if G has a sparse cut. In the extreme case where $\lambda_2(\mathbf{N}_G) = 0$, then we have that G is not connected.

There is an analogous result for $\lambda_n(\mathbf{N}_G)$ and the size of the maximum cut of G. $\lambda_n(\mathbf{N}_G)$ is large if and only if the size of the maximum cut is large - in other words, the graph is close to being bipartite. In the extreme case where $\lambda_n(\mathbf{N}_G) = 2$, then we have that G contains a connected component which is bipartite. We will show the proof of the extreme case and mention what happens if $\lambda_n(\mathbf{N}_G)$ is close to 2.

Lemma 1.1. *We have*

1. $\lambda_n(\mathbf{N}_G) \leq 2$. *So all eigenvalues of* $\mathbf{N}_G$ *are in the range of* $[0, 2]$.
2. $\lambda_n(\mathbf{N}_G) = 2$ *if and only if G contains a connected component which is bipartite.*

Proof. Recall

$$\lambda_n(\mathbf{N}_G) = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\mathbf{x}^\top \mathbf{N}_G \mathbf{x}}{\mathbf{x}^\top \mathbf{x}} = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\mathbf{x}^\top \mathbf{L}_G \mathbf{x}}{\mathbf{x}^\top \mathbf{D}\mathbf{x}}$$

1. To prove $\lambda_n(\mathbf{N}_G) \leq 2$, observe that

$$2\mathbf{x}^\top \mathbf{D}\mathbf{x} - \mathbf{x}\mathbf{L}_G\mathbf{x} = \sum_u 2\deg(u)x_u^2 - \sum_{uv\in E} w_{uv}(x_u - x_v)^2 = \sum_{uv\in E} w_{uv}(x_u + x_v)^2$$

So

$$2 - \lambda_n(\mathbf{N}_G) = 2 - \max_{\mathbf{x}\neq\mathbf{0}} \frac{\mathbf{x}\mathbf{L}_G\mathbf{x}}{\mathbf{x}^\top\mathbf{D}\mathbf{x}} = \min_{\mathbf{x}\neq\mathbf{0}} \frac{2\mathbf{x}^\top\mathbf{D}\mathbf{x} - \mathbf{x}\mathbf{L}_G\mathbf{x}}{\mathbf{x}^\top\mathbf{D}\mathbf{x}} = \min_{\mathbf{x}\neq\mathbf{0}} \frac{\sum_{uv\in E} w_{uv}(x_u + x_v)^2}{\mathbf{x}^\top\mathbf{D}\mathbf{x}} \geq 0.$$

2. ($\Leftarrow$) Now, suppose there is a bipartite component C where $V(C) = L \cup R$ (see Figure 1 for the image to keep in mind). Define a vector $\mathbf{x}$ where

$$x_u = \begin{cases} 1 & u \in L \\ -1 & u \in R \\ 0 & u \notin V(C) \end{cases}$$

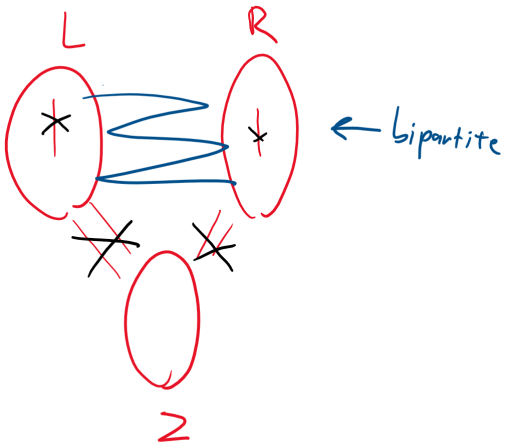
We have (from the proof of (1) in the lemma) that

$$2 - \lambda_n(\mathbf{N}_G) = \min_{\mathbf{x} \neq \mathbf{0}} \frac{\sum_{uv \in E} w_{uv}(x_u + x_v)^2}{\mathbf{x}^\top \mathbf{D}\mathbf{x}} = 0$$

So $\lambda_n(\mathbf{N}_G) = 2$.

($\Rightarrow$) Suppose $\lambda_n(\mathbf{N}_G) = 2$. Then there is some non-zero $\mathbf{x}$ such that $\sum_{uv \in E} w_{uv}(x_u + x_v)^2 = 0$. This happens if and only if $x_u = -x_v$ for all edges $(u, v) \in E$. Define $L = \{u \mid x_u < 0\}$, $R = \{u \mid x_u > 0\}$, and $Z = \{u \mid x_u = 0\}$ (see Figure 1 for an example of how the components relate). Since $\mathbf{x} \neq \mathbf{0}$, we have that $L \cup R$ is non-empty. Further, there are no edges between $L \cup R$ and Z. It follows that $L \cup R$ is a bipartite component of G.

Figure 1: $L \cup R$ form a bipartite component of G. There are no edges between $L \cup R$ and Z. Z may have edges inside of it, but those edges contribute 0 to the sum.



□

When we have that G is connected, then the only component is G itself, and Lemma 1.1 tells us that $\lambda_n(\mathbf{N}_G) = 2$ if and only if G is bipartite i.e. there is a cut that cuts all edges.

If we instead have that $\lambda_n(\mathbf{N}_G)$ is close to 2, then we can show that G is "close to being bipartite" - there is a cut which cuts almost every edge of G. This result was shown by Trevisan - more precisely, Trevisan showed a Cheeger's type inequality for λ_n:

$$\frac{2 - \lambda_n}{2} \leq \beta(G) \leq \sqrt{2(2 - \lambda_n)}$$

where $\beta(G)$ measures bipartiteness of G (see his lecture notes[1]).

[1] https://lucatrevisan.github.io/teaching/expanders2016/lecture05.pdf

2 The Power Method

Last time, we showed that we can construct a sparse cut S of G satisfying $\Phi_G(S) \leq \sqrt{2\lambda_2(\mathbf{N}_G)} \leq 2\sqrt{\Phi(G)}$ by using Fiedler's algorithm. The only slow step for constructing such an S was computing the second eigenvector $\mathbf{v}_2$ of $\mathbf{N}_G$. Now, we will show how to compute an "approximate second smallest eigenvector" in near-linear time. We can then use the approximate second smallest eigenvector in Fiedler's algorithm to obtain a sparse cut satisfying $\Phi_G(S) \leq \sqrt{2(\lambda_2(\mathbf{N}_G) + \varepsilon)}$.

We will begin by presenting how to find an approximation to the largest eigenvector and second largest eigenvector, and then we will show how to use the techniques for compuing the largest and second largest eigenvector to compute an approximate second smallest eigenvector. The technique we will use to compute these approximations is called the Power Method.

2.1 Largest Eigenvalue of M

This algorithm gives an approximation to the largest eigenvalue (and, in fact, its corresponding eigenvector).

POWER
Input: a PSD matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ and parameter k

1. Sample uniformly $\mathbf{x} \in \{-1, 1\}^n$
2. Return $\mathbf{y} = \mathbf{M}^k \mathbf{x}$.

POWER has runtime $O(k(n + \text{nnz}(\mathbf{M}))$ where $\text{nnz}(\mathbf{M})$ is the number of non-zero entries of $\mathbf{M}$.

Let Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ be eigenvalues of $\mathbf{M}$ (we have that all eigenvalues are non-negative because $\mathbf{M}$ is PSD).

Theorem 2.1 (Performance Guarantee of POWER)**.** *For any $\epsilon > 0$, with probability at least 3/16, the algorithm* POWER *returns* $\mathbf{y}$ *such that*

$$\frac{\mathbf{y}^\top \mathbf{M} \mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \geq (1 - \epsilon)\lambda_1 \cdot \frac{1}{1 + 4n(1 - \epsilon)^{2k}}.$$

If we take $k = O(\log(n)/\epsilon)$, then we have that $\frac{\mathbf{y}^\top \mathbf{M} \mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \geq (1 - O(\epsilon))\lambda_1$ - that is we have a multiplicative approximation for the first eigenvector with constant probability. We can amplify this probability to at least $1 - \frac{1}{\text{poly}(n)}$ by repeating POWER independently $O(\log n)$ times and taking the vector y^* which maximizes $\frac{\mathbf{y}^\top \mathbf{M} \mathbf{y}}{\mathbf{y}^\top \mathbf{y}}$.

Analysis of the POWER algorithm. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be eigenvectors of $\mathbf{M}$. We can write

$$\mathbf{M} = \sum_{i=n} \lambda_i \mathbf{v}_i \mathbf{v}_i^\top \text{ and } \mathbf{M}^k = \sum_{i=n} \lambda_i^k \mathbf{v}_i \mathbf{v}_i^\top$$

by spectral decomposition. We can then write

$$\mathbf{x} = \sum_i a_i \mathbf{v}_i$$

where $a_i = \langle \mathbf{x}, \mathbf{v}_i \rangle$ i.e. express $\mathbf{x}$ in the vector space spanned by the eigenvectors of $\mathbf{M}$. We can then write

$$\mathbf{y} = \mathbf{M}^k \mathbf{x} = \sum_i a_i \lambda_i^k \mathbf{v}_i.$$

Let us first see on a high level why the power method works.

Intuition. Suppose $\lambda_1 = 1$ and all $\lambda_2, \ldots, \lambda_n \leq 1/2$. Then we can write

$$\mathbf{y} = a_1 \mathbf{v}_1 + \sum_{i \geq 2} a_i \lambda_i^k \mathbf{v}_i.$$

When $k = \Theta(\log n)$, then each $\lambda_i^k \leq \frac{1}{\text{poly}(n)}$ and the second term becomes vanishingly small. Note that here we need $\mathbf{M}$ to be PSD to guarantee that each λ_i^k was non-negative so that each term with λ_i^k for $i \geq 2$ would actually go to zero - otherwise we might have a negative eigenvalue < -1 and some terms of the sum might not go to zero. Now suppose we can show that $|a_1| = |\langle \mathbf{x}, \mathbf{v}_1 \rangle|$ is not very small, then we get $\mathbf{y} \approx \pm \mathbf{v}_1$, so $\frac{\mathbf{y}^\top \mathbf{M} \mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \approx \lambda_1$.

Making it formal. There were two points we were informal about in our intuition:

1. Why should $|a_1| = |\langle \mathbf{x}, \mathbf{v}_1 \rangle|$ be not too small?

   This is very natural to expect - essentially we are projecting $\mathbf{v}_1$ in a random direction - the magnitude of this projection is likely to be comparable to $\mathbf{v}_1$. We prove this formally in the exercise session. To be exact, we show

   Lemma 2.2. *For any vector $\mathbf{v} \in \mathbb{R}^n$ where $\|\mathbf{v}\| = 1$, sample $\mathbf{x} \in \{-1, 1\}^n$. Then*

   $$\Pr\left[|\langle \mathbf{x}, \mathbf{v} \rangle| \geq \frac{1}{2} \right] \geq \frac{3}{16}.$$

   In particular, with good probability $|a_1|$ is not too small.

2. There is no reason for there to be a large gap between λ_1 and $\lambda_2, \ldots, \lambda_n$. What if λ_2 is close to λ_1? We will show that even if we do not have such a gap, we still obtain a large value for $\frac{\mathbf{y}^\top \mathbf{M} \mathbf{y}}{\mathbf{y}^\top \mathbf{y}}$.

   Fix $\varepsilon > 0$, and define ℓ so that λ_i can be classified as follows: $\lambda_1, \ldots, \lambda_\ell \geq (1 - \varepsilon)\lambda_1$ and $(1 - \varepsilon)\lambda_1 > \lambda_{\ell+1}, \ldots, \lambda_n$.

   We have that $\mathbf{y} = \sum_i a_i \lambda_i^k \mathbf{v}_i$ (shown above) and

   $$\mathbf{y}^\top \mathbf{y} = \sum_i a_i^2 \lambda_i^{2k} \text{ and } \mathbf{y}^\top \mathbf{M} \mathbf{y} = \sum_i a_i^2 \lambda_i^{2k+1}$$

   from the orthogonality of the eigenvectors from the spectral decomposition of $\mathbf{M}$.

   Our goal is to show that $\frac{\mathbf{y}^\top \mathbf{M} \mathbf{y}}{\mathbf{y}^\top \mathbf{y}}$ is big i.e. show a lower bound on the numerator and an upper bound on the denominator.

   By truncating terms in $\mathbf{y}^\top \mathbf{M} \mathbf{y}$ and then applying the bound $\lambda_\ell \geq (1 - \varepsilon)\lambda_1$, we obtain the following lower bound on $\mathbf{y}^\top \mathbf{M} \mathbf{y}$:

   $$\mathbf{y}^\top \mathbf{M} \mathbf{y} \geq \sum_{i=1}^{\ell} a_i^2 \lambda_i^{2k+1} \geq (1 - \epsilon)\lambda_1 \cdot \sum_{i=1}^{\ell} a_i^2 \lambda_i^{2k}.$$

Recall that

$$\mathbf{y}^\top \mathbf{y} = \sum_{i=1}^{\ell} a_i^2 \lambda_i^{2k} + \sum_{i=\ell+1}^{n} a_i^2 \lambda_i^{2k}$$

We can upper bound the $\sum_{i=\ell+1}^{n} a_i^2 \lambda_i^{2k}$ term (and therefore $\mathbf{y}^\top \mathbf{y}$) as follows:

$$\begin{aligned}
\sum_{i=\ell+1}^{n} a_i^2 \lambda_i^{2k} &< (1-\epsilon)^{2k} \quad {\lambda_1}^{2k} \quad \sum_{i=\ell+1}^{n} a_i^2 && (\ell \text{ defined so } (1-\varepsilon)\lambda_1 > \lambda_{\ell+1}, \ldots, \lambda_n) \\
&\leq (1-\epsilon)^{2k} \quad {\lambda_1}^{2k} \quad \|\mathbf{x}\|^2 \\
&\leq (1-\epsilon)^{2k} \quad a_1^2 {\lambda_1}^{2k} \quad \frac{\|\mathbf{x}\|^2}{a_1^2} \\
&\leq (1-\epsilon)^{2k} \quad \sum_{i=1}^{\ell} a_i^2 \lambda_i^{2k} \quad 4n && (\text{using } a_1^2 = |\langle \mathbf{x}, \mathbf{v}_1 \rangle|^2 \geq \tfrac{1}{4})
\end{aligned}$$

So we can upper bound $\mathbf{y}^\top \mathbf{y}$ by

$$\mathbf{y}^\top \mathbf{y} \leq \sum_{i=1}^{\ell} a_i^2 \lambda_i^{2k} \cdot (1 + 4n(1-\epsilon)^{2k})$$

Putting the bounds together, we have with probability at least 3/16

$$\frac{\mathbf{y}^\top \mathbf{M} \mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \geq (1-\epsilon)\lambda_1 \cdot \frac{1}{1 + 4n(1-\epsilon)^{2k}}$$

2.2 Second Largest Eigenvalue of M

What if we want the second largest eigenvalue of $\mathbf{M}$? Just run the same algorithm after translation so that the vector we pick is in the space orthogonal to the first eigenvector $\mathbf{v}_1$.

POWER2
Input: a PSD matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$, vector $\mathbf{v}_1$, and parameter k

1. Sample uniformly $\mathbf{x}' \in \{-1, 1\}^n$
2. $\mathbf{x} \leftarrow \mathbf{x}' - \mathbf{v}_1 \langle \mathbf{x}', \mathbf{v}_1 \rangle$ (and so $\mathbf{x} \perp \mathbf{v}_1$)
3. Return $\mathbf{y} = \mathbf{M}^k \mathbf{x}$.

We can write

$$\mathbf{x} = \sum_{i=2}^{n} a_i \mathbf{v}_i$$

where $a_i = \langle \mathbf{x}, \mathbf{v}_i \rangle$ as above. We only need to sum for $i = 2$ to n because $\mathbf{x} \perp \mathbf{v}_1$ by step 2 of POWER2. We also have that $a_2 \geq 1/2$ with probability at least 3/16. Performing the same analysis as above for

$$\mathbf{y} = \sum_{i=2}^{n} a_i \lambda_i^k \mathbf{v}_i$$

and we have

$$\frac{\mathbf{y}^\top \mathbf{M}\mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \geq (1-\epsilon)\lambda_2 \cdot \frac{1}{1+4n(1-\epsilon)^{2k}}$$

with probability at least 3/16.

Theorem 2.3. *Suppose that* $\mathbf{v}_1$ *is the first eigenvector of* $\mathbf{M}$*. For any* $\epsilon > 0$*, with probability* $\geq 3/16$*, the algorithm* POWER2 *returns* $\mathbf{y} \perp \mathbf{v}_1$ *such that*

$$\frac{\mathbf{y}^\top \mathbf{M}\mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \geq (1-\epsilon)\lambda_2 \cdot \frac{1}{1+4n(1-\epsilon)^{2k}}.$$

2.3 Second Smallest Eigenvalue of $\mathbf{N}_G$

Now that we have an algorithm for the second largest eigenvalue of a PSD matrix, we can show the algorithm to get the second smallest eigenvalue of $\mathbf{N}_G$ to use it for Cheeger's cut. Recall that all eigenvalues of $\mathbf{N}_G$ are in $[0, 2]$. If we instead work with the matrix

$$\mathbf{M} = 2\mathbf{I} - \mathbf{N}_G = \mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}.$$

we can get the second smallest eigenvalue of $\mathbf{N}_G$ by finding the second largest eigenvalue of $\mathbf{M}$. If $0 = \lambda_1 \leq \cdots \leq \lambda_n \leq 2$ are eigenvalues of $\mathbf{N}_G$, then the eigenvalues of $\mathbf{M}$ are

$$2 = 2 - \lambda_1 \geq 2 - \lambda_2 \geq \cdots \geq 2 - \lambda_n \geq 0$$

and so $\mathbf{M}$ is PSD and we can apply POWER2 to $\mathbf{M}$.

To apply POWER2, we need the eigenvector $\mathbf{v}_1$ corresponding to the largest eigenvalue $2 - \lambda_1$ of $\mathbf{M}$. The eigenvector is the same as λ_1 of $\mathbf{N}_G$ which is $\mathbf{v}_1 = \mathbf{d}^{1/2}$. By running POWER2 on $\mathbf{M}$ with $\mathbf{v}_1$, we get $\mathbf{y} \perp \mathbf{d}^{1/2}$ where $\mathbf{y}^\top \mathbf{M}\mathbf{y} \geq (2 - \lambda_2 - \epsilon) \cdot \mathbf{y}^\top \mathbf{y}$.

However

$$\begin{aligned} & \mathbf{y}^\top (2\mathbf{I} - \mathbf{N}_G)\mathbf{y} \geq (2 - \lambda_2 - \epsilon) \cdot \mathbf{y}^\top \mathbf{y} \\ \iff & (\lambda_2 + \epsilon)\mathbf{y}^\top \mathbf{y} \geq \mathbf{y}^\top \mathbf{N}_G \mathbf{y} \end{aligned}$$

so we have that

$$\frac{\mathbf{y}^\top \mathbf{N}_G \mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \leq (\lambda_2 + \epsilon)$$

in time $\widetilde{O}(m/\epsilon)$.

Given $\mathbf{y}$, Fiedler's algorithm computes a cut S where $\Phi_G(S) \leq \sqrt{2(\lambda_2 + \epsilon)}$ in time $O(m+n \log n)$.

However, using $\mathbf{M} = 2(\mathbf{I} - \mathbf{N}_G)$ gives an *additive* approximation, rather than a multiplicative approximation. If we want a $(1 + \epsilon')$ multiplicative approximation, then we need to choose $\epsilon = \epsilon'\lambda_2$. Choosing such ϵ gives a time complexity of $\widetilde{O}(m/(\epsilon'\lambda_2)) \leq \widetilde{O}(m/(\epsilon'\Phi(G)^2))$ because $\Phi(G) \leq \sqrt{2\lambda_2}$ - when $\Phi(G)$ is small, obtaining a $(1 + \epsilon')$ multiplicative approximation is slow.

Question 2.4 (Open Problem)**.** Given a graph G with a promise that either $\Phi(G) \geq 0.1$ or $\Phi(G) \leq 1/\log^{100} n$, can we decide which case it is in $\widetilde{O}(m)$ **deterministically**?

We presented the Fiedler's algorithm which runs in time $\widetilde{O}(m)$, but the algorithm is not deterministic because we sample a vector on the hypercube uniformly at random. The current best known deterministic algorithm runs in time $O(m^{1.01})$ [CGLNPS'20] [2]. If you can solve this problem, you will likely have a successful Ph.D.; Most fast, deterministic algorithms have this problem as a bottleneck, so if you can solve it, then it should lead to several improvements in existing algorithms!

2.4 Let's see examples and get more intuition

Let's interpret what the power method does.

Suppose that the graph G is d-regular. Then we have

$$\mathbf{M} = 2\mathbf{I} - \mathbf{N}_G = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} = \mathbf{I} + \frac{\mathbf{A}}{d}.$$

Let's consider the matrix

$$\mathbf{M}' = \frac{\mathbf{M}}{2} = \frac{\mathbf{I}}{2} + \frac{\mathbf{A}}{2d}$$

This matrix is easier to work with because its eigenvalues are between $[0, 1]$. It is just a scaled version of $\mathbf{M}$.

Consider

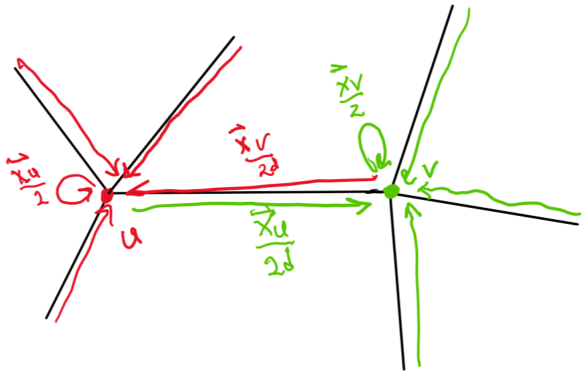
$$\mathbf{x}' = \mathbf{M}'\mathbf{x} = \left(\frac{\mathbf{I}}{2} + \frac{\mathbf{A}}{2d}\right)\mathbf{x}$$

which represents (up to scaling) one matrix multiplication in the power method. We can write

$$\mathbf{x}'_u = \frac{\mathbf{x}_u}{2} + \sum_{(u,v)\in E} \frac{\mathbf{x}_v}{2d}.$$

In total, there are k rounds in the power method and by the equation, in each round, each node u updates its value as a weighted average between its one value and its neighbors' values. This looks like a diffusion process like in Figure 2.

Figure 2: u and v are both updating their values. Because (u, v) are connected, they send some of their value to each other in addition to receiving value from their other neighbors. They also retain half of their value.

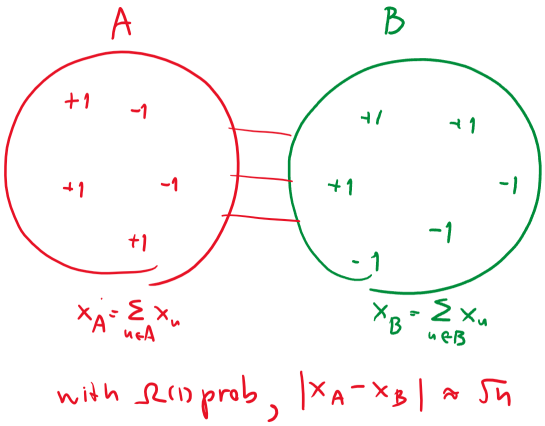


[2] https://arxiv.org/pdf/1910.08025.pdf

See this video[3] about the algorithm in action for cycles and hypercubes.

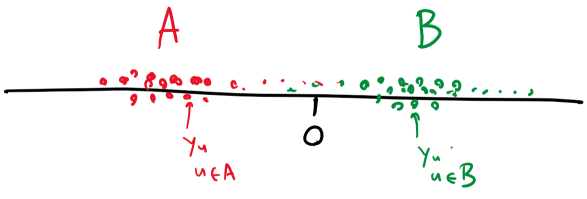
So, intuitively, why should this process give sparse cuts? Suppose there is a balanced sparse cut (A, B) (as in Figure 3) but A and B are themselves well-connected respectively. Denote $x_A = \sum_{u \in A} \mathbf{x}_u, x_B = \sum_{u \in B} \mathbf{x}_u$. In the beginning, since $\mathbf{x}$ is chosen uniformly from the hypercube, in expectation x_A and x_B are 0. Wlog we assume that $x_A \approx -\sqrt{n} < 0$ and $x_B \approx \sqrt{n} > 0$ (since $|x_A| = |x_B| = \Theta(\sqrt{n})$ with $\Omega(1)$ probability.) Then, after updating $\mathbf{x} = (\mathbf{M}')^k \mathbf{x}$, the sums won't change too much, i.e. $x_A \approx -\sqrt{n}$ and $x_B \approx \sqrt{n}$ since there are few edges connecting between A and B whence tiny amount of weight got send between A and B. What's more, since the process allow vertices to average their value with their neighbors, and vertices in A will mainly average with vertices within A, whence the value of vertices in A will be close to each other and most of them will be negative since we started with $x_A < 0$. Similarly, the values of vertices in B will still be close to each other and most of them will be positive.

Figure 3: (A, B) is a balanced sparse cut. Each vertex was assigned to -1 or 1 uniformly.



Fiedler's algorithm just sorts vertices according to $\mathbf{y} = (\mathbf{M}')^k \mathbf{x}$ (as shown in Figure 4) and sweeps through the order for the sparsest cut among the nested cuts. One of these cuts should be close to (A, B) so it should be sparse. This should also hold in the general case.

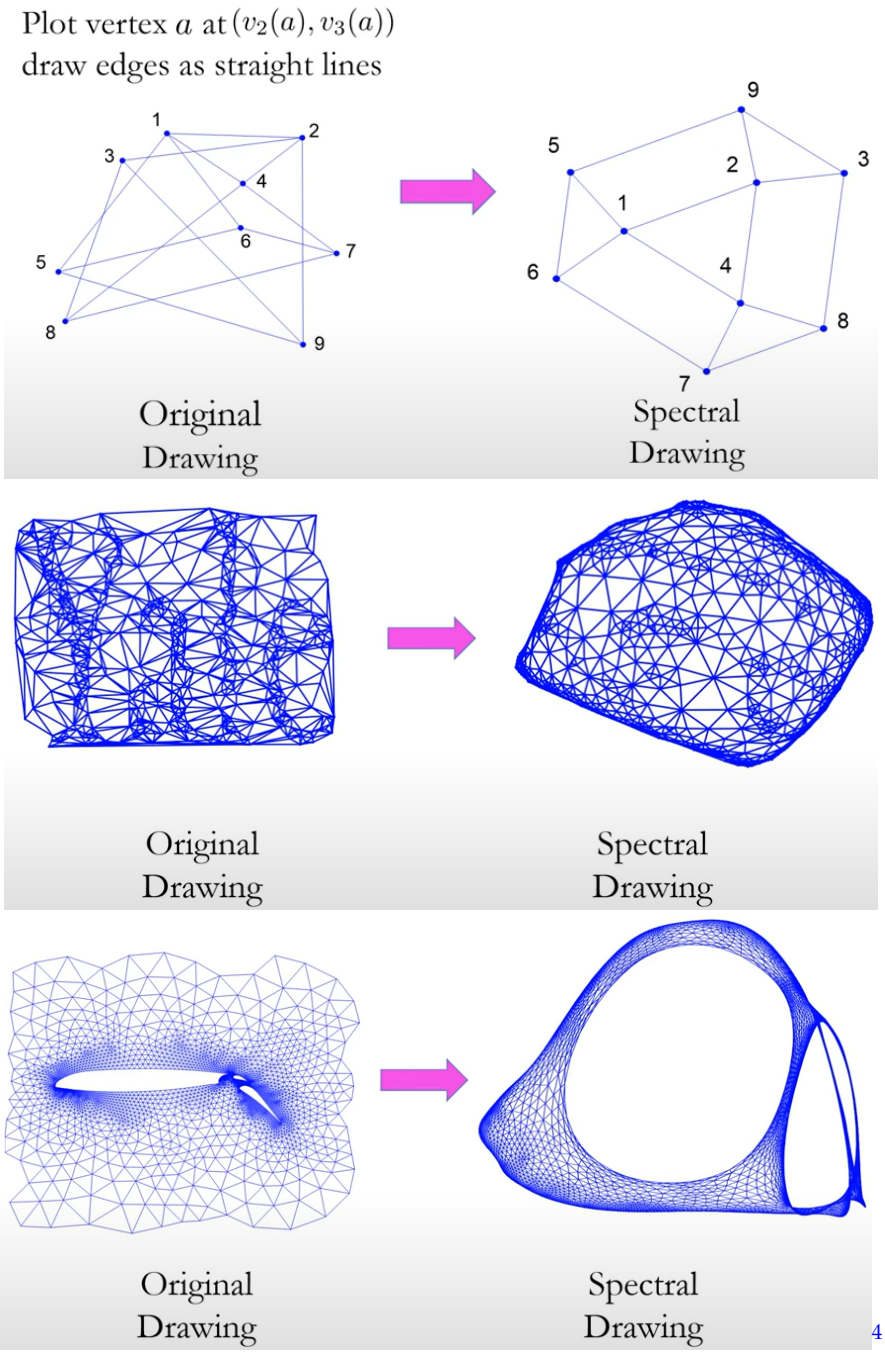
Figure 4: After computing $\mathbf{y} = (\mathbf{M}')^k \mathbf{x}$, Fiedler's algorithm sorts vertices according to $\mathbf{y}$ and sweeps across the nested cuts for the sparsest. Though there are some vertices in B that end with value less than 0 and vertices in A that end with value more than 0, there should be a cut such that there are not so many vertices on the wrong side - this cut should be close to (A, B).



[3] https://www.youtube.com/watch?v=cbYcn8o0Jfg

3 Digression: Spectral Graph Drawing

Let's look at the magic of $\mathbf{v}_2$ and $\mathbf{v}_3$ of L_G.



[4]

Automatically, the second and third eigenvectors tells us a lot about the topology of the graph.

[4]Image from https://www.youtube.com/watch?v=CDMQR422LGM

4 Additional Discussion in Exercise Session

4.1 Removing the $\frac{1}{\Phi(G)^2}$ Dependency from the Power Method

We obtained an additive approximation when we worked with $\mathbf{M} = 2\mathbf{I} - \mathbf{N}_G$ because we were approximating eigenvalues of the form $2 - \lambda_i$. The approximation is multiplicative for $2 - \lambda_i$, but when we view it to an approximation for λ_2, we obtain an additive approximation.

If we instead work with the inverse $\mathbf{N}_G^{-1}$, the eigenvalues should be of the form $1/\lambda_i$, but the first eigenvalue of $\mathbf{N}_G$ is $\lambda_1 = 0$ so $\mathbf{N}_G$ does not have an inverse. Instead, we will work with the **pseudo-inverse** $\mathbf{N}_G^\dagger$. $\mathbf{N}_G^\dagger$ is defined as

$$\mathbf{N}_G^\dagger = \sum_{i:\lambda_i \neq 0} \frac{1}{\lambda_i} v_i v_i^\top$$

where the v_i are from the spectral decomposition of $\mathbf{N}_G^\dagger$ and $\mathbf{N}_G^\dagger$ has eigenvalues

$$0, \frac{1}{\lambda_2} \geq \frac{1}{\lambda_3} \geq \cdots \geq \frac{1}{\lambda_n}.$$

We can directly use $\mathbf{N}_G^\dagger$ to get a multiplicative approximation for eigenvalues of the form $\frac{1}{\lambda_i}$ which can then be used to obtain a multiplicative approximation for λ_i.

Exercise 4.1. Show that $\mathbf{y} = (\mathbf{N}_G^\dagger)^k \mathbf{x}$ satisfies $\frac{\mathbf{y}^\top \mathbf{N}_G \mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \leq (1+\varepsilon)\lambda_2$ for all $\varepsilon > 0$ and $k = O(\log \frac{n}{\varepsilon}/\varepsilon)$.

Proof. Similar to the analysis of POWER we upper bound $\mathbf{y}^\top \mathbf{N}_G \mathbf{y}$ and lower bound $\mathbf{y}^\top \mathbf{y}$. As before, we express $\mathbf{x}$ in the eigenvectors of $(\mathbf{N})_G^\dagger$ so $\mathbf{x} = \sum_{i:\lambda_i \neq 0} a_i \mathbf{v}_i$ where $a_i = \langle \mathbf{v}_i, \mathbf{x} \rangle$.

Observe that $\mathbf{y} = \sum_{i:\lambda_i \neq 0} a_i \frac{1}{\lambda_i^k}$, $\mathbf{y}^\top \mathbf{N}_G \mathbf{y} = \sum_{i:\lambda_i \neq 0} a_i^2 \frac{1}{\lambda_i^{2k-1}}$, and $\mathbf{y}^\top \mathbf{y} = \sum_{i:\lambda_i \neq 0} a_i^2 \frac{1}{\lambda_i^{2k}}$.

Fix $\varepsilon' = \varepsilon/2$. Again, we define ℓ to separate the eigenvalues such that $\lambda_2 \leq \cdots \leq \lambda_\ell \leq (1+\varepsilon')\lambda_2$ and $(1+\varepsilon')\lambda_2 < \lambda_{\ell+1} \leq \cdots \leq \lambda_n$.

Then we obtain the following bounds:

$$\begin{aligned}
\mathbf{y}^\top \mathbf{N}_G \mathbf{y} &= \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}} + \sum_{\substack{i:\lambda_i \neq 0 \\ i > \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}} \\
&\leq \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}} + \frac{1}{(1+\varepsilon')^{2k-1}\lambda_2^{2k-1}} \sum_{\substack{i:\lambda_i \neq 0 \\ i > \ell}} a_i^2 \\
&\leq \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}} + \frac{1}{(1+\varepsilon')^{2k-1}\lambda_2^{2k-1}} ||\mathbf{x}||^2 \\
&= \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}} + \frac{1}{(1+\varepsilon')^{2k-1}\lambda_2^{2k-1}} a_2^2 \frac{||\mathbf{x}||^2}{a_2^2} \\
&= \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}} + \frac{4n}{(1+\varepsilon')^{2k-1}} \frac{a_2^2}{\lambda_2^{2k-1}} \qquad \text{(using } a_2^2 = |\langle \mathbf{x}, \mathbf{v}_2 \rangle|^2 \geq \tfrac{1}{4}\text{)} \\
&\leq \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}} + \frac{4n}{(1+\varepsilon')^{2k-1}} \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} \frac{a_i^2}{\lambda_i^{2k-1}} \\
&= \left(1 + \frac{4n}{(1+\varepsilon')^{2k-1}}\right) \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} \frac{a_i^2}{\lambda_i^{2k-1}}
\end{aligned}$$

$$\begin{aligned}
\mathbf{y}^\top \mathbf{y} &= \sum_{i:\lambda_i \neq 0} a_i^2 \frac{1}{\lambda_i^{2k}} \\
&= \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k}} + \sum_{\substack{i:\lambda_i \neq 0 \\ i > \ell}} a_i^2 \frac{1}{\lambda_i^{2k}} \\
&\geq \frac{1}{(1+\varepsilon')\lambda_2} \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}}
\end{aligned}$$

So

$$\frac{\mathbf{y}^\top \mathbf{N}_G \mathbf{y}}{\mathbf{y}^\top \mathbf{y}} \leq \frac{\left(1 + \frac{4n}{(1+\varepsilon')^{2k-1}}\right) \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} \frac{a_i^2}{\lambda_i^{2k-1}}}{\frac{1}{(1+\varepsilon')\lambda_2} \sum_{\substack{i:\lambda_i \neq 0 \\ i \leq \ell}} a_i^2 \frac{1}{\lambda_i^{2k-1}}} = \left(1 + \frac{4n}{(1+\varepsilon')^{2k-1}}\right)(1+\varepsilon')\lambda_2.$$

When we take $k = O\left(\log\left(\frac{n}{\varepsilon'}\right)/\varepsilon'\right)$, we have that

$$\left(1 + \frac{4n}{(1+\varepsilon')^{2k-1}}\right)(1+\varepsilon')\lambda_2 \leq (1+2\varepsilon')\lambda_2 = (1+\varepsilon)\lambda_2.$$

□

However, to directly use $\mathbf{N}_G^\dagger$ to get a multiplicative approximation for eigenvalues of the form $\frac{1}{\lambda_i}$, we would need to compute $(\mathbf{N}_G^\dagger)^k\mathbf{x}$. Computing $\mathbf{N}_G^\dagger$ itself can be expensive - $\mathbf{N}_G^\dagger$ can be dense, even if G is sparse. Observe that $\mathbf{N}_G^\dagger\mathbf{x}$ is the same as finding $\mathbf{v}$ where

$$\mathbf{N}_G\mathbf{v} = \mathbf{x}.$$

This is a Laplacian linear system. Laplacian linear systems can be solved in near-linear time[5]! (I hope I will have time to cover this).

4.2 Proof of the Paley-Zygmund inequality

Lemma 4.2 (Paley-Zygmund inequality)**.** *If Z is a non-negative random variable with finite variance, then, for every $0 \le \delta \le 1$,*

$$\Pr(Z \ge \delta \cdot \mathbb{E}[Z]) \ge (1-\delta)^2 \cdot \frac{(\mathbb{E}[Z])^2}{\mathbb{E}[Z^2]}$$

Proof. $\mathbb{E}[Z] = \mathbb{E}[Z \cdot \mathbf{1}_{\{Z \ge 0\}}]$ where $\mathbf{1}_{\{Z\ge 0\}}$ is the indicator random variable taking value 1 when $Z \ge 0$ and value 0 otherwise. $\mathbb{E}[Z \cdot \mathbf{1}_{\{Z\ge 0\}}] = \mathbb{E}\left[Z \cdot \mathbf{1}_{\{Z \ge \delta\cdot\mathbb{E}[Z]\}} + Z \cdot \mathbf{1}_{\{Z<\delta\cdot\mathbb{E}[Z]\}}\right]$. We have $\mathbb{E}[Z \cdot 1_{Z<\delta\cdot\mathbb{E}[Z]}] \le \delta \cdot \mathbb{E}[Z]$, so $\mathbb{E}[Z \cdot 1_{Z\ge\delta\cdot\mathbb{E}[Z]}] \ge (1-\delta)\,\mathbb{E}[Z]$.

We can define the inner product on the set of random variables by the expectation of their product i.e. $\langle X, Y\rangle = \mathbb{E}[XY]$. Then by the Cauchy-Schwarz inequality, we have

$$\begin{aligned}
&\langle Z, \mathbf{1}_{\{Z\ge\delta\cdot\mathbb{E}[Z]\}}\rangle^2 \le \langle Z, Z\rangle \cdot \langle \mathbf{1}_{\{Z\ge\delta\cdot\mathbb{E}[Z]\}}, \mathbf{1}_{\{Z\ge\delta\cdot\mathbb{E}[Z]\}}\rangle \\
\iff &\mathbb{E}[Z \cdot \mathbf{1}_{\{Z\ge\delta\cdot\mathbb{E}[Z]\}}]^2 \le \mathbb{E}[Z^2] \cdot \mathbb{E}[\mathbf{1}^2_{\{Z\ge\delta\cdot\mathbb{E}[Z]\}}] \\
\iff &\mathbb{E}[Z \cdot \mathbf{1}_{\{Z\ge\delta\cdot\mathbb{E}[Z]\}}] \le \sqrt{\mathbb{E}[Z^2]} \cdot \sqrt{\mathbb{E}[\mathbf{1}_{\{Z\ge\delta\cdot\mathbb{E}[Z]\}}]}
\end{aligned}$$

Then, $\mathbb{E}[\mathbf{1}_{\{Z\ge\delta\cdot\mathbb{E}[Z]\}}] = \Pr[Z \ge \delta \cdot \mathbb{E}[Z]]$ and by rearrangement and substitution, we obtain the inequality. □

4.3 Inner Product between a Vector with a Random Point from the Hypercube

We mentioned earlier a lemma about the inner product between a vector and a random point from the hypercube in order to prove the performance of POWER. This is a proof of the lemma.

Lemma 4.3. *For any vector $\mathbf{v} \in \mathbb{R}^n$ where $\|\mathbf{v}\| = 1$. Sample $\mathbf{x} \in \{-1, 1\}^n$. Then*

$$\Pr\left[|\langle \mathbf{x}, \mathbf{v}\rangle| \ge \frac{1}{2}\right] \ge \frac{3}{16}.$$

Remark 4.4. The proof of Lemma 4.3 works even if $x \sim \{-1, 1\}^n$ is selected according to a 4-wise independent distribution. This means that the algorithm can be derandomized in polynomial time.

[5] See the Spielman-Teng '04 paper

Proof. From Luca's lecture note[6]. We apply the Paley-Zygmund inequality to the random variable $\langle \mathbf{x}, \mathbf{v} \rangle^2$.

Define Z to be the random variable $\langle \mathbf{x}, \mathbf{v} \rangle = \sum_i x_i v_i$. We compute its first, second, and fourth moments to be:

$$\mathbb{E}[Z] = 0.$$

$$\mathbb{E}[Z^2] = 1.$$

$$\mathbb{E}[Z^4] = 3\left(\sum_i v_i^2\right) - 2\sum_i v_i^4 \leq 3.$$

Then observe that $\mathrm{Var}(Z) = \mathbb{E}[Z^2] - \mathbb{E}[Z]^2 = 1$. Having the first, second, and fourth moment of Z, we can apply the Paley-Zygmund inequality to Z^2 and $\delta = 1/4$, so we obtain

$$\Pr\left[S^2 \geq \frac{1}{4}\right] \geq \left(\frac{3}{4}\right)^2 \cdot \frac{1}{3} = \frac{3}{16}$$

□

[6] https://lucatrevisan.github.io/teaching/expanders2016/lecture08.pdf

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 9: Spectral Equivalences, Expander Mixing Lemma, and Ramanujan Graphs

September 30, 2021

Instructor: Thatchaphol Saranurak

Scribe: Nikhil Shagrithaya

1 Introduction

Last week, we saw that the eigenvalues of the normalized Laplacian of a graph can contain useful information about the 'connectedness' of a graph. In particular, we saw that the second smallest eigenvalue of ($\mathbf{N}_G$) is closely related to the conductance of the graph. The precise relationship is given by Cheeger's inequality:

$$\lambda_2(\mathbf{N}_G)/2 \leq \Phi(G) \leq \sqrt{2\lambda_2(\mathbf{N}_G)}$$

Moreover, we also saw an algorithm which gave us a $O(1)$-approximation of the sparest cut, when the conductance of the graph was $\Omega(1)$. That is, using the power method, we were able to obtain a method cut for which the following was true:

$$\Phi_G(S) \approx \sqrt{2\lambda_2(\mathbf{N}_G)} \leq 2\sqrt{\Phi(G)}$$

2 Spectral equivalency of graph

We can also define a new notion of connectivity for graphs based on the eigenvalues of the normalized Laplacian itself. In order to do this, we need some definitions.

Definition 2.1. We define a partial order on graphs based on their eigenvalues. We say that:

$$H \preccurlyeq^{\text{spec}} G$$

if $\boldsymbol{x}^T\mathbf{L}_H\boldsymbol{x} \leq \boldsymbol{x}^T\mathbf{L}_G\boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^V$. Or equivalently, $\mathbf{L}_H \preceq \mathbf{L}_G$ or $\mathbf{L}_G - \mathbf{L}_H$ is psd (positive semi-definite).

Definition 2.2. We say that $\mathbf{H}$ and $\mathbf{G}$ are **α-spectral-equivalent** if $\mathbf{H}' \preccurlyeq^{\text{spec}} \mathbf{G} \preccurlyeq^{\text{spec}} \alpha\mathbf{H}'$ for some $\mathbf{H}' = c \cdot \mathbf{H}$. We write $\mathbf{H} \approx_\alpha^{\text{spec}} \mathbf{G}$.

Note that $\preccurlyeq^{\text{spec}}$ is stronger than $\preccurlyeq^{\text{cut}}$, because we can define the partial order based on cut in another, equivalent way: $H \preccurlyeq^{\text{cut}} G$ iff $\boldsymbol{x}^T \mathbf{L}_H \boldsymbol{x} \leq \boldsymbol{x}^T \mathbf{L}_G \boldsymbol{x}$ for all $\boldsymbol{x} \in \{0,1\}^V$. Because the partial order based on eignenvalues requires $\boldsymbol{x}^T \mathbf{L}_H \boldsymbol{x} \leq \boldsymbol{x}^T \mathbf{L}_G \boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^V$, and not just $\boldsymbol{x} \in \{0,1\}^V$, $\preccurlyeq^{\text{spec}}$ is a stronger requirement than $\preccurlyeq^{\text{cut}}$, namely $H \preccurlyeq^{\text{spec}} G$ implies $H \preccurlyeq^{\text{cut}} G$.

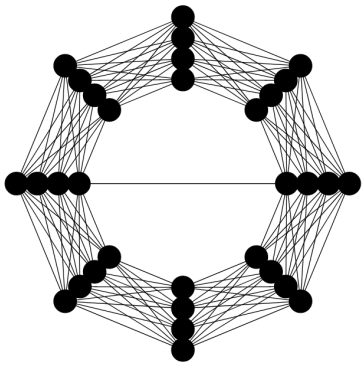
A natural question to then ask, is whether the converse is true, i.e. $H \preccurlyeq^{\text{cut}} G$ implies $H \preccurlyeq^{\text{spec}} G$? And if not, does it hold approximately?

We'll see that the answer is no to both of these questions, in the following lemma:

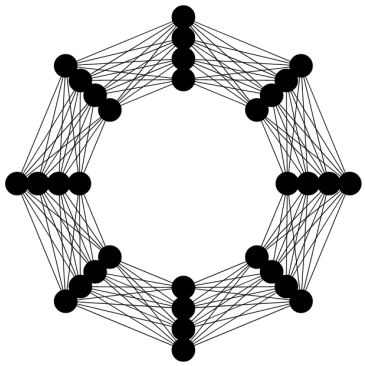
Lemma 2.3 ($\preccurlyeq^{\text{spec}}$ is strictly stronger than $\preccurlyeq^{\text{cut}}$)**.** *There exist graphs G and $\tilde{G}$ such that:*

- $\tilde{G} \preccurlyeq^{\text{cut}} G \preccurlyeq^{\text{cut}} (1+\epsilon)\tilde{G}$, *but*
- $\tilde{G} \preccurlyeq^{\text{spec}} G$ *but* $G \not\preccurlyeq^{\text{spec}} \frac{\epsilon^2 n}{10000}\tilde{G}$

Proof. We will construct unweighted graphs G and $\tilde{G}$ with two parameters: n and $k = \frac{50}{\epsilon}$. Both graphs will have $n \times k$ vertices, having n clusters with k vertices each. The vertex set of $\tilde{G}$ is $\{0, 1, \ldots, n-1\} \times \{1, \ldots, k\}$, where n is even. The graph $\tilde{G}$ will consist of n complete bipartite graphs, connecting all pairs of vertices $\{u, i\}$ and $\{v, j\}$ where $v = u + 1 mod n$. G is identical $\tilde{G}$, except that it has one extra edge going from vertex $\{0, 1\}$ to vertex $\{n/2, 1\}$. The graphs for $n = 8$ and $k = 4$ look like this:



G: $n = 8$ sets of $k = 4$ vertices arranged in a ring and connected by complete bipartite graphs, plus one edge across.



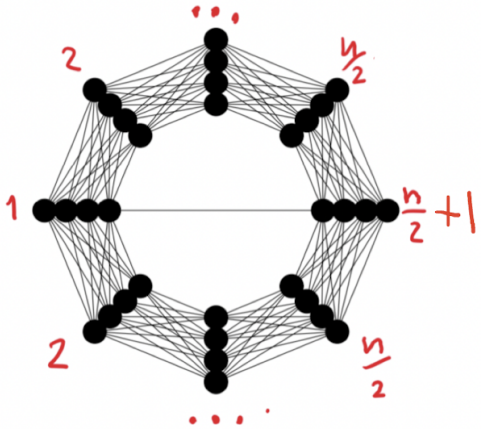
$\tilde{G}$: A good cut sparsifier of G, but a poor spectral sparsifier

[1]

It is easy to see that $\tilde{G} \preccurlyeq^{\text{cut}} G \preccurlyeq^{\text{cut}} (1+\epsilon)\tilde{G}$, because the only cuts whose value will change are those where the extra edge is crossing between them. Because any cut must have size at least k^2, adding one extra edge does not change the cut value by much.

Moreover, $\tilde{G} \preccurlyeq^{\text{spec}} G$ holds because $\tilde{G}$ is a subgraph of G. All that remains is to show $G \not\preccurlyeq^{\text{spec}} \frac{\epsilon^2 n}{10000}\tilde{G}$. Consider the following $\boldsymbol{x}$:

[1] Image from https://arxiv.org/abs/0808.4134



where $\forall u \in \{0, 1, \cdots, n/2\}, \forall i \in [k], x_{\{u,i\}} = u + 1; \forall u \in \{n/2 + 1, \cdots, n - 1\}\ \forall i \in [k], x_{\{u,i\}} = n + 1 - u$. Then, we have:

$$\boldsymbol{x}^T \boldsymbol{L}_G \boldsymbol{x} = nk^2 + \left(\frac{n}{2}\right)^2$$

but

$$\boldsymbol{x}^T \boldsymbol{L}_{\tilde{G}} \boldsymbol{x} = nk^2$$

So $\boldsymbol{x}^T \boldsymbol{L}_G \boldsymbol{x} - \frac{\epsilon^2 n}{10000} \boldsymbol{x}^T \boldsymbol{L}_{\tilde{G}} \boldsymbol{x} = nk^2 > 0 \implies G \not\preccurlyeq^{\text{spec}} \frac{\epsilon^2 n}{10000} \tilde{G}$. □

This lemma shows that spectral-equivalence between graphs are *strictly stronger* than cut-equivalence, and moreover, cut-equivalence cannot even imply a slightly weaker spectral-equivalence, as was the case for flow, because the lemma shows a gap of $\Omega(n)$, which is the largest possible gap.

As an aside, it is easy to check if $H \preccurlyeq^{\text{spec}} G$, because all we need to do is check whether $\boldsymbol{L}_G - \boldsymbol{L}_H$ is positive semi-definite, and this can be done in polynomial time.

Question 2.4 (Open)**.** Is there a **fast** algorithm for checking if $H \preccurlyeq^{\text{spec}} G$ or even $H \approx^{\text{spec}}_{(1+\epsilon)} G$?

Question 2.5 (Open)**.** Is there a **polynomial-time** algorithm for checking if $H \preccurlyeq^{\text{cut}} G$ or even $H \approx^{\text{cut}}_{(1+\epsilon)} G$?

From the previous lecture, checking if $H \preccurlyeq^{\text{flow}} G$ is checking the existence of feasible H-concurrent flow in G, which can be done by solving LP in polynomial time. Since $H \preccurlyeq^{\text{flow}} G \implies H \preccurlyeq^{\text{cut}} G$, Q2.5 can also be done in polytime. Remains to check the existence of faster algorithms for checking $H \preccurlyeq^{\text{flow}} G$ or $H \preccurlyeq^{\text{cut}} G$.

3 Spectral Equivalence between Expanders

In Lecture 03_2, we saw that expanders with the same degree profile are approximately equivalent, in both the cut and the flow sense. But are they also spectral-equivalent?

They indeed are. We begin with the following lemma:

Lemma 3.1. *Suppose G is a ϕ-expander. If $H \preccurlyeq^{\text{deg}} G$, then $H \preccurlyeq^{\text{spec}} O(\frac{1}{\phi^2})G$.*

Proof. We can increase the degree of any vertex in H to be equal to the degree of the corresponding vertex in G by adding self-loops, this will only make our task easier, and importantly, $\mathbf{L}_H$ remains the same. Now, both H and G have the same degree profile $\boldsymbol{d}$ (the degree profile is a vector whose entries are the degrees of the vertices of the graph). For any $\boldsymbol{x} \perp \boldsymbol{d}$, we have:

$$\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x} \geq \frac{\phi^2}{4} \boldsymbol{x}^\top \boldsymbol{L}_H \boldsymbol{x}.$$

The above statement is true because, by Cheeger's, $\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x} \geq \frac{\phi^2}{2} \boldsymbol{x}^\top \boldsymbol{D} \boldsymbol{x}$ for any $\boldsymbol{x} \perp \boldsymbol{d}$ (recall that $\lambda_2(\boldsymbol{N}_G) = \min_{\boldsymbol{x} \perp \boldsymbol{d}} \frac{\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{D} \boldsymbol{x}}$). Also, for any $\boldsymbol{x}$, we claim $2\boldsymbol{x}^\top \boldsymbol{D} \boldsymbol{x} \geq \boldsymbol{x}^\top \boldsymbol{L}_H \boldsymbol{x}$ because:

$$\begin{aligned} 2\boldsymbol{y}^\top \boldsymbol{D} \boldsymbol{y} - \boldsymbol{y}^\top \boldsymbol{L}_H \boldsymbol{y} &= \sum_u 2 \deg_G(u) y_u^2 - \sum_{uv \in E} w_{uv} (y_u - y_v)^2 = \sum_u 2 \deg_H(u) y_u^2 - \sum_{uv \in E} w_{uv} (y_u - y_v)^2 \\ &= \sum_{uv \in E} w_{uv} (y_u + y_v)^2 \geq 0. \end{aligned}$$

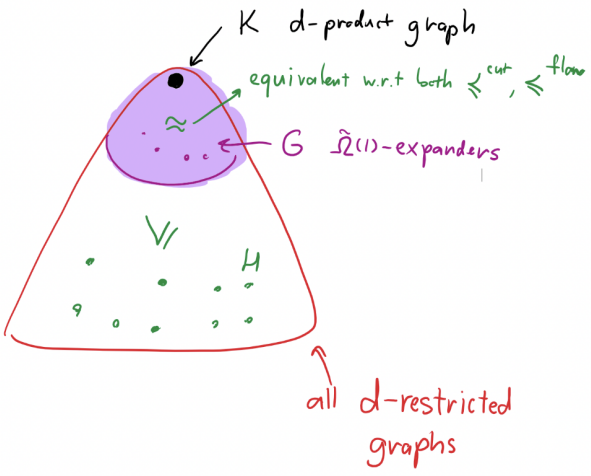
Now we have to show the same inequality also holds for all vectors $\boldsymbol{x}$ such that $\boldsymbol{x} \parallel \boldsymbol{d}$. Let $\boldsymbol{y} = \boldsymbol{x} - \sigma \mathbf{1}$ where $\boldsymbol{y} \perp \boldsymbol{d}$. Then:

$$\boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y} = (\boldsymbol{x} - \sigma \mathbf{1})^\top \boldsymbol{L}_G (\boldsymbol{x} - \sigma \mathbf{1}) = \boldsymbol{x}^T \boldsymbol{L}_G \boldsymbol{x} - \sigma \mathbf{1}^T \mathbf{L}_G \boldsymbol{x} - \boldsymbol{x}^T \mathbf{L}_G \sigma \mathbf{1} + \sigma \mathbf{1}^T \mathbf{L}_G \sigma \mathbf{1}$$

The last two terms are zero because $\mathbf{1}$ is in the kernel of L_G. The second term is also zero, because $\sigma \mathbf{1}^T \mathbf{L}_G \boldsymbol{x} = (\mathbf{L}_G^T \sigma \mathbf{1})^T \boldsymbol{x} = (\mathbf{L}_G^\sigma \mathbf{1})^T \boldsymbol{x} = 0$, where the second equality is true because $\mathbf{L}_G$ is symmetric. A similar argument holds for $\mathbf{L}_H$. Combining everything, we have that $\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x} = \boldsymbol{y}^\top \boldsymbol{L}_G \boldsymbol{y}$ and $\boldsymbol{x}^\top \boldsymbol{L}_H \boldsymbol{x} = \boldsymbol{y}^\top \boldsymbol{L}_H \boldsymbol{y}$, and because we've already shown that $\boldsymbol{x}^\top \boldsymbol{L}_G \boldsymbol{x} \geq \frac{\phi^2}{4} \boldsymbol{x}^\top \boldsymbol{L}_H \boldsymbol{x}$ when $\boldsymbol{x} \perp \boldsymbol{d}$, we are done. □

Corollary 3.2. *Let G and G' be ϕ-expanders with degree profile $\boldsymbol{d}$. We have:*

- *$G \preccurlyeq^{\text{spec}} O(\frac{1}{\phi^2}) G'$ and $G' \preccurlyeq^{\text{spec}} O(\frac{1}{\phi^2}) G$. So G and G' are $O(\frac{1}{\phi^4})$-spectral-equivalent.*
- *For all $\boldsymbol{d}$-restricted graphs H, we have $H \preccurlyeq^{\text{spec}} O(\frac{1}{\phi^2}) G$.*
- We have same picture we saw. It holds for for $\preccurlyeq^{\text{spec}}$



4 Expander Mixing Lemma

4.1 Motivation

Let G be a graph with degree profile $\boldsymbol{d}$. If G has high conductance, then we have:

$$|E_G(S, V \setminus S)| \approx \boldsymbol{d}(S)$$

for all sets S such that $vol(S) \leq vol(V \setminus S)$. But the good conductance only guarantees us that the number of edges going out of a set is high. But what can we say about the number of edges **between two sets ?**

For the $\boldsymbol{d}$-product graph, it can be easily seen that:

$$|E(S, T)| \approx \frac{\boldsymbol{d}(S)\boldsymbol{d}(T)}{\boldsymbol{d}(V)}$$

for all $S, T \subseteq V$. Intuitively, one should think about $\frac{\boldsymbol{d}(S)\boldsymbol{d}(T)}{\boldsymbol{d}(V)}$ as the expectation of $|E(S, T)|$ in the random graph with degree profile $\boldsymbol{d}$. This is a stronger property than conductance, as conductance guarantees this only when $T = V \setminus S$.

A natual question to then ask, is: do expanders satisfies even this stronger property? We will see that this stronger property has applications.

4.2 The Normalized Adjacency Matrix

We begin by defining a variant of the Laplacian matrix, whose eigenvalues will give us a good approximation of the property we defined in the previous subsection. Recall that we previously defined:

$$\boldsymbol{N}_G = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}_G\boldsymbol{D}^{-1/2}$$

, which had eigenvalues satisfying:

$$0 = \lambda_1(\boldsymbol{N}_G) \leq \lambda_2(\boldsymbol{N}_G) \leq \cdots \leq \lambda_n(\boldsymbol{N}_G) \leq 2.$$

We will call

$$\boldsymbol{A}'_G = \boldsymbol{D}^{-1/2}\boldsymbol{A}_G\boldsymbol{D}^{-1/2}$$

as the normalized adjacency matrix. Note that $\boldsymbol{A}'_G = \boldsymbol{I} - \boldsymbol{N}_G$ has eigenvalues satisfying:

$$1 = \lambda_1(\boldsymbol{A}'_G) \geq \lambda_2(\boldsymbol{A}'_G) \geq \cdots \geq \lambda_n(\boldsymbol{A}'_G) \geq -1.$$

The following properties follow from the corresponding ones we proved earlier, about $\mathbf{N_G}$:

- $\lambda_2(\boldsymbol{A}'_G) = 1$ iff G is not connected.
- $\lambda_2(\boldsymbol{A}'_G) < 1 - \Omega(1)$ iff G is a $\Omega(1)$-expander.
- $\lambda_n(\boldsymbol{A}'_G) = -1$ iff G is bipartite.

We further define σ_2 as follows:

$$\sigma_2 := \max\{|\lambda_2(\boldsymbol{A}'_G)|, |\lambda_3(\boldsymbol{A}'_G)|, \ldots, |\lambda_n(\boldsymbol{A}'_G)|\} = \max\{|\lambda_2(\boldsymbol{A}'_G)|, |\lambda_n(\boldsymbol{A}'_G)|\}.$$

It is the second largest eigenvalue in absolute value of $\boldsymbol{A}'_G$. So from what we learned before, if $\sigma_2 \ll 1$ is small, then G is both "far" from being disconnected and bipartite. Moreover, in the following section, we will see that σ_2 describes when two arbitrary disjoint sets are chosen in the graph, how much the edges between them behave like these of $\boldsymbol{d}$-product graph. The smaller σ_2 is, the more similar they are. We will be formalizing this statement, and make it mathematically precise, in the following sections.

4.3 The Statement and the Proof

Lemma 4.1 (Expander Mixing Lemma)**.** *For a graph G with degree profile $\mathbf{d}$, and for every $S, T \subseteq V$, we have*

$$\left| |E(S,T)| - \frac{\boldsymbol{d}(S)\boldsymbol{d}(T)}{\boldsymbol{d}(V)} \right| \leq \sigma_2 \sqrt{\boldsymbol{d}(S)\boldsymbol{d}(T)}.$$

Therefore, when σ_2 is small, $|E(S,T)| \approx \frac{\boldsymbol{d}(S)\boldsymbol{d}(T)}{\boldsymbol{d}(V)}$ holds for all $S, T \subseteq V$.

Proof. We will show a slightly more general statement, which is:

$$\sigma_2 = \max_{\boldsymbol{x},\boldsymbol{y}} \frac{|\boldsymbol{x}^\top (\boldsymbol{A}_G - \boldsymbol{R})\boldsymbol{y}|}{\sqrt{\sum_v \boldsymbol{d}(v)x_v^2} \cdot \sqrt{\sum_v \boldsymbol{d}(v)y_v^2}}$$

where $\boldsymbol{R}$ is equal to $\frac{1}{\boldsymbol{d}(V)}\boldsymbol{d} \cdot \boldsymbol{d}^\top$. Looking at $\boldsymbol{R}$ entry-wise, we get $\boldsymbol{R}_{u,v} = \frac{\boldsymbol{d}(u)\boldsymbol{d}(v)}{\boldsymbol{d}(V)}$. So in other words, $\boldsymbol{R}$ is the adjacency matrix of the $\boldsymbol{d}$-product graph. Now, observe that if $\boldsymbol{x} = \mathbf{1}_S$ and $\boldsymbol{y} = \mathbf{1}_T$, we get $\boldsymbol{x}^\top \boldsymbol{A}_G \boldsymbol{y} = |E(S,T)|$, and $\boldsymbol{x}^\top \boldsymbol{R}\boldsymbol{y}$ equals $\boldsymbol{d}(S)\boldsymbol{d}(T)/\boldsymbol{d}(V)$. Therefore, we have:

$$\sigma_2 \geq \frac{\left| |E(S,T)| - \frac{\boldsymbol{d}(S)\boldsymbol{d}(T)}{\boldsymbol{d}(V)} \right|}{\sqrt{\boldsymbol{d}(S)} \cdot \sqrt{\boldsymbol{d}(T)}}$$

which gives the lemma.

We can rewrite $\boldsymbol{A}'_G$ by making use of eigenvalue decomposition:

$$\boldsymbol{A}'_G = \boldsymbol{v}_1\boldsymbol{v}_1^\top + \lambda_2(\boldsymbol{A}'_G)\boldsymbol{v}_2\boldsymbol{v}_2^\top + \cdots + \lambda_n(\boldsymbol{A}'_G)\boldsymbol{v}_n\boldsymbol{v}_n^\top.$$

By how we defined it, $\lambda_1(\boldsymbol{A}'_G) = 1$. Moreover, $\boldsymbol{v}_1 = \frac{1}{\sqrt{\boldsymbol{d}(V)}}\boldsymbol{d}^{1/2}$. This we can see by combining the fact that $\boldsymbol{A}'_G = \boldsymbol{I} - \boldsymbol{N}_G$ has the same eigenvectors as $\boldsymbol{N}_G$, and that the first eigenvector of $\boldsymbol{N}_G$ is $\frac{1}{\sqrt{\boldsymbol{d}(V)}}\boldsymbol{d}^{1/2}$.

So we have:

$$\begin{aligned}
\sigma_2 &= \max_{z,w} \frac{|z^\top (A'_G - v_1 v_1^\top) w|}{\|z\| \cdot \|w\|} \\
&= \max_{x,y} \frac{|x^\top D^{1/2} (A'_G - v_1 v_1^\top) D^{1/2} y|}{\sqrt{\sum_v d(v) x_v^2} \cdot \sqrt{\sum_v d(v) y_v^2}} && z \mapsto D^{1/2} x, w \mapsto D^{1/2} y \\
&= \max_{x,y} \frac{|x^\top (A_G - R) y|}{\sqrt{\sum_v d(v) x_v^2} \cdot \sqrt{\sum_v d(v) y_v^2}}
\end{aligned}$$

The first equality is by the *variational characterization of singular values* (we prove this below). The last equality is because $A'_G = D^{-1/2} A_G D^{-1/2}$, and we can verify that

$$\begin{aligned}
D^{1/2} v_1 v_1^\top D^{1/2} &= D^{1/2} (\frac{1}{\sqrt{d(V)}} d^{1/2}) (\frac{1}{\sqrt{d(V)}} d^{1/2})^\top D^{1/2} \\
&= \frac{1}{d(V)} d \cdot d^\top \\
&= R
\end{aligned}$$

This completes the proof.

□

Corollary 4.2. *Let G be a d-regular graph. For every $S, T \subseteq V$, we have*

$$\left| |E(S,T)| - \frac{d \cdot |S| \cdot |T|}{dn} \right| \leq \sigma_2 d \sqrt{|S| \cdot |T|}.$$

It turns out that the converse of the Expander Mixing Lemma also holds.

Lemma 4.3. [2]*Let G be a d-regular graph. If*

$$\left| |E(S,T)| - \frac{d \cdot |S| \cdot |T|}{dn} \right| \leq \theta d \sqrt{|S| \cdot |T|}$$

for all two disjoint sets S, T. Then $\sigma_2 = O(\theta(1 + \log(\frac{d}{\theta}))$.

5 Ramanujan Graphs: The Best Expander w.r.t. Eigenvalues

To simplify the discussion, we will only talk about d-regular in this section. A natural question to ask at this point is: How small σ_2 can be? A smaller σ_2 means that $E(S,T)$ is closer to $\frac{d|S||T|}{dn}$, for all $S, T \subseteq V$.

It turns out that σ_2 can be as small as $\frac{2\sqrt{d-1}}{d} \leq \frac{2}{\sqrt{d}}$. Graphs that achieve this bound are called **Ramanujan graphs**. Furthermore, this bound is tight! Roughly, we have that:

[2]https://link.springer.com/article/10.1007/s00493-006-0029-7

Theorem 5.1. *For every n_0 and d_0, there exist $n \in [n_0, O(n_0)]$ and $d \in [d_0, O(d_0)]$ such that, there is a d-regular graph G with n vertices satisfying:*

$$\sigma_2 \leq \frac{2\sqrt{d-1}}{d} = O(\frac{1}{\sqrt{d}}).$$

That is, its second eigenvalue (in absolute values) of the normalized adjacency matrix is at most $\frac{2\sqrt{d-1}}{d}$.

Question 5.2 (Open)**.** Can we prove that this holds for all n and d?

Adam Marcus, Daniel Spielman and Nikhil Srivatsava [3] proved that it holds for bipartite graphs. That is, $\lambda_n(A'_G) = -1$, but $\max_{i \neq 1,n} \lambda_i(A'_G) \leq \frac{2\sqrt{d-1}}{d}$. This led to the resolution of the *Kardison Singer problem*[4]

5.1 Tightness of Ramanujan Graphs

One can ask whether Ramanujan expanders are the best expanders with respect to eigenvalues. That is: does there exist another family of graphs for which σ_2 is strictly smaller than $\frac{2\sqrt{d-1}}{d}$?

It is known that this is not the case, it has been shown that $\sigma_2 \geq \frac{2\sqrt{d-1}}{d}(1 - o(1))$.[5] That is, Ramanujan graphs are the best expanders w.r.t. eigenvalues. In this lecture, we will show a slightly less general version of this bound.

Lemma 5.3. *For any d-regular graph where $d \leq 0.99 \cdot n$, we have $\sigma_2 \geq \Omega(\frac{1}{\sqrt{d}})$.*

Proof. We have

$$\begin{aligned} \mathrm{Tr}[(A'_G)^2] &= \sum_{i=1}^{n} \lambda_i(A'_G)^2 \\ &\leq 1 + (n-1)\sigma_2^2 \end{aligned}$$

upon rearranging the terms, we get:

$$\sigma_2 \geq \sqrt{\frac{\mathrm{Tr}[(A'_G)^2] - 1}{(n-1)}}$$

Note $(A_G^2)_{u,v}$ is just the number of 2-step walks from u to v. So $\mathrm{Tr}[A_G^2] = nd$. Also, when G is d-regular, we have $A'_G = A_G/d$, and therefore, $\mathrm{Tr}[(A'_G)^2] = (nd)/d^2 = n/d$ holds. Substituting these values into the last inequality above, we get:

$$\sigma_2 \geq \sqrt{\frac{n/d - 1}{(n-1)}} = \sqrt{\frac{(n-d)}{d(n-1)}} = \Omega(1/\sqrt{d})$$

when $d \leq 0.99 \cdot n$. □

[3] https://arxiv.org/abs/1304.4132

[4] *https://arxiv.org/abs/1306.3969*

[5] https://www.sciencedirect.com/science/article/pii/0012365X9190112F?via%3Dihub

5.2 Some Properties of Ramanujan Graphs

Let G be a d-regular Ramanujan graph. Then:

Corollary 5.4. *For any $S, T \subseteq V$, if $|S| \cdot |T| \cdot d > 4n^2$, then $E(S,T) \neq \emptyset$.*

That is, if we take two vertex sets which are large enough, then in Ramanujan graphs, there is guaranteed to be an edge between those two sets.

Proof. We prove the contra-positive. Suppose $E(S,T) = \emptyset$. The Expander Mixing Lemma for d-regular graphs gives us:

$$\left| |E(S,T)| - \frac{d \cdot |S| \cdot |T|}{dn} \right| \leq \sigma_2 d \sqrt{|S| \cdot |T|}$$

which implies that

$$\frac{d|S| \cdot |T|}{n} \leq 2\sqrt{d} \cdot \sqrt{|S| \cdot |T|} \iff |S| \cdot |T| \cdot d \leq 4n^2.$$

□

Example 5.5. Say $d = \sqrt{n}$. Consider S and T. If $|S|, |T| \geq 2n^{3/4}$, then $E(S,T) \neq \emptyset$. But in fact, if $|S|, |T| \geq 4n^{3/4}$ which are a bit bigger than what we required before, then there will be considerable amount of edges between then, i.e. $|E(S,T)| = \Omega(d|S||T|/n) = \Omega(n)$.

Corollary 5.6. *Any independent set S has size at most $2n/\sqrt{d}$.*

Proof. As $E(S,S) = \emptyset$ by the definition of an independent set, then $|S|^2 \cdot d \leq 4n^2$. So $|S| \leq 2n/\sqrt{d}$.
□

Exercise 5.7. Is there a d-regular graph that is better than Ramanujan graph? Say, for all S, T where $|S|, |T| \geq 2\sqrt{n}$, we have $E(S,T) \neq \emptyset$?

6 Expander: Conductance vs Eigenvalue

Recall Cheeger's inequality:

$$\frac{\lambda_2(\boldsymbol{N}_G)}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G)}.$$

Roughly, it claims the following fact: $\Phi(G)$ is big $\iff$ $\lambda_2(\boldsymbol{N}_G)$ is big. But in the *very well-connected* regime, the implication may not necessarily go both ways.

6.1 Eigenvalue can be stronger than Conductance

When the conductance $\Phi(G) = 1 - o(1)$ is almost maximum, we cannot always conclude that $|E(S,T)| \approx \frac{\boldsymbol{d}(S)\boldsymbol{d}(T)}{\boldsymbol{d}(V)}$ for all $S, T \subseteq V$. Take the case of the star graph: it has conductance 1. But for S, T such that they are equal sized partitions of the outer vertices, we get $|E(S,T)| = 0$, whereas $\frac{\boldsymbol{d}(S)\boldsymbol{d}(T)}{\boldsymbol{d}(V)}$ is roughly $O(n)$.

In general, by Cheeger's inequality, $\lambda_2(\boldsymbol{N}_G) \geq \Phi(G)^2/2 = \frac{1}{2} - o(1)$. Therefore, we can only bound $\sigma_2 \leq \frac{1}{2} + o(1)$. In this particular case, even when $\mathrm{vol}(S), \mathrm{vol}(T) = \mathrm{vol}(V)/2$, the Expander Mixing Lemma does not guarantee an edge between S and T. That is, $E(S,T) = \emptyset$ possibly.

6.2 Conductance can be stronger than Eigenvalue

Going the other way, there are cases when the eigenvalue can imply strong expansion, but the conductance of that graph might not be as good. To see one such case, suppose G is d-regular and $\sigma_2 \leq O(\frac{1}{\sqrt{d}})$ is almost minimum. Then, we have:

$$\lambda_2(\boldsymbol{N}_G) \geq 1 - \frac{1}{O(\sqrt{d})}.$$

By Cheeger's inequality, we only get:

$$\Phi(G) \geq \lambda_2(\boldsymbol{N}_G)/2 \geq \frac{1}{2} - o(\frac{1}{O(\sqrt{d})}).$$

Therefore, Cheeger's does not allow us to get strong lower bounds on the conductance, in this case. In particular, it does not imply $\Phi(G) \approx (1-\epsilon)$.

A d-regular graph G where $\Phi(G) \approx (1-\epsilon)$ is called a **lossless expander**, and it has lots of applications in:

- Complexity theory (pseudo-randomness)
- Streaming algorithms (deterministic sparse recovery).
- Dynamic algorithms (dynamic matching)[6]

Note that d should be small, and G should be regular. (Otherwise easy: consider clique and star). There is a bipartite version which is important as well.

6.3 Proof of variational characterization of singular values

Theorem 6.1. *Let $\mathbf{A'_G} = \mathbf{I} - \mathbf{N_G}$. Let σ_2 denote the second largest eigenvalue in terms of absolute value. Then,*

$$\sigma_2 = \max_{z,w} \frac{|z^\top (\boldsymbol{A}'_G - \boldsymbol{v}_1 \boldsymbol{v}_1^\top) w|}{\|z\| \cdot \|w\|}$$

Proof. For any matrix $\boldsymbol{M}$, we write the singular value decomposition of $\boldsymbol{M}$ as

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal = \sum_i \sigma_i(\boldsymbol{M}) \cdot \boldsymbol{u}_i \boldsymbol{v}_i^\top$$

where $\boldsymbol{u}_i, \boldsymbol{v}_i$ are unit vectors and $\sigma_i(\boldsymbol{M})$ are the singular values of $\boldsymbol{M}$. Note that this generalizes eigenvalue decomposition, and holds for any $m \times n$ matrix $\mathbf{M}$. Chapter 3 of this book gives a gentle introduction to singular value decomposition and mentions a lot of applications:

https://home.ttic.edu/~avrim/book.pdf

We will need the following fact:

[6] https://arxiv.org/abs/2108.10461

Fact 6.2. *For each i, we have :*

$$\boldsymbol{M}\boldsymbol{v}_i = \sigma_i(\boldsymbol{M})\boldsymbol{u}_i.$$

Furthermore, the first (largest) singular value $\sigma_1(\boldsymbol{M})$ of $\boldsymbol{M}$ is such that

$$\sigma_1(\boldsymbol{M}) = \|\boldsymbol{M}\|_2 = \max_{\boldsymbol{x}} \frac{\|\boldsymbol{M}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}.$$

Lemma 6.3 (Variational Characterization of the First Singular Values)**.** *We have*

$$\sigma_1(\boldsymbol{M}) = \max_{\boldsymbol{z},\boldsymbol{w}} \frac{\boldsymbol{z}^\top \boldsymbol{M}\boldsymbol{w}}{\|\boldsymbol{z}\| \cdot \|\boldsymbol{w}\|}.$$

Proof. For any unit vector $\boldsymbol{z}$ and $\boldsymbol{w}$, by Cauchy-Schwarz we have

$$\boldsymbol{z}^\top \boldsymbol{M}\boldsymbol{w} \leq \|z\| \cdot \|\boldsymbol{M}\boldsymbol{w}\| \leq \|z\| \cdot \|\boldsymbol{M}\|_2 \cdot \|\boldsymbol{w}\| = \sigma_1(\boldsymbol{M})\|z\| \cdot \|\boldsymbol{w}\|.$$

That is,

$$\sigma_1(\boldsymbol{M}) \geq \max_{\boldsymbol{z},\boldsymbol{w}} \frac{\boldsymbol{z}^\top \boldsymbol{M}\boldsymbol{w}}{\|\boldsymbol{z}\| \cdot \|\boldsymbol{w}\|}.$$

But we know that this equality is attained for the unit vectors $\boldsymbol{u}_1$ and $\boldsymbol{v}_1$ because

$$\boldsymbol{u}_1^\top \boldsymbol{M}\boldsymbol{v}_1 = \boldsymbol{u}_1^\top(\sigma_1(\boldsymbol{M})\boldsymbol{u}_1) = \sigma_1(\boldsymbol{M}).$$

This prove the lemma. □

Claim 6.4. *For symmetric matrix $\boldsymbol{M}$, $\sigma_1(\boldsymbol{M}) = \max\{|\lambda_1(\boldsymbol{M})|, |\lambda_n(\boldsymbol{M})|\}$.*

Proof. We have that

$$\begin{aligned}(\sigma_1(\boldsymbol{M}))^2 = \|\boldsymbol{M}\|_2^2 &= \max_{\boldsymbol{x}} \frac{\|\boldsymbol{M}\boldsymbol{x}\|^2}{\|\boldsymbol{x}\|^2} \\ &= \max_{\boldsymbol{x}} \frac{\boldsymbol{x}^\top \boldsymbol{M}^\top \boldsymbol{M}\boldsymbol{x}}{\boldsymbol{x}^\top \boldsymbol{x}} \\ &= \lambda_1(\boldsymbol{M}^2).\end{aligned}$$

But we know that $\lambda_1(\boldsymbol{M}^2) = (\max\{|\lambda_1(\boldsymbol{M})|, |\lambda_n(\boldsymbol{M})|\})^2$. This proves the claim. □

When we let $\boldsymbol{M} = \boldsymbol{A}'_G - \boldsymbol{v}_1\boldsymbol{v}_1^\top$, we have

$$\begin{aligned}\sigma_1(\boldsymbol{M}) &= \max\{|\lambda_1(\boldsymbol{A}'_G - \boldsymbol{v}_1\boldsymbol{v}_1^\top)|, |\lambda_n(\boldsymbol{A}'_G - \boldsymbol{v}_1\boldsymbol{v}_1^\top)|\} \\ &= \max\{|\lambda_2(\boldsymbol{A}'_G)|, |\lambda_n(\boldsymbol{A}'_G)|\} \\ &= \sigma_2\end{aligned}$$

by the definition of σ_2 from the lecture. (Basically, we defined $\sigma_2 = \sigma_2(\boldsymbol{A}'_G)$.) We can conclude now that

$$\sigma_2(\boldsymbol{A}'_G) = \sigma_1(\boldsymbol{M}) = \max_{\boldsymbol{z},\boldsymbol{w}} \frac{|\boldsymbol{z}^\top(\boldsymbol{A}'_G - \boldsymbol{v}_1\boldsymbol{v}_1^\top)\boldsymbol{w}|}{\|\boldsymbol{z}\| \cdot \|\boldsymbol{w}\|}$$

by the variational characterization of $\boldsymbol{M}$.

□

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 10: Deterministic Vertex Connectivity via Ramanujan Graphs

September 30, 2021

Instructor: Thatchaphol Saranurak

Scribe: Chaitanya Nalam

- Today, we will show an application of Ramanujan graphs to fast graph algorithms.

1 Vertex Connectivity

Let $G = (V, E)$ be an unweighted and undirected graph. We introduce some basic definitions related to vertex connectivity below.

Definition 1.1 (Vertex Connectivity)**.** Given a connected graph G, the smallest number of vertices needed to be deleted from G to disconnect G.

Definition 1.2 (Vertex cut)**.** For $G = (V, E)$, partition (L, S, R) on set of vertices V is said to be a vertex cut if $L, R \neq \phi$ and $E_{G \setminus S}(L, R) = \phi$, i.e. vertices in S separates L from R. $|S|$ is called the size of vertex cut.

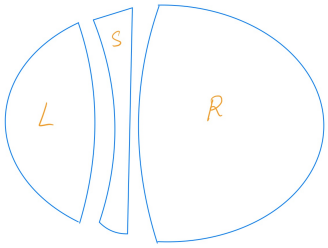


Figure 1: Vertex Cut

The size $|S^*|$ of the smallest possible vertex cut (L^*, S^*, R^*) is equal to the vertex connectivity of graph G denoted by $\kappa(G)$ or just κ.

Definition 1.3 ((s,t)−vertex cut)**.** A set $S \subset V$ is said to be a (s,t)− vertex cut for vertices $s, t \in V$ if removing the nodes in S will disconnect vertex s from t. It is also called (s,t)−separator.

Definition 1.4 ((s,t)− vertex mincut)**.** The smallest possible (s,t)− vertex cut is called (s,t)− vertex mincut or minimum (s,t)−separator and denote its size by $\kappa(s,t)$.

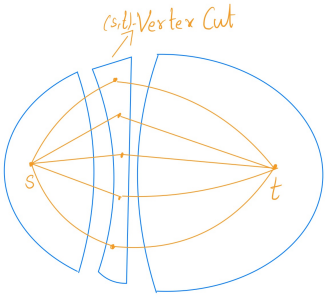


Figure 2: (s,t)-Vertex Cut

1.1 Naive Method

By max flow min cut theorem $\kappa(s,t)$ is the maximum number of vertex disjoint paths. Hence, we can compute $\kappa(s,t)$ in one max-flow call.

Hence, $\kappa(G)$ is obtained by taking minimum over $\kappa(s,t)$ all possible (s,t) pairs in the graph and finding the minimum separator of the graph.

$$\kappa(G) = \min_{s,t} \kappa(s,t)$$

However, this naive method takes $O(n^2)$ max flow calls. Today we will see a deterministic algorithm that takes $O(n^{1.5} \log n)$ max flow calls based on the paper by [Gabow'00].

1.2 State of the Art

There is a deterministic algorithm by [Gabow'00] that takes $O(m \cdot (n + \min\{\kappa^{5/2}, \kappa n^{3/4}\}) = O(mn^{1.75})$ time when κ is as big as $O(n)$. [1] It is the only fast algorithm that exploits Ramanujan Graphs as far as we know. This algorithm is much slower than the current algorithm that we present.

There is a randomized algorithm by [LNP**S**Y'21] which takes polylog(n) "max flow calls". Current best algorithm for max flow runs in $O(m^{4/3+o(1)})$ time. [2] The max flow calls are made on smaller graphs with fewer edges. The total size of the graph that they run max flow is still near-linear $\tilde{O}(m)$.

1.3 Overall Plan

Given an unweighted undirected graph G, the goal would be to return the minimum vertex cut (L, S, R) of size $\kappa(G)$.

Note: **We assume** $\delta \leq 9n/10$ to avoid annoying corner case where δ denote the minimum vertex degree.

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.1674&rep=rep1&type=pdf
[2] https://arxiv.org/abs/2104.00104

2 Algorithm for vertex connectivity

There are two cases depending on how balanced the vertex cut is. We first assume that the minimum vertex cut is balanced and find it. Next we reduce the unbalanced case to balanced case.

Definition 2.1 ($\beta-$balanced vertex cut)**.** A vertex-cut (L, S, R) is said to be β**-balanced** if $|L| \geq \beta$ and $|R| = \Omega(n)$.

The main idea underlying the algorithm is as follows. Let (L^*, S^*, R^*) be the minimum vertex cut of size k. If we some how know a vertex $x \in L^*$ and $y \in R^*$, then we can do a max flow call to find minimum $(x, y)-$ vertex cut (L, S, R) whose size is same as $|S^*|$ (although exact cut may be different). This is because S^* is a $(x, y)-$vertex cut whence $\kappa(x, y) \leq k$; otoh since (L^*, S^*, R^*) is minimal, $\kappa(x, y) \geq k \implies \kappa(x, y) = k$.

However, we do not know which pair of vertices belongs to L^*, R^* respectively, whence the naive way is to try all possible pairs however this may take as large as $O(n^2)$ max flow calls.

But can we reduce the number of max flow calls using the "promise" that our vertex cut is $\beta-$balanced?

Yes, this is our next idea for an improved algorithm. By using the structural property of the Ramanujan Graphs we can decrease the number of pairs on which we need to call max flow, for finding the $\beta-$balanced cut. We will state the structural property of the Ramanujan Graph here without proof.

Corollary 2.2. *Let $H = (V, E)$ be a d-regular Ramanujan graph. For any $L, R \subseteq V$, if $|L| \cdot |R| \cdot d > 4n^2$, then $E(L, R) \neq \emptyset$.*

We want to identify a vertex pair such that they belong to either sides of the minimum vertex cut (L^*, S^*, R^*). From the above corollary we have that whenever $|L| \cdot |R| \cdot d > 4n^2$ then there always exist an edge from L to R.

Since we want to find a vertex pair between L^* and R^* when $|L^*| \geq \beta$ and $|R^*| = \Omega(n)$ we can as well set $d = \Theta(n/\beta)$ such that $|L^*| \cdot |R^*| \cdot d > 4n^2$ is satisfied and then by the property of Ramanujan Graph (H) we have a guaranteed edge between L^*, R^*.

However, we do not know which edge is between L^* and R^* (or really, we don't know where L^* and R^* are) so we use all edges of Ramanujan Graph (E_H) as candidate pairs for calling max flow and take the minimum value of the vertex cut between all of them. By the property of Ramanujan Graph and the balanced nature of minimum vertex cut we are guaranteed to find such a pair which gives us minimum cut.

Lemma 2.3. *If there exists a β-balanced vertex-mincut (L^*, S^*, R^*), then we can find a vertex-cut of size at most k in $O(n \times \frac{n}{\beta})$ max flow calls.*

Proof. We use the edges of H, E_H to get candidate pairs for finding the balanced cut. Since there can be at most $n \cdot d$ edges in a $d-$regular graph, $|E_H| = O(n^2/\beta)$ which implies at most $O(n^2/\beta)$ many max flow calls. □

Note that H is totally different graph from G but shares the same vertices.

3 Reduction to Balanced Case

We saw above that we can improve over the naive $O(n^2)$ max flow calls to $O(n^2/\beta)$ calls when we know that minimum vertex cut is balanced. However we do not know if the (L^*, S^*, R^*) is balanced. So in this section we present a way to reduce the unbalanced case to balanced case.

Our plan would be to gradually modify the graph to make it balanced and then solve it in the above mentioned way. We measure the balanced-ness of the minimum cut using a condition called the **gap condition**. We keep on improving gap condition until that it is sufficient to imply "every" minimum cut is $\beta-$balanced.

Note that balanced case requires that at least one of the all possible minimum cuts need to be balanced. But gap condition ensures that all minimum cuts are balanced.

3.1 Gap Condition

We know that $\kappa(G) = \kappa \leq \delta$ because a vertex with degree δ can be separated from the rest of the graph by removing its neighbours which are of size δ. Hence minimum cut can only be either smaller or equal.

Since $\kappa \leq \delta$, the difference between the min degree and min cut, $\delta - \kappa$ is called **Gap**. Why is this quantity useful? It is formalized in the lemma below.

Lemma 3.1. *Every mincut (L^*, S^*, R^*) is β-balanced where $\beta = \delta - \kappa$.*

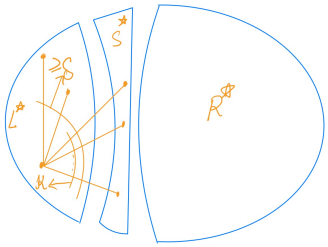


Figure 3: Gap Condition

Proof. Let $x \in L^*$ (we know L^* is non empty) and $N(x)$ be the set of x's neighbors. We have $N(x) \subseteq L^* \cup S^*$ as it cannot have edges to R^*.

$$
\begin{aligned}
|N(x)| &\geq \delta && \text{(minimum degree)} \\
|N(x)| &\leq |L^*| + |S^*| = |L^*| + \kappa \\
|L^*| &\geq \delta - \kappa
\end{aligned}
$$

There are always $\delta - \kappa$(**the gap** between minimum degree and minimum cut) vertices on both sides of the minimum cut. □

So to make the minimum cut balanced we need to increase the gap $\delta - \kappa$. We need more structural understanding of the vertex connectivity to see why we can do this and how to do this.

Definition 3.2 (x-rooted vertex connectivity)**.** The minimum number of vertices needed to disconnect x from any other vertex in the graph.

$\kappa(x) = \min_{y \neq x} \kappa(x, y)$ which can be computed in $n - 1$ max flows.

Lemma 3.3. *If* $\kappa(x) > \kappa(G)$*, then* $\kappa(G \setminus x) = \kappa(G) - 1$.

Proof. If $\kappa(x) > \kappa(G)$ then for all possible min cuts (L^*, S^*, R^*), $x \in S^*$ if not $\kappa(x) = \kappa(G)$.

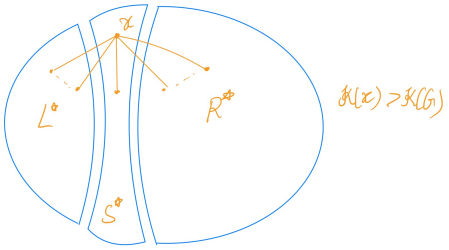


Figure 4: Remove vertex to decrease κ

$\kappa(G \setminus x)$ cannot be $< \kappa(G) - 1$ because then we can add x back and get a min cut for graph G of size $< \kappa(G)$ which is contradiction.

$\kappa(G \setminus x)$ need not be $> \kappa(G) - 1$ because we already have a cut of size $= \kappa(G) - 1$ which is $S^* \setminus \{x\}$ where S^* is any min cut separator and we also know that $x \in S^*$. □

We use the above structural lemma as follows. Compute $\kappa(x)$ along with the corresponding cut. If $\kappa(x) = \kappa$ then we have a found a vertex cut of optimal size. Otherwise $\kappa(x) > \kappa$ then remove x from the graph and compute $\kappa(G \setminus x)$ and according to above lemma we know that $\kappa(G) = \kappa(G \setminus x) + 1$.

From now on we use κ to always refer to the minimum vertex cut size of the current graph which is $G \setminus x$ after x is removed.

However, if we repeat the above step it may take $O(n)$ steps to find min-cut as κ can be $O(n)$. Since each step takes $O(n)$ max flows we are again at square one. Recall our target is to increase the gap between δ and κ.

Using the above step we have reduced κ by 1. So gap should increase if δ has stayed the same. However, by removing x, δ can also decrease by 1 if $N(x)$ contains some vertex of minimum degree, in which case the gap cannot be increased.

Let F be the set of neighbors of x such that $\forall x' \in F$, the degree of x' become $\delta - 1$ after removing x. So all nodes in F need to be "fixed" so that their degree become δ again. We can fix them without affecting the min cut size for which we need the following structural lemma.

Lemma 3.4. *If* $\kappa(x, y) > \kappa$*, then* $\kappa(G \cup (x, y)) = \kappa(G)$.

Proof. $\kappa(x, y) > \kappa$ then x, y always belong to same side either $L^* \cup S^*$ or $S^* \cup R^*$ for every possible min cut (L^*, S^*, R^*). If not $\kappa(x, y)$ would have been κ. So adding an edge between x, y (if it does not already exist) does not affect any minimum vertex cut. □

For any vertex w compute $\kappa(w)$ and if it is $= \kappa$ then we are done, else we can add edges to any other vertex from this without effecting the min cut as it belongs to S^* for every min cut (L^*, S^*, R^*).

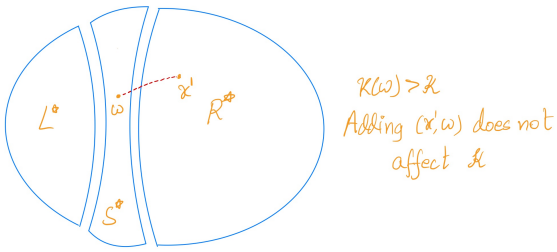


Figure 5: Add edge (w, x') to G

- Let $F_w = \{x' \in F \mid (x', w) \notin E\}$ be all nodes that need to be fixed, and **not** adjacent to w.
- For each $x' \in F_w$, we can add the edge (x', w) into the graph without changing κ.
- So all nodes in F_w are now fixed!

However, F_w might be very small when compared to F and may take many max flows to fix all vertices in F. We show below that it is not the case.

Lemma 3.5. *Let G' be the current graph after removing x. There exists $w \in V(G')$ such that*

$$|F_w| \geq |F| \cdot (1 - \frac{\delta - 1}{|V(G')|})$$

.

Proof. We have $\text{vol}_{G'}(F) \leq |F|(\delta - 1)$ as each vertex in F is of degree $\delta - 1$. So there is a vertex $w \in V(G')$ such that $|E_{G'}(w, F)| \leq \frac{|F|(\delta-1)}{|V(G')|}$.

That is, w is adjacent to at most $\frac{|F|(\delta-1)}{|V(G')|}$ vertices in F. So we can fix at least $|F| \cdot (1 - \frac{\delta-1}{|V(G')|})$ vertices in F by adding edges to w. □

We will see that the algorithm will guarantee $|V(G')| \geq n - \sqrt{n}$, and we assume $\delta \leq \frac{9}{10}n$ from the beginning. Hence, $|F_w| \geq |F| \cdot (1 - \frac{\delta-1}{|V(G')|}) \geq |F|/100$. So we fix at least $(1/100)^{th}$ fraction of vertices in F every time by incurring an overhead of n max flows for computing $\kappa(w)$.

We find w by scanning through all possible vertices and find the vertex w that fixes at least $(1/100)^{th}$ fraction of vertices in F, which is guaranteed to exist based on the above lemma. So it takes a total of $O(\log n)$ rooted vertex connectivity calls to fix all vertices. Once all vertices are fixed we have successfully increased the minimum degree to δ without increasing the minimum vertex cut $\kappa(G \setminus x) = \kappa(G) - 1$ and hence increasing the gap by 1. This took a total of $O(n \log n)$ max flow calls.

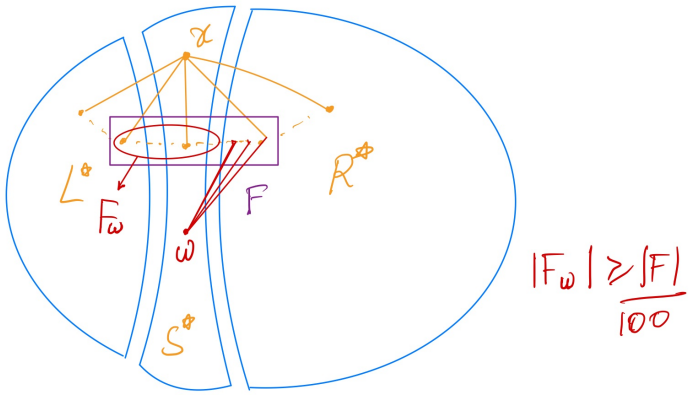


Figure 6: Vertex Cut

Remark 3.6. Note that there is an easy way to just find any arbitrary non-neighbour w of a vertex for a $x' \in F$ and compute $\kappa(x', w)$.

- If $\kappa(x', w) = \kappa$, since we track of minimum vertex cut we found so far, we are done.
- If $\kappa(x', w) > \kappa$, then we can fix $x' \in F$ by adding edge to w.

iHowever we do not choose the above said process and use $O(\log n)$ rooted vertex connectivity calls because, if later we can speed up the algorithm for problem of finding rooted vertex connectivity faster than $o(n)$ max flow calls which immediately implies an improved algorithm for vertex connectivity problem.

Finding rooted vertex connectivity in $o(n)$ max flow calls is an independent and very interesting open problem.

3.2 Summary of the Algorithm in the Unbalanced Case

Algorithm 1: $IncreaseGap(G)$

Result: Graph in which the gap, $\delta - \kappa$, is increased by 1
Choose any $x \in V$. Compute $\kappa(x)$ and the corresponding cut
Let $G' = G \setminus x$
Let $F = \{x' \in N(x) \mid \deg_{G'}(x') = \delta - 1\}$
while $F \neq \phi$ **do**
  Iterate over V to find w such that $|F_w| \geq |F|/100$
  Compute $\kappa(w)$ and the corresponding cut and set $\hat{\kappa}_{G'} \leftarrow \min\{\hat{\kappa}_{G'}, \kappa(w)\}$
  For each $x' \in F_w$, add edge (x', w) to G'
end

Note that if $\kappa(x) > \kappa$, then the gap is increased by 1 and our candidates for minimum cut are $\kappa(x)$ and $\hat{\kappa}_{G'} + 1$; otherwise if $\kappa(x) = \kappa$ or $\kappa(w) = \kappa$, then we had found the minimum vertex cut of G and $\kappa(G)$. If we repeated this process for β times and in each round we have $\kappa(x) > \kappa$ we could increase the gap by β, the vertex min cut in the resulting graph is β-balanced. Meanwhile we keep

track of the current smallest cut and update it using $\min\{\kappa(x), \hat{\kappa}_{G'} + 1\}$ and the corresponding cut. In this way, if at any round we found κ, then it will be recorded.

We finally use the balanced case to solve this graph for minimum vertex cut in the resulting graph and then ripple back through the deletions of vertices to find the minimum cut among all possible candidates.

3.3 Correctness

All the while we are comparing our rooted vertex connectivities with the minimum vertex connectivity κ however we do not know it before hand.

It is not an issue as we only over estimate the minimum cut. If we ever find the minimum cut exactly somewhere in our algorithm. Since we store all the cut values and take the minimum. Hence, we are guaranteed to find the minimum cut.

If we never found the minimum cut while reducing to balanced case, then according to our algorithm we will find the minimum cut after reducing it to balanced case. So, either way the minimum of all our cuts will give us the minimum cut.

3.4 Runtime

- We call $\beta \times O(n \log n)$ max flows to reduce the problem to the β-balanced case.
- In the β-balanced case, we can solve the problem in $O(n \times \frac{n}{\beta})$ max flows.
- So a total of $O(\frac{n^2}{\beta} + \beta n \log n)$ max flow calls. By choosing $\beta = \sqrt{n}$, you can solve the problem in $O(n^{1.5} \log n)$ max flow calls.

4 Exercises and Open Questions

Exercise 4.1. Show an algorithm for vertex connectivity that takes $O(n \cdot \kappa)$ max flow calls.

Proof. At the i^{th} iteration, pick an arbitrary point $x_i \in V$ and compute $\kappa(x)$ using $O(n)$ max flow calls and record $k_i = \min_{j \le i} \kappa(x_j)$. If $k_i \le i - 1$, then return k_i. The algorithm is going to terminate since as long as $i > \kappa$, then we are guaranteed to pick a point that is not in S^* where (L^*, S^*, R^*) is a minimum vertex cut, whence the algorithm halts in at most $\kappa + 1$ iterations. The algorithm is correct since if $i \le \kappa$ and $k_i > \kappa \ge i$ then the algorithm won't halt. □

Question 4.2 (Open)**.** *Can we deterministically solve vertex connectivity using $O(n)$ max flow calls or even less?*

Note the that for randomized case, it is just polylog(n) max flows now. Something should be possible!

Question 4.3 (Open)**.** *The state of the art for **weighted** vertex connectivity.*

- *Randomized:* $\tilde{O}(mn)$
- *Deterministic:* $\tilde{O}(m^2)$

- So if you can find
  - a randomized algorithm using $o(n)$ max flow
  - a deterministic algorithm using $o(n^2)$ max flow
- This would improve the state of the art.
- Any hardness from fine-grained complexity would be interesting too!

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 11: Expander: Probabilistic view

October 5, 2021

Instructor: Thatchaphol Saranurak

Scribe: Aditya Anand

1 A probabilistic view of expanders

In this lecture, we shall study yet another characterization of expanders - **expanders are graphs on which random walks mix rapidly.** We shall see that there is a direct connection between the second eigenvalue of the normalized laplacian and the mixing time of a random walk on graphs.

2 Random Walks

Let G be a graph with degree profile $\boldsymbol{d}$. We define a random walk step starting from a given vertex as follows.

Definition 2.1. (Random Walk Step) Given an undirected unweighted graph G and a vertex u, the result of one step of a random walk from u is a uniformly random neighbour of u in G.

The same definition can naturally be extended to weighted graphs.

Definition 2.2. (Random Walk Step in Weighted Graphs) Given an undirected weighted graph G with weight function $w : V(G) \to \mathbb{R}$ and a vertex u, the result of one step of a random walk from u is the random vertex obtained by sampling a single vertex from $N_G(u)$ where $x \in N_G(u)$ is sampled with probability $\frac{w(u,x)}{\sum_{v \in N_G(u)} w(u,v)}$.

Today's lecture is motivated by the following natural questions, which can be better understood with an example.

Example 2.3. Given a cycle of length n, if we start at some vertex u and do a random walk for a sufficiently large number of steps, what is the distribution of vertices we end up with?

Intuitively, we understand that the answer is the uniform distribution. But can we prove this? We can also ask how many steps does it take so that $\Pr[\text{end at } u] = \frac{1 \pm \epsilon}{n}$ for all u - that is, how fast do we converge to the uniform distribution?

For a general graph, these questions become

- Do random walks converge?

Figure 1: Random walk illustration

- If so, to what distribution does the random walk converge?
- What is the rate of convergence - how fast does the random walk "mix" or converge to the final distribution?

The goal of this lecture will be to answer these questions, and understand these in the context of expanders - we will show that expanders have small mixing times.

3 Walk Matrices

Let us now formalize the notion of a random walk. Let $\boldsymbol{p}_t \in \mathbb{R}^V$ denote the probability distribution over vertices after time t. Suppose initially we start at a vertex u, so $\boldsymbol{p}_0 = \mathbf{1}_u$. Then $\boldsymbol{p}_1 = \frac{1}{\deg(u)} \cdot \mathbf{1}_{N(u)}$ where $N(u)$ is the set of neighbors of u. It is clear that

$$\boldsymbol{p}_{t+1}(u) = \sum_{(v,u)\in E} \frac{w(u,v)}{d(v)} \boldsymbol{p}_t(v).$$

Definition 3.1. (Walk matrix) The walk matrix $\boldsymbol{W}$ is the matrix $\boldsymbol{W}$ that satisfies $\boldsymbol{p}_{t+1} = \boldsymbol{W}\boldsymbol{p}_t$. From the above expression, we can see that

$$\boldsymbol{W} = \boldsymbol{A}\boldsymbol{D}^{-1}.$$

After t random walk steps, we have

$$\boldsymbol{p}_t = \boldsymbol{W}\boldsymbol{p}_{t-1} = \boldsymbol{W}^t \boldsymbol{p}_0.$$

We can now formalize the questions from the previous section as follows.

- Convergence - What is the limit of $\boldsymbol{p}_t$ when $t \to \infty$ (if it exists)?
- Rate of convergence - How fast does $\boldsymbol{p}_t$ converge to that limit?

Unfortunately, it turns out that the above limit may not exist for a general graph. In fact, consider a trivial graph consisting of only one edge (u, v). If we do a random walk starting from u, then at every odd step we are at v with probability 1 and at every even step we are at u with probability 1 - from which it is clear that the limit $\boldsymbol{p}_t$ when $t \to \infty$ does not exist.

In fact this limit does not exist for any bipartite or disconnected graph[1] - at every step the total probability mass on one partite set is zero. This suggests that we need a different notion of random walk to ensure convergence for every graph. Does there exist a natural random walk that converges for any graph? The answer is indeed yes, as we shall see in the subsequent section.

4 Lazy Random Walk

We define a lazy random walk step as follows.

Definition 4.1. (Lazy Random Walk Step) Given an undirected unweighted graph G and a vertex u, the result of one step of a lazy random walk is u with probability $\frac{1}{2}$, otherwise it is the result of a random walk step with u.

Let $\boldsymbol{p}_t$ be the probability mass vector after t lazy random walk steps. We now define a new walk matrix $\widetilde{\boldsymbol{W}}$ corresponding to the lazy random walk, so that we have

$$\boldsymbol{p}_{t+1} = \widetilde{\boldsymbol{W}}\boldsymbol{p}_t,$$

What is the matrix W? Since we stay put with probability $\frac{1}{2}$ and do a random walk with probability $\frac{1}{2}$, it follows that

$$\begin{aligned}\widetilde{\boldsymbol{W}} &= \boldsymbol{I}/2 + \boldsymbol{W}/2 \\ &= \boldsymbol{I}/2 + \boldsymbol{A}\boldsymbol{D}^{-1}/2\end{aligned}$$

Let us now return to our trivial example graph with one edge (u, v) and ask the question if the limit of $\boldsymbol{p}_t = \widetilde{\boldsymbol{W}}^t \boldsymbol{p}_0$ when $t \to \infty$ exists. It is clear that $p_t(u) = p_t(v) = \frac{p_{t-1}(v)+p_{t-1}(u)}{2}$, and hence the lazy random walk converges to $p_t(u) = p_t(v) = \frac{1}{2}$ in just one step.

5 The Stable Distribution

Theorem 5.1. *In a connected undirected graph G, for any $\boldsymbol{p}_0 \in \mathbb{R}^V$,*

$$\lim_{t\to\infty} \widetilde{\boldsymbol{W}}^t \boldsymbol{p}_0 = \frac{1}{\boldsymbol{d}(V)} \cdot \boldsymbol{d}$$

Essentially, the above result says that irrespective of the initial distribution, the lazy random walk converges on any graph. Further, it converges to the same distribution, where each vertex has probability mass proportional to its degree.

Proof. We have

$$\begin{aligned}\boldsymbol{p}_t &= \widetilde{\boldsymbol{W}}^t \boldsymbol{p}_0 \\ &= \boldsymbol{D}^{1/2}\boldsymbol{D}^{-1/2}\widetilde{\boldsymbol{W}}^t\boldsymbol{D}^{1/2}\boldsymbol{D}^{-1/2}\boldsymbol{p}_0\end{aligned}$$

[1] It can be shown that this random walk does not converge if and only if G is disconnected or bipartite.

$$= \boldsymbol{D}^{1/2}(\boldsymbol{D}^{-1/2}\widetilde{\boldsymbol{W}}\boldsymbol{D}^{1/2})^t\boldsymbol{D}^{-1/2}\boldsymbol{p}_0$$

Recall that

$$\widetilde{\boldsymbol{W}} = \frac{1}{2}(\boldsymbol{I} + \boldsymbol{A}\boldsymbol{D}^{-1})$$

So

$$\boldsymbol{D}^{-1/2}\widetilde{\boldsymbol{W}}\boldsymbol{D}^{1/2} = \frac{1}{2}(\boldsymbol{I} + \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}) = \boldsymbol{I} - \frac{1}{2}\boldsymbol{N}.$$

Now, we write

$$\boldsymbol{p}_t = \boldsymbol{D}^{1/2}(\boldsymbol{I} - \frac{1}{2}\boldsymbol{N})^t\boldsymbol{D}^{-1/2}\boldsymbol{p}_0.$$

Note that $\boldsymbol{N}$ and $\boldsymbol{I} - \frac{1}{2}\boldsymbol{N}$ have the same eigenvectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ and since

$$0 = \lambda_1(\boldsymbol{N}) \leq \cdots \leq \lambda_n(\boldsymbol{N}) \leq 2,$$

we have

$$1 = \lambda_1(\boldsymbol{I} - \frac{1}{2}\boldsymbol{N}) \geq \cdots \geq \lambda_n(\boldsymbol{I} - \frac{1}{2}\boldsymbol{N}) \geq 0$$

where

$$\lambda_i(\boldsymbol{I} - \frac{1}{2}\boldsymbol{N}) = 1 - \lambda_i(\boldsymbol{N})/2$$

We can make use of the eigen-decomposition as follows

$$\boldsymbol{D}^{-1/2}\boldsymbol{p}_0 = \sum_i c_i\boldsymbol{v}_i \qquad \text{where } c_i = \left\langle \boldsymbol{v}_i, \boldsymbol{D}^{-1/2}\boldsymbol{p}_0 \right\rangle$$

Therefore it follows that

$$\begin{aligned}
\boldsymbol{p}_t &= \boldsymbol{D}^{1/2}(\boldsymbol{I} - \frac{1}{2}\boldsymbol{N})^t \sum_i c_i\boldsymbol{v}_i \\
&= \boldsymbol{D}^{1/2} \sum_i \left(\lambda_i(\boldsymbol{I} - \frac{1}{2}\boldsymbol{N})\right)^t c_i\boldsymbol{v}_i \\
&= \boldsymbol{D}^{1/2}c_1\boldsymbol{v}_1 + \boldsymbol{D}^{1/2} \sum_{i\geq 2} \left(1 - \frac{\lambda_i(\boldsymbol{N})}{2}\right)^t c_i\boldsymbol{v}_i
\end{aligned}$$

Since $\lambda_i(\boldsymbol{N}) > 0$ for $i \geq 2$ because G is connected, we have

$$\boldsymbol{p}_t = \boldsymbol{D}^{1/2}c_1\boldsymbol{v}_1 \text{ when } t \to \infty$$

Recall that the first eigenvector $\boldsymbol{v}_1$ of $\boldsymbol{N}$ is $\boldsymbol{v}_1 = \frac{\boldsymbol{d}^{1/2}}{\sqrt{\boldsymbol{d}(V)}}$. So, we have

$$c_1 = \left\langle \frac{\boldsymbol{d}^{1/2}}{\sqrt{\boldsymbol{d}(V)}}, \boldsymbol{D}^{-1/2}\boldsymbol{p}_0 \right\rangle = \left\langle \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{d}(V)}}, \boldsymbol{p}_0 \right\rangle = \frac{1}{\sqrt{\boldsymbol{d}(V)}}$$

and

$$\boldsymbol{p}_t = \boldsymbol{D}^{1/2}\frac{1}{\sqrt{\boldsymbol{d}(V)}}\frac{\boldsymbol{d}^{1/2}}{\sqrt{\boldsymbol{d}(V)}} = \frac{1}{\boldsymbol{d}(V)} \cdot \boldsymbol{d}$$

when $t \to \infty$. □

6 Rate of Convergence

We call $\boldsymbol{\pi} = \lim_{t\to\infty} \widetilde{\boldsymbol{W}}^t \boldsymbol{p} = \frac{1}{\boldsymbol{d}(V)} \cdot \boldsymbol{d}$ the **stable/stationary distribution.**

Observation 6.1.

$$\boldsymbol{\pi} = \widetilde{\boldsymbol{W}}\boldsymbol{\pi}.$$

That is, doing a random walk from the stationary distribution keeps it intact.

Proof. Suppose that $\pi_{t-1}(v) = \frac{d(v)}{d(V)}$ for each vertex v. Fix a vertex x. The total probability mass that lands at x from y after one more random walk step is $\frac{1}{d(y)}\frac{d(y)}{d(V)}$ if $(x, y) \in E(G)$. Now summing over all such y we get $\pi_t(x) = \sum_{y\in N_G(x)} \frac{1}{d(V)} = \frac{d(x)}{d(v)} = \pi_{t-1}(x)$. □

This brings us to our next question on the rate of convergence. How fast does $\boldsymbol{W}^t\boldsymbol{p}$ converge to $\boldsymbol{\pi}$ as a function of t?

Theorem 6.2. *For all $a, b \in V$ and t, suppose $\boldsymbol{p}_0 = \mathbf{1}_a$. Then*

$$|\boldsymbol{p}_t(b) - \boldsymbol{\pi}(b)| \le \sqrt{\frac{\boldsymbol{d}(b)}{\boldsymbol{d}(a)}}\left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t.$$

That is, $\lambda_2(\boldsymbol{N})$ dictates how fast $\boldsymbol{p}_t$ converges to the stable distribution $\boldsymbol{\pi}$. On $\Omega(1)$ expanders, we have $\lambda_2(\boldsymbol{N}) \ge \Omega(1)$ and the bound decreases exponentially.

Proof. From previous analysis we have

$$\boldsymbol{p}_t = \widetilde{\boldsymbol{W}}^t\boldsymbol{p}_0 = \boldsymbol{D}^{1/2}c_1\boldsymbol{v}_1 + \boldsymbol{D}^{1/2}\sum_i \left(1 - \frac{\lambda_i(\boldsymbol{N})}{2}\right)^t c_i\boldsymbol{v}_i$$

where

$$c_i = \left\langle \boldsymbol{v}_i, \boldsymbol{D}^{-1/2}\boldsymbol{p}_0 \right\rangle$$

So

$$\begin{aligned}\boldsymbol{p}_t(b) = \mathbf{1}_b^\top\boldsymbol{p}_t &= \mathbf{1}_b^\top\boldsymbol{D}^{1/2}c_1\boldsymbol{v}_1 + \mathbf{1}_b^\top\boldsymbol{D}^{1/2}\sum_i \left(1 - \frac{\lambda_i(\boldsymbol{N})}{2}\right)^t c_i\boldsymbol{v}_i \\ &= \boldsymbol{\pi}(b) + \mathbf{1}_b^\top\boldsymbol{D}^{1/2}\sum_i \left(1 - \frac{\lambda_i(\boldsymbol{N})}{2}\right)^t c_i\boldsymbol{v}_i\end{aligned}$$

That is,

$$\boldsymbol{p}_t(b) - \boldsymbol{\pi}(b) = \mathbf{1}_b^\top\boldsymbol{D}^{1/2}\sum_i \left(1 - \frac{\lambda_i(\boldsymbol{N})}{2}\right)^t c_i\boldsymbol{v}_i$$

and we only need to bound the right hand side. To do this, observe that

$$c_i = \left\langle \boldsymbol{v}_i, \boldsymbol{D}^{-1/2}\boldsymbol{p}_0 \right\rangle = \frac{1}{\sqrt{\boldsymbol{d}(a)}}\boldsymbol{v}_i^\top\mathbf{1}_a$$

also

$$\mathbf{1}_b^\top \boldsymbol{D}^{1/2} \boldsymbol{v}_i = \sqrt{\boldsymbol{d}(b)} \mathbf{1}_b^\top \boldsymbol{v}_i$$

We have

$$\mathbf{1}_b^\top \boldsymbol{D}^{1/2} \sum_i \left(1 - \frac{\lambda_i(\boldsymbol{N})}{2}\right)^t c_i \boldsymbol{v}_i = \sqrt{\frac{\boldsymbol{d}(b)}{\boldsymbol{d}(a)}} \sum_i \left(1 - \frac{\lambda_i(\boldsymbol{N})}{2}\right)^t \cdot \mathbf{1}_b^\top \boldsymbol{v}_i \boldsymbol{v}_i^\top \mathbf{1}_a$$

Now, we are ready to bound the second term

$$\begin{aligned}
&\left| \sum_i \left(1 - \frac{\lambda_i(\boldsymbol{N})}{2}\right)^t \cdot \mathbf{1}_b^\top \boldsymbol{v}_i \boldsymbol{v}_i^\top \mathbf{1}_a \right| \\
&\leq \left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t \sum_i |\mathbf{1}_b^\top \boldsymbol{v}_i| \cdot |\boldsymbol{v}_i^\top \mathbf{1}_a| \\
&\leq \left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t \sqrt{\sum_i (\mathbf{1}_b^\top \boldsymbol{v}_i)^2} \cdot \sqrt{\sum_i (\mathbf{1}_a^\top \boldsymbol{v}_i)^2} && \text{by Cauchy-Schwartz} \\
&= \left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t \|\mathbf{1}_b\| \cdot \|\mathbf{1}_a\| && \text{as } \boldsymbol{v}_i \text{ form an orthonormal basis} \\
&= \left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t
\end{aligned}$$

So we can now conclude that

$$|\boldsymbol{p}_t(b) - \boldsymbol{\pi}(b)| \leq \sqrt{\frac{\boldsymbol{d}(b)}{\boldsymbol{d}(a)}} \left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t.$$

□

7 Mixing Time

Definition 7.1. We say that a walk after step t has ϵ**-mixed** if

$$|\boldsymbol{p}_t(b) - \boldsymbol{\pi}(b)| \leq \epsilon \boldsymbol{\pi}(b)$$

for all vertices b.

In particular, if the walk at step t is ϵ-mixed, the **total variation** between $\boldsymbol{p}_t$ and $\boldsymbol{\pi}$ which is defined as$\|\boldsymbol{p}_t - \boldsymbol{\pi}\|_{TV} = \frac{1}{2} \sum_b |\boldsymbol{p}_t(b) - \boldsymbol{\pi}(b)| \leq \frac{1}{2} \sum_b \epsilon \boldsymbol{\pi}(b) = \frac{\epsilon}{2}$ is at most $\epsilon/2$.

Definition 7.2. (Mixing Time) Fix some random walk process. The **mixing time** $\tau_{\text{mix}}(G, \epsilon)$ of the graph G is given by

$$\tau_{\text{mix}}(G, \epsilon) = \max_{\boldsymbol{p}_0} \min_t \{t \mid \|\boldsymbol{p}_t - \boldsymbol{\pi}\|_{TV} \leq \epsilon\}.$$

In English, mixing time is the minimium amount of steps for total variation to be ϵ small starting with any possible initial distribution.

7.1 Mixing time and eigenvalues: $\tau_{\text{mix}}(G, \epsilon) \approx 1/\lambda_2(\boldsymbol{N}_G)$

The theorem below shows that $\tau_{\text{mix}}(G, \epsilon)$ is just another way to think about $\lambda_2(\boldsymbol{N}_G)$!

Theorem 7.3. *We have that*

$$\Omega\left(\frac{\log(\frac{1}{\epsilon})}{\lambda_2(\boldsymbol{N}_G)}\right) \leq \tau_{\text{mix}}(G, \epsilon) \leq O\left(\frac{\log(\frac{\boldsymbol{d}(V)}{\epsilon \boldsymbol{d}_{\min}})}{\lambda_2(\boldsymbol{N}_G)}\right)$$

where $\boldsymbol{d}_{\min} = \min_u \boldsymbol{d}(u)$.

Proof. We omit the proof of lower bound, and only prove the upper bound below.

First we show that the "worst" initial distribution that takes longest to mix the walk is

$$\boldsymbol{p}_0 = \mathbf{1}_{a^*}$$

for some vertex a^*.

Observation 7.4. *Pick any initial probability distribution $\boldsymbol{p}_0$ on the vertices. Then there exists a vertex a^* such that the probability distribution $\mathbf{1}_{a^*}$ mixes slower - that is, $\tau_{mix}(\boldsymbol{p}_0) < \tau_{mix}(\mathbf{1}_{a^*})$.*

Proof. We get $\|p_t - \boldsymbol{\pi}(t)\|_{TV} = \frac{1}{2}\sum_b |\boldsymbol{p}_t(b) - \boldsymbol{\pi}(b)|$. Let $p_t(a, b)$ denote the fraction of mass starting from a landing on b. Then clearly

$$\begin{aligned}
\|p_t - \boldsymbol{\pi}(t)\|_{TV} &= \frac{1}{2}\sum_b |\sum_a p_0(a)\boldsymbol{p}_t(a, b) - \boldsymbol{\pi}(b)| \\
&= \frac{1}{2}\sum_b |\sum_a (p_0(a)\boldsymbol{p}_t(a, b) - p_0(a)\boldsymbol{\pi}(b))| \\
&\leq \frac{1}{2}\sum_b \sum_a |p_0(a)\boldsymbol{p}_t(a, b) - p_0(a)\boldsymbol{\pi}(b)| \text{(Triangle inequality)} \\
&= \frac{1}{2}\sum_b \sum_a p_0(a)|\boldsymbol{p}_t(a, b) - \boldsymbol{\pi}(b)| \\
&= \frac{1}{2}\sum_a p_0(a) \sum_b |\boldsymbol{p}_t(a, b) - \boldsymbol{\pi}(b)| \\
&\leq \frac{1}{2}\max_a \sum_b |\boldsymbol{p}_t(a, b) - \boldsymbol{\pi}(b)|.
\end{aligned}$$

Suppose $a = a^*$ maximizes $\sum_b |\boldsymbol{p}_t(a, b) - \boldsymbol{\pi}(b)|$. Then it is clear that the random walk starting at a^* takes more time to mix than the random walk starting with the initial distribution p_0, and hence we are done.

□

We are now ready to prove the result. Fix a vertex a^* from which we perform the lazy random walk. That is, fix $p_0 = \mathbf{1}_{a^*}$ We have

$$|\boldsymbol{p}_t(b) - \boldsymbol{\pi}(b)| \leq \sqrt{\frac{\boldsymbol{d}(b)}{\boldsymbol{d}(a^*)}} \left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t.$$

So we ask that is the smallest t where

$$\begin{aligned}
\sqrt{\frac{\boldsymbol{d}(b)}{\boldsymbol{d}(a^*)}} \left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t &\leq \frac{\epsilon \boldsymbol{d}(b)}{\boldsymbol{d}(V)} && \iff \\
\left(1 - \frac{\lambda_2(\boldsymbol{N})}{2}\right)^t &\leq \epsilon \frac{\sqrt{\boldsymbol{d}(a^*)\boldsymbol{d}(b)}}{\boldsymbol{d}(V)} && \iff \\
\exp(-\Theta(\frac{t\lambda_2(\boldsymbol{N})}{2})) &\leq \epsilon \frac{\sqrt{\boldsymbol{d}(a^*)\boldsymbol{d}(b)}}{\boldsymbol{d}(V)} && \iff \text{using } 1 - x \approx \exp(-x) \\
-\Theta(\frac{t\lambda_2(\boldsymbol{N})}{2}) &\leq \ln(\epsilon \frac{\sqrt{\boldsymbol{d}(a^*)\boldsymbol{d}(b)}}{\boldsymbol{d}(V)}) && \iff \\
t &\geq \Theta(\frac{2}{\lambda_2(\boldsymbol{N})}) \ln(\frac{\boldsymbol{d}(V)}{\epsilon\sqrt{\boldsymbol{d}(a^*)\boldsymbol{d}(b)}}) && \iff
\end{aligned}$$

from which we get

$$\tau_{\text{mix}}(G, \epsilon) \leq O\left(\frac{\log(\frac{\boldsymbol{d}(V)}{\epsilon \boldsymbol{d}_{\min}})}{\lambda_2(\boldsymbol{N}_G)}\right).$$

□

7.2 Examples

- Let's try to answer the question about cycles from the beginning of the lecture.
  - Why uniform distribution? - we saw that the probability mass on a vertex is proportional to its degree, and hence we have the uniform distribution as the stationary distribution.
  - How fast does the distribution converge? For a cycle G, $\lambda_2(N_G) = \frac{1}{n^2}$. Hence the mixing time is $O(n^2 \log n)$.
- When it is not clear what is $\tau_{\text{mix}}(G, \epsilon)$ or $\lambda_2(\boldsymbol{N}_G)$, knowing one implies $O(\log n)$-approximation of the other.
  - When G is a $\tilde{\Omega}(1)$-expander, we know $\lambda_2(\boldsymbol{N}_G) = \tilde{\Omega}(1)$. So $\tau_{\text{mix}} = \tilde{O}(1)$.
  - When G is a path, we know $\tau_{\text{mix}} \approx n^2$ (why?), so $\lambda_2(\boldsymbol{N}_G) \approx 1/n^2$.
    We can see this from the conductance of path. Another good reason is that the expected transition distance after n steps of a random walk is of order $\sqrt{n}$,[2] whence it takes roughly n^2 steps to move n units of distance away from the starting point.

[2] https://mathworld.wolfram.com/RandomWalk1-Dimensional.html

- When G is a dumbell, $\tau_{\text{mix}} \approx n^2$ (why?), so $\lambda_2(\boldsymbol{N}_G) \approx 1/n^2$
  The idea is that there is roughly $1/n$ of probability for a the current point to move from one clique to an endpoint of the bridge, and it takes another $1/n$ of probability to move from one endpoint of the bridge to the other (so that the point will enter the other clique.)
- When G is a Bolas graph, $\tau_{\text{mix}} \approx n^3$ (why?), so $\lambda_2(\boldsymbol{N}_G) \approx 1/n^3$.
  Similarly, it takes roughly $1/n$ of probability to move from one clique to an endpoint of the bridge, and another $1/n^2$ of probability to move to the other endpoint by previous result for path.

- Actually, the Bolas graph is the *unweighted* graph which is hardest to mix.

Exercise 7.5 (Worst mixing time in unweighted graphs)**.** In the homework, you will show that for any unweighted graph G, $\lambda_2(\boldsymbol{N}_G) \geq \Omega(1/n^3)$. So $\tau_{\text{mix}}(G, 1/\text{poly}(n)) = O(n^3 \log n)$.

8 Applications of Random Walks

8.1 Short path in $\tilde{O}(\sqrt{n})$ time on Expanders

- **Question:** Given a ϕ-expander (undirected, unweighted) $G = (V, E)$ and $s, t \in V$, can we find a "short" path from s to t in sublinear time?

- Algorithm SHORTPATH
  - Perform a lazy random walk from s for $\tau_{\text{mix}}(G, \frac{1}{n^2}) = O(\frac{\log n}{\phi^2})$ steps.
  - Repeat for $O(\sqrt{n} \log n)$ times.
  - Let S be the set of vertices where the walks end.
  - We now do the same walk from t. Let T be the similar set of vertices.
  - If a there is a vertex $v \in S \cap T$, return a path from s to t as the union of paths between s and v and v and t.

  Note that if $S \cap T \neq \emptyset$, then we obtain an (s, t) path of length $O(\tau_{\text{mix}}(G, \frac{1}{n^2}))$. The running time of the algorithm is clearly $\tilde{O}(\sqrt{n} \cdot \tau_{\text{mix}}(G, \frac{1}{n^2}))$.

  Observation 8.1. *$S \cap T \neq \emptyset$ whp.*

  Proof. (Sketch) Essentially birthday paradox. Since the walks have mixed sufficiently, S and T are essentially uniformly random vertices (since G is connected). Thus the probability of their intersection being non-empty is the probability that given n bins, if we independently throw $O(\sqrt{n} \log n)$ balls uniformly at random, two of the balls land in the same bin. To analyze this formally, note that the probability that no two of the balls collide is at most $(1 - \frac{1}{n})(1 - \frac{2}{n}) \ldots (1 - \frac{\sqrt{n} \log n}{n})$ which is at most $e^{-\sum_{i=1}^{\sqrt{n} \log n} \frac{i}{n}} \sim e^{-\log^2 n} \ll \frac{1}{n^c}$ for any constant c. □

  Corollary 8.2. *If G is an $\Omega(1)$-expander, then algorithm* SHORTPATH *returns a $O(\log n)$-length (s, t)-path in $\tilde{O}(\sqrt{n})$ time.*

8.2 Connectivity in $O(\log n)$ bits of space

- **Question:** Given a (undirected and unweighted) graph $G = (V, E)$ and $s, t \in V$, is there a $s - t$ path?
- We shall assume that the adjacency lists of G are *read-only* and we can use only $O(\log n)$ bits of *working memory*.
- **The algorithm**
  - Start from s.
  - Perform lazy random walks for $O(n^5 \log^2 n)$ steps.
  - Whenever we visit t, return YES.
  - If we never visit t, return NO.
- If s and t are not connected, clearly the algorithm return NO.

Observation 8.3. *If s and t are in the same connected component, then the random walk will meet t whp.*

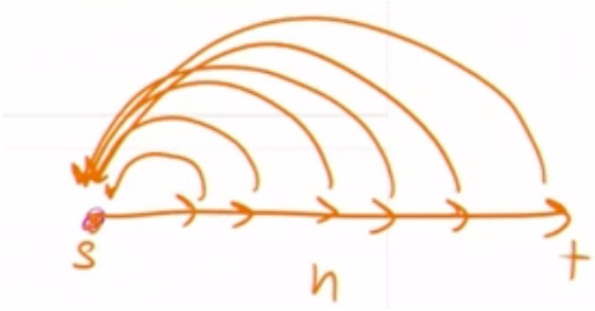
Proof. Let C be the connected component containing s and t.

We have $\tau_{\text{mix}}(C, \frac{1}{n^2}) \leq O(n^3 \log n)$ (as C is an unweighted graph - by exercise 7.5)

Therefore after every $O(n^3 \log n)$ steps, we visit t w.p. $\frac{d(t)}{d(C)} \geq \frac{1}{n^2}$. It follows that after $O(n^5 \log^2 n)$ steps, we never visit t w.p. at most $(1 - \frac{1}{n^2})^{O(n^2 \log n)} \leq 1/\text{poly}(n)$. □

Question 8.4. *Is it possible to improve the time to $\tilde{O}(n^2)$ or even $\tilde{O}(n)$ while using $O(\text{polylog}(n))$ space?*

For directed graph, the above algorithm won't work. Indeed, consider the graph in the below figure-in expectation it takes 2^n steps for s to reach t. An interesting open problem is that: is there an (even randomized) s-t connectivity algorithm use $O(n^{0.99})$ space runs in polylog(n) time? The current best known deterministic algorithm uses $O(n/2^{\sqrt{\log n}})$ space.



8.3 Sampling a Spanning Tree

The problem of sampling (almost uniformly) a spanning tree has been studied extensively in the literature. However, the "right" algorithm has been recently shown last year [3]. We will

[3] https://arxiv.org/abs/2004.07220

only discuss the high level ideas involved in the sampling process and skip the technical details.

- **Question:** Given a connected graph G, sample (almost)-uniformly at random a spanning tree T of G.
- **Key Idea:**
  - Given a spanning tree T, consider the following "flip" operation that
    - Choose one of the $m - (n-1)$ non-tree edges e
    - Look at the unique cycle C in $T \cup e$
    - Choose e' in C and remove e'.
    - We obtain another spanning tree $T' = T \cup e \setminus e'$.

Figure 2: The flip operation

- Consider the super graph $\mathcal{G}$ where
  - each node is a spanning tree
  - We add an edge between T and T' iff T can be flipped to T'.
- **Key idea (roughly speaking)**: although $\mathcal{G}$ is exponentially big, $\mathcal{G}$ is an expander. So it takes only $O(\log |V(\mathcal{G})|)$ random walk steps for get to sample a uniform spanning tree.

- Algorithm:
  - Start with any spanning tree.
  - Randomly flip for $O(m \log m)$ steps.
  - Return the tree.
- We can simulate the tree flipping using the link-cut tree data structure in $O(\log n)$ time. So the overall running time is $O(m \log^2 n)$.

9 Conclusion

Exercise 9.1. Suppose that the graph G is d-regular. Recall from the previous lecture that the power method for computing $\lambda_2(\boldsymbol{N})$ just computes $\widetilde{\boldsymbol{W}}^{O(\log(n)/\epsilon)}\boldsymbol{x}$ where $\boldsymbol{x}$ is a some random vector.

9.1 Alternate notions of expansion

Recall the various characterizations of expanders we have seen:

1. Robustness view: Robust towards edge deletions
2. Cut view: No sparse cuts, high conductance
3. Flow view: Can embed any degree restricted demand with small congestion
4. Spectral view: Small second eigenvalue of normalized laplacian
5. Probabilistic view: Random walks mix fast

Most of the theory we have seen is about conductance (i.e. edge expansion) in undirected graphs. But what about other notions?

Question 9.2. *Survey the equivalence we have seen so far for vertex expansion and/or directed graphs.*

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 14.0: The Cut-Matching Game: Deterministic Cut Player

October 12 & 14, 2021

Instructor: Thatchaphol Saranurak

Scribe: Yi Tang

The cut-matching game provides an *implicit* construction of expanders, where it "forces an adversary" to construct an expander.

1 The Game

The cut-matching game involves two players: the cut player and the matching player. The game starts with an empty graph G_0 with vertex set V; $|V| = n$. For each round $i = 1, 2, \ldots$:

- The cut player chooses a cut (A_i, B_i) (assuming $|A_i| \leq |V|/2$).
- The matching player gives a maximal matching M_i between A_i and B_i, with size $|A_i|$.
- Set $G_i = G_{i-1} \cup M_i$ (retaining parallel edges).

The game ends when G_i is an expander (say an $\Omega(1/\text{polylog}(n))$-expander). For simplicity, denote the total number of rounds by I and $G = G_I$.

In the cut-matching game, the objective of the cut player is to minimize the total number of rounds I by choosing the cuts cleverly, while the matching player behaves adversarially, trying to delay the process as much as possible.

In the following sections we will show strategies for the cut player so that no matter how badly the matching player behaves, the game yields an expander in $I = \text{polylog}(n)$ rounds.

2 Heuristics: Balanced Sparse Cuts

First of all we observe the following heuristics for the cut player:

- The chosen cut should be balanced, so that M_i is large and contains plenty of edges.
- The chosen cut should be sparse, or should contain a sparse cut inside, so that M_i makes progress in fixing sparse cuts, which is necessary for increasing the global conductance.

It turns out that these two heuristics are enough, while it is not trivial at all to formalize.

3 High-level Goal for Analysis: Random Walks that Embeds a Clique

In this section we show a sufficient condition for the game to end and yield an expander.

Consider the following random walk:

- Start from some vertex v_0.
- For each step/round $i \in [I]$:
  - If $v_{i-1} \notin M_i$ then stay put at $v_i = v_{i-1}$.
  - Otherwise if $(v_{i-1}, u) \in M_i$ for some vertex u then take one (lazy) random walk step according to M_i, i.e.,
    - with probability $1/2$, stay put at $v_i = v_{i-1}$;
    - with probability $1/2$, move to $v_i = u$.

Based on the random walk, define $p_i(u, v)$ to be the probability of reaching v at step i in the random walk starting from u.

Let $\mathbf{P}_i \in \mathbb{R}^{V \times V}$ be the matrix with value $p_i(u, v)$ at entry (u, v). During the random walk, $\mathbf{P}_i$ evolves as follows:

- $\mathbf{P}_0 = \mathbf{I}$.
- For $v \notin M_i$, the column satisfies $\mathbf{P}_i(\cdot, v) = \mathbf{P}_{i-1}(\cdot, v)$.
- For $(u, v) \in M_i$, the columns satisfy $\mathbf{P}_i(\cdot, u) = \mathbf{P}_i(\cdot, v) = (\mathbf{P}_{i-1}(\cdot, u) + \mathbf{P}_{i-1}(\cdot, v))/2$.

I.e., in each round $i \in I$, the columns of $\mathbf{P}_i$ are obtained by averaging the columns of $\mathbf{P}_{i-1}$ according to the matchings in M_i.

By induction, it is easy to see that the columns of $\mathbf{P}_i$ always sum to 1, i.e., $\sum_u \mathbf{P}_i(u, v) = 1$ for all v and i (as in each round the columns either remain the same or get averaged). (Cf. the rows always sum to 1 by definition.)

Consider all together the random walks starting from all vertices, and regard the random walk as distributing masses, with initial mass 1 at each vertex at the beginning. Then the total mass going through edge $(u, v) \in M_i \subseteq G$ during the random walks is exactly

$$\underbrace{\frac{1}{2}\sum_a \mathbf{P}_{i-1}(a, u)}_{\text{from } u \text{ to } v} + \underbrace{\frac{1}{2}\sum_b \mathbf{P}_{i-1}(b, v)}_{\text{from } v \text{ to } u} = \frac{1}{2} + \frac{1}{2} = 1 \,,$$

since the edge $(u, v) \in M_i$ is involved only during round i in the random walk (recall that G retains parallel edges).

Let K be the 1-product graph (i.e., clique with weight $1/n$ on every edge). The following lemma tells that a sufficient condition for the game to end and yield a graph that embeds a clique is to have p_i be well-mixed.

Lemma 3.1. *If $p_I(u, v) \geq 1/(2n)$ for all u, v, then $K/2 \preceq^{flow} G$ (and thus $K/2 \preceq^{cut} G$).*

Proof. Consider the flow that "follows the random walk," i.e., routes commodities in the same way as the random walk distributes masses (see Figure 1). Indeed, it suffices to construct a flow that routes at least $1/2n$ on each edge of K in G with congestion at most 1 since we could scale them with constant factors to make them satisfies the demands exactly. We construct the flow

inductively by considering adding one matching M_t at a time and route $(p_t(i,j)+p_t(j,i))$ amount of flow between i and j in G_t, where $(p_I(i,j)+p_I(j,i)) \geq 1/n > \kappa_K(\{i,j\})$ by premise. For the base case when $t=0$ where we have no edges, $\forall (a,b) \in V \times V$, when $a \neq b$, we set $f^0_{a,b} := \mathbf{0}$ and $f^0_{a,a} : P^0_{a,a} \mapsto p_0(a,a) = 1$, where $P^0_{a,a} =< a >$. When a new edge $\{u,v\} \in M_{t+1}$ is added to G_t, $\forall a \in V$, the amount of flow that we need to route between a and u increases by $\left(\frac{1}{2}p_t(a,v) - \frac{1}{2}p_t(a,u)\right)$. In order to maintain the congestion yet satisfies the new demand, we let

$$f^{(t+1)}_{a,u} : \begin{cases} P^{(t)}_{a,u} \mapsto \frac{1}{2} f^{(t)}_{a,u}(P_{a,u}), & \text{for } P^{(t)}_{a,u} \text{ is a path from } a \text{ to } u \text{ in } G_t \\ P^{(t)}_{a,v}; < v,u > \mapsto \frac{1}{2} f^{(t)}_{a,v}(P_{a,v}), & \text{for } P^{(t)}_{a,v}; < v,u > \text{ is the concatenation of } P^{(t)}_{a,v} \text{ and } < v,u > \\ P^{(t+1)}_{a,u} \mapsto 0, & \text{o.w.} \end{cases}$$

as described in Figure 2. A Symmetric analysis and construction applies to $f^{(t+1)}_{a,v}$. In this way, the congestion on edges in G_t won't change and by the analysis above the lemma and inducting on t we know the total amount of flow going through $\{u,v\}$ is exactly 1, which means the congestion of the flow is also exactly 1. Hence $K/2 \preceq^{\text{flow}} G$. □

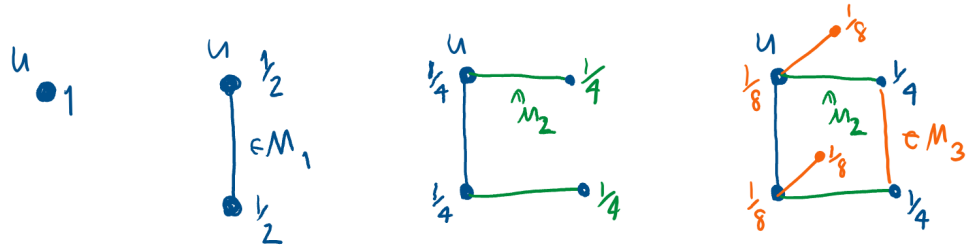


Figure 1: The distribution of masses from u to each vertex during the lazy random walk based on M_t can also be regarded as the amount of flow that we need to route from u to each vertex in the flow-embedding that we constructed at each inductive step. Namely, after M_t is added, we need that $\forall x \in V, ||f_{u,x}|| = p_t(u,x)$.

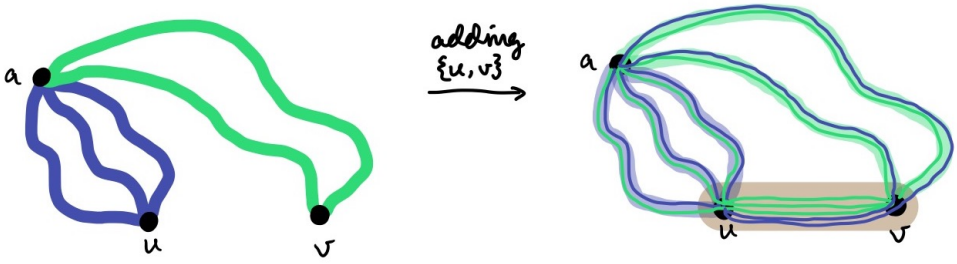


Figure 2: The figure shows how flows from a to u and a to v changes when $\{u,v\}$ was added. The amount of flow on $\{u,v\} \in M_{t+1}$ when it was added is $\frac{1}{2}\sum_{a\in V}(f^{(t)}_{a,v}(P_{a,v}) + f^{(t)}_{a,u}(P_{a,u})) = \frac{1}{2}\sum_{a\in V}(p_t(a,v)+p_t(a,u)) = 1$ by inducting on t, and the amount of flow on edges in G_t remains the same.

On the other hand, since $G = G_I$ has maximum degree I, we know $G \preceq^{\text{cut}} 2I \cdot K$. Then $K/2 \preceq^{\text{cut}} G \preceq^{\text{cut}} 2I \cdot K$, so K and G are $O(I)-$cut-equivalent. More precisely, the flow-embedding that embeds $K/2$ into G act as a certificate of G being an $\Omega(1/maxdeg(G))$-expander. Hence if $I = \text{polylog}(n)$ then $maxdeg(G) = \text{polylog}(n)$ and we obtain $\Omega(1/\text{polylog}(n))$-expanders as desired.

We will show two algorithms/strategies for the cut player: a randomized one and a deterministic one. The randomized one uses the sufficient condition established in this section, while the deterministic one uses something else yet similar. Nevertheless the deterministic one is actually more intuitive and will be presented first.

4 Notations

In this section we introduce some notations useful in the analysis.

Definition 4.1. A cut $S \subseteq V$ is *β-balanced* if $\min\{|S|, |V \setminus S|\} \geq \beta|V|$.

Definition 4.2. The *expansion* of cut $S \subseteq V$ in graph G is defined by

$$\Psi_G(S) = \frac{|E(S, V \setminus S)|}{\min\{|S|, |V \setminus S|\}} .$$

The expansion of the graph G is defined by $\Psi(G) = \min_{\emptyset \neq S \subsetneq V} \Psi_G(S)$.

Recall that this notion of expansion is the same as d-expansion with $d(v) = 1$ for all v.

Note that since $|S| \leq \mathrm{vol}_G(S) \leq d_{\max}(G) \cdot |S|$ (assuming unweighted graphs), we have $\Phi(G) \leq \Psi(G) \leq d_{\max}(G) \cdot \Phi(G)$, which means the expansion $\Psi(G)$ is a good proxy to the conductance $\Phi(G)$, especially when $d_{\max}(G) = \mathrm{polylog}(n)$.

5 Deterministic Exponential-Time Algorithm

Theorem 5.1 ([KKOV07])**.** *There is a deterministic exponential-time algorithm for the cut player so that after $I = O(\log n)$ rounds, $\Psi(G) = \Omega(1)$ (and thus $\Phi(G) = \Omega(1/\log n)$).*

Proof. The deterministic algorithm works as follows:

- If there exists a $(1/4)$-balanced sparse cut (A_i, B_i) with $\Psi_{G_{i-1}}(A_i) \leq 1/100$ then choose such a cut (A_i, B_i).
- Otherwise there must be a cut (A_i, B_i) with $|A_i| \leq n/4$ and $\Psi(G_{i-1}[B_i]) > 1/400$; then the cut player choose (A_i, B_i), and assert that it is the last cut to choose and leads to $\Psi(G_i) > 1/1200$.

Note that the $(1/4)$-balanced cut (A_i, B_i) with $\Psi_{G_{i-1}}(A_i) \leq 1/100$ really follows the heuristics of choosing a *balanced sparse cut*. The correctness and efficiency of the algorithm will be analyzed throughout this section. □

5.1 Entropy Potential

For vertex u and round i, define the *entropy potential* by

$$\Pi_i(u) = \mathrm{H}[p_i(u, \cdot)] = -\sum_v p_i(u, v) \log p_i(u, v) .$$

Also define the total entropy potential by $\Pi_i = \sum_u \Pi_i(u)$.

Note that the entropy of a probability distribution over n elements is bounded between 0 and $\log n$, and is maximized by (and only by) the uniform distribution, thus measuring how well-spread/well-mixed the distribution is. This gives $\Pi_i(u) \leq \log n$ and $\Pi_i \leq n \log n$.

5.2 All Rounds except the Last

The following lemma shows that in the algorithm, the entropy potential Π_i increase by $\Omega(n)$ in each round, thus concluding that there are at most $I = O(\log n)$ rounds in the algorithm.

Lemma 5.2. *If there exists an $\Omega(1)$-balanced cut (A_i, B_i) with $\Psi_{G_{i-1}}(A_i) \le 1 - \delta$ where $\delta = \Omega(1)$, then $\Pi_i - \Pi_{i-1} = \Omega(n)$.*

(Recall that the algorithm uses (1/4)-balanced cut with $\delta = 99/100$.)

Proof. Denote $p_i(S, T) = \sum_{u \in S, v \in T} p_i(u, v)$, which represents the total mass going from S to T in the first i rounds. Consider $p_{i-1}(A_i, B_i)$, the total mass going from A_i to B_i before round i. By the same analysis as in Section 3, this total mass is at most $\sum_{j<i} |M_j|/2 = |E_{G_{i-1}}(A_i, B_i)|/2$ (and the total mass going from B_i to A_i corresponds to the other half). Then since $\Psi_{G_{i-1}}(A_i) \le 1 - \delta$, we know

$$p_{i-1}(A_i, B_i) \le |E_{G_{i-1}}(A_i, B_i)|/2 \le (1 - \delta)|A_i|/2 \,.$$

This implies that there are at least $\Omega(\delta|A_i|) = \Omega(n)$ vertices $a \in A_i$ where $p_{i-1}(a, B_i) \le 1/2 - \delta/3$. We say such vertices to be *confined* (see Figure 3).

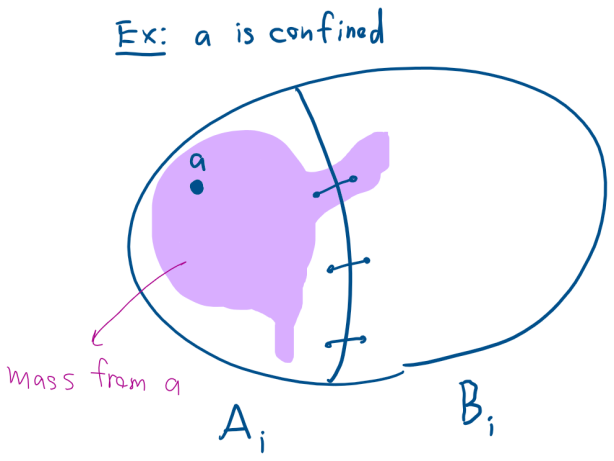


Figure 3: Illustration of a confined vertex a for cut (A_i, B_i).

For a confined vertex $a \in A_i$, intuitively, little mass at a goes into B_i and much of the mass remains in A_i, so there must be a large fraction of $(u, v) \in M_i$ such that $p_{i-1}(a, u) \gg p_{i-1}(a, v)$. Formally, we say an edge $(u, v) \in M_i$ to be *a-spreading* if $p_{i-1}(a, u) \ge (1 + \delta) \cdot p_{i-1}(a, v)$. For an a-spreading edge (u, v), since $p_{i-1}(a, u)$ and $p_{i-1}(a, v)$ differs notably, we expect it to contribute large entropy increase after averaging the probabilities according to M_i.

Partition A_i into $S_a = \{u \in A_i : (u, v) \in M_i \text{ is } a\text{-spreading}\}$ and $N_a = \{u \in A_i : (u, v) \in M_i \text{ is } a\text{-non-spreading}\}$, and let $N_a^* = \{v \in B_i : (u, v) \in M_i, u \in N_a\}$. Note that $\sum_{u \in A_i} p_{i-1}(a, u) = p_{i-1}(a, A_i) = 1 - p_{i-1}(a, B_i)$, while for the a-non-spreading edges,

$$\sum_{u \in N_a} p_{i-1}(a, u) \le (1 + \delta) \sum_{v \in N_a^*} p_{i-1}(a, v) \le (1 + \delta) \cdot p_{i-1}(a, B_i) \,.$$

Therefore, for the a-spreading edges,

$$\sum_{u \in S_a} p_{i-1}(a, u) \ge 1 - (2 + \delta) \cdot p_{i-1}(a, B_i) \ge \delta/6 + \delta^2/3 = \Omega(1) \,.$$

Observe that for $1 \geq p \geq (1 + \Theta(1)) \cdot q \geq 0$,

$$-(p+q)\log\frac{p+q}{2} + p\log p + q\log q = \Omega(p)\,.$$

(This follows by noting that the expression on the left hand side is decreasing in q.) Hence adding an a-spreading edge $(u, v) \in M_i$ then averaging the probabilities according to the random walk process, $\Pi_{i-1}(a)$ increases by $\Omega(p_{i-1}(a, u))$. Summing over all a-spreading edges and all confined vertices a, the total entropy increases at least by $\Omega(n) \cdot \sum_{u \in S_a} \Omega(p_{i-1}(a, u)) = \Omega(n)$, as desired. □

5.3 Last Round

The following lemmas show that the assertions made in the algorithm are always valid.

Lemma 5.3. *If there is no* $(1/4)$*-balanced cut* B *with* $\Psi_{G_{i-1}}(B) \leq \psi$*, then there exists cut* (A_i, B_i) *such that* $|A_i| \leq n/4$ *and* $\Psi(G_{i-1}[B_i]) > \psi/3$*.*

(Recall that the algorithm uses $\psi = 1/100$.)

Proof. Consider the following procedure:

- Start with $H \leftarrow G_{i-1}$, $A \leftarrow \emptyset$.
- While there exists a sparse cut S with $\Psi_H(S) \leq \psi/3$ and $|S| \leq |V_H|/2$, update $H \leftarrow H \setminus S$, $A \leftarrow A \cup S$.
- Return H, A.

Obviously the procedure returns H with $\Psi(H) > \psi/3$. Then it suffices to show that the procedure also returns A with $|A| \leq n/4$, so that setting $A_i = A$ (and hence $G_{i-1}[B_i] = H$) satisfies all the desired properties.

Suppose towards contradiction that $|A| > n/4$. Since A is initialized to be $\emptyset$, it means there is a certain round where $|A| \leq n/4$ while $|A \cup S| > n/4$. Note that $|S| \leq |V_H|/2 \leq n/2$. Hence $A \cup S$ is a $(1/4)$-balanced cut. Moreover, $A \cup S$ is the disjoint union of a sequence of cuts $\{S_j\}_{j \in [J]}$ with $\Psi_H(S_j) \leq \psi/3$, so the cut size of $A \cup S$ in G_{i-1} is at most $(\psi/3) \cdot |A \cup S| \leq (\psi/3) \cdot (3n/4) = \psi n/4$ (as every edge cut by $A \cup S$ in G_{i-1} must have been cut by some S_j in the corresponding iterate of H). This implies $\Psi_{G_{i-1}}(A \cup S) \leq (\psi n/4)/(n/4) = \psi$, which contradicts the given condition. □

Lemma 5.4. *For any cut* (A_i, B_i) *(assuming* $|A_i| \leq n/2$*),* $\Psi(G_i) > \min\{\Psi(G_{i-1}[B_i])/2, 1\}$*.*

(Recall that the algorithm uses $|A_i| \leq n/4$ and $\Psi(G_{i-1}[B_i]) > 1/400$.)

Proof. Intuitively, for any cut S in G_i, if a large fraction of S is in A_i, then the expansion of S is large as the edges in M_i goes from $S \cap A_i$ to B_i; while if a large fraction of S is in B_i, then the expansion of S is large as well since $G_{i-1}[B_i]$ has large expansion.

Let $\psi = \Psi(G_{i-1}[B_i])$. Consider an arbitrary cut S in G_i, with $S = S_A \cup S_B$ where $S_A = S \cap A_i$ and $S_B = S \cap B_i$. Suppose without loss of generality that $|S_B| \leq |B_i|/2$, and let S_B^* be the matching of S_B under M_i, i.e., $S_B^* = \{u \in A_i : (u, v) \in M_i, v \in S_B\}$ (clearly $|S_B^*| \leq |S_B|$ as $|A_i| \leq n/2$). Then by

direct calculations without case analysis,

$$\begin{aligned}\Psi(G_i) &\geq \frac{|\partial_{G_{i-1}[B_i]} S_B| + |S_A \oplus S_B^*|}{|S_B| + |S_A|} \\ &\geq \frac{\psi|S_B| + |S_A \setminus S_B^*|}{|S_B| + |S_A|} \\ &= \frac{\psi|S_B| + |S_A \setminus S_B^*|}{|S_B| + |S_A \cap S_B^*| + |S_A \setminus S_B^*|} \\ &\geq \frac{\psi|S_B| + |S_A \setminus S_B^*|}{2|S_B| + |S_A \setminus S_B^*|} \geq \min\{\psi/2, 1\} .\end{aligned}$$

(Here $\oplus$ denotes symmetric difference of sets.) □

5.4 Why Exponential Time?

The algorithm costs exponential time because of finding sparse cuts; specifically, the algorithm involves the task of finding either a (1/4)-balanced cut S with $\Psi_{G_{i-1}}(S) \leq 1/100$ or a subgraph $G_{i-1}[B_i]$ of size $|B_i| \geq 3n/4$ with $\Psi(G_{i-1}[B_i]) > 1/400$, and the latter implies, say, all (3/8)-balanced cuts in G_{i-1} have expansions at least 1/3200. So the algorithm decides the expansion of a graph (with regard to only the (3/8)-balanced cuts though) up to a constant factor. However, it is NP-hard to decide whether or not $\Psi(G) \leq \psi$ given (G, ψ) [MS90] (and unique-game-hard to decide up to any constant factor [CKK+06, KV15]).

This complexity issue can be resolved by using polylog(n)-approximate sparse cuts, which can be computed efficiently. In particular, by Lemma 5.4, finding $\Psi(G_{i-1}[B_i]) \geq 1/\text{polylog}(n)$ instead of $\Psi(G_{i-1}[B_i]) \geq \Omega(1)$ is still good enough for obtaining $\Psi(G) \geq 1/\text{polylog}(n)$.

References

[CKK+06] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complexity*, 15(2):94–114, 2006. Preliminary version in CCC 2005.

[KKOV07] Rohit Khandekar, Subhash A. Khot, Lorenzo Orecchia, and Nisheeth K. Vishnoi. On a cut-matching game for the sparsest cut problem. Technical Report UCB/EECS-2007-177, EECS Department, University of California, Berkeley, Dec 2007.

[KV15] Subhash A. Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrability gap for cut problems and embeddability of negative-type metrics into ℓ_1. *J. ACM*, 62(1):Art. 8, 39, 2015. Preliminary version in FOCS 2005.

[MS90] David W. Matula and Farhad Shahrokhi. Sparsest cuts and bottlenecks in graphs. volume 27, pages 113–123. 1990. Computational algorithms, operations research and computer science (Burnaby, BC, 1987).

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 15.0: The Cut-Matching Game: Randomized Cut Player

October 14 & 21, 2021

Instructor: Thatchaphol Saranurak

Scribe: Yi Tang

Recall: the Cut-Matching Game

In the cut-matching game, there are two players: the cut player and the matching player. The game starts with an empty graph G_0 with vertices V ($|V| = n$). In each round i, the cut player chooses a cut (A_i, B_i) (assuming $|A_i| \leq |V|/2$), the matching player then chooses a maximal matching M_i between A_i and B_i (with size $|A_i|$), and the game sets $G_i = G_{i-1} \cup M_i$. The game ends when G_i is an expander (say an $\Omega(1/\text{polylog}(n))$-expander). Denote the total number of rounds by I and $G = G_I$.

The objective of the cut player is to minimize I and get an expander as fast as possible by cleverly choosing the cuts, while the matching player does the opposite and tries to delay the process as much as possible. We consider strategies/algorithms for the cut player under completely adversarial matching player.

The (lazy) random walk based on M_i is useful in analyzing the game strategies. In each step/round i of the random walk, from vertex u, if $u \notin M_i$ then the walk stays put; otherwise if $(u, v) \in M_i$ then the walk stays put with probability $1/2$ and goes to v with probability $1/2$.

Let $p_i(u, v)$ be the probability of reaching v at step i in the random walk starting from u, and $\mathbf{P}_i \in \mathbb{R}^{V \times V}$ be the matrix with value $p_i(u, v)$ at entry (u, v). Also let $\mathbf{M}_i$ be the lazy random walk matrix for matching M_i (see Figure 1 for the structure of $\mathbf{M}_i$). Then $\mathbf{P}_0 = \mathbf{I}$, and $\mathbf{P}_i = \mathbf{P}_{i-1}\mathbf{M}_i = \mathbf{M}_1 \cdots \mathbf{M}_i$. Note that $\mathbf{M}_i$ is symmetric and sparse and has only $\Theta(n)$ non-zero entries.

6 Randomized Near-Linear-Time Algorithm

Theorem 6.1 ([KRV09]). *There is a randomized $O(n \log^4 n)$-time algorithm for the cut player so that after $I = O(\log^2 n)$ rounds, $\Psi(G) = \Omega(1)$ (and thus $\Phi(G) = \Omega(1/\log^2 n)$).*

Proof. The randomized algorithm repeat the following subroutine I times: on round i,

- Sample a random (unit) vector $\mathbf{r}_i \in \mathbb{R}^V$ satisfying $\mathbf{r}_i \perp \mathbf{1}$.
- Compute $\mathbf{x} = \mathbf{P}_i^\top \mathbf{r}_i = \mathbf{M}_i \cdots \mathbf{M}_1 \mathbf{r}_i \in \mathbb{R}^V$.
- Return the *median cut* A_i that contains the $n/2$ vertices whose corresponding values x_u in $\mathbf{x}$ are the smallest.

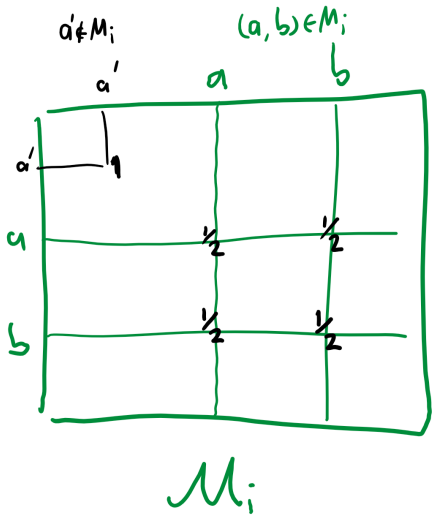


Figure 1: Structure of the (lazy) random walk matrix $\mathbf{M}_i$ for matching M_i.

The correctness and efficiency of the algorithm will be analyzed throughout this section. □

The randomized algorithm still closely follows the heuristics of choosing a *balanced sparse cut*. Firstly, the median cut is always a balanced cut. Moreover, if there exists a sparse cut S, then similar to the situation in the power method (though here the random walk mixes by merely M_i instead of the entire graph in round i), the values x_u in $\mathbf{x}$ for $u \in S$ and $u \in V \setminus S$ would be well-separated with high probability, and hence the median cut would contain the sparse cut S.

6.1 ℓ_2 Potential

For vertex u and round i, define the ℓ_2 *potential* by

$$\Pi_i(u) = \|\mathbf{P}_i(\cdot, u) - \mathbf{1}/n\|_2^2 .$$

Also define the total ℓ_2 potential by $\Pi_i = \sum_u \Pi_i(u)$.

Note that $\Pi_i(u) \geq 0$, and the equality holds for and only for uniform $\mathbf{P}_i(\cdot, u)$, so the potential measures how well-spread the distribution $\mathbf{P}_i(\cdot, u)$ is.

By calculation, $\Pi_0(u) = (1 - 1/n)^2 + (n-1)/n^2 = (n-1)/n$, and $\Pi_0 = n - 1$.

6.2 Small Potential Implies Done

Proposition 6.2. *If* $\Pi_I \leq 1/(4n^2)$, *then* $\Psi(G) \geq 1/4$.

Proof. Note that Π_I is a sum of squares, so if $\Pi_I \leq 1/(4n^2)$, then $p_I(u, v) \geq 1/n - 1/(2n) = 1/(2n)$ for all u, v. Then by Lemma 3.1 in the previous lecture, $K/2 \preceq^{\text{flow}} G$ and thus $K/2 \preceq^{\text{cut}} G$. This implies that for any cut S,

$$|\partial_G S| \geq \frac{|S| \cdot |V \setminus S|}{2n} \geq \frac{1}{4} \min\{|S|, |V \setminus S|\} .$$

I.e., $\Psi(G) \geq 1/4$. □

6.3 Potential Reduction = Total Squared Distance between Matched Vertices

Lemma 6.3. *The potential decrease in round i is precisely*

$$\Pi_{i-1} - \Pi_i = \frac{1}{2} \sum_{(u,v)\in M_i} \|\mathbf{P}_{i-1}(\cdot, u) - \mathbf{P}_{i-1}(\cdot, v)\|^2 .$$

Proof. Let $\mathbf{x}_{i,u} = \mathbf{P}_i(\cdot, u) - \mathbf{1}/n$. Then $\Pi_i = \sum_u \|\mathbf{x}_{i,u}\|^2$. For $u \notin M_i$, $\mathbf{x}_{i,u}$ is the same as $\mathbf{x}_{i-1,u}$ and the potential for u remains the same. For $(u, v) \in M_i$, the potential decrease for u, v is $\|\mathbf{x}_{i-1,u}\|^2 + \|\mathbf{x}_{i-1,v}\|^2 - 2\|(\mathbf{x}_{i-1,u} + \mathbf{x}_{i-1,v})/2\|^2$. Observe the identity $a^2+b^2-2((a+b)/2)^2 = (a-b)^2/2$. By applying the identity coordinate-wise, the potential decrease for u, v is precisely $\|\mathbf{x}_{i-1,u} - \mathbf{x}_{i-1,v}\|^2/2$. The claim follows by summing over all $(u, v) \in M_i$. □

6.4 Intuition of the Remaining Analysis

Imagine $\mathbf{P}_i \in \mathbb{R}^{V\times V}$ as an embedding of the vertices in the n-dimensional space, with u mapped to $\mathbf{P}_i(\cdot, u)$. By Lemma 6.3, if the points for the vertices are "well-separated" (see the left of Figure 2), and the cut is a good separation, then there would be large potential decrease (no matter how the matching behaves) as the distances between vertices on two sides are large.

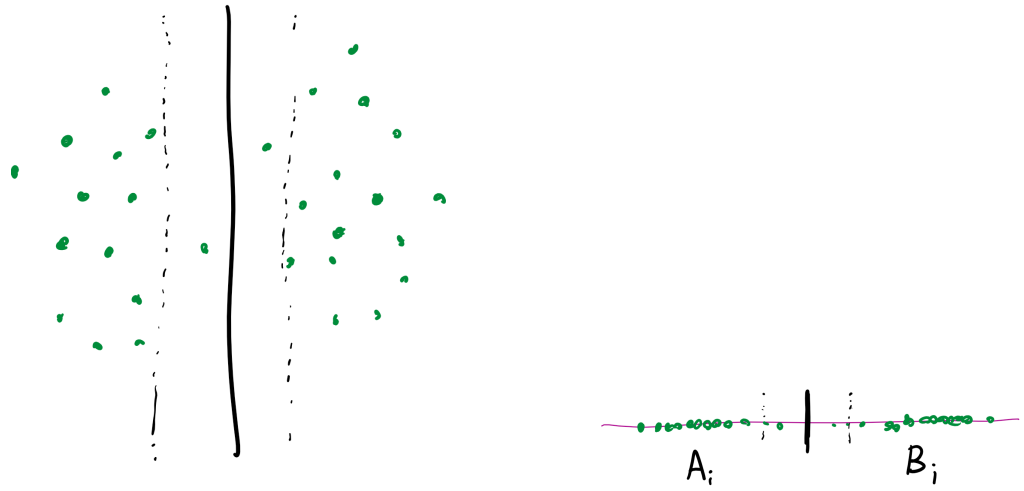


Figure 2: Left: Example of well-separated points, where the bold line is a good separation. Right: Random projection preserves distances well, so a good separation after projection is also a good separation before projection.

For efficiency, instead of computing a good separation in the n-dimensional space, we project onto a 1-dimensional subspace in a random direction $\mathbf{r}$ (see the right of Figure 2). This gives $\mathbf{x} = \mathbf{P}_i^\top \mathbf{r}$, with u mapped to $x_u = \mathbf{r}^\top \mathbf{P}_i(\cdot, u)$. Random projection has the property of preserving pairwise distances well. So the median cut after projection should approximate a good separation.

6.5 Random Projection Preserves Distances

Lemma 6.4 (Gaussian behavior of projections)**.** *For any vector $\mathbf{v} \in \mathbb{R}^d$ and random unit vector $\mathbf{r} \in \mathbb{S}^{d-1}$,*

1. $\mathrm{E}_{\mathbf{r}}[(\mathbf{r}^\top \mathbf{v})^2] = \|\mathbf{v}\|^2/d$;
2. $\Pr_{\mathbf{r}}[(\mathbf{r}^\top \mathbf{v})^2 \geq \alpha \cdot \|\mathbf{v}\|^2/d] \leq e^{-\alpha/4}$, *for all* $\alpha \leq d/16$.

Proof. By Johnson-Lindenstrauss lemma. □

Note that all the involved vectors $\mathbf{P}_i(\cdot, u) - \mathbf{1}/n$ as well as $\mathbf{P}_i(\cdot, u) - \mathbf{P}_i(\cdot, v)$ live in the $(n-1)$-dimensional subspace orthogonal to $\mathbf{1}$. So by constraining $\mathbf{r} \perp \mathbf{1}$, we can apply the above lemma to the $(n-1)$-dimensional subspace. Also note that by setting $\alpha = \Theta(\log n)$ the probability in the above lemma is inverse polynomial. Then we have the following lemma as a corollary.

Lemma 6.5. *It holds that:*

1. $\|\mathbf{P}_{i-1}(\cdot, u) - \mathbf{P}_{i-1}(\cdot, v)\|^2 = (n-1)\,\mathrm{E}[(x_u - x_v)^2]$, *and* $\|\mathbf{P}_{i-1}(\cdot, u) - \mathbf{1}/n\|^2 = (n-1)\,\mathrm{E}[x_u^2]$;
2. $\|\mathbf{P}_{i-1}(\cdot, u) - \mathbf{P}_{i-1}(\cdot, v)\|^2 \geq (n-1)/\Theta(\log n) \cdot (x_u - x_v)^2$ *with high probability* $1 - 1/\mathrm{poly}(n)$.

6.6 Median Cut is Good

Lemma 6.6. *With median cut,* $(n-1)\,\mathrm{E}[\sum_{(u,v)\in M_i}(x_u - x_v)^2] \geq \Pi_{i-1}$.

Proof. Suppose η is the median in $\mathbf{x}$. Note that with median cut, M_i is a perfect matching. Then by calculation,

$$\sum_{(u,v)\in M_i} (x_u - x_v)^2 \geq \sum_{(u,v)\in M_i} (x_u - \eta)^2 + (x_v - \eta)^2 = \sum_u (x_u - \eta)^2 = \sum_u x_u^2 - 2\eta \sum_u x_u + n\eta^2 \geq \sum_u x_u^2 \,.$$

where the last ineqaulity is because $\sum_u x_u = 0$ as $\mathbf{x} = \mathbf{P}_{i-1}^\top \mathbf{r} \perp \mathbf{1}$. Hence by Lemma 6.5, Item 1,

$$(n-1)\,\mathrm{E}\Big[\sum_{(u,v)\in M_i} (x_u - x_v)^2\Big] \geq (n-1)\,\mathrm{E}\Big[\sum_u x_u^2\Big] = \sum_u \|\mathbf{P}_{i-1}(\cdot, u) - \mathbf{1}/n\|^2 = \Pi_{i-1} \,. \qquad \square$$

6.7 Summary of Correctness

Putting together Lemma 6.3, Lemma 6.5, Item 2, and Lemma 6.6,

$$\begin{aligned}
\mathrm{E}[\Pi_{i-1} - \Pi_i] &= \frac{1}{2}\,\mathrm{E}\Big[\sum_{(u,v)\in M_i} \|\mathbf{P}_{i-1}(\cdot, u) - \mathbf{P}_{i-1}(\cdot, v)\|^2\Big] \\
&\geq \frac{1}{\Theta(\log n)} \cdot (n-1)\,\mathrm{E}\Big[\sum_{(u,v)\in M_i} (x_u - x_v)^2\Big] - \frac{\mu}{\mathrm{poly}(n)} \\
&\geq \frac{1}{\Theta(\log n)} \cdot \Pi_i - \frac{\mu}{\mathrm{poly}(n)} \,.
\end{aligned}$$

Here $\mu = \max_{\mathbf{r}}(n-1)/\Theta(\log n) \cdot \sum_{(u,v)\in M_i}(x_u - x_v)^2 \leq \Theta(n^2/\log n)$ (as $|x_u| \leq \|\mathbf{P}_i(\cdot, u)\|_2 \leq \|\mathbf{P}_i(\cdot, u)\|_1 = 1$). So with large enough hidden degree in the term $\mathrm{poly}(n)$ (which in turn comes from large enough hidden constant in $\alpha = \Theta(\log n)$), the term $\mu/\mathrm{poly}(n)$ remains $1/\mathrm{poly}(n)$.

Then we have

$$\mathrm{E}[\Pi_i] \leq \Big(1 - \frac{1}{\Theta(\log n)}\Big) \cdot \Pi_{i-1} + \frac{1}{\mathrm{poly}(n)} \,.$$

This implies after $I = O(\log^2 n)$ rounds, with high probability, $\Pi_I \leq 1/(4n^2)$ and thus $\Psi(G) \geq 1/4$.

6.8 Near-Linear Time

Note that $\mathbf{M}_i$ has $\Theta(n)$ non-zero entries. Then $\mathbf{x} = \mathbf{P}_i^\top \mathbf{r} = \mathbf{M}_i \cdots \mathbf{M}_1 \mathbf{r}$ can be computed in $O(n \cdot i)$ time. Moreover it takes $O(n \log n)$ time to sort and find the median cut in $\mathbf{x}$ (or actually $O(n)$ time to find the median cut without a thorough sorting). Hence the total running time of the randomized algorithm is $\sum_{i=1}^{I} O(n \cdot i) = O(nI^2) = O(n \log^4 n)$.

7 State of the Art

There is an improvement to the randomized algorithm where we have $\Psi(G) = \Omega(\log n)$ instead of $\Psi(G) = \Omega(1)$ at the end, while still using $I = O(\log^2 n)$ rounds (so that $\Phi(G) = \Omega(1/\log n)$ instead of $\Phi(G) = \Omega(1/\log^2 n)$), summarized in the following theorem.

000

Theorem 7.1 ([OSVV08])**.** *There is a randomized $\widetilde{O}(n)$-time algorithm for the cut player so that after $I = O(\log^2 n)$ rounds, $\Psi(G) = \Omega(\log n)$ (and thus $\Phi(G) = \Omega(1/\log n)$).*

We have seen a slow deterministic algorithm and a fast randomized algorithm, and there is also an algorithm which almost gives the best of both worlds, summarized in the following theorem.

Theorem 7.2 ([CGL+20])**.** *There is a deterministic $O(n^{1+o(1)})$-time algorithm for the cut player so that after $I = O(\log n)$ rounds, $\Phi(G) = \Omega(1/n^{o(1)})$.*

This leaves open the following question.

Question 7.3. Is there a deterministic $\widetilde{O}(n)$-time algorithm for the cut player so that after $I = \text{polylog}(n)$ rounds, $\Phi(G) = \Omega(1/\text{polylog}(n))$?

An affirmative answer to this question would improve a lot of $O(m^{1+o(1)})$-time deterministic algorithms to $\widetilde{O}(m)$.

Moreover, in the randomized algorithm, it is not clear how much precision we need for the random unit vector $\mathbf{r}$; in particular, the following question asks whether "one bit per dimension" could suffice.

Question 7.4. In the randomized algorithm, can we simply sample $\mathbf{r} \in \{-1, 1\}^V$ instead (and translate it so that $\mathbf{r} \perp \mathbf{1}$)?

8 Further Topics

8.1 Directed Graphs

One can generalize the cut-matching game to direct graphs, where in each round i:

- The cut player chooses two disjoint subsets of vertices A_i, B_i with the same size.
- The matching player chooses two perfect matchings $M_i^{\rightarrow}, M_i^{\leftarrow}$ in both directions A_i to B_i and B_i to A_i.
- Set $G_i = G_{i-1} \cup M_i^{\rightarrow} \cup M_i^{\leftarrow}$.

One can also define the *directed expansion* to be

$$\Psi_G(S) = \frac{\min\{|E(S, V \setminus S)|, |E(V \setminus S, S)|\}}{\min\{|S|, |V \setminus S|\}}.$$

Under this notion of expansion, the deterministic and the randomized algorithms can be generalized to direct graphs using the same idea: entropy-based deterministic algorithm [BPGS20] and ℓ_2-norm-based randomized algorithm [Lou10].

8.2 Lower Bounds

Theorem 8.1 ([OSVV08])**.** *In order to get* $\Psi(G) \geq \Omega(1)$*, we need* $I \geq \Omega(\sqrt{\log n})$*.*

Theorem 8.2 ([She09])**.** *In order to get* $\lambda_2(\mathbf{L}_G) \geq \Omega(1)$*, we need* $I \geq \Omega(\log n / \log\log n)$*.*

Question 8.3. How does the lower bound on I change when we require only $\Psi(G) \geq \Omega(1/\text{polylog}(n))$?

9 Exercises

Exercise 9.1. Improve the algorithms for the cut player so that the resulting graphs not only are expanders but also have approximately a given degree profile $\mathbf{d}$.

Solution. First consider the case where $\mathbf{d} = R \cdot \mathbf{1}$, where R is the number of rounds in the game, i.e. we want uniform-degree expanders. The randomized algorithm already satisfies this requirement as it always chooses balanced cuts. The deterministic algorithm can also be easily modified to satisfies this requirement: in each round i, instead of choosing a cut A_i decided by the algorithm, we choose a balanced cut $A'_i \supseteq A_i$ instead. Observe that this does not break the entropy-based analysis in Lemma 5.2 in the previous lecture.

For general $\mathbf{d}$, discretize it by $\mathbf{d} \approx R \cdot \mathbf{w}$. A first attempt could be to duplicate the vertices according to $\mathbf{w}$, run the (*uniform-degree*) cut-matching game for R rounds, and contract the vertices correspondingly at the end. However an issue is that there might be many edges among the duplicate vertices and the contraction would mess up the degree profile. Even if those edges are retained as self-loops, the running time of the cut player would depend on $\mathbf{d}(V)$, which could be huge and thus the dependency is undesired.

Instead, one can consider the weighted cut-matching game that exactly fits the purpose:

- The cut player finds a $\frac{1}{4}$-*balanced* sparse cut (A'_i, B'_i), where $\mathbf{w}(A'_i) \geq \frac{1}{4}\mathbf{w}(V)$, then chooses the cut (A_i, B_i) where $A_i \supseteq A'_i$ where $\mathbf{w}(A_i) = \mathbf{w}(B_i)$.
- The matching player chooses a $\mathbf{w}$-*matching* between A_i and B_i, which is a bipartite subgraph with degree profile $\mathbf{w}$.

The deterministic and the randomized algorithm can be generalized to this weighted case. For the randomized algorithm, we need to define the matrix $\mathbf{M}_i$ for lazy random walk on M_i in the way such that for each edge $\{a, b\} \in M_i$, add $1/(2\deg(a))$ to $\mathbf{M}_i(a, a), \mathbf{M}_i(a, b)$ and add $1/(2\deg(b))$ to $\mathbf{M}_i(b, b), \mathbf{M}_i(b, a)$; otherwise if $a \notin M_i$ then let $\mathbf{M}_i(a, a) = 1$. By running the $\mathbf{w}$-weighted cut-matching game for R rounds, an expander with approximate degree profile $\mathbf{d}$ can be obtained. The running time for the randomized algorithm is proportional to the maximum nonzero entries in $\mathbf{M}_i$, *i.e.* $O(\max_i \mathbf{w}(A_i) + n)$ whence the running time is $\widetilde{O}(\max_i \mathbf{w}(A_i) + n)$. □

Exercise 9.2. Improve the algorithms for the cut player so that they can handle matching players choosing *fractional matchings*. A fraction matching between A_i and B_i is a weighted bipartite graph where each edge can have fractional weight and the degrees of vertices in A_i ($|A_i| \leq |B_i|$) are exactly 1 and those of B_i are at most 1.

Solution. For the deterministic algorithm, we don't need to change anything except that we now find the balanced sparse cut in the weighted graph; and for randomized algorithm, we need to define the matrix $\mathbf{M}_i$ for lazy random walk on M_i in the way such that for each edge $\{a, b\} \in M_i$ with weight α, add $\alpha/2$ in entry $\mathbf{M}_i(a, a), \mathbf{M}_i(b, b), \mathbf{M}_i(a, b), \mathbf{M}_i(b, a)$; otherwise if a is not matched in M_i then let $\mathbf{M}_i(a, a) = 1$. This can be verified by redoing all the proofs, but conceptually, a factional matching is a convex combination of integral matchings, and thus the random walk over factional matchings can only mix faster than the random walk over integral matchings.

The running time of the (randomized) algorithm becomes $\widetilde{O}(\max \#\text{ of edges in fractional matching}) = \widetilde{O}(m)$ instead of $\widetilde{O}(n)$, since the fractional matchings may contain more edges whence $nnz(\mathbf{M}_i)$ =4(# of edges in fractional matching)+$O(n) = O(m)$. □

References

[BPGS20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, SCC, and shortest paths via directed expanders and congestion balancing. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science*, pages 1123–1134. IEEE Computer Soc., Los Alamitos, CA, [2020] ©2020.

[CGL+20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science*, pages 1158–1167. IEEE Computer Soc., Los Alamitos, CA, [2020] ©2020.

[KRV09] Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4):Art. 19, 15, 2009. Preliminary version in STOC 2006.

[Lou10] Anand Louis. Cut-matching games on directed graphs. *CoRR*, abs/1010.1047, 2010.

[OSVV08] Lorenzo Orecchia, Leonard J. Schulman, Umesh V. Vazirani, and Nisheeth K. Vishnoi. On partitioning graphs via single commodity flows. In *STOC'08*, pages 461–470. ACM, New York, 2008.

[She09] Jonah Sherman. Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$-approximations to sparsest cut. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2009*, pages 363–372. IEEE Computer Soc., Los Alamitos, CA, 2009.

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 16: Approximating Expansion via The Cut-Matching Game

October 29, 2021

Instructor: Thatchaphol Saranurak

Scribe: Chaitanya Nalam

Finding exact conductance (expansion) of a graph is NP-hard. In this course we have seen two methods to approximate the conductance.

- Cheeger's Inequality where we find a cut S where $\Phi_G(S) \leq \phi$ or reports $\Phi(G) \geq \Omega(\phi^2)$ in $O(\frac{m}{\phi^2})$. Approximation ratio is $\frac{1}{\phi}$ and can be very bad when ϕ is small.
- Leighten-Rao relaxation finds a cut S where $\Phi_G(S) \leq \phi$ or reports $\Phi(G) \geq \Omega(\phi/\log n)$. Although approximation ratio is good i.e. is $O(\log n)$. It is slow and takes the time to solve LP.

Today we are going to see an algorithm using the cut-matching game described in previous lecture, that can give best of both worlds i.e. good approximation and fast running time. In particular

- Approximation ratio: $O(\log^2 n)$ (#rounds in the cut-matching game)
- Time: $O(\log^2 n)$ max flow calls. (Best known running time for max flow is $\tilde{O}(m + n^{1.5})$ time).

We will see two applications based on cut-matching game which are:

1. Approximating expansion/conductance
2. Finding balanced sparse cuts

The cut-matching game framework is very flexible and can be used to solve many variants of the above problems. However, in this notes we focus on undirected, unweighted graph with n vertices and m edges.

1. Approximating Expansion

Recall that the expansion of a graph is denoted by $\Psi(G)$ and defined as

$$\Psi(G) = \min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{|S|, |V \setminus S|\}}$$

where $E(S, V \setminus S)$ is the edges set crossing the cut S. We will discuss later how we can change the algorithm to make it work for approximation conductance or $d-$expansion.

To approximate expansion to $O(\log^2 n)$ factor given a graph $G = (V, E)$ and parameter ϕ, either

- Find a cut S where $\Psi_G(S) \leq \phi$ (or)
- Report that $\Psi(G) \geq \Omega(\phi/\log^2 n)$.

In the first case if we return a sparse cut S then we are done, however in second case what does it mean by reporting $\Psi(G) \geq \Omega(\phi/\log^2 n)$.

What is the certificate or witness for saying that it has $\Omega(\phi/\log^2 n)$ expansion?

Answer: Embedding an expander of expansion $\Omega(1)$ in the graph G with at most $O(\log^2 n/\phi)$ congestion proves that expansion of the graph is high as well.

1.1. Lemma. *If* $X \preccurlyeq^{\text{flow}} O(\frac{\log^2 n}{\phi}) \cdot G$ *and* $\Psi(X) \geq \Omega(1)$, *then* $\Psi(G) \geq \Omega(\phi/\log^2 n)$.

Proof. For any $S \subseteq V$, we have

$$\begin{aligned} |E_G(S, V \setminus S)| &\geq \Omega(\frac{\phi}{\log^2 n}) \cdot |E_X(S, V \setminus S)| && \text{as } X \preccurlyeq^{\text{cut}} O(\frac{\log^2 n}{\phi}) \cdot G \\ &\geq \Omega(\frac{\phi}{\log^2 n}) \cdot \min\{|S|, |V \setminus S|\} && \text{as } \Psi(X) \geq \Omega(1) \end{aligned}$$

□

How can we embed an expander? This is where cut-matching game comes to our rescue. According to cut-matching game expander can be thought of as union of carefully selected matchings. If we embed every matching with a lesser congestion i.e. $O(1/\phi)$ then we can embed the whole expander which is union of $O(\log^2 n)$ matchings with at most $O(\log^2 n/\phi)$ congestion as required.

So from now on our goal is to either find a sparse cut S (first case) or find a matching embedding with $1/\phi$ congestion (towards the second case).

2. A Key Subroutine: Sparse Cut (or) Matching Embedding

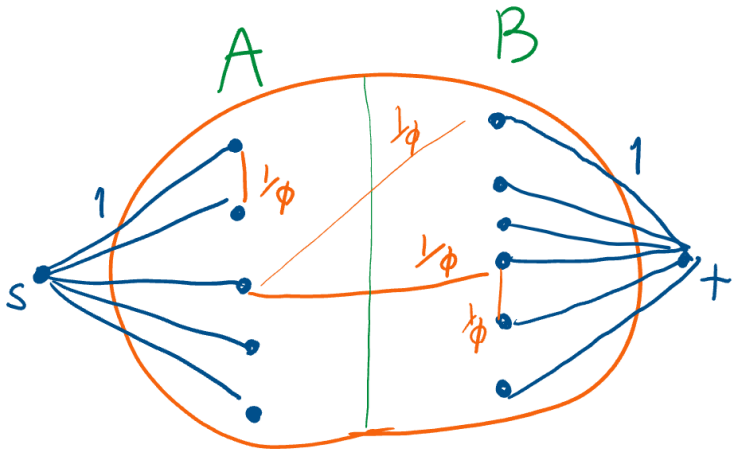
Formally we need to solve the following problem which will be a key subroutine to be used in cut-matching game.

Input: A graph $G = (V, E)$, a cut (A, B) where $|A| \leq |B|$, and a parameter ϕ

Output: Either

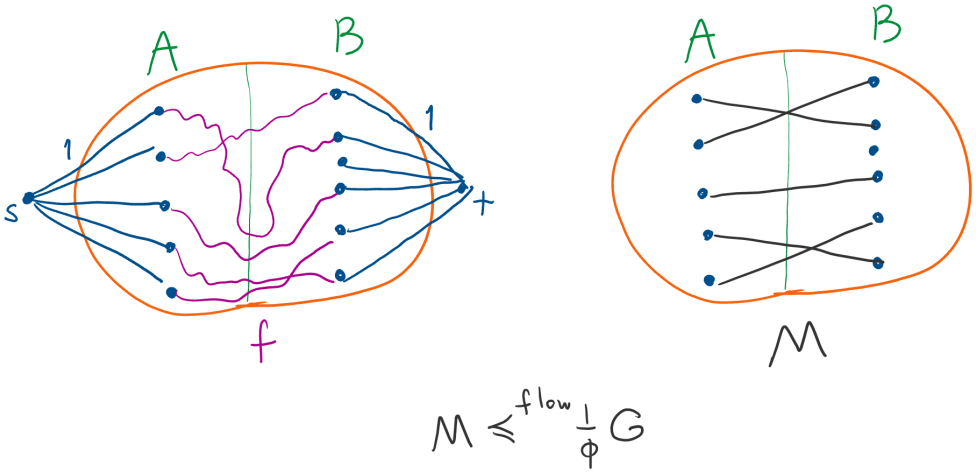
- A matching M between A and B of size $|A|$ and $M \preccurlyeq^{\text{flow}} \frac{1}{\phi} \cdot G$ (i.e. M is embedded into G with congestion at most $1/\phi$)
- A cut S where $\Psi_G(S) < \phi$.

In the graph G we are given a cut (A, B) with $|A| \leq |B|$. Consider the following (s, t)-flow problem where all the vertices in A are joined to a super source s and all the vertices in B are joined to a super sink t with capacities 1 and scale the capacities of edges in the graph G by $1/\phi$. Since it is unweighted graph initially new capacities would be equal to $1/\phi$. Let the resulting graph be $G_{s,t}$.



Solving the max flow min cut problem in $G_{s,t}$ will give us two cases. Either the max flow would be $|A|$ or less than it. It cannot be greater because the total out flow from s can have at most value equal to $|A|$.

Case 1: Max flow value is $|A|$. Let f be the corresponding $(s,t)-$ max flow. As f is an integral flow we can perform flow-path decomposition which can be done in $O(m)$ time using link-cut trees. There are $|A|$ paths exactly from s to t which have to go through a vertex in A and B as shown in figure below. For each of such path add the endpoints $a \in A, b \in B$ of the path (path excluding s,t) as an edge to set M.



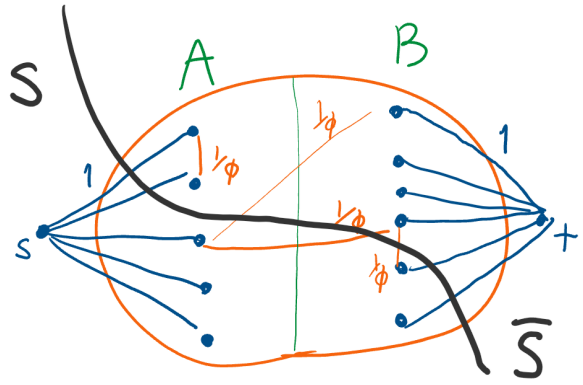
Observe that M is a matching because any vertex in A cannot send flow more than 1 receiving from s and any vertex in B cannot send more than 1 flow to t. Hence at most 1 path can be incident on any vertex as end points. Matching is of size $|A|$ and since this is a feasible flow returned by max flow algorithm which means congestion in original graph G is at most $1/\phi$.

$$M \preccurlyeq^{\text{flow}} \frac{1}{\phi} \cdot G$$

Case 2: Max flow is $< |A|$. So we have a cut whose size is $< |A|$ in graph $G_{s,t}$. Let that cut be S', where $s \in S'$ and $\overline{S'} = V(G_{s,t}) \backslash S'$.

$$c(E_{G_{s,t}}(S', \overline{S'})) < |A|$$

Consider the set $S = S' \setminus \{s\}$.



2.1. Claim. $\Psi_G(S) < \phi$.

Proof. From the picture above, observe that the flow from some vertices in $A \cap S$ is trapped in S and could not flow to other side to reach t even though the capacities are scaled by $1/\phi$. Hence this is a sparse cut. Formalized as follows.

$$|A \setminus S| + |B \cap S| + \frac{1}{\phi}|E_G(S, \overline{S})| = c(E_{G_{s,t}}(S, \overline{S})) < |A|$$
$$\frac{1}{\phi}|E_G(S, \overline{S})| < |A \cap S| - |B \cap S| \leq |S|$$

For another direction, we can also write

$$|B \setminus \overline{S}| + |A \cap \overline{S}| + \frac{1}{\phi}|E_G(S, \overline{S})| = c(E_{G_{s,t}}(S, V \setminus S)) < |A| \leq |B|$$
$$\frac{1}{\phi}|E_G(S, \overline{S})| < |B \cap \overline{S}| - |A \cap \overline{S}| \leq |\overline{S}|$$

So

$$|E_G(S, \overline{S})| < \phi \min\{|S|, |\overline{S}|\}.$$

□

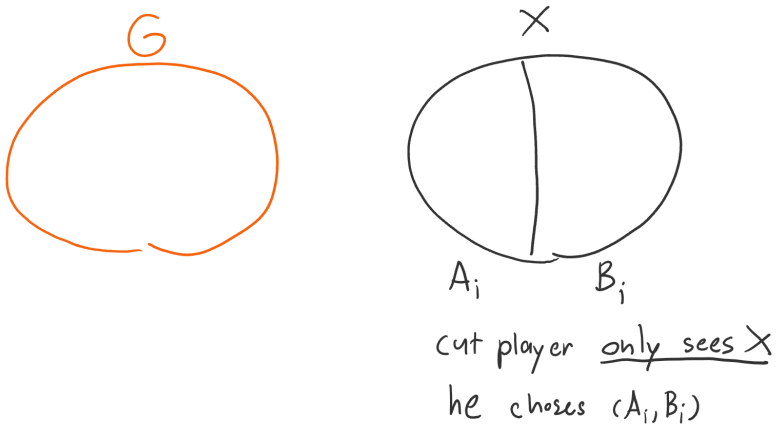
Now that we have embedded a matching we are one step closer to embedding an expander as it is union of matchings. Using this sub-routine and cut-matching game we formalize embedding an expander in the next section.

3. Combining the Two Ingredients

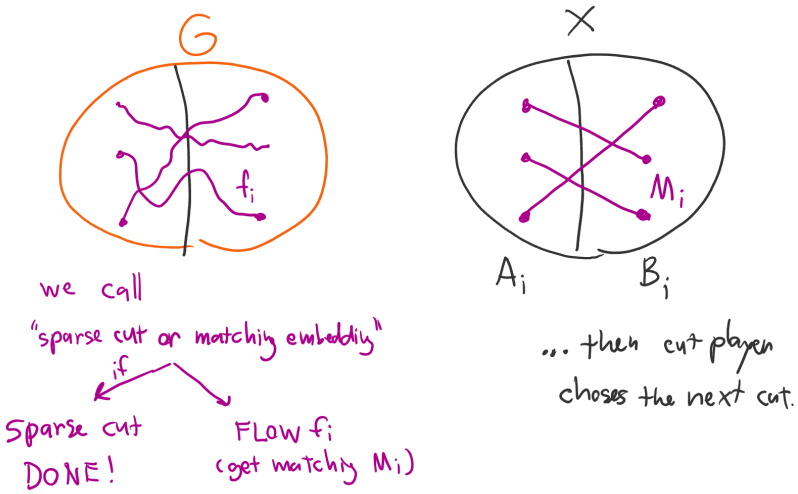
Recall that we have two players in the cut-matching game. Cut player chooses a cut (balanced cut containing sparse cut) from the graph constructed so far and matching player matches all the vertices on the smaller side by adding a matching to the graph.

For approximating expansion we either need to find a sparse cut or embed and expander. We combine our sub-routine "Sparse Cut or Matching Embedding" and cut-matching game as follows.

Cut player starts with an empty graph X and finds a cut A_i, B_i where i represents round in cut-matching game.



Given a cut A_i, B_i matching player calls the sub-routine "Sparse Cut or Matching Embedding". If a sparse cut is found then we are done by just returning that sparse cut in G, if not we get a flow from which matching player adds a matching to X between vertices in A_i, B_i. Observe that as per requirements of cut-matching game whenever we get a matching it perfectly matches every vertex in A_i.



Now again the cut player chooses a new cut A_{i+1}, B_{i+1} and game repeats.

There is a different way of thinking this process. One wants to find a sparse cut out of all possible subsets. However we want to be faster in that we should be finding sparse cut if it exists in $\tilde{O}(m)$ time instead of 2^n time. So we start searching for it by applying max flow min cut in an orientation (joining some vertices to s and other to t) and see if we can get a sparse cut. If we do not get one then we have to find different orientation of the graph where applying max flow

min cut will give us a chance to hit the sparsest cut. However we do not know which direction to proceed i.e. what vertices to combine to s and t.

This exact hint is given by X which keep tracks of all the directions you have explored till now. How? By storing all the matchings (directions) you have explored until now and suggesting you a new cut (which is sparse).

Why such a cut is useful hint to you? Because in X that "direction" characterized by the cut is having less edges across it. That means it is not explored enough and might have higher chance of containing the sparse cut.

Summarizing the formal algorithm:

Algorithm 1: $SparseCutOrExpanderEmbed(G, \phi)$

Initialize: $G = (V, E)$ be the input graph
$X_0 = (V, \{\})$ empty graph on V and it will be expander at the end
Let $R = O(\log^2 n)$ be the number of rounds in the cut-matching game
Result: Sparse cut (or) Expander embedding
for $i = 1, \cdots, R$ **do**
  Cut player reads X_{i-1} and choose a cut (A_i, B_i)
  Call Max flow on the constructed graph $G_{s,t}$ from (A_i, B_i)
  if *Flow value* $= |A_i|$ **then**
    Construct a matching from the end points of the flow's path decomposition;
    $M_i \preccurlyeq^{\text{flow}} \frac{1}{\phi} \cdot G$ and $X_i \leftarrow X_{i-1} \cup M_i$
  else
    return Sparse cut $S = S' \setminus \{s\}$ and we proved $\Psi_G(S) < \phi$
  end
end
$\Psi(X_R) = \Omega(1)$ by the cut-matching guarantee
$X_R \preccurlyeq^{\text{flow}} \frac{R}{\phi} \cdot G$ by the union of flows found so far
return X_R along with flow embedding as witness for $\Psi(G) \geq \frac{\phi}{R}$ as proved earlier.

Runtime analysis:

- Total running time of the cut-player:
  - $\tilde{O}(n)$ randomized (from previous class by KRV[1]).
  - $n^{1+o(1)}$ deterministic (by this paper[2]).
- Total running time of the matching player („Sparse Cut or Matching Embedding"):
  - $R = O(\log^2 n)$ max flow calls.

So in total runtime of $\tilde{O}(1)$ max flow calls we can either find a sparse cut S where $\Psi_G(S) \leq \phi$ or report that $\Psi(G) \geq \Omega(\phi/\log^2 n)$.

[1] https://dl.acm.org/doi/10.1145/1538902.1538903
[2] https://arxiv.org/pdf/1910.08025.pdf

4. Balanced Sparse Cut

Lets look at another problem of finding a balanced sparse cut which is useful has many applications. In the above sub-routine we found a sparse cut however there is no guarantee on how balanced it is. Sparse cut we obtain in previous sub-routine might have only few vertices. Formally the problem is as follows.

Problem:

Given a graph $G = (V, E)$ and parameters ϕ, β, we will either

- Find a cut S where $\Psi_G(S) \leq \phi$ and $\beta \leq |S| \leq n/2$, or
- Report that, for any cut S where $2000\beta R \leq |S| \leq n/2$, we have $\Psi_G(S) \geq \Omega(\phi/R)$.
  - Any cut that is much more balanced than β cannot be much sparse than ϕ.

4.1. Question. *How is such a guarantee useful?*

We can use it and do binary search to find

- "most balanced ϕ-sparse cut" or
- "sparsest β-balanced cut"

However, because of the nature of our guarantee we have to allow some gap of approximation in both expansion and balance.

As we have seen for previous problem, what does it mean to say that no cut which is more balanced is not much worse. In this case we need to return something that witnesses this guarantee. So we define our algorithmic goal as follows and we also prove why such an algorithmic goal would imply the required guarantee.

Our Algorithmic Goal: Given a graph $G = (V, E)$ and parameters ϕ, β, we will either

- Find a cut S where $\Psi_G(S) \leq \phi$ and $|S| \geq \beta$, or
- Return an "almost-expander" X' and a set F of "fake edges"
  - $\Psi(X) \geq 1/1000$ where $X = X' \cup F$
  - $X' \preccurlyeq^{\text{flow}}_{\frac{R}{\phi}} \cdot G$
  - $|F| \leq \beta R$

Note that we embed X' an "almost expander" into G instead of X which is an expander. X' is "almost expander" because it is close to becoming an expander by the addition of set of edges F which is of size at most βR. We call F "fake" because we do not embed edges in F into G.

Below we prove the lemma why such a witness would imply the required guarantee.

4.2. Lemma. *Given the above X' and F, every cut S where $2000\beta R \leq |S| \leq n/2$ is such that $\Psi_G(S) \geq \Omega(\phi/R)$.*

Proof. For any $S \subseteq V$ of large enough size i.e. $|S| \geq 2000\beta R$, we have

$$|E_X(S, V \setminus S)| \geq |S|/1000 \geq 2\beta R$$

First inequality follows as X is an $\Omega(1)-$expander. As we will prove later that $|F| \leq \beta R$ we have that

$$|F| \leq |E_X(S, V \setminus S)|/2.$$

Since, F is at most half the size of cut $(S, V \setminus S)$ in X. Cut size of $(S, V \setminus S)$ in X' is at least half the size of the cut in X as X and X' just differ by the edges in F. (Note that this is only true when the cut is more balanced i.e. $|S| \geq 2000\beta R$)

$$|E_{X'}(S', V \setminus S')| \geq |E_X(S, V \setminus S)| - |F| \geq |E_X(S, V \setminus S)|/2.$$

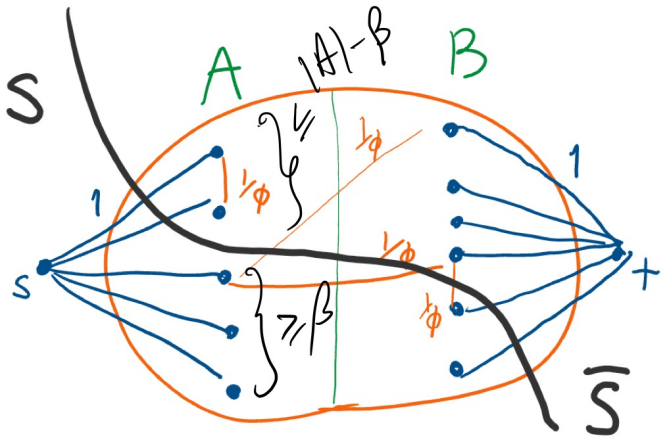
$$\begin{aligned}
|E_G(S, V \setminus S)| &\geq \frac{\phi}{R} \cdot |E_{X'}(S, V \setminus S)| && \text{as } X' \preccurlyeq^{\text{cut}} \frac{R}{\phi} \cdot G \\
&\geq \frac{\phi}{R} \cdot \frac{|E_X(S, V \setminus S)|}{2} && \text{as shown above for } |S| \geq 2000\beta R \\
&\geq \Omega(\frac{\phi}{\log^2 n}) \cdot \min\{|S|, |V \setminus S|\} && \text{as } \Psi(X) \geq \Omega(1) \\
\Psi_G(S) &\geq \Omega(\frac{\phi}{\log^2 n})
\end{aligned}$$

□

We solve the "Balanced Sparse Cut or Matching Embedding with Fake edges" similar to above problem. Instead of dividing into two cases based on value of flow at $|A|$ our new threshold would be $|A| - \beta$.

Why such a threshold would help?

Because when the value of flow is $\leq |A| - \beta$ there are at least β vertices that belong to A which are trapped in the cut $S = S' \setminus \{s\}$ as shown in diagram below. So the cut S is $\beta-$balanced.



If the flow is $> |A| - \beta$ then the matching M' would be of size at least $|A| - \beta$ so we get a flow embedding $M' \preccurlyeq^{\text{flow}} \frac{1}{\phi} G$. However because we want the matching to be of size $|A|$ i.e. whole of A should be matched for the cut-matching game to proceed. We now introduce some "fake" edges F that fill the gap and they can be at most β i.e. $|F| \leq \beta$ as we have got at least $|A| - \beta$ flow. But these are the edges created by us and not end points of flow path decomposition these are not embedded in G. Note that cut-matching game does not impose any constraints on the matching given by the matching player except that it is $A-$perfect. Since $|B| \geq |A|$ there exist β many vacant vertices on B side that we can match to unmatched vertices on A to form F whose union with M' gives $M = M' \cup F$.

Note that union of matchings M form an expander X after R rounds. But what is embedded in G is X' which is the union of edges in M' obtained in each of R rounds. Note that set of fake edges F are bounded by βR as we only get at most β in each round.

Given above explanation it is an easy exercise to show formally this works.

4.3. Exercise. Consider the graph $G_{s,t}$ with super source s and super sink t as defined before. Let F be the (s,t)-max flow value in $G_{s,t}$.

- If $|F| \geq |A| - \beta$, then we can find M' and F above.
- If $|F| < |A| - \beta$, then we can find S above.

4.4. Exercise. Show how to combine "Balanced Sparse Cut or Matching Embedding with Fake Edge" subroutine with the cut-matching game to solve the goal.

5. Extending to Capacitated graphs

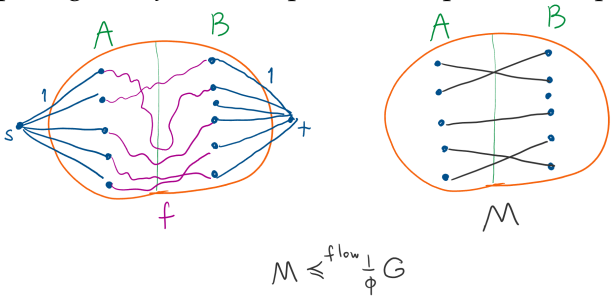
Till now we have assumed that our graph is unweighted however we will show in this section that cut-matching framework is good for extending to other variants. There are some problems though which we will see as we proceed and discuss solutions for the same.

Note that the definition of the expansion is different.

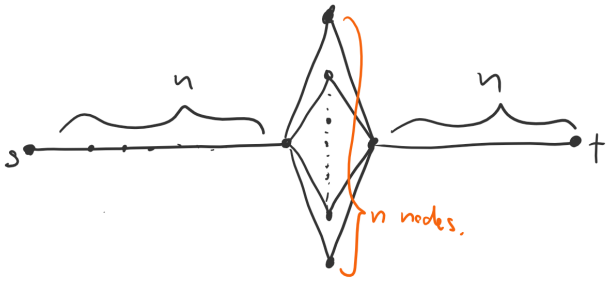
$$\Psi(G) = \min_{(S,V\setminus S)} \frac{c(E(S, V \setminus S))}{\min\{|S|, |V \setminus S|\}}.$$

Problem 1:

Flow-path decomposition in "Sparse Cut or Matching Embedding" subroutine. Recall that, after computing flow f, we compute a flow-path decomposition f.



In capacicated graph, explicit flow-path decomposition might take $\Omega(mn)$ time. In fact, just the total length of paths (ignore computation time) might be $\Omega(mn)$. For example, look at max flow in this graph



In general, there can be $\Omega(m)$ paths, each of length $\Omega(n)$. This is called the flow-decomposition barrier. It cannot be more than m paths because each and every time we extract a path we are maxing out at least one edge that belongs to that path making the residual capacity to 0 and this can be done at most m times.

However, using link-cut trees data structure again, we can still extract **the endpoints of these paths** in $O(m \log n)$ time, without listing the paths themselves.

From this, we get a **fractional matching** M where

- M has at most m non-zero weighted edges. (Before, it was an integral matching with at most n edges).
- M still has matching value $|A|$, i.e. $\sum_{e \in E(M)} w(e) = |A|$.

Problem 2:

Fractional (not integral) matching in the cut-matching game.

In round i of the cut-matching game, given a cut (A_i, B_i), we get a fractional matching M_i of size $|A_i|$ instead of an integral matching. Although we are not formally proving it here Cut-matching game would extend seamlessly to this setting. But the total running time of the cut player is $\tilde{O}(m)$ instead of $\tilde{O}(n)$ as fractional matching consists of at most $O(m)$ edges in each round because of which cut player have to find a cut (A_i, B_i) from a graph of size $\tilde{O}(m)$.

6. $d-$expansion

Cut matching game can be extended for approximating the general version of expansion i.e. $d-$expansion $\Phi_d(G)$. We need to adjust the two main ingredients of the cut matching game which are:

- Sparse Cut or $d-$Matching Embedding
- The Cut Matching Game for constructing $d-$expander

Sparse Cut or $d-$Matching Embedding:

- **Input**: a graph $G = (V, E)$, a cut (A, B) where $\boldsymbol{d}(A) \leq \boldsymbol{d}(B)$, and a parameter ϕ

- **Output:** return either
  - A "bipartite $\boldsymbol{d}$-matching" M between A and B of size $\boldsymbol{d}(A)$ such that $M \preccurlyeq^{\text{flow}} \frac{1}{\phi} \cdot G$. That is,
    - for all $a \in A$, $\deg_M(a) = \boldsymbol{d}(a)$
    - for all $b \in B$, $\deg_M(b) \leq \boldsymbol{d}(b)$
  - A cut S where $\Phi_{G,\boldsymbol{d}}(S) < \phi$.

Note that above sub routine can be solved by max flow min cut by joining the vertices to either s or t with capacities of their respective d values. We have to flow-path decomposition to get the endpoints of the $d-$Matching too. We restrict ourselves to integer $d-$expansions so that we can get an integral matching possibly with multi edges.

A Cut Matching Game (degree profile d version)

Given a target degree profile $\boldsymbol{d} : [n] \to \mathbb{Z}_{\geq 1}$, we can formally show how to construct an expander $G = (V, E)$ with n vertices and where each node v has degree $\boldsymbol{d}(v) \leq \deg_G(v) \leq \boldsymbol{d}(v) \cdot O(\log^2 n)$ using the cut-matching game framework, where in each round the matching player returns a "bipartite $\boldsymbol{d}$-matching".

The number of rounds should be $O(\log^2 n)$. The running time per round of the cut-player should be poly(n) or even $\tilde{O}(|E(G)|)$. That is, they should be independent from $\boldsymbol{d}(V)$.

7. Conclusion: Flexibility of The Cut-Matching Framework

- The framework of the cut-matching game for approximating expansion is very very flexible.
- We saw that it is useful for approximating
  - Expansion, conductance, and $\boldsymbol{d}$-expansion. (But not H-expansion for general H)
  - Balanced version for all of the above.
  - But these are kinds of edge expansion in undirected graphs (captured by theory we saw in the first half of the course).
- **Important:** But the cut-matching framework also works for
  - Vertex expansion
  - Directed graph version of all the above
  - Hypergraph version of all the above
- The total running time of each version $\approx$ polylog(n) calls to max flow on that graphs.

  For example,
  - vertex expansion: vertex capacitated max flow
  - many kinds of expansion in hypergraph: max flow on hypergraphs (which is even a special case of vertex capacitated max flow)

- many kinds of expansion in directed graphs using max flow on directed graphs

7.1. *Remark.* All variants of "Sparse Cut or Matching Embedding" can be solved (with slightly worse guarantee) using approximate max flow too.

- We showed that approximate sparsest cuts and balanced cuts reduce to polylog(n) max flow calls.
- Actually, they can be reduced to polylog(n) approximate max flow calls.
  - There are fast approximate max flow algorithms in **undirected** graphs
    - $\tilde{O}(m)$ time: edge capacitated graphs[3]
    - $m^{1+o(1)}$ time: vertex capacitated graphs, and hypergraphs[4]
  - So the total running time is almost always almost-linear in undirected graphs.

8. Reflection: How did we avoid solving multi-commodity flow?

Let Graph G be with degree profile $\boldsymbol{d}$ and K be the $\boldsymbol{d}$-product graph with same degree profile. Graph G can have conductance at least ϕ iff G can allow a flow embedding of $\phi \cdot K$ graph with at most $O(\log n)$ congestion i.e.

$$\Phi(G) \geq \phi \iff \phi K \preccurlyeq^{\text{flow}} O(\log n) G$$

So the largest value of ϕ for which we can embed $\phi \cdot K$ in G is the conductance of the graph G up to $\log n$ factors. Essentially approximating conductance of a graph is same as solving the multi-commodity flow problem by embedding K the $d-$product graph which have $|E(K)| = O(n^2)$ demands.

However there is another idea to reduce the number of demands that we need to embed. From flow view of expander we also know that for any $\tilde{\Omega}(1)$-expander X with degree profile $\boldsymbol{d}$ we have $X \approx^{\text{cut}}_{\text{polylog}(n)} K$ and so up to a polylog(n) factor,

$$\Phi(G) \geq \phi \iff \phi X \preccurlyeq^{\text{flow}} G$$

which need only $\tilde{O}(n)$ demands to be embedded.

So how did we approximate the conductance of the graph G without solving the maximum congestion flow problem?

Problem: As long as we fix an expander X_0, checking if $\phi X_0 \preccurlyeq^{\text{flow}} G$ is a multi-commodity flow problem with $\Omega(n)$ demands.

Idea: The best thing that *single-commodity flow subroutine* can do is embed a large matching by just joining the matching vertices by creating a super sink and super source which just increase the number of vertices by 2. But we cannot embed arbitrary matchings as they may not help us finally embed an expander X.

Solution: The cut-matching game is developed for this situation. It "forces" the union of arbitrary matchings to become an expander X by cleverly choosing the cuts in each round. Note that

[3] https://arxiv.org/pdf/1304.2077.pdf
[4] https://arxiv.org/abs/2101.07149

X is not fixed and not chosen by us, but it is an expander and that is good enough. Moreover this takes just $O(\log^2(n))$ *single-commodity flow* calls but embedding a matching each time.

This is the real motivation behind the development of the cut-matching game.

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 17: Expander Decomposition: Existence and Construction

October 28, 2021

Instructor: Thatchaphol Saranurak

Scribe: Tian Zhang

We have seen a rich theory around expanders. They are great for so many reasons. **It would be wonderful if we can exploit expanders in non-expander graphs too.** Now come the punchline: **It turns out that we can!** This is where expander decomposition comes into play.

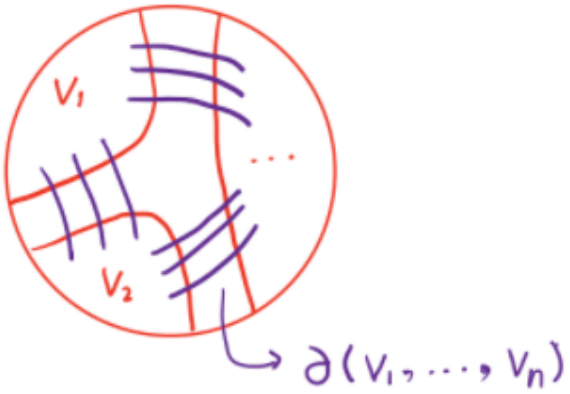
- Today: Existence and construction of expander decomposition.
- Next lecture: Applications of it.

1. Overview

Expander Decomposition = What Happens when Keep Finding Sparse Cuts. Let's say $G = (V, E)$ is unweighted. Let's fix parameter ϕ. Imagine the following process:

- DECOMP(G):
  - If $\Phi(G) \geq \phi$, return $\{V\}$
  - Else, there is a cut S where $\Phi_G(S) < \phi$, return DECOMP($G[S]$) $\cup$ DECOMP($G[V \setminus S]$)

If DECOMP(G) = $\{V_1, \ldots, V_k\}$, then we know each $\Phi(G[V_i]) \geq \phi$, i.e. $G[V_i]$ is a ϕ-expander.



- **Question:** How many edges we have „cut" throughout the whole process?
  - Formally, Let $\partial(V_1, \ldots, V_k)$ denote the edges crossing from V_i to V_j, $j \neq i$. What is $|\partial(V_1, \ldots, V_k)|$?
- **Answer:** $O(\phi m \log m)$ where $m = |E(G)|$.
- **Analysis:**
  - For each sparse cut $(S, V \setminus S)$ where $\text{vol}(S) \leq \text{vol}(V)/2$, we have that $|E(S, V \setminus S)| \leq \phi \text{vol}(S)$.
  - Charge the edges in $E(S, V \setminus S)$ to nodes in the smaller side S.
    - where each node $u \in S$ is charged $\phi \deg(u)$.
  - How many times a node can be charged?
    - $O(\log m)$, as we charge to the side where the volume is halved.

The following theorem concludes what we get.

1.1. Theorem (Expander Decomposition)**.** *For any unweighted graph $G = (V, E)$ with m edges and any $\phi \in [0, 1]$, there is a partition of vertices* $\textsc{Decomp}(G) = \{V_1, \ldots, V_k\}$ *such that*

- *$G[V_i]$ is a ϕ-expander, i.e., $\Phi(G[V_i]) \geq \phi$, and*
- *at most $\phi \log m$-fraction of edges crosses the partition.*

Intuitively, this is a maxim that you should keep

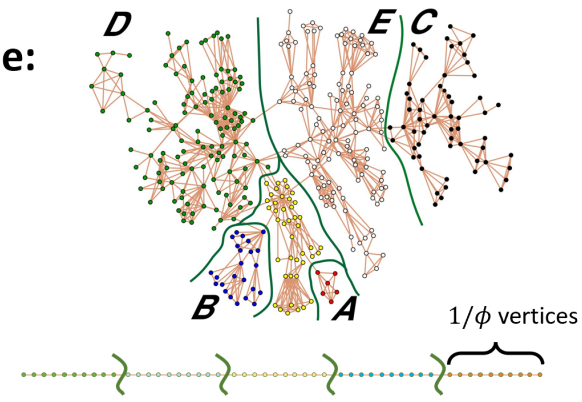
Any Graph = Disjoint Expanders + Small Fraction of Edges

Let's see some examples:

1.2. Example. What is $\textsc{Decomp}(G)$ when G is...

- Stars or Cliques
- Grids
- Planar graphs

Example:



However, **the algorithm is slow for two reasons**

1. **Checking $\Phi(G)$ exactly is NP-hard.**
   Easy to fix. Use approximation algorithms. Let's say that we can find approximate sparse cut near-linear time (and we can). There is still another problem.

2. **We might need to recurse to the bigger side $\Omega(n)$ time.**
   So the running time can be $\Omega(mn)$.

2. Fast Algorithm: Reduction to Balanced Sparse Cut

To bound the recursion depth (hopefully $O(\log m)$), we want to find the most-balanced $\phi-$sparse cut each time.

Let $\textsc{MaxBal}(G, \phi)$ be the volume of most-balanced ϕ-sparse cut.

$$\textsc{MaxBal}(G, \phi) = \max\{\text{vol}(S) \mid \Phi_G(S) < \phi \text{ and } \text{vol}(S) \leq \text{vol}(V \setminus S)\}$$

If $\Phi(G) \geq \phi$ (no ϕ-sparse cut), define $\textsc{MaxBal}(G, \phi) = 0$.

2.1. Exercise. Show an algorithm $\textsc{BalCut}(G, \phi)$ that either outputs

- $S = \emptyset$ and reports that $\Phi(G) \geq \phi$
- $S \neq \emptyset$ where $\text{vol}(S) \leq \text{vol}(V \setminus S)$ such that
  - $\Phi_G(S) < \phi \cdot c_{\text{exp}}$
  - $\text{vol}(S) \geq \textsc{MaxBal}(G, \phi)/c_{\text{bal}}$

where $c_{\text{exp}}, c_{\text{bal}} = O(\log^2 n)$ are approximation factor on expansion and balance. Note the algorithm described above approximately solves the most-balanced $\phi-$sparse cut problem.

$\textsc{BalCut}(G, \phi)$ can be solved via the cut-matching framework and takes polylog(n) (approximate) max flow calls. A natural way for using $\textsc{BalCut}$ for $\textsc{Decomp}$ would be this.

- $\textsc{Decomp}(H)$:
  - Let $S \leftarrow \textsc{BalCut}(H, \phi)$
  - If $S = \emptyset$ (i.e. $\Phi(H) \geq \phi$),
    - return $\{V(H)\}$
  - Else (i.e. $\Phi_G(S) < c_{\text{exp}} \cdot \phi$)
    - return $\textsc{Decomp}(H[S]) \cup \textsc{Decomp}(H[V \setminus S])$.
- Then, we call $\textsc{Decomp}(G)$.

2.1. Lucky Setting: Always Balanced Cuts

Suppose $\textsc{BalCut}(G, \phi)$ always returns a balanced cut S where $\text{vol}(S), \text{vol}(V \setminus S) \geq \text{vol}(V)/10$.

- **Question:** What is the total running time?

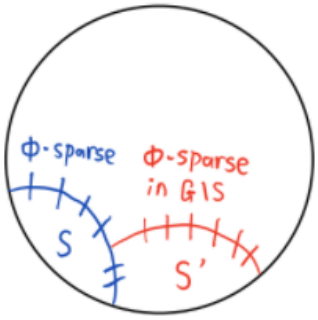
- **Answer:** There will be $O(\log m)$ recursion depth. Within the same depth, graphs are disjoint. Total time is $\tilde{O}(m)$.
- **Problem:** But what if all ϕ-sparse cuts are not balanced?
  - $\textsc{BalCut}(G, \phi)$ cannot return balanced cut.
  - So recursion depth is big?

2.2. Ideal Setting: Exact Algorithm

Suppose magically we can solve the exact version of $\textsc{BalCut}(G, \phi)$ where $c_{\text{exp}}, c_{\text{bal}} = 1$ in linear time.

Let $S \leftarrow \textsc{BalCut}(G, \phi)$ s.t. $\text{vol}(S) = \textsc{MaxBal}(G, \phi)$ and $\Phi_G(S) < \phi$. So S is the most-balanced cut among all ϕ-sparse cut.

- **The Setting:**
  - Suppose S is not very balanced (say, $\text{vol}(S) \leq \text{vol}(V)/10$). (Otherwise it is good.)
  - Let's look at we get when we recurse $\textsc{Decomp}(G[V \setminus S], \phi)$ on the bigger side.
  - Let $S' \leftarrow \textsc{BalCut}(G[V \setminus S], \phi)$.
- **Question:** Can S' be not balanced again (say, $\text{vol}(S') \leq \text{vol}(V \setminus S)/10$)?
- **Answer**: No.
  - This is because $S \cup S'$ would be a ϕ-sparse cut in G which is more balanced than the ϕ-sparse cut S.



  - Contradict the guarantee of S.
- **Conclusion:** That is, we cannot get very unbalanced cut two consective times on the bigger side. So the recursion depth is still $O(\log m)$!

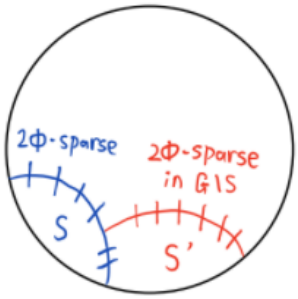
2.3. How Things Goes Wrong with Approximation

As we have seen,

- If we are always lucky, then the algorithm is fast.

- Else if we are not lucky but we have an exact algorithm that returns the most -balanced sparse cut, it's also fine.
- However, the reality is that we are not only unlucky but also have an approximate algorithm only.

The argument in 2.2 does not work even when $c_{\text{exp}} = 2$ (or anything > 1) and $c_{\text{bal}} = 1$. It's because We know that there is no ϕ-sparse cut which is more balanced than S, but S itself and $S \cup S'$ is only 2ϕ-sparse. There is no contradiction and the recursion depth seems to be still $\Omega(n)$.



2.4. The Trick: Adjusting Parameters

In reality, we have $c_{\text{exp}}, c_{\text{bal}} = O(\log^2 n)$. Let's see how bad it can be...

Suppose $S \leftarrow \textsc{BalCut}(G, \phi)$ is super unbalanced, say $\text{vol}(S) = O(1)$. Ok, S might not be useful. **But the algorithm just tells us something quite strong:** $\textsc{MaxBal}(G, \phi) \le O(c_{\text{bal}}) = O(\log^2 n)$.

Now comes **the key trick**:

- What if we run $\textsc{BalCut}(G, \phi/c_{\text{exp}})$ instead of $\textsc{BalCut}(G, \phi)$?
  - The cut returned must have conductance at most $c_{\text{exp}}(\phi/c_{\text{exp}}) = \phi$.
  - Can $\textsc{BalCut}(\cdot, \phi/c_{\text{exp}})$ keep returning $O(1)$-volume on the bigger side of the recursion?
  - Not more than $O(c_{\text{bal}})$ times! Otherwise the union of these cuts will of have volumes more than $O(c_{\text{bal}}) \ge \textsc{MaxBal}(G, \phi)$.
  - Contradiction.

2.5. Formal Algorithm

- **Parameters**:
  - $\varepsilon = 0.01$ (actually, we later we will set $\varepsilon = \Theta(\sqrt{\log n})$, but let's say $\varepsilon = 0.01$ for now)
  - $L = 1/\varepsilon$
  - Let $\phi_1 \ge \cdots \ge \phi_{L+1} = \phi$ where $\phi_{\ell+1} = \phi_\ell / c_{\text{exp}}$. So $\phi_1 = \phi \cdot \log^{O(L)} n$
  - $m = |E(G)|$
- $\textsc{Decomp}(H, \ell)$:

- Let $S \leftarrow \textsc{BalCut}(H, \phi_{\ell+1})$
- If $S = \emptyset$ (i.e. $\Phi(H) \geq \phi_{\ell+1}$),
  - return $\{V(H)\}$
- Else (i.e. $\Phi_H(S) < c_{\exp} \cdot \phi_{\ell+1} = \phi_\ell$ and $\mathrm{vol}(S) \geq \textsc{MaxBal}(H, \phi_{\ell+1})/c_{\mathrm{bal}}$))
  - If $\mathrm{vol}(S) \geq m^{1-\ell\varepsilon}/c_{\mathrm{bal}}$,
    - return $\underbrace{\textsc{Decomp}(H[S], 1)}_{\text{"left" recursion}} \cup \underbrace{\textsc{Decomp}(H[V \setminus S], \ell)}_{\text{"right" recursion}}$.
  - Else,
    - return $\underbrace{\textsc{Decomp}(H, \ell + 1)}_{\text{"down" recursion}}$

- Then, we call $\textsc{Decomp}(G, 1)$.
- Compare the difference:
  - We now have **levels**.
  - How we recurse depends on how balanced the cut S is.

2.6. Analysis

It is enough to analyze the depth of the recursion. (Why? Answer: The sum of the subgraphs at each level is just the original graph)

Consider a recursion tree $\mathcal{T}$:

- Each edge is labeled
  - „left" : small side
  - „right": big side
  - „down": same graph but increment level.
- Each leaf $x \in \mathcal{T}$ corresponds to an expander (because we stop the recursion).

Then, consider a recursion tree $\mathcal{T}$:

2.2. Lemma. *P can contains at most $O(\log m)$ left edges.*

Bizonyítás. Every time we go left, the volume is halved. □

2.3. Lemma. *Between two consecutive left edges in P, there are at most L many down edges.*

Bizonyítás. Between two consecutive left edges in P, the level never decreases. It increments for each down edge. After L down edges, we are at level $\ell = L + 1$. The condition

$$\mathrm{vol}(S) \geq m^{1-\ell\varepsilon}/c_{\mathrm{bal}} = 1/c_{\mathrm{bal}}$$

always holds. So we cannot have any more down recursion. □

- It remains to prove that between any left/down edges in P, there cannot be too many right edges.
- That is, there cannot be too many consecutive right edges in P.
- Now, we observe that the algorithm maintains this invariant:

2.4. Lemma. *When* $\textsc{Decomp}(H, \ell)$ *is called, we have* $\textsc{MaxBal}(H, \phi_\ell) < m^{1-(\ell-1)\varepsilon}$.

Bizonyítás. For $\ell = 1$, $\textsc{MaxBal}(H, \phi_1) \le m$ trivially holds (note $\mathrm{vol}(V)/2 = m$).

For $\ell > 1$, note that ℓ is incremented only by „down" edge. Before ℓ is increased, we have

$$\mathrm{vol}(S) \ge \textsc{MaxBal}(H, \phi_{\ell+1})/c_{\mathrm{bal}}, \text{ and}$$
$$\mathrm{vol}(S) < m^{1-\ell\varepsilon}/c_{\mathrm{bal}}$$

and then we set $\ell \leftarrow \ell + 1$. So the lemma holds. □

2.5. Lemma. *There is at most* $R = m^\varepsilon \cdot c_{\mathrm{bal}}$ *consecutive right edges in* P.

Bizonyítás.

- Consider the *right subpath* $P' \subseteq P$ of length R.
- Let H_1 be the graph corresponds to the node at the closest to root of P'. $H_2, \ldots, H_{R+1}$ are defined similarly.
- The algorithm calls $\textsc{Decomp}(H_1, \ell), \ldots, \textsc{Decomp}(H_R, \ell)$.
  - Note that the level ℓ never changes with the right edges.
  - We have $\textsc{MaxBal}(H_1, \phi_\ell) < m^{1-(\ell-1)\varepsilon}$ by the invariant of the algorithm.
- Let $S_i = \textsc{BalCut}(H_i, \phi_{\ell+1})$.
  - $\Phi_{H_i}(S_i) < c_{\mathrm{exp}} \cdot \phi_{\ell+1} = \phi_\ell$.
  - $\mathrm{vol}_{H_i}(S_i) \ge m^{1-\ell\varepsilon}/c_{\mathrm{bal}}$ for all i.
- Let $\overline{S} = S_1 \cup \cdots \cup S_R$.
- Observe that $\Phi_{H_1}(\overline{S}) < \phi_\ell$.
  - Because union of ϕ_ℓ-sparse cut is a ϕ_ℓ-sparse cut.
  - This is true if $\mathrm{vol}_{H_1}(\overline{S}) \le \mathrm{vol}_{H_1}(V(H_1))/2$.
  - But this might not be true. (**Exercise: How to fix this?)**
- Suppose $R > m^\varepsilon \cdot c_{\mathrm{bal}}$. Then,

$$\mathrm{vol}_{H_1}(\overline{S}) \ge R \cdot m^{1-\ell\varepsilon}/c_{\mathrm{bal}} > m^{1-(\ell-1)\varepsilon} = \textsc{MaxBal}(H_1, \phi_\ell).$$

- This is a contradiction. So $R \le m^\varepsilon \cdot c_{\mathrm{bal}}$.

□

2.6. Corollary. *The length of root-to-leaf path P is at most $O(\log n) \times L \times m^{\varepsilon} \cdot c_{\text{bal}} = \tilde{O}(m^{\varepsilon})$.*

- From here, we can conclude that expander decomposition can be computed in almost-linear time if we pay a $m^{o(1)}$ factor

2.7. Theorem (Fast Expander Decomposition)**.** *For any unweighted graph $G = (V, E)$ with m edges and any $\phi \in [0, 1]$ and a parameter $\varepsilon > 0$, we can compute a partition of vertices $\text{DECOMP}(G) = \{V_1, \ldots, V_k\}$ such that*

- *$G[V_i]$ is a ϕ-expander, i.e., $\Phi(G[V_i]) \geq \phi$, and*
- *at most $\phi \log^{O(1/\varepsilon)}$ m-fraction of edges crosses the partition.*

The algorithm call BALCUT *in graphs of $\tilde{O}(m^{1+\varepsilon})$ total number of edges. So it takes $\tilde{O}(m^{1+\varepsilon})$ total time.*

If we set $\varepsilon = O(1/\sqrt{\log m})$, then

- the running time is $\tilde{O}(m) \cdot 2^{O(\sqrt{\log m})} = m^{1+o(1)}$
- the fraction of crossing edges is $\phi \log^{O(\sqrt{\log m})} m = \phi m^{o(1)}$. (Instead of $\phi \log m$).

3. State of The Art

You actually saw the fastest algorithm.
There is another incomparable algorithm:

3.1. Theorem (Another Fast Expander Decomposition)**.** *For any unweighted graph $G = (V, E)$ with m edges and any $\phi \in [0, 1]$ and a parameter $\varepsilon > 0$, we can compute in $\tilde{O}(m/\phi)$ time a partition of vertices $\text{DECOMP}(G) = \{V_1, \ldots, V_k\}$ such that*

- *$G[V_i]$ is a ϕ-expander, i.e., $\Phi(G[V_i]) \geq \phi$, and*
- *at most $\phi \log^3$ m-fraction of edges crosses the partition.*
- This is faster and better for large ϕ, say $\phi \geq 1/\text{polylog}(n)$.

4. Flexibility of The Algorithm

Once you define a notion of sparse cut. The definition of expander decomposition follows. (Just keep finding a sparse cut).

4.1. Exercise. Try to state expander decomposition for these versions

- ***d***-expansion
- hypergraph

- vertex expansion
- directed graphs!

So the definition itself is flexible.But the algorithm we saw today is flexible too. Why because it is just a reduction to BALCUT,

- BALCUT can be solved based on the cut-matching game.
- The cut-matching game is very flexible to all notation of expansion.

The upshot is that, based on the lecture today, you compute all variant of expander decomposition using polylog(n) approx max flow calls!

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 18: Cut, Edge Sparsification via Expander Decomposition

November 2nd, 2021

Instructor: Thatchaphol Saranurak

Scribe: Nikhil Shagrithaya

1 Recap: Expander Decomposition

In the previous lecture, we discussed the process of expander decomposition, which is used to 'break' any graph into a disjoint union of expanders. More formally, we proved the following:

Theorem 1.1 (Expander Decomposition)**.** *For any unweighted graph $G = (V, E)$ with m edges and any $\phi \in [0, 1]$, there is a partition of vertices* $\textsc{Decomp}(G) = \{V_1, \ldots, V_k\}$ *such that*

- *$G[V_i]$ is a ϕ-expander, i.e., $\Phi(G[V_i]) \geq \phi$, and*
- *at most $\phi \cdot c_{\text{cross}}$-fraction of edges crosses the partition, where we chose c_{cross} to be equal to $\log m$.*

We also analyzed the running time, and saw that if we allowed c_{cross} to be equal to $m^{o(1)}$ instead of $\log m$, we can compute expander decomposition in almost linear time, that is, in $m^{1+o(1)}$ time. Note that this is algorithm was randomized, but it can be made deterministic (using $m^{1+o(1)}$-time deterministic cut player in the cut-matching game[1]).

Another incomparable algorithm exists, which when $c_{\text{cross}} = \log^3 m$, can compute in time $m \log^4(m)/\phi$ [2] (note the inverse dependence on ϕ). It is still an open question whether an expander decomposition algorithm exists which has $c_{\text{cross}} = \text{polylog}(m)$, and runs in time $m\text{polylog}(m)$.

Today, we see how expander decomposition can be used for "compressing" any graph. This is an extremely useful concept in theory and practice.

2 Repeated Expander Decomposition

Let's start with first simple, but powerful tool:

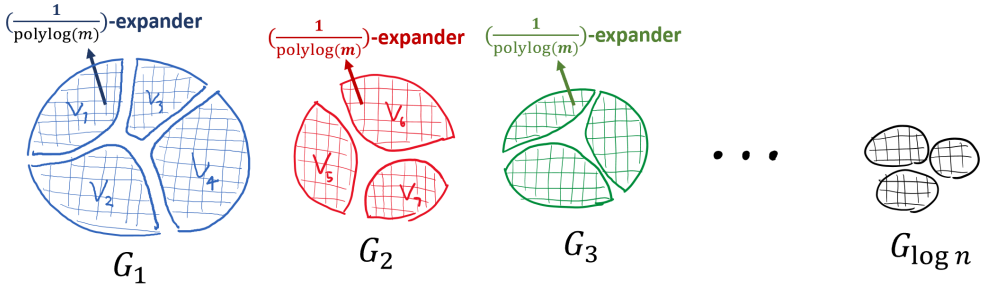
Theorem 2.1 (Repeated Expander Decomposition)**.** *For any unweighted graph $G = (V, E)$ with n vertices and m edges, there is a partition of* ***edges*** *E into $G_1, \ldots, G_{\log m}$ such that G_i contains disjoint union of $\Omega(\frac{1}{\log m})$-expanders.*

This technique has various applications, some of them being:

[1] https://arxiv.org/pdf/1910.08025.pdf

[2] https://arxiv.org/pdf/1812.08958.pdf

- (randomized): in time $m\text{polylog}(m)$ but with $(1/\text{polylog}(n))$-expanders.
- (deterministic): in time $m^{1+o(1)}$ but with $(1/m^{o(1)})$-expanders.



Intuitively, we are saying that **any graph can be decomposed into a union of expanders that only have a small overlap**.

3 Spanners

A spanner H of a graph G is a graph which has fewer edges, but still preserves all pairwise distances upto a multiplicative constant. Formally, given a graph G, a α**-spanner** H of G is a subgraph of G such that the following holds:

$$\text{dist}_G(u,v) \leq \text{dist}_H(u,v) \leq \alpha \text{dist}_G(u,v)$$

Our goal will be to find such a graph H which has the least number of edges possible. For general graphs, the best result is that it is possible to construct a $(2k-1)$-spanner of size $O(n^{1+1/k})$. This is a tight result, assuming the Erdos conjecture. In this lecture, we will show a simpler which is not as good as the best known, but a lot simpler: how to construct a polylog(n)-spanner of $\tilde{O}(n)$ size, using repeated expander decomposition.

3.1 Spanners are Composable

If we consider G as a union of two subgraphs, $G = G_1 \cup G_2$ (i.e, G_1 and G_2 share the same vertex set as G, but the $E(G)$ is partitioned between G_1 and G_2. Now, let H_1 be an α-spanner of G_1, and let H_2 be an α-spanner of G_2. Then, we get the following lemma:

Lemma 3.1. *$H = H_1 \cup H_2$ is an α-spanner of G.*

Proof. First, note that it is enough to show this: For every edge $e = (u,v) \in G$, we have $\text{dist}_H(u,v) \leq \alpha$ (why?). Now, if $e \in G_1$, then $\text{dist}_H(u,v) \leq \text{dist}_{H_1}(u,v) \leq \alpha$. Otherwise, if $e \in G_2$, then $\text{dist}_H(u,v) \leq \text{dist}_{H_2}(u,v) \leq \alpha$. Because every edge in G has to be either in G_1 or G_2, we are done. □

3.2 Spanners of Expanders are easy to find

Given a ϕ-expander G, it is very easy to find a $O(\log(m)/\phi)$-spanner H of G:

Proposition 3.2. *Let H be a shortest path tree (BSF tree) of a ϕ-expander G. Then, H is $O(\log(m)/\phi)$-spanner of G.*

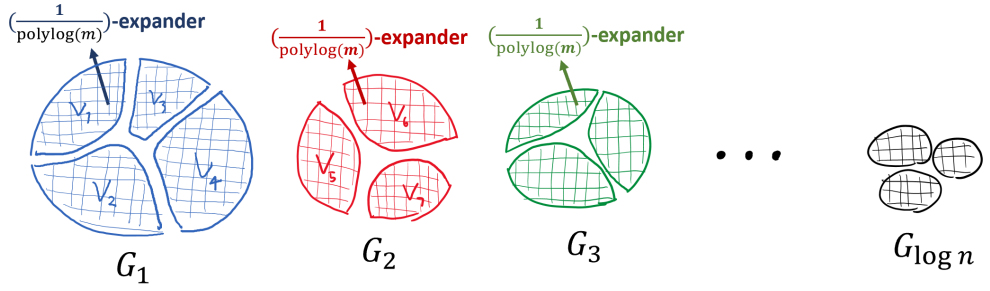
Proof. Recall that any ϕ-expander has diameter at most $D = O(\log(m)/\phi)$. Therefore, for any edge $(u, v) \in G$, we have $\text{dist}_H(u, v) \leq D$. □

Note: we did not exploit any expander here, and it is easy to find spanners as long as G has small diameter.

3.3 Final Step: Repeated expander decomposition.

We now know two things: that spanners are composable, and that the spanners of expanders are super easy. Therefore, the most obvious thing to do now is to decompose the graph into a union of expanders, and find spanners for those expanders, and then compose them to obtain a spanner for the whole graph. We will use repeated expander decomposition to obtain the union of expanders.

To express the above paragraph in a more detailed manner, we will sparsify each expander, that is, for each expander X in G_i, we will get a polylog(m)-spanner of size $O(V(X))$ (via shortest path tree). Then we will take the union of all sparsified expanders:



and by Lemma 3.1, we will get a polylog(m)-spanner of G of size $O(n \log n)$.

What is the running time? It is $\tilde{O}(m)$, because expander decomposition takes $\tilde{O}(m)$ time, and the time taken to find the shortest path tree on each expander: linear in the number of edges on each expander. Since these expanders are edge disjoint, we will only spend $O(m)$ time in total, for finding shortest path trees for all expanders in our expander decomposition.

Exercise 3.3. What if G is not unweighted? Suppose G has integral weights between $[1, \text{poly}(n)]$. Show how to get polylog(m)-spanner of G of size $O(n \log^2(n))$.

Exercise 3.4. What if we want a deterministic algorithm?

4 Cut Sparsifiers

In a manner similar to how we defined the concept of spanners, we have cut sparsifiers, where we want a graph with fewer edges, but still preserves the 'cut' information of the original graph. Formally, given a graph G, H is a **α-cut sparsifier** of G if H is α-cut-equivalent to G, i.e. $G \preccurlyeq^{\text{cut}} H \preccurlyeq^{\text{cut}} \alpha G$. We want H have a "small" number of edges, say $|E(H)| = \tilde{O}(|V(G)|)$ (**Note:** Even if G is unweighted, H must be weighted graph (why?).

4.1 Cut Sparsifiers are Composable

Again, we have a similar situation, as in the case of spanners:

Consider this situation, let $G = G_1 \cup G_2$, and let H_1 be an α-cut sparsifier of G_1, and H_2 be an α-cut sparsifier of G_2. Then:

Exercise 4.1. $H = H_1 \cup H_2$ is an α-cut sparsifier of G.

4.2 Known Construction of Cut Sparsifiers

Let $G = (V, E)$ be any general graph. Then for each edge $e = (u, v) \in E$, let $\lambda_e = \text{mincut}_G(u, v)$ denote the size of minimum cut that separates u, v. Now, consider this algorithm:

- For each $e \in E$,
  - add e to E_H with probability p_e where $p_e \geq \frac{\log^2 n}{\epsilon^2 \lambda_e}$.
  - If e was added to E_H, then set weight of e to be $1/p_e$.
- Return $H = (V, E_H)$.

The following is shown by Fung et al. [3]. We will not prove this.

Theorem 4.2. *$(1-\epsilon)G \preccurlyeq^{\text{cut}} H \preccurlyeq^{\text{cut}} (1+\epsilon)G$ with high probability.*

Therefore, $(1+\epsilon)H$ is a $(1+\epsilon)$-cut-sparsifier of G. Another lemma whose proof we will omit is:

Lemma 4.3. $\sum_{e \in E} \frac{1}{\lambda_e} \leq n - 1$.

From this lemma, we see that the expected size of $E(H)$ is:

Proposition 4.4. *If $p_e = \frac{\log^2 n}{\epsilon^2 \lambda_e}$, then $\mathbb{E}[|E(H)|] = \sum_{e \in E} p_e = \sum_{e \in E} \frac{\log^2 n}{\epsilon^2 \lambda_e} \leq n \log^2(n)/\epsilon^2$.*

In order to get an H such that $|E(H)| \leq O(n \frac{\log^2 n}{\epsilon^2})$ with high probability, we will just repeat the algorithm $O(\log n)$ times, and one of them will be good enough.

4.3 Cut Sparsifiers of Expanders are (Algorithmically) Easy

We want to apply the approach above on expanders., in the hope that it we can optimize it in some manner, by utilizing some nice expander properties. But at first glance, this algorithm looks very slow, because computing λ_e for every edge e can be too expensive. However, we just need a lower bound $\tilde{\lambda}_e \leq \lambda_e$, and then we can set $p_e = \frac{\log^2 n}{\epsilon^2 \tilde{\lambda}_e} \geq \frac{\log^2 n}{\epsilon^2 \lambda_e}$.

The lower bound should not be too small, because $\mathbb{E}[|E(H)|] \approx \sum_{e \in E} \frac{1}{\tilde{\lambda}_e}$. Now, suppose our graph G is a ϕ-expander. It turns out that it is actually very easy to get a somewhat accurate estimate of each λ_e.

Exercise 4.5. Given a ϕ-expander $G = (V, E)$ and $s, t \in V$,

$$\Omega(\phi) \cdot \min\{\deg(s), \deg(t)\} \leq \text{mincut}_G(s, t) \leq \min\{\deg(s), \deg(t)\}.$$

Sketch. Let K be the $\deg_G$-product graph. Recall that we have $\Omega(\phi) \cdot K \preccurlyeq^{\text{cut}} G \preccurlyeq^{\text{cut}} 2K$. Observe that $\text{mincut}_K(s, t) = \Theta(\min\{\deg(s), \deg(t)\})$. □

[3] https://epubs.siam.org/doi/abs/10.1137/16M1091666?journalCode=smjcat

For each edge $e = (u, v)$, we will just our set our estimate of λ_e as $\tilde{\lambda}_e = \Omega(\phi) \cdot \min\{\deg(s), \deg(t)\}$. Then we get:

$$p_e = \frac{\log^2 n}{\epsilon^2 \tilde{\lambda}_e} \leq O(\frac{\log^2 n}{\phi \epsilon^2 \min\{\deg(s), \deg(t)\}}).$$

As $p_e = \frac{\log^2 n}{\epsilon^2 \lambda_e}$ for all e, we have $(1-\epsilon)G \preccurlyeq^{\text{cut}} H \preccurlyeq^{\text{cut}} (1+\epsilon)G$ whp.

But what about the size of the edge set? Actually, we know that $\tilde{\lambda}_e$ and λ_e are within $O(\phi)$ factor. We can use the fact that $\sum_{e \in E} \frac{1}{\lambda_e} \leq n-1$ to conclude that $\mathbb{E}[|E(H)|] = O(n \frac{\log^2(n)}{\epsilon^2 \phi})$. Alternatively, we can prove this directly too.

Lemma 4.6. *$\sum_{e \in E} \frac{1}{\tilde{\lambda}_e} \leq O(n/\phi)$ and so $\mathbb{E}[|E(H)|] = O(n \frac{\log^2(n)}{\epsilon^2 \phi})$.*

Proof. We have

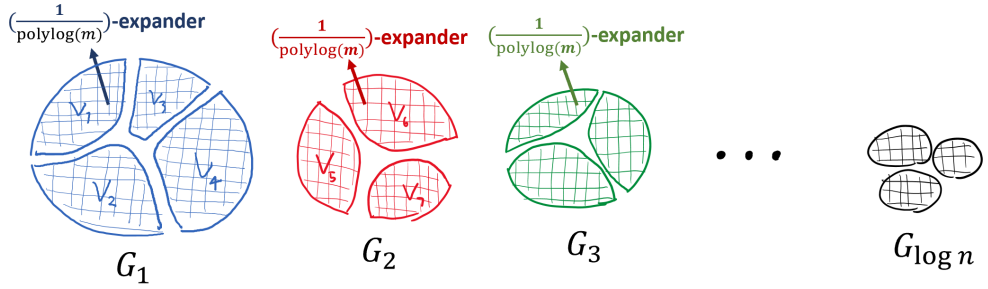
$$\sum_{e \in E} \frac{1}{\tilde{\lambda}_e} = O(\frac{1}{\phi} \sum_{(u,v) \in E} \frac{1}{\min\{\deg(u), \deg(v)\}}) = O(\frac{1}{\phi} n)$$

as each vertex u is counted $\deg(u)$ times. □

Finally, we can see we can compute $\tilde{\lambda}_e$ for all e, quite quickly, within $O(m)$ time.

4.4 Final Step: Repeated expander decomposition.

Similarly as in the case of spanners, we exploit the facts that cut sparsifiers are composable, and that cut sparsifiers of expanders are easy. We apply repeated expander decomposition to get a union of expanders such that each expander X in G_i, we get a $(1+\epsilon)$-cut-sparsifier of size $O(|V(X)| \frac{\log^2(n)}{\epsilon^2 \phi})$ where $\phi = 1/\text{polylog}(m)$.



In the end we combine the sparsifier to get a $(1+\epsilon)$-cut-sparsifier of G of size $\tilde{O}(n/\epsilon^2)$. The total running time required to execute all these operations is again $\tilde{O}(m)$ (why?).

4.5 Some Literature on $(1+\epsilon)$-Cut Sparsifiers

We just saw a randomized algorithm running in $\tilde{O}(m)$ time to obtain a cut sparsifier of size $\tilde{O}(n/\epsilon^2)$. This is worse than the state-of-the-art, but algorithmically very simple. Moreover, this approach can be extended to the dynamic setting.

This was first studied by Benczur and Karger: $O(n \log(n)/\epsilon^2)$ size, randomized, $\tilde{O}(m)$ time algorithm. See lecture notes by Debmalya too[4]

The state of the art is due to Batson, Spielman, Srivastava: they give $O(n/\epsilon^2)$ size, even deterministic, but slow algorithm (very different approach) [5]. Also see Lee and Sun: $O(n/\epsilon^2)$ size, randomized, $\tilde{O}(m)$.[6]. Lastly, it is open to give a deterministic algorithm which gives a $\tilde{O}(n/\epsilon^2)$ size cut sparsifier, running in $\tilde{O}(m)$ time.

[4]https://courses.cs.duke.edu//fall19/compsci638/fall19_notes/lecture8.pdf
[5]https://arxiv.org/pdf/0808.0163.pdf
[6]https://arxiv.org/pdf/1702.08415.pdf

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 21: Part 1 - Push-Relabel Flow Algorithms Introduction

December, 2021

Instructor: Thatchaphol Saranurak

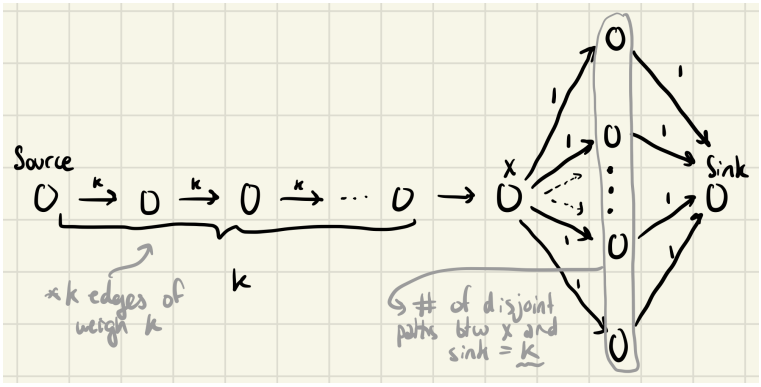
Scribe: Syed Akbari

Contents

1 Introduction and Overview

In this lecture, we will be introducing a new classical framework for working with flow problems: the push-relabel framework. Like the augmenting path framework we've done most of our work in so far (or blocking flow and multiplicative weight update frameworks if you're familiar with them), push-relabel refers to a general strategy or way of thinking about and working with flow problems.

Push-relabel has lots of really neat properties, as it is fast, dynamic, and is one of the most used algorithms in practice. The key property however that makes the push-relabel paradigm so powerful is that it allows for "amortization"; roughly speaking, this means it divides up the work very well locally on each node and edge, such that it can solve complex flow problems defined on all n nodes of the graph in only one call of the push-relabel algorithm, whereas most strategies we've seen require n calls . Consider the following flow problem as a motivating example, where there are there are k edges of k weight each going between the source and x, and between x and the sink, there are k disjoint paths, each of unit length:[1]



For this example, let k be some large number, like 100,000. Consider the augmenting-path based algorithm for solving this problem, the Ford-Fulkerson algorithm, which runs in $\Omega(k^2)$ time on this graph. Notice that at each iteration, the algorithm wastes time re-exploring the long path between the source and x, even though it doesn't change. What feels more intuitive would be to simple route the flow once in $O(k)$ time from the source to x, then from x to the sink, push the the flow again in simply $O(k)$ time through each of the disjoint paths- kind of like water flowing down all the available paths of least resistance. So intuitively, we see an an approach solving this $O(k)$ time overall.

Push-relabel is a framework built to, first, capture this essential idea in a way to work with any arbitrary graph, and second, be flexible enough to deal with all sorts of flow problems. To capture these two ideas in their full generality however, there are a several definitions that must first be introduced, but fortunately, they are all fairly intuitive to understand. Using these definitions, we can define the general push-relabel algorithm. After this point, we will first show a basic example of push-relabel in use, to find max-flow, and then a very cool modern example of push-relabel, to

[1] This example is drawn from: https://www.youtube.com/watch?v=0hI89H39USg&t=184s

efficiently find sparse-cuts and an efficient matching player strategy for the cut-matching game.

2 Definitions

You are all familiar at this point with the basic ideas of flow, demand, and source/sink with (s,t)-flows. Below however, we will redefine each of these in the context of push-relabel. They will still essentially be very similar to the (s,t)-flow style definitions, but they are just generalizations that run the engine under-the-hood for push-relabel. Redefining these three will allow us to define one new idea, preflow, after which we will have everything we need to start talking about the algorithm itself.

All the definition below are defined, given the following:
Let $G = (V, E, c)$ be an undirected capacitated graph where edge capacities are c$\in \mathbb{R}^E_{\geq 0}$.
For any $S \subseteq V$ let $\delta(S) = c(E(S, V \setminus S))$ denote the size of the cut S.

2.1 Flow

2.1.1 Definitions

Definition 2.1 (flow)**.** A **flow** $f : V \times V \to \mathbb{R}$ satisfies $f(u,v) = -f(v,u)$ and $f(u,v) = 0$ for $\{u,v\} \notin E$.

- The notation $f(u,v) > 0$ means that **mass** is routed in the direction from u to v.

Definition 2.2 (Congestion)**.** The **congestion** of f is $\max_{\{u,v\}\in E} \frac{|f(u,v)|}{c(u,v)}$. If the congestion is at most 1, we say that f is **feasible** or **respects capacities**.

- So in feasible flow, we have $-c(u,v) \leq f(u,v) \leq c(u,v)$. (Negative just means sending mass from v to u).

Definition 2.3 (Flow out/in of f)**.** The **net flow going out of** u is $f_{out}(u) = \sum_{v\in V} f(u,v)$. As you would expect, **net flow coming into** u is $f_{in}(u) = \sum_{v\in V} f(v,u)$.

Definition 2.4 (Flow out/in of S)**.** Similarly, the **net flow going out of** S is $f_{out}(S) = \sum_{u\in S} f_{out}(S)$ and the **net flow coming into** S is $f_{in}(S) = \sum_{u\in S} f_{in}(S)$.

2.1.2 Points to note

For building intuition, it's nice to play around with the definitions to see what are simple things we can deduce immediately. For example, directly from these definitions we get:

- $f_{out}(u) + f_{in}(u) = 0$ for all u.
- $f_{out}(V) = \sum_u f_{out}(u) = 0$

Also, compare this new definition of flow with the old (s,t)-definition. For any old-school (s,t)-flow f, $f_{out}(u) = 0$ for $u \neq s,t$, and f must respect capacity, whereas for our modern generalized idea of flow, both of these constraints are relaxed.

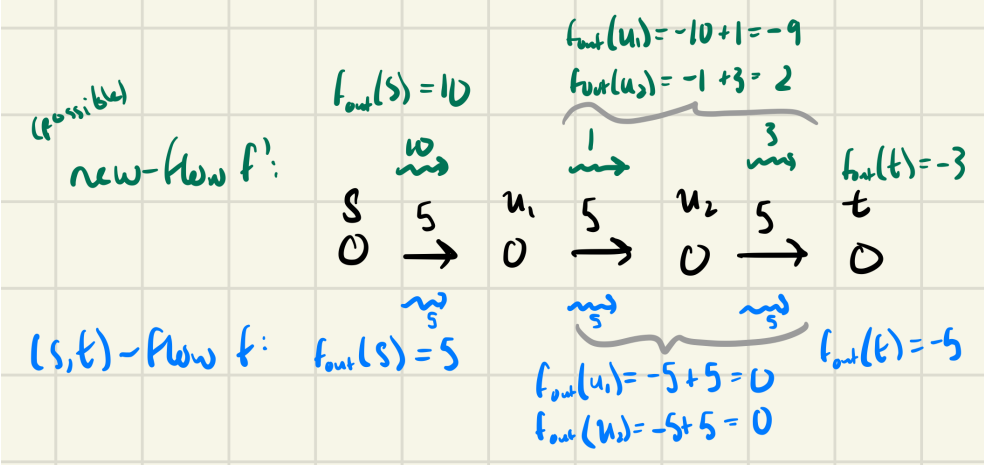


Figure 1: Relaxation of $f_{out}(u) = 0$, and edges can be greater than capacity.

2.2 Demand Functions

2.2.1 Definitions

Definition 2.5 (Demand function)**.** A **demand function** is simply some function mapping vertices in V to a number, so $\mathsf{dem} : V \to \mathbb{R}$.

Definition 2.6 (Flow satisfying demand)**.** We say that flow f **satisfies demand** dem if $\mathsf{dem}(u) = f_{out}(u)$ for all $u \in V$.

Definition 2.7 (Flow satisfying demand)**.** We say that flow f **satisfies demand** dem if $\mathsf{dem}(u) = f_{out}(u)$ for all $u \in V$.

Definition 2.8 (Total demand)**.** For any $S \subseteq V$, let $\mathsf{dem}(S) = \sum_{v \in S} \mathsf{dem}(v)$ be the **total demand** on S.

2.2.2 Points to note

Consider any of demand function we define (using this new idea of demand) such that:

$$\mathsf{dem}(u) = \begin{cases} \mathsf{val} & \text{if } u = s \\ -\mathsf{val} & \text{if } u = t \\ 0 & \text{if } u \neq s, t \end{cases}$$

A flow f satisfies this demand if and only if it is an (s,t)-flow. clearly then, this new definition is a generalization of our old-school (s,t)-flow idea of demand.

We also proved the following two ideas in HW2:

- A demand dem is satisfiable iff $\sum_u \mathsf{dem}(u) = 0$ (assuming that G is connected).
- A demand dem is satisfiable by some feasible flow iff $|\mathsf{dem}(S)| \leq \delta(S)$ for all $S \subseteq V$. *Hint*: Max-flow min-cut.

2.3 Source/Sink Functions

2.3.1 Definitions

Definition 2.9 (Source/sink function)**.** Let $\Delta : V \to \mathbb{R}_{\geq 0}$ be a **source function** and $T : V \to \mathbb{R}_{\geq 0}$ be a **sink function**. (Note: they map to non-negative numbers)

- $\Delta(v)$ specifies the amount of mass initially placed on v.

  Definition 2.10 (Source value for $S \subseteq V$)**.** $\Delta(S) = \sum_{u \in S} \Delta(u)$

- $T(v)$ specifies the capacity of v as a sink.

  Definition 2.11 (Sink value for $S \subseteq V$)**.** $T(S) = \sum_{u \in S} T(u)$

Definition 2.12 (Flow 'at' a node)**.** $f_{at}(v) = \Delta(v) + f_{in}(v)$, so the flow 'at' a v is the **net flow ending at** v after the routing f.

Definition 2.13 (Flow source-sink)**.** f satisfies source-sink (Δ, T) iff:

1. $f_{out}(v) \leq \Delta(v)$ for all v
   (i.e. the net flow out of v is at most its initial mass)
2. $f_{at}(v) \leq T(v)$ for all v
   (i.e. the net flow ends at v is at most its sink capacity)

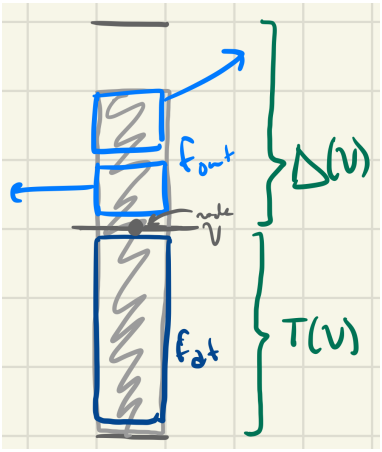


Figure 2: $f_{out}(v) \leq \Delta(v)$ and $f_{at}(v) \leq T(v)$.

2.3.2 Points to note

These ideas of source and sink functions can be thought of as connecting the previous two generalized ideas of flow and demand, into a workable framework in terms of flows that "satisfy (Δ, T)".

Consider the following problem: Given $A, B \subseteq V$ where $|A| \ll |B|$:

- route the flow such that every vertex in A should send 1 unit of flow to some node in B, and...

- each vertex in B can receive at most 1 unit.

Thinking about this in terms of dem and (s,t)-flows, there is no clear idea of how to approach this. We could try to add a new super source node and new super sink node, and then work off of that, but if we want to solve this problem in the original graph, there is no natural way to do so. However, "(Δ, T) satisfaction" can deal with these types of problems quite naturally. Consider the following exercise.

Exercise 2.1. Given a satisfiable demand dem (i.e. $\sum_u \mathsf{dem}(u) = 0$), we can define (Δ, T) that capture the dem. Let

$$\Delta(u) = \begin{cases} \mathsf{dem}(u) & \text{if } \mathsf{dem}(u) \geq 0 \\ 0 & \text{else} \end{cases}$$

$$T(u) = \begin{cases} -\mathsf{dem}(u) & \text{if } \mathsf{dem}(u) < 0 \\ 0 & \text{else} \end{cases}$$

Observe that f satisfies dem iff f satisfies (Δ, T).

Proof. Suppose f satisfies dem. Then, we have $f_{out}(u) = \mathsf{dem}(u) \leq \Delta(u)$ and also $f_{at}(u) = \Delta(u) + f_{in}(u) = \Delta(u) - \mathsf{dem}(u)$, which means

$$f_{at}(u) = \begin{cases} \mathsf{dem}(u) - \mathsf{dem}(u) = 0 \leq T(u) & \text{if } \mathsf{dem}(u) \geq 0 \\ 0 - \mathsf{dem}(u) \leq T(u) & \text{if } \mathsf{dem}(u) < 0. \end{cases}$$

Next, suppose f satisfies (Δ, T). Then, we have

$$f_{out}(u) \leq \Delta(u)$$

and

$$\Delta(u) - f_{out}(u) \leq T(u) \iff f_{out}(u) \geq \Delta(u) - T(u).$$

So for u where $\mathsf{dem}(u) \geq 0$, we have

$$f_{out}(u) = \Delta(u) = \mathsf{dem}(u).$$

Now, for u where $\mathsf{dem}(u) < 0$, we have

$$\mathsf{dem}(u) \leq f_{out}(u) \leq 0.$$

We claim that $\mathsf{dem}(u) = f_{out}(u)$ too in this case. Otherwise, suppose for contradiction that there is u where $\mathsf{dem}(u) < f_{out}(u) \leq 0$. Then, we get a contradiction below

$$0 = \sum_{u:\mathsf{dem}(u)\geq 0} \mathsf{dem}(u) + \sum_{u:\mathsf{dem}(u)<0} \mathsf{dem}(u) < \sum_{u:\mathsf{dem}(u)\geq 0} f_{out}(u) + \sum_{u:\mathsf{dem}(u)<0} f_{out}(u) = 0.$$

□

Exercise 2.2. We have

- (Δ, T) is satisfiable by some flow iff $\Delta(V) \leq T(V)$. (Assuming that G is connected.)
- (Δ, T) is satisfiable by some feasible flow iff $\Delta(S) - T(S) \leq \delta(S)$ for all $S \subseteq V$. *Hint*: Max-flow min-cut.

Proof. The work here carries over straightforwardly from 2.2.2. □

2.4 Preflow

2.4.1 Definitions

Given source/sink functions (Δ, T)...

Definition 2.14 (Preflow)**.** We say that f is a **preflow** w.r.t. (Δ, T) iff:

- $f_{out}(v) \leq \Delta(v)$ for all v
- **but we allow** $f_{at}(v) > T(v)$
  (this is the 'excess' unit(s) of flow left that push-relabel is trying to 'push' away to some sink, so we aren't 'finished' building the flow, hence the name 'preflow')

Definition 2.15 (Absorbed mass)**.** Let $\text{ab}_f(v) = \min\{f_{at}(v), T(v)\}$ be the **absorbed mass** on v (note: nonnegative).

Definition 2.16 (Saturated sink)**.** When $\text{ab}_f(v) = T(v)$, we say that v**'s sink is saturated**.

Definition 2.17 (Excess)**.** Let $\text{ex}_f(v) = f_{at}(v) - \text{ab}_f(v)$ be the **excess on** v (note: nonnegative).

2.4.2 Points to note

The first key point to note is that f satisfies source sink (Δ, T) if and only if there is no excess.

Proposition 2.3. *f satisfies (Δ, T) iff $\text{ex}_f(v) = 0$ for all v.*

Proof. $\text{ex}_f(v) = 0 \iff f_{at}(v) = \text{ab}_f(v) \iff f_{at}(v) \leq T(v)$. □

Proposition 2.4. *Let f be a preflow w.r.t (Δ, T). Then, $f_{at}(v) \geq 0$ for all v.*

Proof. We have $f_{at}(v) = \Delta(v) + f_{in}(v) = \Delta(v) - f_{out}(v)$, but $f_{out}(v) \leq \Delta(v)$. □

So all three of the terms we care about, $\text{ex}_f, \text{ab}_f, f_{at}$, are nonnegative, and can be understood naturally in terms of each other.

Exercise 2.5. For any vertex set $S \subseteq V$, we have

$$\underbrace{\Delta(S)}_{\text{initial mass}} = \underbrace{f_{out}(S)}_{\text{net flow out}} + \underbrace{\text{ab}_f(S)}_{\text{absorbed mass}} + \underbrace{\text{ex}_f(S)}_{\text{excess}}$$

Proof. For any u, we have two inequalities

$$f_{at}(u) = \Delta(u) + f_{in}(u)$$
$$f_{at}(u) = \text{ex}_f(u) + \text{ab}_f(u).$$

So

$$\Delta(u) = f_{out}(u) + \text{ex}_f(u) + \text{ab}_f(u)$$

and we are done by summing over $u \in S$. □

3 The Push-Relabel Framework

Given the above terminology, we have everything we need to describe the structure and actual algorithm of the PUSH/RELABEL framework.

3.1 Inputs, Structure, and Invariants of the Algorithm

Inputs:

- graph $G = (V, E, c)$
- source function Δ
- sink function T
- parameter h (height)

Maintains (keeps track of these structures at all steps) :

- a preflow f w.r.t. (Δ, T)
- a label for each vertex $\ell : V \to \{0, \ldots, h\}$

Definition 3.1 (Valid state)**.** We say that (f, ℓ) (aka: flow f w.r.t the status of the vertex labels ℓ at the current iteration in the algorithm) is a (G, Δ, T)**-valid state** iff:

1. For all nodes $u, v \in V$, whenever $\ell(u) > \ell(v) + 1$, then f fully saturates (u, v) from u to v (i.e. $f(u, v) = c(u, v)$)
2. For all nodes u s.t. $\ell(u) \geq 1$, f has completely filled the sink for u, or in other words, u is fully absorbed (i.e. $\text{ab}(u) = T(u)$).

Definition 3.2 (Valid solution)**.** We say that (f, ℓ) is a (G, Δ, T)**-valid solution** if (f, ℓ) is a (G, Δ, T)**-valid state**, and additionally, we have that

1. If $\ell(u) < h$, then u has no excess (i.e. $\text{ex}(u) = 0$)

Given these inputs and structures, we can state the key property for the algorithm. Assuming we are given a (G, Δ, T)-valid state as input, the goal is to keep (f, ℓ) a (G, Δ, T)-valid state at all times. So the two **invariants** come from the two parts of definition 3.1:

1. *"For all nodes $u, v \in V$, whenever $\ell(u) > \ell(v) + 1$, then f fully saturates (u, v) from u to v (i.e. $f(u, v) = c(u, v)$)"*

   This can be understood as saying that "only edges between nodes in adjacent levels in ℓ have some left over capacity over which you can still route some flow".

   It can be helpful to think of the directed **residual graph** $G_f = (V, E_f)$ where $E_f = \{(u, v) \mid r_f(u, v) > 0\}$, where $r_f(u, v) = c(u, v) - f(u, v)$ is the **residual capacity** of (u, v). In these terms, this variant can be understood as saying that all the "downward edges" in G_f cannot skip levels.

2. *"For all nodes u s.t. $\ell(u) \geq 1$, f has completely filled the sink for u, or in other words, u is fully absorbed (i.e. $\text{ab}(u) = T(u)$)."*

   This is basically saying that all the nodes at non zero levels have no more sink space left inside them, so provided they have any excess, it must be routed elsewhere.

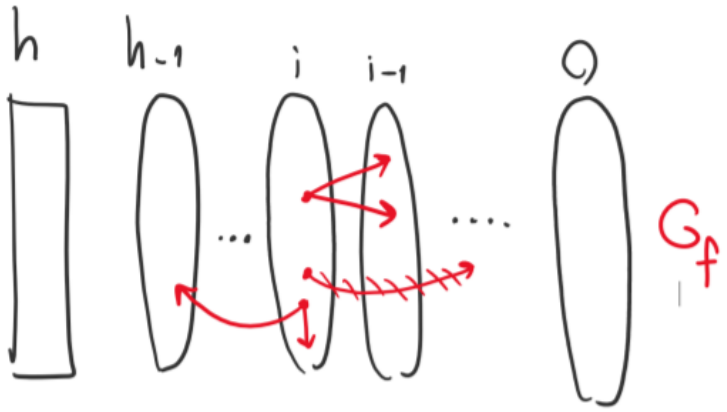


Figure 3: Residual graph interpretation of invariant 1.

The goal of the algorithm is then to, while keeping in line with these invariants, end with (f, ℓ) as a (G, Δ, T)**-valid solution**.

Initialization:
You can think of G, Δ, and T, as being the "raw inputs", so in practice, these would be given to you. h, as will become clearer below, is something you set, depending on the type of problem you are working with.

Now, while the algorithm will be maintaining and updating f and ℓ at each iteration, you can choose what state you want to start them off in. As we want the algorithm to keep (f, ℓ) in a (G, Δ, T)-valid state at all times (recall why we have the two invariants we have), it makes sense that we would like to feed it in some (G, Δ, T)-valid state. $f \equiv 0$ and $\ell \equiv 0$ is trivially a (G, Δ, T)-valid state and works as an input, but a really neat fact that you will see in section 4 below is that actually every (G, Δ, T)-valid state works as input! We will use this fact to get some interesting local and dynamic properties of push-relabel next lecture.

3.2 Algorithm

These two simple definitions will allow us to describe the algorithm. The role each definition plays will become clear by looking at the algorithm.

Definition 3.3 (Active vertex)**.** A vertex u is **active** if $\ell(u) < h$ and $\text{ex}(u) > 0$.

- In other words, a vertex is active if the algorithm hasn't given up on it ($\ell(u) < h$), and if there is still some excess that needs to be routed to a sink.

Definition 3.4 (Admissible arc)**.** An arc (u, v) in G_f is **admissible** if $r_f(u, v) > 0$ and $\ell(v) = \ell(u) - 1$.

- In other words, an arc is admissible if it has some capacity left over that's not being used, and is also a valid arc to send flow through, keeping in line with invariant 1.

Now, we will actually describe the algorithm, first through words and intuition, then we will provide the pseudo code.

Algorithm in words:

1. Find an **active** vertex u.
   - If there isn't that means that, either, we've given up on all of them ($\forall u \in V\ \ell(u) \geq h$), or all the excess has been successfully routed to the sinks ($\forall u \in V\ ex(u) = 0$) and we have in a (G, Δ, T)-valid solution. In either case, it makes sense to terminate.
   - Alternately, if there is some active node, that means we've got a node that has some excess to route, preventing us from having a (G, Δ, T)-valid solution, and we still haven't give up on it. So, we can continue onward in the algorithm to try to push the excess off.
2. If u has some **admissible** edge (u, v) attached to it, push as much excess as possible down that edge.
   - An edge (u, v) in G is admissible if the flow going through it hasn't yet maxed out it's capacity ($r_f(u, v) > 0$, and if it's attached to some node that's in the immediately smaller ℓ level below it.
   - Intuitively, you should think of ℓ as the height. So if a node is in ℓ level 3, it clearly can't send flow upwards to 4, and also, before it can send flow to ℓ level 2, the flow has to flow over ℓ level 3. Moreover, for the nodes v in level 2, because they are more than one level below u, by invariant 1, the edges (u, v) are have no capacity left anyways ($\ell(v) < \ell(u) - 1 \implies r_f(u, v) = 0$).
3. Else, **relabel** u by putting it in a higher level.
   - If there all the edges going from u to a lower level are full, then it makes sense to try the edge between u and the nodes in a higher level. To do this, we increase u's height.
4. Repeat!
   - This process can only terminate once there are no active edges (so in step 1).

Algorithm pseudocode:

BASICPUSHRELABEL$(G, \Delta, T, h, (f, \ell))$
. **Assertion**: (f, ℓ) is (G, Δ, T)-valid.
. **While** $\exists$ active vertex u (i.e. $\ell(u) < h$ and ex$(u) > 0$)
. . PUSH/RELABEL(u).

PUSH/RELABEL(u)
. **If** $\exists$ admissible arc (u, v) (i.e. $r_f(u, v) > 0$, $\ell(u) = \ell(v) + 1$)
. . PUSH(u, v).
. **Else** RELABEL(u).

PUSH(u, v)
. **Assertion**: u is active and (u, v) is admissible.
. $\psi = \min\big(\text{ex}(u), r_f(u, v)\big)$
. Send ψ units of supply from u to v: $f(u, v) \leftarrow f(u, v) + \psi$

RELABEL(u)
. **Assertion**: u is active and there is no admissible arc (u, v)
. $\ell(u) \leftarrow \ell(u) + 1$.

3.3 Analysis

3.3.1 Input/Output

For the following lemmas, assume (f, ℓ) is initialized to a (G, Δ, T)-valid state at the beginning of the algorithm.

Lemma 3.1. *(f, ℓ) remains (G, Δ, T)-valid state throughout.*

Proof. To show that (f, ℓ) remains a (G, Δ, T)-valid state, it suffices to show that the two invariants are respected by every operation.

- Invariant 1: "$\forall u, v \in V \quad \ell(u) > \ell(v) + 1 \implies f(u, v) = c(u, v)$".

  We only change labels in RELABEL, so given a valid state, this invariant could only be broken in RELABEL. But if $f(u, v) < c(u, v)$ after calling RELABEL(u), then (u, v) was admissible before calling RELABEL(u), so PUSH would have been called instead of RELABEL, hence a contradiction. Therefore invariant 1 is maintained.

- Invariant 2: "if $\ell(u) \geq 1$, then $\text{ab}(u) = T(u)$":

  Because $\ell(u)$ increased from 0 to 1, that means that u must have been active at some point. This means that it must have had some non-zero excess ($\text{ex}(u) > 0$) at some point, but $\text{ex}(u) > 0$ only if u's sink was already full/fully absorbed ($\text{ab}(u) = T(u)$). Now because $\text{ab}(u)$ is never decreased, as we only PUSH excess out, $\text{ab}(u) = T(u)$ will be maintained throughout.

□

Lemma 3.2. *After termination, (f, ℓ) is a (G, Δ, T)-valid solution.*

Proof. Given a valid state, by the previous lemma we know that (f, ℓ) will remain a valid state at termination. We only terminate if there is no active vertex, so only when $\forall u \in V \ \ell(u) = h$ or $\text{ex}(u) = 0$, so if it terminates, we have the additional condition that makes (f, ℓ) a (G, Δ, T)-valid solution. □

Corollary 3.3. *Suppose after termination, $\forall u \in V \ \ell(u) < h$. Then f is a feasible flow satisfying (Δ, T).*

Proof. Note, from the above proof, that we only terminate when $\forall u \in V \ \ell(u) = h$ or $\text{ex}(u) = 0$, so if no node is at level h, then every node has zero excess. By proposition 2.2, we get then that f satisfies (Δ, T). Because we never PUSH flow that's greater than the capacity of an edge, $\forall (u, v) \in E$ $|f(u, v)| \leq c(u, v)$, so $\frac{|f(u,v)|}{c(u,v)} \leq 1$, giving us that f is also feasible.

□

These lemmas basically give us the confirmation that our algorithm 'makes sense'. Now, we need to work to see how long it takes.

3.3.2 Counting push and relabel calls

Note that $\ell(u)$ only increases, and only stops increasing one $\ell(u) = h$. This is an easy to see fact that will be relevant in bounding the time.

Lemma 3.4 (Relabel)**.** *The total number of* RELABEL *calls is at most* nh*.*

Proof. n vertices, and each is labeled at most h times. □

Bounding the number of PUSH calls is a bit more challenging. Note that whenever we call PUSH, we push either $\text{ex}(u)$ flow, or $r_f(u, v)$ flow- whichever is smaller. So, it makes sense to break bounding PUSH calls into two cases, those which "fill up edges" ($r_f(u, v) \leq \text{ex}(u)$), and those which don't ($\text{ex}(u) < r_f$):

Definition 3.5 (saturating PUSH)**.** This is a push where the edge (u, v) is "filled up". It happens only when $r_f(u, v) \leq \text{ex}(u)$ before the PUSH, and after the PUSH, $f(u, v) = c(u, v)$ (so $r_f(u, v) = 0$).

Definition 3.6 (unsaturating PUSH)**.** This is a push where the edge still has space; if a push isn't saturating, it is unsaturating. Any unsaturating $\text{PUSH}(u, v)$ will push $\text{ex}(u)$ units of flow.

Now we can bound each of these types of PUSH calls.

Lemma 3.5 (saturating PUSH)**.** *The total number of saturating* PUSH *is at most* mh*.*

Proof. Given some edge (u, v), let's count the number of saturating pushes we can make on it. Suppose we make our first saturating $\text{PUSH}(u, v)$. Note that $\ell(u) = \ell(v) + 1$.

Even though the edge is full right now, suppose at some future iteration, there is some reverse direction $\text{PUSH}(v, u)$ that empties out the edge somewhat. Note that v can only push to u if $\ell(v) = \ell(u) + 1$, so $\ell(v)$ must have increased 2 levels. Similarly, for there to be another saturating $\text{PUSH}(u, v)$ for us to count, $\ell(u)$ must also increase 2 levels.

By induction, there can be at most $\frac{h}{2}$ saturating $\text{PUSH}(u, v)$. There are m edges, and for each edge, you can have a saturating push in both directions, there at most $2m \times (\frac{h}{2}) = mh$ saturating pushes. □

Lemma 3.6 (unsaturating PUSH)**.** *The total number of unsaturating* PUSH *is at most* $O(mh^2)$*.*

Proof. Consider the following potential function, which is summing up the levels of each of the nodes with excess:

$$\Lambda = \sum_{v:\text{ex}(v)>0} \ell(v).$$

Because every unsaturating $\text{PUSH}(u, v)$ pushes exactly $\text{ex}(u)$ flow off from u, thereby making $\text{ex}(u) = 0$, $\text{PUSH}(u, v)$ could remove $\ell(u)$ from the sum. However, $\text{PUSH}(u, v)$ could also give

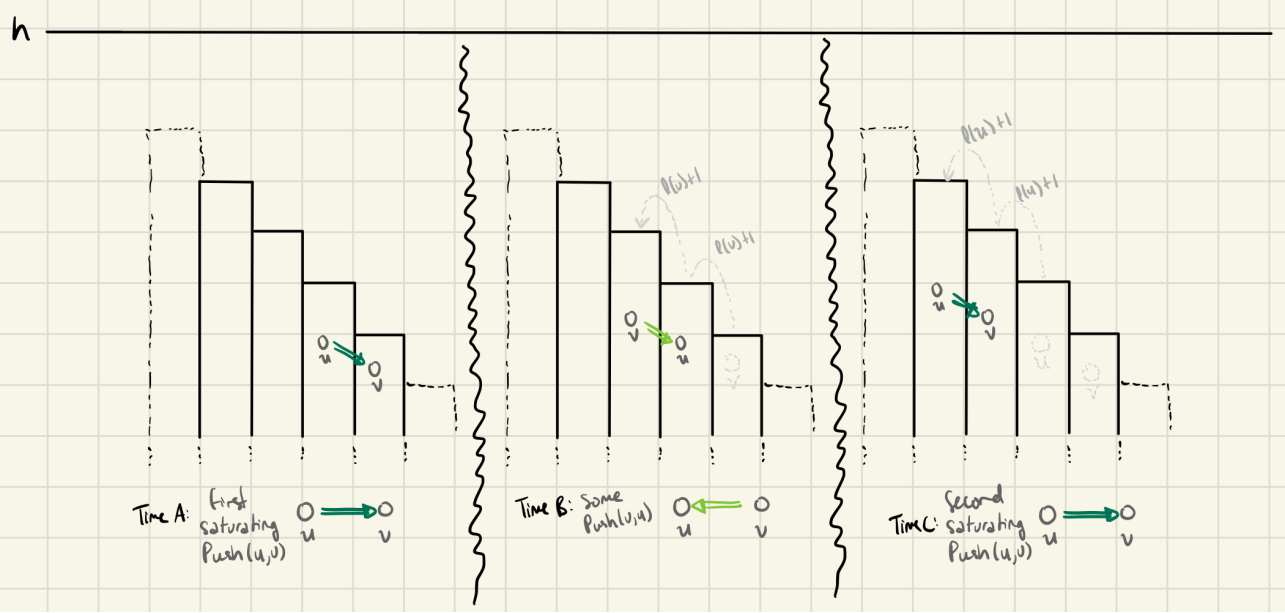


Figure 4: Proof of saturating-pushes bound visualization.

v excess, and thereby add $\ell(v)$ to the sum. However, $\ell(u) = \ell(v) - 1$, so at the very least, every unsaturating PUSH(u, v) decreases Λ by 1.

Consider a saturating PUSH(u, v). This won't take away the excess at u, but could potentially give v excess; therefore, it can't decrease Λ, but could potentially add $\ell(v)$ to Λ. $\ell(v)$ could be at most $h - 1$, so each saturating PUSH(u, v) adds at most $h - 1$ to the sum. There can only be mh saturating pushes (by lemma 3.5), so in one full run of the algorithm, saturating PUSH calls can increase Λ by at most $mh(h - 1)$.

Each RELABEL call increases Λ by at most 1, and the total number of RELABEL calls is at most nh (by lemma 3.3), so in one full run of the algorithm, RELABEL calls can increase Λ by at most nh. Finally, note that Λ can start off as at most nh, because each node can be of level h or less.

Combining all of these, we get that the total number of unsaturating PUSH will be at most $O(mh(h-1) + 2nh) = O(mh^2)$ □

3.3.3 Basic Implementation

We've bounded the number of PUSH and RELABEL calls. To finish understanding the algorithm and its performance, we need to specify how to implement the remaining two pieces quickly: getting an active node, and getting an admissible edge. Recall that we look for active nodes at the start of each iteration of the while loop, and given an active node, we decide to PUSH or RELABEL depending on whether that node has an admissible edge or not.

Active node: This is the easy one. Maintaining a simple queue will give us an active node in

$O(1)$ time.

Admissible edge: Given an active node u, the trivial solution would be to look through each of the edges connected to it to find an admissible edge; so every time you wanted to find an admissible edge for u, it would take $O(deg(u))$ time to find an admissible edge and PUSH, or conclude there aren't any and RELABEL.

We can make this faster however. Note that if an edge (u, v) is not-admissible, it is guaranteed to remain not-admissible until $\ell(u)$ changes.* So, we can maintain an ordered list for each node u with each of it's edges, separated into two types: (1) the edges that have been confirmed as not-admissible since $\ell(u)$ was last updated, and (2) the edges that haven't yet been checked since $\ell(u)$ was last updated. So, if you have u as your active node and are looking for an admissible edge, just look through the nodes in (2), transferring them to (1) if you see they aren't admissible, until you find an admissible edge. In a later iteration, if you end up with u as your active node again and $\ell(u)$ hasn't been updated yet, you just need to look through the, now shorter, list of edges in (2), to find an admissible edge.

So before RELABEL(u) is called again, we only spend $O(\deg(u))$ time scanning through u's neighbors to check for admissible edges, so we can charge $O(\deg(u))$ time to each RELABEL(u). Therefore, the total time on RELABEL is $\sum_u O(\deg(u))h = O(mh)$.

*Note: In this context, u and v see the edge (u, v) differently. It's possible u sees (u, v) as not admissible and v sees it as admissible, because they can both only push flow away from themselves (u sends flow (u, v) and v sends flow (v, u)). Think of the residual graph understanding on invariant 1 on page 8.

3.4 Total Running Time

Each PUSH takes only $O(1)$ time, and there are $O(mh)$ saturating PUSH calls, and $O(mh^2)$ unsaturating PUSH calls. We know from the previous section that we can charge RELABEL with $O(mh)$ total time, so combining everything, we get that the total running time is $O(mh^2)$.

4 Applications

4.1 Max-Flow (h = $n + 1$)

4.1.1 Inputs/Outputs:

We are presenting the "decision version" of the (s,t)-max flow problem. Note that if we can solve this problem, we can solve the standard (s,t)-max flow by a simple binary search.

Inputs: G, Δ, T

Output: Return one of the following:

- YES: with a feasible flow f satisfying (Δ, T)

- NO: with the cut S where $\Delta(S) - T(S) > \delta(S)$
  (certifying that no feasible flow satisfying (Δ, T) exists, by exercise 2.2)

4.1.2 Algorithms and Analysis:

Algorithm: Simply call BASICPUSHRELABEL on (G, Δ, T) with parameter $h = n + 1$.

Analysis: BASICPUSHRELABEL can end in one of two states: (1) no vertex u with $\ell(u) = h$, so $\text{ex}(V) = 0$, or (2) some vertex u with $\ell(u) = h$, so $\text{ex}(V) > 0$.

Case (1): By corollary 3.3, we have f as a feasible flow satisfying (Δ, T).

Case (2): We need to find a cut S that witnesses the failure of f to be a feasible flow, so we will construct a cut S where $\Delta(S) - T(S) > \delta(S)$.

First, notice that, as $h = n + 1$, there will be some level k where there are no vertices, i.e. $L_k = \{u \mid \ell(u) = k\} = \emptyset$ (each node can only be in one level). We will argue below that the cut S as the level cut $L_{>k} = \{u \mid \ell(u) > k\}$ works. Recall:

$$\underbrace{\Delta(S)}_{\text{initial mass}} = \underbrace{f_{out}(S)}_{\text{net flow out}} + \underbrace{\text{ab}_f(S)}_{\text{absorbed mass}} + \underbrace{\text{ex}_f(S)}_{\text{excess}}$$

Now, note the following:

- $f_{out}(S) = \delta(S)$: This is because all the nodes in S are more than one level above all the nodes in $V \setminus S$, so the flow out of S to $V \setminus S$ will be exactly the weight of the edges between S and $V \setminus S$.
- $\text{ab}_f(S) = T(S)$: All the nodes in S are above level 0, and must respect invariant 2, so they much all be fully absorbed.
- $\text{ex}_f(S) > 0$: The excess for the whole graph is always pooled up in level h (at termination, there are no active nodes, so any node not at level h has no excess), but we $L_h \subseteq S$. Since the excess of the graph is nonzero (no feasible flow), $L_h > 0$, so $\text{ex}_f(S) > 0$.

Plugging these in above, we get:

$$\Delta(S) > \delta(S) + T(S) \implies \Delta(S) - T(S) > \delta(S)$$

as desired.

4.2 Sparse Cut or Matching Embedding (h = $O(\frac{log(n)}{\phi})$)

4.2.1 Inputs/Outputs:

Inputs:

- $G = (V, E)$

- a cut (A, B) s.t. $vol(A) \leq vol(B)$
- a parameter ϕ

Outputs: Return one of the following:

- a cut S where $\Phi_G(S) < 1.1\phi$ (a sparse cut)
- a bipartite $(\deg_G)$-matching M of size vol(A) where $M \preccurlyeq^{\text{flow}} \frac{1}{\phi} \cdot G$
  - For all $a \in A$, $\deg_M(a) = \deg_G(a)$
  - For all $b \in B$, $\deg_M(b) \leq \deg_G(b)$

4.2.2 Algorithm and Analysis

Setting Δ and T: First, how to set Δ and T comes quite naturally from the matching we're trying to find. As we're trying to send $\deg(a)$ flow out for each node $a \in A$, and accept $\deg(b)$ flow in for each node $b \in B$, we set:

$$\Delta(u) = \begin{cases} \deg(u) & u \in A \\ 0 & u \notin A \end{cases} \text{ and } T(u) = \begin{cases} \deg(u) & u \in B \\ 0 & u \notin B \end{cases}$$

Scaling G: PUSH/RELABEL find flows, but only feasible flows. We're trying to embed M with congestion ϕ, so we can just scale down G to G_c, where $c(u, v) = 1/\phi$, and then tun the algorithm on G_c.

So, we can know that we will run PUSH/RELABEL on (G_c, Δ, T). What is left now is to set h. We will first go through an easy option, which gives us what we want but can be slow, and then will work through a more efficient, but somewhat more complicated example.

Option 1 (h = n + 1): Consider once more the two cases at the end of the algorithm, (1) where $\text{ex}(V) = 0$, and (2) where $\text{ex}(V) > 0$:

Case (1): Using the same reasoning from the analysis for Max-Flow (5.1.2), there exists a feasible flow f satisfying (Δ, T). f embeds M, and by flow-path decomposition, f corresponds to the (deg_G)-matching M that we want.

Case (2): Once more, using the reasoning for Max-Flow (5.1.2), we can easily find a cut S where $f_{out}(S) = \delta(S)$, so we get:

$$\begin{aligned} \frac{1}{\phi}\delta_G(S) = \delta_{G_c}(S) = f_{out}(S) &= \Delta(S) - \text{ab}(S) - \text{ex}(S) \\ &< \Delta(S) - T(S) \end{aligned}$$

Now, we just need to show that $\Delta(S) - T(S)$ is less that the volume of the cut (i.e. $\Delta(S) - T(S) \leq \min\{\text{vol}(S), \text{vol}(\bar{S})\}$), which will give us that $\frac{1}{\phi}\delta_G(S) < \Delta(S) - T(S) \leq \min\{\text{vol}(S), \text{vol}(\bar{S})\} \implies$

$\delta_G(S) \leq \phi \min\{\text{vol}(S), \text{vol}(\bar{S})\}$ (i.e. that S is a ϕ-sparse cut).

The total source $\Delta(S)$ in S is at most $\text{vol}(S \cap A)$, which is clearly less than or equal to $\text{vol}(S)$, so $\Delta(S) \leq \text{vol}(S)$. So, $\Delta(S) - T(S) \leq \text{vol}(S)$. Simply working through definitions again shows that $\Delta(S) - T(S) \leq \text{vol}(\bar{S})$ as well. This gives us that $\Delta(S) - T(S) \leq \min\{\text{vol}(S), \text{vol}(\bar{S})\}$, and by the above, that S is a ϕ-sparse cut.

Option 2 (h = $O(\frac{\log n}{\phi})$): So, we know that option 1 works, but it is slow. We will show here that we can set h to be considerably smaller, and get almost the same result. The only caveat is that in the case of finding a sparse cut, it might not be quite a ϕ-sparse cut, but it will be close enough 1.1ϕ-sparse. Consider once more the two end cases, (1) where $\text{ex}(V) = 0$, and (2) where $\text{ex}(V) > 0$:

Case (1): Exactly as above, we have a feasible flow f satisfying (Δ, T).

Case (2): In the case of h = n + 1, once we found an S where the flow out was saturated in G_c, so where $f_{out}(S) = \delta_{G_c}(S) = \frac{1}{\phi}\delta_G(S)$ in G, we were done. But notice that we are also done if we can find an S where the net flow is big enough:

$$f_{out}(S) \geq \frac{1}{\phi}\delta_G(S) - O(\min\{\text{vol}(S), \text{vol}(\bar{S})\})$$

so almost saturated, with some slack. So, as we've done the last few times, we just need to prove there exists a level cut where this property holds. The following lemma gives this to us.

Lemma 4.1. *There is a level k where $L_{\geq k} = \{u \mid \ell(u) \geq k\}$ and*

$$\begin{aligned} f_{out}(L_{\geq k}) &\geq \delta_{G_c}(L_{\geq k}) - \frac{\phi}{10} \cdot \min\{\text{vol}_{G_c}(L_{\geq k}), \text{vol}_{G_c}(L_{<k})\} \\ &= \frac{1}{\phi}\delta(L_{\geq k}) - \frac{1}{10} \cdot \min\{\text{vol}(L_{\geq k}), \text{vol}(L_{<k})\} \end{aligned}$$

Proof. Consider the graph G_{con} (G 'consecutive'), where $G_{con} \subset G_c$ contains only the edges between consecutive levels $\bigcup_i E_{G_c}(L_i, L_{i-1})$. The key point to note is that there is a level $k \in [h - \frac{50 \log n}{\phi}, h]$ where $\delta_{G_{con}}(L_{\geq k}) < \frac{\phi}{20}\text{vol}_{G_{con}}(L_{\geq k})$.

To see this, note that $\text{vol}_{G_{con}}(L_{\geq k-1}) \geq \delta_{G_{con}}(L_{\geq k}) + \text{vol}_{G_{con}}(L_{\geq k})$, so if the claim is not true, then $\text{vol}_{G_{con}}(L_{\geq h-i}) \geq (1 + \frac{\phi}{20})^i \geq n^4$ when $i \geq \frac{50 \log n}{\phi}$, which would give us a contradiction. This is why we need $h \geq \frac{100 \log n}{\phi}$.

Using this argument as a ball growing argument from both sides, and using the k with the level cut with smaller cut size, we get that there is a k where:

$$\delta_{G_{con}}(L_{\geq k}) < \frac{\phi}{20} \cdot \min\{\text{vol}_{G_{con}}(L_{\geq k}), \text{vol}_{G_{con}}(L_{<k})\} \leq \frac{\phi}{20} \cdot \min\{\text{vol}_{G_c}(L_{\geq k}), \text{vol}_{G_c}(L_{<k})\}.$$

Now because all the edges that skip levels are saturated, we get that:

$$f_{out}(L_{\geq k}) \geq \delta_{G_c}(L_{\geq k}) - 2\delta_{G_{con}}(L_{\geq k})$$

which in turn implies that:

$$f_{out}(L_{\geq k}) = \frac{1}{\phi}\delta(L_{\geq k}) - \frac{1}{10}\min\{\text{vol}(L_{\geq k}), \text{vol}(L_{<k})\}$$

as was desired. □

4.3 Balanced Sparse Cut or Matching Embedding With Fake Edge

So we've now got a powerful way to use PUSH/RELABEL for finding a sparse cut, or a (deg_G)-matching embedding. Note that this is really just the step of a matching player in a cut-matching game. We will now show how to use PUSH/RELABEL for finding a balanced sparse cut, which can be used for finding balanced sparse cuts via the cut matching framework, and in turn, for computing expander decomposition.

Inputs/Outputs: **Inputs**:

- $G = (V, E)$
- a cut (A, B) s.t. $vol(A) \leq vol(B)$
- a parameter ϕ
- a parameter β

Outputs: Return one of the following:

- a cut S where $\Phi_G(S) < 2\phi$ and $\text{vol}(S), \text{vol}(\bar{S}) > \beta$ (a balanced sparse cut)
- a matching M' and a set of fake edge F where
  - $M = M' \cup F$ is a bipartite $(\deg_G)$-matching between A and$B$ of size $\text{vol}(A)$
  - for all $a \in A$, $\deg_M(a) = \deg_G(a)$
    - for all $b \in B$, $\deg_M(b) \leq \deg_G(b)$
    - $|F| \leq \beta$
  - $M \preccurlyeq^{\text{flow}} \frac{1}{\phi} \cdot G$

Algorithm: We can use the same algorithm as last time (5.2.2), so calling BASICPUSHRELABEL on (G_c, Δ, T) with parameter $h = O(\frac{\log n}{\phi})$. The difference now, is that we will change what we do based on the threshold of $\text{ex}(V)$ at the end, from 0 to β.

Case (1): Suppose $\text{ex}(V) \leq \beta$. This means that you were close to embedding the matching, but there were β units of flow that doesn't go from source to sink. So, using the same argument as the last few times with corollary 3.3, via flow-path decomposition, f corresponds to the $(\deg_G)$-matching M' of size $\geq \text{vol}(A) - \beta$.

Case (2): Suppose otherwise, that $\text{ex}(V) > \beta$. The excess can only be on L_h, where the excess for each node in L_h can be at most the degree, so we have that $\text{vol}(L_h) \geq \text{ex}(V) > \beta$. Moreover, $S \supseteq L_h$, so $\text{vol}(S) > \beta$. Similarly, we can show that $\text{vol}(L_0) > \beta$, so $\text{vol}(\bar{S}) > \beta$ as well.

5 Faster Implementation With Dynamic Trees

Recall from section 3.4 that the overall running time, using the basic implementation we discussed, is: $O(mh^2)$. The bottleneck here are the unsaturating PUSH calls, as everything else costs $O(mh)$.

Our goal here is to improve $O(mh^2)$ to $O(mh \log n)$, by making (almost) all pushes saturating, so only $O(mh)$ operations. With this improvement, our running time for max-flow would improve from $O(mn^2)$ to $O(mn \log n)$, and for finding a ϕ-sparse cut would improve from $O(m \log^2(n)/\phi^2)$ to $O(m \log^2(n)/\phi)$.

To implement this strategy, we will use **top tree** data structure.[2] We will discuss their use from a high level perspective, enough so that we can use it to improve PUSH/RELABEL.

5.1 Top Tree Data Structure

Suppose you have a collection of rooted trees $\mathcal{T} = \{T_1, \ldots, T_k\}$, where each edge e is associated with value $\mathsf{val}(e)$. Morally speaking, the **top tree data structure** allows you to maintain 'anything' you'd like to dynamically on these trees. Most likely, it can handle anything you'd like to throw at it. Here are some example of the types of things it can handle, quite surprisingly, in $O(\log n)$ time:

Updates:

- link(u, v): add an edge (u, v) to combine two trees (as long as $\mathcal{T}$ remains sets of rooted tree)
- cut(u, v): delete edge (u, v) to cut a tree into two
- change-path(x, y, v): for each $e \in P_{xy}$ in the unique path from x to y, set $\mathsf{val}(e) \leftarrow \mathsf{val}(e) + v$.
- change-root(u): make u a root
- many many more..

Queries:

- root(u): return the root of $T \ni u$
- min-path(x, y): return $\min_{e \in P_{xy}} \mathsf{val}(e)$. (Can return max or sum too)
- min-subtree(x): return $\min_{e \in T_x} \mathsf{val}(e)$ where T_x is the subtree rooted at x. (Can return max or sum too)
- size-subtree(u): return size of T_u
- meet(x, y, z): find the unique "meeting vertex" of $P_{xy} \cap P_{yz} \cap P_{zx}$.
- many many more..

[2] Top trees go by many names in the literature, such as dynamic trees, link-cut trees, topological trees, etc. Read more about the detail of how they work here: https://arxiv.org/pdf/cs/0310065.pdf

A warning to keep in mind though is that, while great in theory, top trees seem to be a lot slower in practice.

Question 5.1 (Research Question). *Is there some problem on trees that is easy in the static setting, but hard on the dynamic setting?*

5.2 Top Tree PUSH/RELABEL Implementation

5.2.1 Rephrased Invariants

Start off with imagining each vertex as the root of it's own tree in L_0, so $T_u = \{u\}$ for each node u. We want to maintain the following invariants:

1. Each vertex u is incident to at most one tree edge (u, v) where $\ell(v) = \ell(u) - 1$. In other words, every vertex can have only one parent in $\mathcal{T}$ in another level.
2. Only active vertices can be the root of a tree.

- The $\mathsf{val}(u, v)$ for each edge in $\mathcal{T}$ is equal to the residual capacity of the edge $r_f(u, v)$. This isn't quite an invariant, as it is can be implicitly maintained very quickly:
  - When we call link(u, v), we always set change-value$(u, v, r_f(u, v))$.
  - When we call cut(u, v), we always set $f(u, v) \leftarrow c(u, v) - \mathsf{val}(u, v)$.

  However, it's an important thing to keep in mind when understanding how this works.

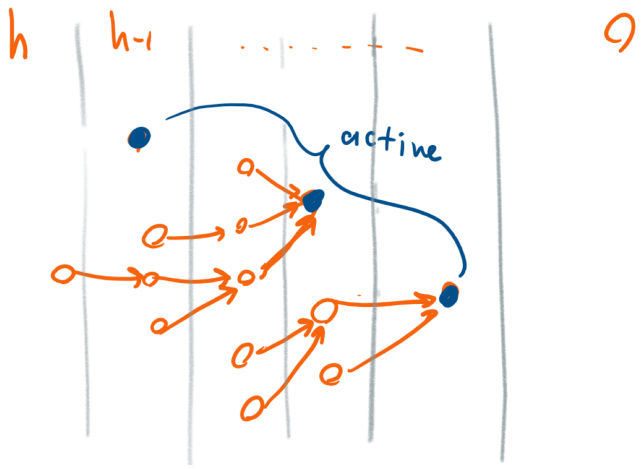


Figure 5: Top tree invariants visualization.

5.2.2 Updated Algorithm

Can start off with initializing $f \equiv 0$ and $\ell \equiv 0$, so every node is a root of its own tree.

TREEPUSHRELABEL$(G, \Delta, T, h, (f, \ell))$
. **Assertion**: (f, ℓ) is (G, Δ, T)-valid.
. **While** $\exists$ active tree root vertex u
. . PUSH/RELABEL(u).

PUSH/RELABEL(u)
. **If** $\exists$ admissible arc (u, v)
. . TREEPUSH(u, v).
. **Else** RELABEL(u).

TREEPUSH(u, v)
. **Assertion**: u is active and (u, v) is admissible.
. link(u, v)
. **While** $u \neq$ root(u) and ex$(u) > 0$
. . $\psi = \min\left(\text{ex}(u), \text{min-path}(u, \text{root}(u))\right)$
. . Send ψ units of supply from u to root(u): change-path$(u, \text{root}(u), -\psi)$
. . for all $e \in P_{u,\text{root}(u)}$ where $\mathsf{val}(e) = r_f(e) = 0$, perform cut$(e)$.
//Exercise: implement this using $O(1)$ top-tree-operations per cut.

RELABEL(u)
. **Assertion**: u is active and there is no admissible arc (u, v)
. for all tree-children c of u, cut(c, u).
. $\ell(u) \leftarrow \ell(u) + 1$.

5.2.3 Analysis

We can use the same implementation details for getting an active vertex of an admissible edge quickly: a queue for active edges, and an ordered list for edges. The changes using top trees are with how we PUSH.

In TREEPUSH, we can have lots of intermediate pushes until we get to the root. Lets call these intermediate push operations PUSH. We can define **saturating/unsaturating** PUSH as we did before: if $\psi = \text{min-path}(u, \text{root}(u))$, then this is a **saturating** PUSH, and otherwise, it is **unsaturating**.

The updated running time for RELABEL and this saturating PUSH follow very directly from the analysis in 3.4, except we update for the $\log n$ operations with top trees. So:

- RELABEL is called $O(nh)$ times, so total time is $O(mh \log n)$
  - We charge $O(\deg(u))$ to RELABEL(u) for the cost of scanning through list of neighbors only once.
  - Also, for each RELABEL, we call "cut" $O(\deg(u))$ tree operations, each of cost $O(\log n)$.
- There are $O(mh)$ saturating PUSH calls, so the total time is $O(mh \log n)$

It just remains now to bound the previous bottleneck, unsaturating PUSH calls. First, note that #unsaturating PUSH $\leq$ #links, but #links $\leq$ #cuts + $n - 1$. Also, note that #cuts $\leq$ #edges that become saturated + #edges scanned during RELABEL. But, because #saturating pushes $\leq mh$ and #edges scanned during RELABEL $\leq \sum_u \deg(u)h \leq 2mh$, we get that the total cost of unsaturating PUSH is $O(\#\text{link} \cdot \log n) = O(mh \log n)$.

Therefore, the overall running time is $O(mh \log n)$.

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 22: Local and Dynamic Push-Relabel Flow Algorithms for Local Sparse Cuts and Expander Pruning

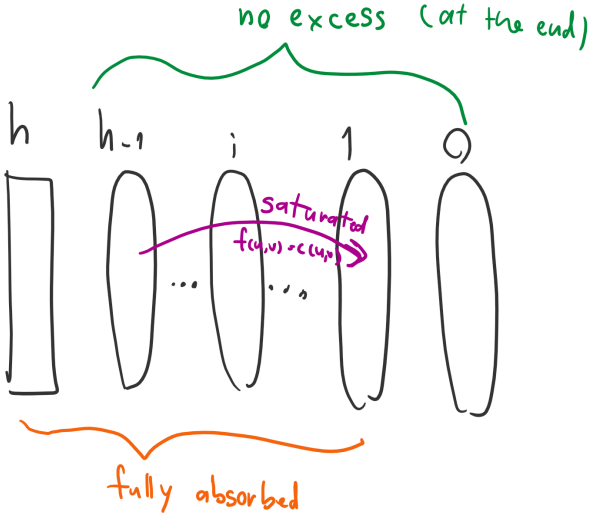
November 18th, 2021

Instructor: Thatchaphol Saranurak

Scribe: Nikhil Shagrithaya

1 Recap

Recall that we discussed PUSH/RELABEL Flow in the last class, where given the parameters G, the input graph, Δ, T, the sink and source functions respectively, and parameter h, the number of levels an active vertex can attain, and additionally, given a (G, Δ, T)-valid state (f, ℓ), we were required to output a (G, Δ, T)-valid solution.



Definition 1.1. We say that (f, ℓ) is a (G, Δ, T)**-valid state** if

1. If $\ell(u) > \ell(v) + 1$, then (u, v) is saturated from u to v (i.e. $f(u, v) = c(u, v)$)
2. If $\ell(u) \geq 1$, then u is fully absorbed (i.e. $\text{ab}(u) = T(u)$).

Definition 1.2. We say that (f, ℓ) is a (G, Δ, T)**-valid solution** if additionally we satisfy the following: if $\ell(h) < h$, then u has no excess (i.e. $\text{ex}(u) = 0$)

The time taken by this algorithm is equal to $O(mh^2)$, by using the basic implementation. However, by using the top trees data structure, we can bring down the running time to $O(mh \log n)$.

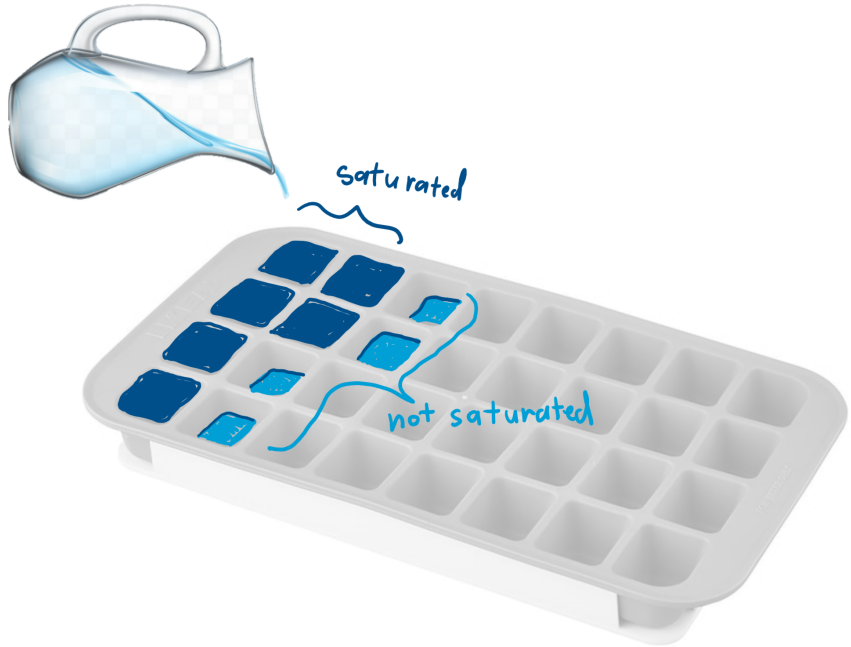
There are many applications of push-relabel, some of them being:

- **Exact Max flow:** Actually, this is a "decision version" of Max Flow. But for convenience, whenever we call BASICPUSHRELABEL with $h = n + 1$, I will just say that we call an "Exact Max Flow" subroutine. The key property here is that if the algorithm returns a cut S, then $f_{out}(S) = \delta(S)$.
- **ϕ-sparse cut or matching embedding:** We set $h = \Theta(\log(n)/\phi)$ here. The key property is: if the algorithm returns a cut S, then $f_{out}(S) \geq \delta(S) - \frac{\phi}{100} \min\{\text{vol}(S), \text{vol}(\overline{S})\}$.

In this lecture, we will look at **Local and Dynamic** Push-Relabel Techniques.

2 When can flow algorithms be local? The Ice Tray Analogy

Imagine pouring water into an ice tray:



The way the water flows through the tray is intuitively **local** in nature. The time taken to pour out the water is proportional to only the **volume of the source water, not the size of the ice tray.**

Let's abstract out the reason why it is local. Every node is a "large enough" sink. (Otherwise, imagine that all cells in the ice tray almost cannot absorbed any flow, except that furthest cell. Then, the water would need to "read" the whole ice tray to find a sink.) Moreover, every node forwards flow to the adjacent node only when its sink is saturated. (This is how PUSH/RELABEL works!)

3 Local PUSH/RELABEL

With the above intuition in mind, let's try to analyze PUSH/RELABEL when every node is a "large enough" sink. Formally, we consider PUSH/RELABEL in the following setting.

Parameters: G, Δ, T, h where $T(u) = \deg(u)$ for all $u \in V$ where $\deg(u)$ is the unweighted degree of u. Additionally, we think of all quantities as being integral:

- integral source function $\Delta : V \to \mathbb{Z}_{\geq 0}$
- integral edge capacities $c : E \to \mathbb{Z}_{>0}$

We also require the same guarantee: given a (G, Δ, T)-valid state (f, ℓ), output a (G, Δ, T)-valid solution. There are nice applications in this specific setting.

We will show a variant of PUSH/RELABEL, LOCALPR, that solves the above flow problem in local time. Specifically, the running time is $O(\Delta(V)h)$, so it is only proportional to total source, not graph size. This was first shown in [HRW'17].[1]

3.1 Algorithm Description

We will require one additional invariant: that $\text{ex}(u) \leq \deg(u)$ for all $u \in V$, unless that excess is initially placed at u. To maintain this, we edit the basic PUSH/RELABEL algorithm using the red part below.

LOCALPR$(G, \Delta, T, h, (f, \ell))$
. **Assertion**: (f, ℓ) is (G, Δ, T)-valid.
. **While** $\exists$ active vertex u (i.e. $\ell(u) < h$ and $\text{ex}(u) > 0$)
. . Let v be the active vertex with smallest $\ell(v)$.
. . PUSH/RELABEL(u).

PUSH/RELABEL(u)
. **If** $\exists$ admissible arc (u, v) (i.e. $r_f(u, v) > 0$, $\ell(u) = \ell(v) + 1$)
. . PUSH(u, v).
. **Else** RELABEL(u).

PUSH(u, v)
. **Assertion**: u is active, (u, v) is admissible, and $\text{ex}(v) = 0$
. $\psi = \min\left(\text{ex}(u), r_f(u, v), \deg(v) - \text{ex}(v)\right)$ //Observe that $\psi \geq 1$. Why?
. Send ψ units of supply from u to v: $f(u, v) \leftarrow f(u, v) + \psi$

RELABEL(u)
. **Assertion**: u is active and there is no admissible arc (u, v)
. $\ell(u) \leftarrow \ell(u) + 1$.

Observe how the excess on each node u evolves. Initially, it is possible that $\text{ex}(u) > \deg(u)$. Also, $\text{ex}(u)$ can only decrease until $\text{ex}(u) = 0$. The vertex u keeps pushing initial excess and u never receives additional mass until $\text{ex}(u) = 0$. Only then, $\text{ex}(u)$ can increase, but $\text{ex}(u) \leq \deg(u)$ always holds.

[1] They call this variant a *Unit-Flow* algorithm https://arxiv.org/pdf/1704.01254.pdf

3.2 Running Time

Lemma 3.1. *The total time spent on* RELABEL *is* $O(\Delta(V)h)$.

Proof. According to the algorithm, we only relabel vertices when their level is at least 1. But because each $u \in L_{\geq 1}$ is fully absorbed, i.e. $\text{ab}(u) = \deg(u)$, whenever we call RELABEL(u), we charge the cost of $O(\deg(u))$ to absorbed mass at u, each of $O(1)$. Recall the original analysis of PUSH/RELABEL: we charge $O(\deg(u))$ time to each RELABEL(u). This is the time for finding an admissible edge by scanning through list of neighbors.

How much time is each unit of absorbed mass charged? It is at most h. Since total absorbed mass is trivially at most $\Delta(V)$. The total time is then $O(\Delta(V)h)$.

To summarize the above argument in a more algebraic way, we have the total cost spent on RELABEL is:

$$\begin{aligned}\sum_{u \in L_{\geq 1}} O(1) \cdot \deg(u) \cdot h &= O(1) \cdot \text{ab}(L_{\geq 1}) \cdot h \\ &\leq O(1) \cdot \Delta(V) \cdot h\end{aligned}$$

□

Now we analyze the number of times the PUSH algorithm is called.

Lemma 3.2. *The total number of* PUSH *is at most* $2\Delta(V)h$. *So the total time spent on* PUSH *is* $O(\Delta(V)h)$.

Proof. Consider the potential function

$$\Lambda = \sum_v \text{ex}(v)\ell(v).$$

Each PUSH decreases Λ by at least $\psi \geq 1$, as ψ units of excess is moved from level i to $i-1$. Moreover, each RELABEL(u) increases Λ by at most $\text{ex}(u)$. If $\text{ex}(u) > \deg(u)$, all these excess at u must be initial excess of u. We charge 1 to each unit of this initial excess. Each unit of mass is charged, as initial excess, at most h. If $\text{ex}(u) \leq \deg(u)$, then $\text{ex}(u) \leq \text{ab}(u)$. We charge 1 to each unit of the absorbed mass. Each unit of mass is charged, as absorbed mass, at most h.

So in total, RELABEL increases Λ by at most $\Delta(V) \times 2h$. So #PUSH $\leq 2\Delta(V)h$.

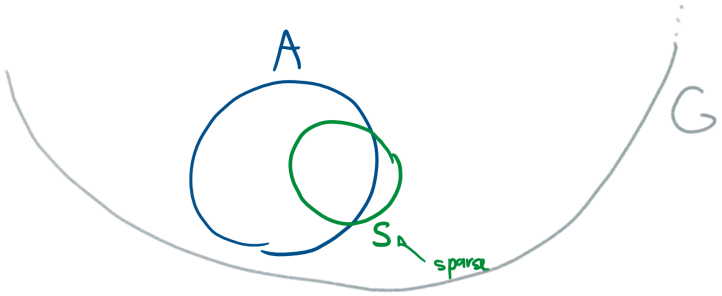
□

4 Local Sparse Cut: Application of Local PUSH/RELABEL

Consider this situation: Given a vertex set $A \subset V$ with a promise that there is some ϕ-sparse cut S^* "close to" A. More formally,

$$\text{vol}(S^* \cap A) \geq \sigma \text{vol}(S^*)$$

for some $\sigma > 0$.



Our objective is to find a sparse cut, but we don't want to read the whole graph.

The idea is that we call LocalPR on input (G_c, Δ, T, h), where G_c is the the graph G with edge capacity $c(e) = 1/\phi$ for all $e \in E(G)$.

$$\Delta(u) = \begin{cases} \frac{1}{\sigma} \deg_G(u) & \text{if } u \in A \\ 0 & \text{if } u \notin A \end{cases}$$

and

$$T(u) = \deg_G(u) \forall u$$

$h = \Theta(\frac{\log n}{\phi})$ similar to when we use Push/Relabel for finding ϕ-sparse cuts.

Can LocalPR terminate with no excess? No, because $\Delta(S^*) > \delta_{G_c}(S^*)$:

$$\Delta(S^*) = \frac{1}{\sigma} \text{vol}_G(S^* \cap A) \geq \text{vol}_G(S^*)$$

and

$$\delta_{G_c}(S^*) = \frac{1}{\phi} \delta_G(S^*) < \text{vol}_G(S^*)$$

So LocalPR terminates with some excess, i.e., some vertex u has label $\ell(u) = h$.

There is a level cut $L_{\geq k} = \{u \mid \ell(u) \geq k\}$ where

$$\Phi_G(L_{\geq k}) \leq 1.1\phi.$$

Exercise: This follows from exactly the same analysis as in previous class. Note: We also have that $L_{\geq k}$ is small: $\text{vol}(L_{\geq k}) \leq \Delta(V) = \text{vol}(A)/\sigma$. The time take is $O(\Delta(V)h) = O(\frac{\text{vol}(A)}{\sigma\phi} \log n)$.

Exercise: show that $h = \Theta(\frac{\log(\text{vol}(A)/\sigma)}{\phi})$ works too.

To conclude:

Theorem 4.1. *Let $A \subset V$ be a vertex set where $\text{vol}(A) < \sigma \text{vol}(V)$ for some $\sigma > 0$. (Why we need this?).*
Suppose there is S^* where $\Phi_G(S^*) < \phi$ and $\text{vol}(S^* \cap A) \geq \sigma \text{vol}(S^*)$.
Then, we can find a cut S where $\Phi_G(S) < 1.1\phi$ and $\text{vol}(S) \leq \text{vol}(A)/\sigma$ in time $O(\frac{\text{vol}(A)}{\sigma\phi} \log(\text{vol}(A)/\sigma))$

4.1 Big picture: Local Clustering Algorithms

So far, we have seen 2 local algorithms for finding "clusters" in graphs. Localized PUSH/RELABEL for Local sparse cut. (Today). Localized Ford-Fulkerson for Local cut of size k. There is another class of local algorithms based on random walks or localized PageRank.

Many more interesting questions about local clustering algorithm:

- Local densest subgraph?
- Local graph with many triangles?
- On hypergraphs?

5 Expander Pruning: A Stronger Robustness Characterization of Expanders

The question we ask here is: can you "repair" an expander?

We formally describe the problem that captures "repairing" expanders:

Problem 5.1 (One-batch Expander Pruning)**.** Given a ϕ-expander $G = (V, E)$ and an edge set $D \subset E$, let $G' = G \setminus D$. We need to find a *prune set* $P \subset V$ such that:

- $G'[V \setminus P]$ is a $\Omega(\phi)$-expander, and
- $\mathrm{vol}_G(P) = O(|D|/\phi)$.

We will show how to solve this problem. This gives a *stronger robustness characterization of expanders*. Recall the characterization from the first lecture: after d edge deletions, there is P with $\mathrm{vol}(P) = O(d/\phi)$ where $G'[V \setminus P]$ is connected, and P contains all small disconnected components.

In this stronger characterization, we will show that after d edge deletions, there is a P with $\mathrm{vol}(P) = O(d/\phi)$, where $G'[V \setminus P]$ is still $\Omega(\phi)$-expander.

In fact, we can show something even stronger: we can repair an expander under sequence of deletions **at all times.**

Problem 5.2 (Expander Pruning)**.** Given a ϕ-expander $G = (V, E)$ and an online sequence of edge deletions $D = (e_1, e_2, \ldots, e_d)$, let G_i be the graph G after time i (after deleting $e_1, \ldots, e_i$). We need to maintain the prune set $P \subset V$ where P_i denote the set P after time i with the following properties:

- $G_i[V \setminus P_i]$ is a $\Omega(\phi)$-expander.
- $\mathrm{vol}_G(P_i) \leq 8i/\phi$, and $P_{i-1} \subseteq P_i$ (it is an incremental set)

Expander pruning is a very important tool for almost all dynamic algorithms based on expanders. Expanders are great, but everything we have talked so far is for static graphs. Expander pruning allows us to exploit the power of expanders in the dynamic setting.

6 One-batch Expander Pruning

In this lecture, we only consider the easier version of the problem (the one-batch version). Once we understand the technique for this version, it easily extends to the full dynamic version.

6.1 Terminology

Consider an unweighted graph G and a vertex set $U \subseteq V$. (Everything that follows can be extended to capacitated graphs.) Let $G\{U\}$ denote the following **induced subgraph with boundary vertices**. Start with the induced subgraph $G[U]$ and then for each **boundary edge** $e = (u, v) \in E(U, V \setminus U)$ with endpoint $u \in U$, create a new vertex x_e and add the edge (x_e, u) to $G\{U\}$.

Let $\partial_G \langle U \rangle = V(G\{U\}) \setminus U$ be the set of **boundary vertices**. For each boundary vertex $x_e \in \partial_G \langle U \rangle$ where $e = (u, v)$ and $v \in U$, we say that v is the **partner** of x_e. $G\{U\}$ is a **near ϕ-expander** if:

$$\delta_{G\{U\}}(S) \geq \phi \text{vol}_{G\{U\}}(S)$$

for all $S \subseteq U$ where $\text{vol}(S) \leq \text{vol}(U)/2$.

Intuitively, this means that $G[U]$ is almost an expander, but we count boundary edges as well. Note that, for any U, we have $\deg_{G\{U\}}(u) = \deg_G(u) \forall u$ and $\text{vol}_{G\{U\}}(S) = \text{vol}_G(S) \forall S \subseteq U$. Therefore, degree and volume remain fixed even when we consider different U.

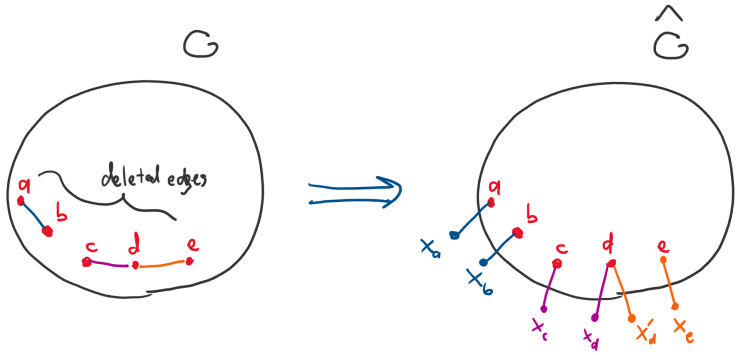
6.2 Reformulation

If we can solve the following problem, then we can solve the one-batch expander pruning problem:

Problem 6.1 (Pruning near expanders)**.** Let $G = (V, E)$ and $U \subset V$ where $G\{U\}$ is a near ϕ-expander. Compute a *prune set* $P \subseteq U$ and $U' = U \setminus P$ such that:

- $G[U']$ is a $\Omega(\phi)$-expander
- $\text{vol}(P) \leq O(|\partial_G \langle U \rangle|/\phi)$.

Intuitively, we want to make a near-expander an expander by pruning small part of it. Given a ϕ-expander $G = (V, E)$ and a deleted edge $D \subset E$, create a graph $\widehat{G}$ as follows:



Observe that $\widehat{G}\{V\}$ is a near ϕ-expander because for all $S \subseteq V$ where $\text{vol}(S) \leq \text{vol}(V)/2$, we have

$$\delta_{\widehat{G}\{V\}}(S) \geq \delta_G(S) \geq \phi \text{vol}_G(S) = \text{vol}_{\widehat{G}\{V\}}(S)$$

We also have $\text{vol}(P) \leq O(|\partial_{\widehat{G}} \langle V \rangle|/\phi) = O(|D|/\phi)$, and $\widehat{G}[V \setminus P] = G'[V \setminus P]$ where $G' = G \setminus D$

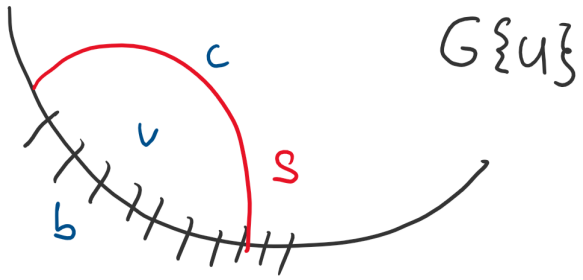
6.3 Useful Structure

Lemma 6.2. *Suppose that $G\{U\}$ is a near ϕ-expander, but $G[U]$ is not a $\phi/6$-expander. Then, there is $S \subset U$ where*

$$|E(S, U \setminus S)| \leq |E(S, V \setminus U)|/5.$$

Define the following quantities:

- $c = |E(S, U \setminus S)|$ (cut edges)
- $b = |E(S, V \setminus U)|$ (boundary edges)
- $v = \text{vol}(S)$



We have:

$$c < \frac{1}{6}\phi v \text{ and } c + b \geq \phi v$$

and so $b \geq \frac{5}{6}\phi v$ and so

$$c \leq b/5.$$

Note that, if $G[U]$ is not a $\phi/1000$-expander, we would have $c \leq b/999$. You should think of $c \ll b$ for intuition.

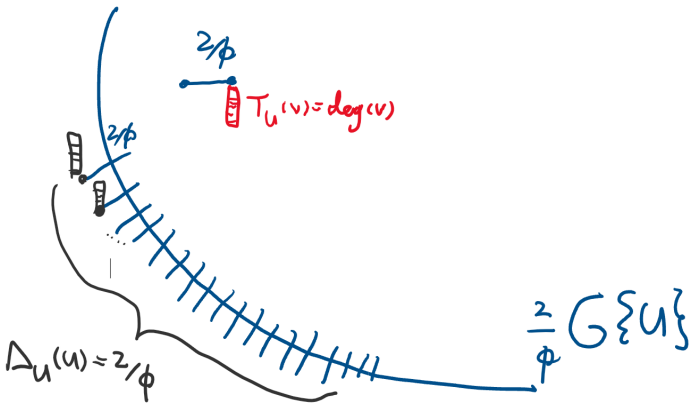
6.4 Certifying that a Near-expander is an Expander

The **U-boundary source function** $\Delta_U : V(G\{U\}) \to \mathbb{R}_{\geq 0}$ is defined as:

$$\Delta_U(u) = \begin{cases} 1 & \text{if } u \in \partial_G \langle U \rangle \\ 0 & \text{if } u \in U \end{cases}$$

The **U-sink function** $T_U : V(G\{U\}) \to \mathbb{R}_{\geq 0}$ is defined as

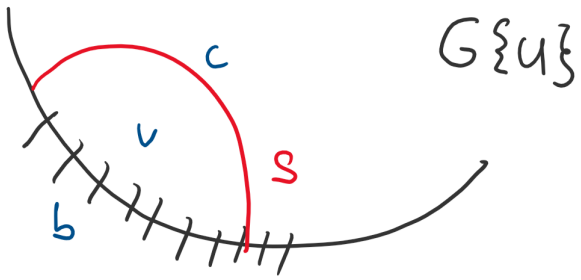
$$T_U(u) = \begin{cases} 0 & \text{if } u \in \partial_G \langle U \rangle \\ \deg_G(u) & \text{if } u \in U \end{cases}$$



Lemma 6.3. *Let $G\{U\}$ be a near ϕ-expander. If there is a feasible flow f in $\frac{2}{\phi}G\{U\}$ satisfying $(\frac{2}{\phi}\Delta_U, T_U)$, then $G[U]$ is a $\phi/6$-expander.*

In words, we can certify that $G[U]$ is already an expander, if there is a feasible flow f in $\frac{2}{\phi}G\{U\}$ where **every** boundary edge on sends flow at its full capacity, and each non-boundary u vertex absorbed at most $\deg(u)$.

Proof. Suppose for contradiction, there is an $S \subset U$ where $\mathrm{vol}(S) \le \mathrm{vol}(U)/2$



where $c \le b/5$. (Think of $c \ll b$ for intuition.)

Now, if f is feasible, we have

$$\underbrace{\frac{2}{\phi}c}_{\text{flow out}} \ge \underbrace{\frac{2}{\phi}b}_{\text{initial mass}} - \underbrace{v}_{\text{absorbed mass}}$$

But the absorbed mass is at most $v \le \frac{1}{\phi}(b+c)$. For intuition, if $c \ll b$, the absorbed mass is essentially at most half of the initial mass. So half of flow must go out, which means that $c \ge b/2$ (why?). But this is a contradiction because $c \ll b$.

More formally, we have $v \le \frac{1}{\phi}(b+c) \le \frac{6}{5\phi}b$ and so

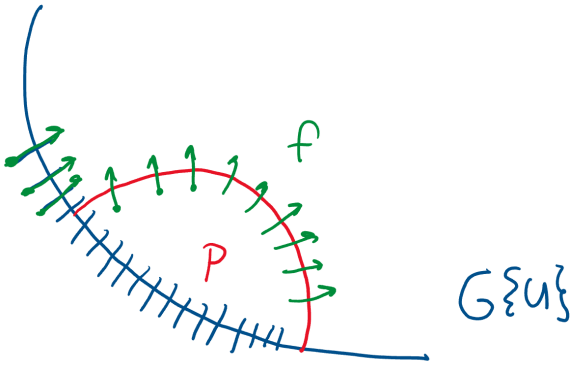
$$\frac{2}{\phi}c \ge \frac{4}{5\phi}b \iff c \ge \frac{2}{5}b$$

which contradicts the fact that $c \le b/5$. □

6.5 Warm up: Slow Algorithm by Solving Exact Max Flow

Suppose we call BASICPUSHRELABEL on $(\frac{2}{\phi}G\{U\}, \frac{2}{\phi}\Delta_U, T_U)$ with $h = n + 1$, and also suppose there is no excess. Then we get a feasible flow and so $G[U]$ itself is a $\phi/6$-expander. Then we are done, with $P = \emptyset$.

But what if there is excess? In this case, BASICPUSHRELABEL returns a cut $P \subseteq V(G\{U\})$. Let f be the preflow in $\frac{2}{\phi}G\{U\}$ maintained byBASICPUSHRELABEL. Recall that $f_{out}(P) = \delta_{\frac{2}{\phi}G\{U\}}(P)$. For $u \notin P$, we have $\mathrm{ex}(u) = 0$.



Quick question: For each boundary vertex $x_e \in P$, if its unique adjacent vertex $u \in P$, then $x_e \in P$ too. We will show that we can just return P as our prune set.

Lemma 6.4. *Let $U' = U \setminus P$. Then $G[U']$ is a $\phi/6$-expander.*

Proof. Consider the preflow f computed by BASICPUSHRELABEL, and let f' be obtained from f by restricting to $G\{U'\}$. Observe that f' is a feasible flow in $\frac{2}{\phi}G\{U'\}$ satisfying $(\frac{2}{\phi}\Delta_{U'}, T_{U'})$. Note that $G\{U'\}$ is a near ϕ-expander (as $G\{U\}$ is a near ϕ-expander). So, $G[U']$ is a $\phi/6$-expander by Lemma 6.3.

□

Lemma 6.5. $\mathrm{vol}_G(P) \le O(|\partial_G \langle U \rangle|/\phi)$

Proof. For each $u \in P$, we have $\mathrm{ab}(u) = T(u) = \deg_G(u)$. So

$$\mathrm{vol}_G(P) = \mathrm{ab}(P) \le \frac{2}{\phi}\Delta_{U'}(V(G\{U\}) = \frac{2}{\phi}|\partial_G \langle U \rangle|.$$

□

To summarize: our goal is a partition (P, U') of U such that P is not too big, and there is a feasible flow in $\frac{2}{\phi}G\{U'\}$ where every U'-boundary edge can send flow at its full capacity. This implies that $G[U']$ is a $\phi/6$-expander.

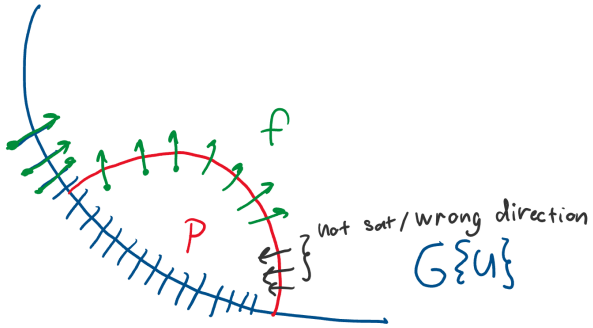
Exercise 6.6. Show that every U'-boundary edge can send flow by at least 0.9 fraction of its capacity, we can still conclude that $G[U']$ a $\Omega(\phi)$-expander.

Actually, suppose there is a subroutine that returns (P, U') and only guarantees that every U'-boundary edge can send flow by at least α fraction of its capacity for some small $\alpha > 0$, show how to adjust the algorithm to certify that $G[U']$ is a $\Omega(\alpha\phi)$-expander. *Hint: Work with $\frac{2}{\alpha\phi}G\{U\}$ instead.*

Exercise 6.7. Instead of calling BASICPUSHRELABEL, we can call LOCALPUSHRELABEL too.

6.6 What is Wrong if We Don't Call Exact Max Flow?

Say we call BASICPUSHRELABEL with $h < n + 1$. If BASICPUSHRELABEL terminates with excess, then the returned cut P can be such that $f_{out}(P) < \delta_{\frac{2}{\phi}G\{U\}}(P)$.



When h is big enough, like $h = O(\frac{\log n}{\phi})$, we know the portion of "bad cut edges" is quite small.

This motivates the following idea: the (restriction of) flow f is almost a good enough certificate of $G[U \setminus P]$. Therefore, we can work with $U' \leftarrow U \setminus P$ and try to reuse flow f.

6.7 Fast Algorithm via Dynamic PUSH/RELABEL

Initialize the following as follows:

- $U_1 = U$, $f_1 \equiv 0$, $\ell_1 \equiv 0$.
- $h = O(\frac{\log n}{\phi})$
- $t = 1$

While LOCALPUSHRELABEL$(\frac{2}{\phi}G\{U^{(t)}\}, \frac{2}{\phi}\Delta_{U^{(t)}}, T_{U^{(t)}}, h, (f^{(t)}, \ell^{(t)}))$ terminates with some excess, let $P^{(t)}$ be the returned cut such that

$$\begin{aligned} f^{(t)}_{out}(P^{(t)}) &\geq \delta_{\frac{2}{\phi}G\{U^{(t)}\}}(P^{(t)}) - \frac{\phi}{100}\mathrm{vol}_{\frac{2}{\phi}G\{U^{(t)}\}}(P^{(t)}) \\ &= \delta_{\frac{2}{\phi}G\{U^{(t)}\}}(P^{(t)}) - \frac{1}{50}\mathrm{vol}_{G\{U^{(t)}\}}(P^{(t)}) \end{aligned}$$

Let $U^{(t+1)} \leftarrow U^{(t)} \setminus P^{(t)}$, and let $(f^{(t+1)}, \ell^{(t+1)})$ be obtained from $(f^{(t)}, \ell^{(t)})$ by restricting from $\frac{2}{\phi}G\{U^{(t)}\}$ to $\frac{2}{\phi}G\{U^{(t+1)}\}$.

Exercise: what to do exactly at new boundary vertices?

Exercise 6.8. After restriction, $(f^{(t+1)}, \ell^{(t+1)})$ is a $(\frac{2}{\phi}G\{U^{(t+1)}\}, \frac{2}{\phi}\Delta_{U^{(t+1)}}, T_{U^{(t+1)}})$-valid state.

So the algorithm description is valid. By Lemma 6.3, we havethe following:

Lemma 6.9. *When* LOCALPUSHRELABEL$(\frac{2}{\phi}G\{U^{(t)}\}, \frac{2}{\phi}\Delta_{U^{(t)}}, T_{U^{(t)}}, h, (f^{(t)}, \ell^{(t)}))$ *terminates with no excess, then* $G[U^{(t)}]$ *is a* $\phi/6$*-expander.*

Lemma 6.10. *The total new amount of mass before round* $t + 1$ *is* $\frac{2}{\phi}|\partial_G \langle U^{(t+1)} \rangle| - f^{(t+1)}_{out}(\partial_G \langle U^{(t+1)} \rangle \leq \frac{1}{50}\mathrm{vol}_{G\{U^{(t)}\}}(P^{(t)})$.

Lemma 6.11. *The total destroyed mass before round $t+1$ is* $\mathrm{vol}_{G\{U^{(t)}\}}(P^{(t)} \cap U^{(t)})$.

Lemma 6.12. *The total amount of mass we ever added is at most* $2 \times \frac{2}{\phi}|\partial_G \langle U \rangle|$.

Lemma 6.13. *Let* $P = \bigcup_t P^{(t)}$. *We have* $\mathrm{vol}(P) \leq 2 \times \frac{2}{\phi}|\partial_G \langle U \rangle|$.

All that remains is to bound the total running time. But we saw that the total running time is proportional to the total amount of mass times h. So the total running time is $2 \times \frac{2}{\phi}|\partial_G \langle U \rangle| \times h = O(|\partial_G \langle U \rangle| \frac{\log n}{\phi^2})$.

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 23: Boundary-Linked Decomposition

November 30, 2021

Instructor: Thatchaphol Saranurak

Scribe: Ashwin Sreevatsa

1 Formal definitions

1.1 Well-linked Sets

Intuition: **Well-linked sets** of graphs are subsets of the vertices of a graph that are well-connected.

Let $G = (V, E)$ be a graph and $T \subseteq V$ is a set of terminals. T is **α-well-linked** in G if, for any disjoint sets $A, B \subseteq T$, we have $\text{mincut}_G(A, B) \geq \alpha \min\{|A|, |B|\}$.

Note that for any cut $(S, \overline{S})$ in G, $\delta_G(S) = \text{mincut}_G(S, \overline{S}) \geq \text{mincut}_G(S \cap T, \overline{S} \cap T) \geq \alpha \min\{|S \cap T|, |\overline{S} \cap T|\}$.
Also note that V is α-well-linked $\iff \Psi(G) \geq \alpha$.

We can also introduce the concept of well-linkedness to edges. Consider a graph $G = (V, E)$, construct a **split graph** $G' = (V', E')$ from G by converting each edge $e = (u, v) \in E$ from graph G into $(u, x_e), (x_e, v)$ in G'. Note that $V' = V \cup \{\text{the set of split nodes of } G\}$, $E' = \{(u, x_e), (x_e, v) \text{ for each edge } e = (u, v) \in E\}$. (Figure 1)

Let $X_E = \{x_e | e \in E\}$ be the set of split nodes of G. Then, we can show that $\Phi(G) \geq \phi \iff X_E$ is ϕ-well-linked in G'.
We can also consider well-linked sets from the **flow perspective**. Let K be the **all-to-all demand** between T when K is the **d**-product graph, with $\text{d}(u) = 1$ if $u \in T$ and $\text{d}(u) = 0$ otherwise.

If $K \preccurlyeq^{\text{flow}} G$, then for any demand H where $V(H) = T$ and H has maximum degree $O(1)$, we have $H \preccurlyeq^{\text{deg}} O(1)K \implies H \preccurlyeq^{\text{flow}} O(1)K \preccurlyeq^{\text{flow}} O(1)G$.
This means that if the all-to-all demand between T is routable, then any demand between T where each node exchanges at most $O(1)$ units of flow must be routable with $O(1)$ congestion.

Exercise 1.1. Given a graph G and a vertex set T, show an algorithm for polylog(n)-approximating the well-linkedness of T.

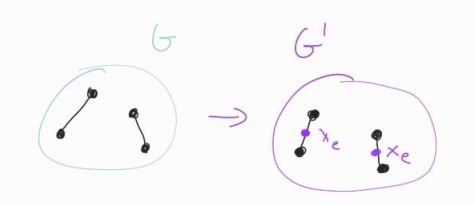


Figure 1: An example of a graph with split edges

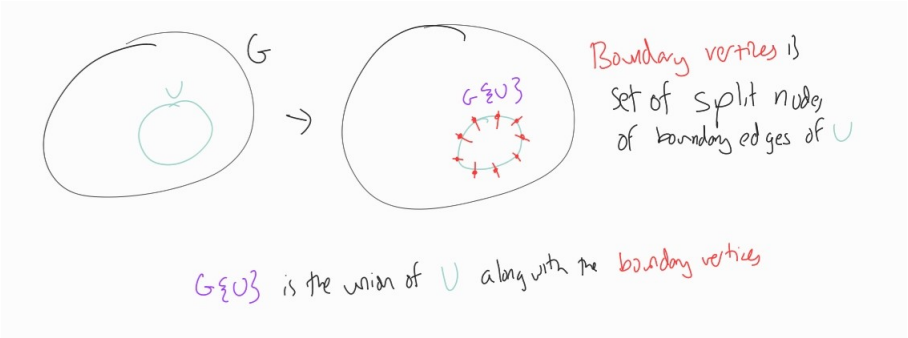


Figure 2: Boundary vertices of a graph with vertex subset U

Proof. We have that T is α-well-linked in $G \iff$ the all-to-all demand between T is routable in G with congestion $O(\frac{\log n}{\alpha})$ from a previous exercise. Since we have an algorithm for finding the minimum congestion routable flow for the all-to-all demand in T, and this value will simply be a log(n) approximation of the well-linkedness. □

1.2 Boundary-linked Graphs

Let $G = (V, E)$ be a graph with $U \subseteq V$ a subset of the vertices. Then $G\{U\}$ is the union of the subgraph U and its **boundary vertices** (figure 2). Formally, for every edge $e = (u, v) \in E$ with $u \in U$ and $v \in V/U$, create a new vertex x_e such that $(u, x_e), (x_e, v) \in E$ and $(u, x_e) \in G\{U\}$. (Figure 2)

Let $\partial_G \langle U \rangle = V(G\{U\}) \setminus U$ be the set of boundary vertices. Then, we say $G\{U\}$ is α-boundary-linked if $\partial_G \langle U \rangle$ is α-well-linked in $G\{U\}$.

An additional formulation of this notion of boundary-linkedness is with respect to flow:

- the all-to-all demand between $\partial_G \langle U \rangle$ is routable with congestion $\tilde{O}(1/\alpha)$, or equivalently
- constant-degree expanders on $\partial_G \langle U \rangle$ is embeddable into G with congestion $\tilde{O}(1/\alpha)$.

2 Boundary-linked Decomposition

Intuition: we want to take some induced subgraph $G\{U\}$ with boundary vertices and partition U into $U_1, U_2, ..., U_k$ to produce a new set of induced subgraphs $G\{U_1\}, G\{U_2\}, ..., G\{U_k\}$. This will

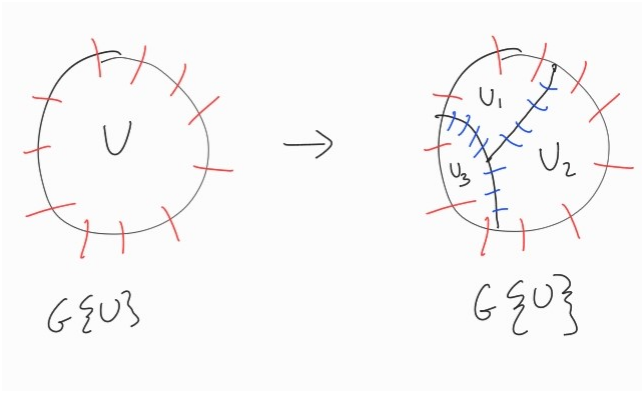


Figure 3: Boundary-linked Decomposition

be an α**-boundary-linked decomposition** (figure 3) if the new induced subgraphs need to satisfy 2 constraints:

1. $G\{U_i\}$ is α-boundary-linked
2. There aren't too many new boundary edges
   (a) Formally we want the total number of new boundary edges to be $\sum_i |\partial_G \langle U\rangle_i \backslash \partial_G \langle U\rangle| = c_{\text{bound}} \cdot \alpha |\partial_G \langle U\rangle|$
   (b) $c_{\text{bound}} = O(\log |\partial_G \langle U\rangle|)$ for existential result
   (c) $c_{\text{bound}} = n^{o(1)}$ for fast algorithms

2.1 Motivation: Vertex Sparsifiers

One application of this method of boundary-linked decomposition is for vertex sparsifiers (figure 4). In essence, we can 'compress' $G\{U\}$ into a new graph H by taking each set U_i and contracting it into a single vertex u_i in H.
As it turns out, H preserves cut size between all subsets of boundary vertices.

Theorem 2.1. *For any two set of boundary vertices $A, B \subseteq \partial_G \langle U\rangle$, we have*

$$\text{mincut}_{G\{U\}}(A, B) \leq \text{mincut}_H(A, B) \leq \frac{1}{\alpha} \cdot \text{mincut}_{G\{U\}}(A, B)$$

Proof. The first half of the inequality is easy to prove. For any (A, B) mincut in H, that cut must necessarily exist in $G\{U\}$ because graph H is a 'compression' of graph $G\{U\}$. So we have $\text{mincut}_{G\{U\}}(A, B) \leq \text{mincut}_H(A, B)$.

The second inequality is the harder inequality to prove. Let (X, Y) be an (A, B)-mincut in $G\{U\}$. We want to find another (A, B)-cut (X', Y') in H that is only blown up by a factor of $\frac{1}{\alpha}$. If this can be satisfied, then we have $\text{mincut}_H(A, B) \leq \frac{1}{\alpha} \cdot \text{mincut}_{G\{U\}}(A, B)$ and we're done.

The way we do this is by taking the (X, Y) cut, and for every set U_i that it crosses, redirect the

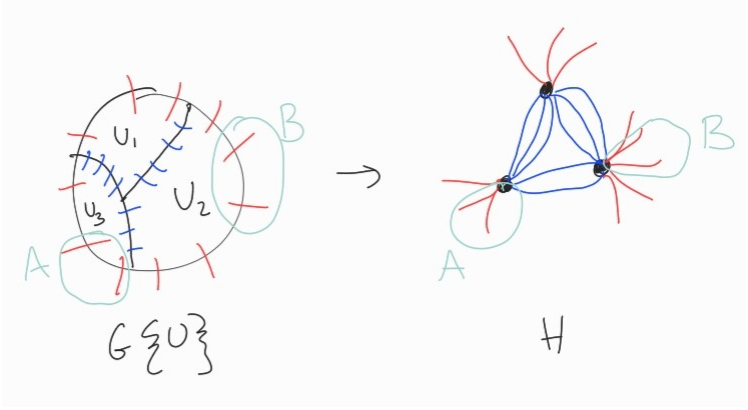


Figure 4: Vertex Sparsifier

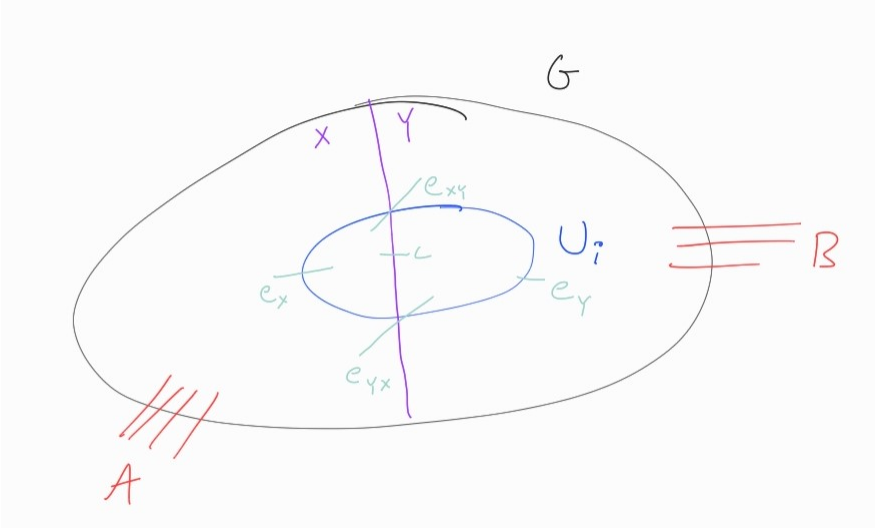


Figure 5: An example of a graph with split edges

cut so that it doesn't cross U_i. Since we can redirect this cut 2 different ways, choose the direction that cuts less edges.

Notice figure 5.

- $e_X = |E_G(X \cap U_i, X \setminus U_i)|$
- $e_Y = |E_G(Y \cap U_i, Y \setminus U_i)|$
- $e_{XY} = |E_G(X \cap U_i, Y \setminus U_i)|$
- $e_{YX} = |E_G(Y \cap U_i, X \setminus U_i)|$
- $c = |E_G(X \cap U_i, Y \cap U_i)|$

We have that $e_X + e_{XY}$ is the number of boundary vertices whose endpoint in U_i is on the X side, whereas $e_Y + e_{YX}$ is the number of boundary vertices whose endpoint in U_i is on the Y side. Assume without loss of generality that $e_X + e_{XY} \leq e_Y + e_{YX}$. Then, we want to route the (X', Y')

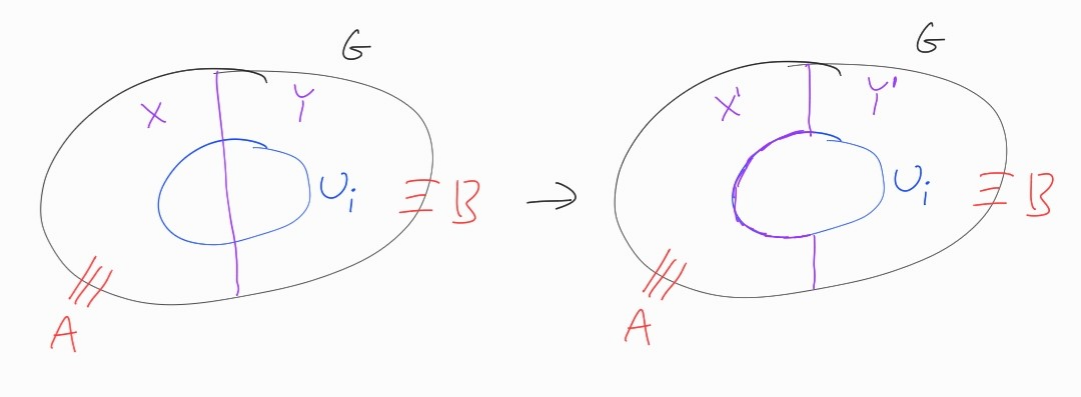


Figure 6: Boundary-linked decomposition algorithm (Algorithm 1)

cut on the X side of U_i.

Note that $e_X \leq e_X + e_{XY} \leq \frac{1}{\alpha}c$, where the second inequality holds by U_i being α-boundary-linked. So we have $E(X,Y) = c + e_{XY} + e_{YX} \geq \alpha \cdot e_X + \alpha \cdot e_{YX} = \alpha \cdot (e_X + e_{YX}) = \alpha \cdot E(X',Y')$. Then, $E(X',Y') \leq \frac{1}{\alpha}E(X,Y)$ and $\text{mincut}_H(A,B) \leq \frac{1}{\alpha} \cdot \text{mincut}_{G\{U\}}(A,B)$. □

How large would the graph H be?

Lemma 2.2. $|E(H)| = \underbrace{|\partial_G \langle U \rangle|}_{\text{old boundary}} + \underbrace{\sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle|}_{\text{new boundary}} = O(c_{\text{bound}}|\partial_G \langle U \rangle|)$.

From the construction of boundary-linked decompositions, we know that the number of new boundary edges $= \sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle| \leq c_{\text{bound}} \cdot \alpha |\partial_G \langle U \rangle|$, so the second equality will hold.

As a result, H is a vertex sparsifier of $G\{U\}$. We have reduced the number of vertices and edges to be proportional to $\partial_G \langle U \rangle$ while still approximately preserving all of the cuts for $\partial_G \langle U \rangle$.

2.2 Existence of Boundary-linked Decomposition

The generic algorithm for showing the existence of expander decomposition used the following idea:

- If there is no "sparse" cut, we are done
- Otherwise, find the "sparse" cut, cut the graph into 2 subgraphs, and recurse on both sides.

We can use this same algorithm for existence of boundary-linked decomposition, with "sparsity" defined via boundary-linkedness. (Figure 6, Algorithm 1)

We know that $G\{U_i\}$ will be α-boundary-linked for all i. What we need to do now is to bound the number of new boundary vertices $\sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle|$.

Goal: Ideally, we want $\sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle| = O(|\partial_G \langle U \rangle|)$.

Algorithm 1 Boundary-linked Decomposition Algorithm: Decomp($G\{U\}, \alpha$)

Require: $G\{U\}, \alpha$
  if $\partial_G \langle U \rangle$ is α-well-linked **then**
    return $\{U\}$
  else
    there must be a cut $(S, \overline{S}) \in G\{U\}$ where $\delta_G(S) < \alpha \min\{|S \cap \partial_G \langle U \rangle|, |\overline{S} \cap \partial_G \langle U \rangle|\}$
    return Decomp($G\{U \cap S\}, \alpha$) $\cup$ Decomp($G\{U \cap \overline{S}\}, \alpha$)
  end if

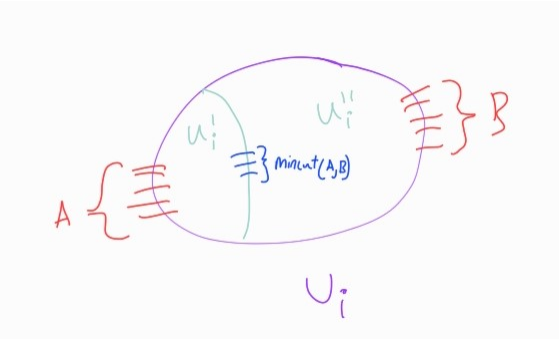


Figure 7: Sparse cut of U_i

Strategy:

- Put "money" into each vertex of the graph
- Every time we create a new boundary vertices, we will pay \$1
- **Invariant**: Each boundary vertex in $\partial_G \langle U_i \rangle$ has at least $4\alpha \log_{3/2}(\partial_G \langle U_i \rangle)$ dollars, for all *current* sets U_i. We will need to reallocate money at each step in order to maintain the invariant
- We will show that the initial money is enough to 'pay' for all of the new boundary vertices created

The point of all of this is to bound the number of new boundary vertices. Since we must spend \$1 for each new boundary vertex created, the total number of created boundary vertices is at most the total initial money, or $O(\alpha \cdot |\partial_G \langle U \rangle| \cdot \log |\partial_G \langle U \rangle|) = O(|\partial_G \langle U \rangle|)$. [Note that this requires $\alpha = O(\frac{1}{\log |\partial_G \langle U \rangle|})$]

Proof. Consider U_i in figure 7. We find a sparce cut in $G\{U_i\}$ and recurse on $G\{U_i'\}$ and $G\{U_i''\}$. We have

- $A = \partial_G \langle U \rangle_i' \cap \partial_G \langle U \rangle_i$
- $B = \partial_G \langle U \rangle_i'' \cap \partial_G \langle U \rangle_i$
- $C = \text{mincut}_{G\{U_i\}}(A, B)$

Assume w.l.o.g that $|A| \leq |B|$. We will use money from A to pay for the new boundary edges created.
First note that $|\partial_G \langle U_i' \rangle| = |A| + |C| \leq |A| + \alpha \cdot |A| = (1 + \alpha) \cdot |A| \leq \frac{4}{3}|A| \leq \frac{2}{3}|\partial_G \langle U_i \rangle|$.

(The first equality holds from how $\partial_G \langle U_i' \rangle$ is defined, the second inequality holds from the fact that U_i is not well-linked and C is a mincut, the third equality holds from algebra, the fourth inequality holds from $\alpha = O(\frac{1}{\log |\partial_G \langle U \rangle|})$, the final inequality holds from $|A| \leq |B|$ and $|A| + |B| = \partial_G \langle U_i \rangle$.)
This gives us $\log_{3/2} |\partial_G \langle U_i' \rangle| \leq \log_{3/2} |\partial_G \langle U_i \rangle| - 1$.

Based on our invariant, we know that for each $a \in A$, a has $4\alpha \log_{3/2} |\partial_G \langle U \rangle_i|$. After the partition into U_i', U_i'', a has a little bit of extra money (since the new graph has less boundary edges than before). a only needs $4\alpha \cdot |\log_{3/2}| \cdot (\partial_G \langle U_i' \rangle) \leq 4\alpha \cdot |\log_{3/2}| \cdot (\partial_G \langle U_i \rangle - 1) = 4\alpha \cdot |\log_{3/2}| \cdot (\partial_G \langle U_i \rangle) - 4\alpha$ dollars.So a has 4α extra dollars it can spend.

After cutting U_i, we collect all of the extra dollars from A, which will be $4\alpha|A|$. We will use this money for creating all of the boundary vertices and for maintaining the invariants on both sides of the cut (for both U_i' and U_i'').

The total money needed =

- $2|C| \longrightarrow$ (\$1 for each of the $|C|$ boundary edges in both U_i' and U_i'') +
- $|C| \cdot 4\alpha \log_{3/2}(\partial_G \langle U_i' \rangle) \longrightarrow$ (to maintain the invariant in U_i') +
- $|C| \cdot 4\alpha \log_{3/2}(\partial_G \langle U_i'' \rangle) \longrightarrow$ (to maintain the invariant in U_i'')

We have $2|C| + |C| \cdot 4\alpha \log_{3/2}(\partial_G \langle U_i' \rangle) + |C| \cdot 4\alpha \log_{3/2}(\partial_G \langle U_i'' \rangle) \leq 2|C| + |C| \cdot 8\alpha \log_{3/2}(\partial_G \langle U_i \rangle) \leq 2|C| + 2|C| = 4|C| \leq 4|A|\alpha$.
(The first inequality holds from the fact that the number of boundary edges of U_i is an upper bound on the number of boundary edges of U_i' and U_i'', the second inequality holds from $\alpha \leq \frac{1}{4 \log_{3/2}(\partial_G \langle U_i \rangle)}$, the third equality holds from algebra, the fourth inequality holds from U_i not being α-well-linked and (U_i', U_i'') being a mincut).

Since we have enough extra money to pay for all of the costs of adding boundary edges and maintaining invariants, we are done. In other words, $G\{U_i\}$ is α-boundary-linked for all i, and $\sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle| = O(\alpha |\partial_G \langle U \rangle| \log |\partial_G \langle U \rangle|)$. □

2.3 How to compute it fast?

Exercise 2.1. Give a polynomial time algorithm for boundary-linked decomposition with slightly worse guarantee.

1. Show that an algorithm that, given $G\{U\}$, either
   - reports that $\partial_G \langle U \rangle$ is $\alpha/\text{polylog}(n)$-well-linked, or.
   - finds a cut certifying that $\partial_G \langle U \rangle$ is not α-well-linked
2. Use this subroutine to get boundary-linked decomposition. The only change in the analysis is that when we stop the recursion at $G\{U_i\}$, we only guarantee that $G\{U_i\}$ is $\alpha/\text{polylog}(n)$-boundary-linked

Exercise 2.2. Use the cut-matching game framework to compute α-boundary-linked decomposition in almost-linear time, where $\alpha \geq 1/n^{o(1)}$ and $\sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle| = O(\alpha |\partial_G \langle U \rangle| n^{o(1)})$. *Hint*: Find most balanced sparse cut.

3 Boundary-linked Expander Decomposition

We've shown that we can partition graphs based on boundary-linkedness and based on expanders (expander decomposition). Can we do both at the same time?

Terminology: U is (α, ϕ)-linked in G if $G\{U\}$ is α-boundary-linked and $G\{U\}$ is a ϕ-expander.

If U is (α, ϕ)-linked, then

- Edges in $G\{U\}$ are $\frac{1}{\phi}$-well-linked. (The all-to-all demand between edges of $G\{U\}$ is routable in $G[U]$ with congestion $O(\frac{\log n}{\phi})$).
- Boundary edges in $G\{U\}$ are $\frac{1}{\alpha}$-well-linked. (The all-to-all demand between boundary edges of $G\{U\}$ is routable in $G[U]$ with congestion $O(\frac{\log n}{\phi})$).

We want to show an algorithm with the following characteristics:

- **Input:** $G\{U\}, \phi$
- **Output:** a partition $U_1, U_2, ..., U_k$ of U such that:
  - U_i is (α, ϕ)-linked in G (with $\alpha = \frac{1}{\Theta(\log n)}$)
  - $\sum_i |\partial_G \langle U_i \rangle| = O(|\partial_G \langle U \rangle| + \phi \text{vol}_G(U) \log m)$.

3.1 Induced subgraph with k-boundary-self-loops

To conclude that U is (α, ϕ)-linked, we introduce the following notion:

Let $G[U]^k$ denote an **induced subgraph with k-boundary-self-loops** (figure 8). What this means is that we start with the induced subgraph $G[U]$ and add k self-loops (or a self-loop with capacity k) on each boundary edge $e = (u, v) \in E(U, V \setminus U)$ with endpoint $u \in U$. (Figure TODO)

Lemma 3.1. If $G[U]^{\alpha/\phi}$ is a ϕ-expander, then U is (α, ϕ)-linked.

Proof. We can prove the cases separately.

$G\{U\}$ is a ϕ-expander because

- For any cut $(S, \overline{S})$ in $G\{U\}$, let $(S', \overline{S}') = (S \cap U, \overline{S} \cap U)$ be a cut in $G[U]^{\alpha/\phi}$.
- We have $\delta_{G\{U\}}(S) \geq \delta_{G[U]^{\alpha/\phi}}(S')$ but $\text{vol}_{G[U]}(S) \leq \text{vol}_{G[U]^{\alpha/\phi}}(S')$.
- So $\Phi_{G[U]}(S) \geq \Phi_{G[U]^{\alpha/\phi}}(S') \geq \phi$

$G\{U\}$ is α-boundary-linked.

- Let A, B be a partition of $\partial_G \langle U \rangle$.

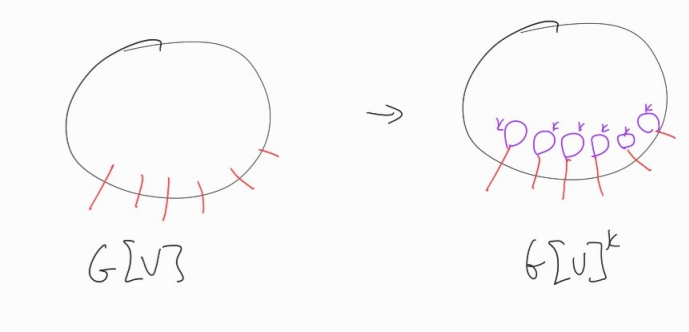


Figure 8: Induced subgraph with k-boundary-self-loops

- For any (A, B)-mincut $(S, \overline{S})$ in $G\{U\}$, let $(S', \overline{S}') = (S \cap U, \overline{S}' \cap U)$ be a cut in $G[U]^{\alpha/\phi}$.
- We have

$$\begin{aligned}
\text{mincut}_{G\{U\}}(A, B) &= \delta_{G\{U\}}(S) \\
&= \delta_{G[U]^{\alpha/\phi}}(S') \\
&\geq \phi \cdot \min\{\text{vol}_{G[U]^{\alpha/\phi}}(S'), \text{vol}_{G[U]^{\alpha/\phi}}(\overline{S}')\} \\
&= \phi \cdot \min\{\frac{\alpha}{\phi} \cdot |A|, \frac{\alpha}{\phi} \cdot |B|\} \\
&= \alpha \cdot \min\{|A|, |B|\}
\end{aligned}$$

□

3.2 Existence of Boundary-linked Expander Decomposition

The algorithm and analysis for boundary-linked expander decomposition will be very similar to that of the general boundary-linked decomposition.

Input:

- an induced subgraph $G\{U\}$ with boundary vertices
- a parameter ϕ.

Output: a partition of $U_1, \ldots, U_k$ of U such that

- $G[U_i]^{\alpha/\phi}$ is a ϕ-expander
  - $\alpha = 1/4 \log_{3/2}(\text{vol}(G[U]^{\alpha/\phi})) = \Theta(1/\log m)$
- $\sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle| = O(|\partial_G \langle U \rangle| + \phi \text{vol}_G(U) \log m)$

Analysis strategy:

- We'll put money on "edges" of graphs, and whenever we create a new boundary vertex, we pay \$1.
- **Invariant**: Each vertex u in $G[U_i]^{\alpha/\phi}$ has at least $\deg_{G[U_i]^{\alpha/\phi}}(u) \times 4\phi \log_{3/2}(\text{vol}(G[U_i]^{\alpha/\phi}))$ dollars.

Algorithm 2 Boundary-linked Expander Decomposition Algorithm: Decomp($G\{U\}^{\alpha/\phi}, \phi$)

Require: $G\{U\}^{\alpha/\phi}, \phi$
  if $G\{U\}^{\alpha/\phi}$ is ϕ-expander **then**
    return $\{U\}$
  else
    there must be a cut $(S, \overline{S}) \in G\{U\}$ that is a ϕ-sparse cut
    return Decomp($G\{U \cap S\}^{\alpha/\phi}, \phi$) $\cup$ Decomp($G\{U \cap \overline{S}\}^{\alpha/\phi}, \phi$)
  end if

- The total number of new boundary vertices is bounded by the total initial money, $O(|\partial_G \langle U \rangle| + \phi \mathrm{vol}_G(U) \log m)$.

Proof. Suppose we find a ϕ-sparse cut (U_i', U_i'') in $G[U_i]^{\alpha/\phi}$ and we recurse on $G[U_i']^{\alpha/\phi}$ and $G[U_i'']^{\alpha/\phi}$.

We have $c < \phi \min\{a, b\}$.

Define

- $a = \mathrm{vol}_{G[U_i]^{\alpha/\phi}}(U_i')$
- $b = \mathrm{vol}_{G[U_i]^{\alpha/\phi}}(U_i'')$
- $c = \delta_{G[U_i]^{\alpha/\phi}}(U_i')$

Assume w.l.o.g. that $a \le b$.
The volume of the bigger side does not increase: $\mathrm{vol}(G[U_i'']^{\alpha/\phi}) \le b + \frac{\alpha}{\phi} c \le b + a = \mathrm{vol}(G[U_i]^{\alpha/\phi})$
The volume of the smaller side decreases by a constant factor: $\mathrm{vol}(G[U_i']^{\alpha/\phi}) \le \frac{2}{3}\mathrm{vol}(G[U_i]^{\alpha/\phi})$ because

$$\begin{aligned} \mathrm{vol}(G[U_i']^{\alpha/\phi}) &\le a + \frac{\alpha}{\phi} c \\ &\le \frac{4}{3} a && \text{as } c \le \phi a \text{ and } \alpha \ll 1/3 \\ &\le \frac{2}{3} \mathrm{vol}(G[U_i]^{\alpha/\phi}). \end{aligned}$$

- So $\log_{3/2}(\mathrm{vol}(G[U_i']^{\alpha/\phi})) \le \log_{3/2}(\mathrm{vol}(G[U_i]^{\alpha/\phi}) - 1$.

After cutting U_i,

- we can collect $a \cdot 4\phi$ dollars (from endpoints of edges in $G[U_i]^{\alpha/\phi}$ incident to U_i').
- money needed for the new boundary vertices is at most
  - $2c \longrightarrow$ [\$1 per new boundary]
  - $\frac{\alpha}{\phi} c \cdot 4\phi \log_{3/2}(\mathrm{vol}(G[U_i']^{a/\phi})) \longrightarrow$ [invariant on $G[U_i']^{a/\phi}$]
  - $\frac{\alpha}{\phi} c \cdot 4\phi \log_{3/2}(\mathrm{vol}(G[U_i'']^{a/\phi})) \longrightarrow$ [invariant on $G[U_i'']^{a/\phi}$]
- So we have enough money to maintain the invariant.

This completes the proof.

- $G[U_i]^{\alpha/\phi}$ is a ϕ-expander for all i.
- $\sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle| = O(|\partial_G \langle U \rangle| + \phi \mathrm{vol}_G(U) \log m)$

□

Some additional resources here: [And10], [CC13], [GRST21].

References

[And10] Matthew Andrews. Approximation algorithms for the edge-disjoint paths problem via raecke decompositions. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 277–286. IEEE, 2010.

[CC13] Chandra Chekuri and Julia Chuzhoy. Large-treewidth graph decompositions and applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 291–300, 2013.

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2212–2228. SIAM, 2021.

Tree Flow Sparsifier via Expander Hierarchy

December 11, 2025

1 Cut/Flow Sparsifier w.r.t. Terminals

Definition 1.1. Given a graph $G = (V, E)$, a **cut sparsifier** H of G for terminal set S with quality q satisfies the following

1. $V(H) \supseteq S$
2. For every $A, B \subseteq S$, we have

$$\text{mincut}_G(A, B) \leq \text{mincut}_H(A, B) \leq q \cdot \text{mincut}_G(A, B).$$

Example 1.2. We saw this concept from the previous class.

- Let $G\{U\}$ be a graph with boundary
- Let H be obtained from $G\{U\}$ as follows
  - Let $U_1, \ldots, U_k$ be a α-boundary-linked decomposition of $G\{U\}$.
  - Contract each U_i into a vertex u_i.
  - Note that the size of H is just proportional to the boundary $|E(H)| = O(|\partial_G \langle U \rangle|)$
- We showed that H is a cut sparsifier of $G\{U\}$ for terminal set $\partial_G \langle U \rangle$ with quality $\frac{1}{\alpha}$.

Definition 1.3. Given a graph $G = (V, E)$, a **flow sparsifier** H of G for terminal set S with quality q satisfies the following

1. $V(H) \supseteq S$
2. Let $D = (S, E')$ be a demand between terminals S.

$$\mathsf{mcf}(H, D) \leq \mathsf{mcf}(G, D) \leq q \cdot \mathsf{mcf}(H, D)$$

   where $\mathsf{mcf}(G, D)$ is the minimum congestion for routing D in G. In words, if D is routable in G, then D is routable in H. Conversely, if D is routable in H, then D is routable in G with congestion q.

Exercise 1.4. [TODO in scribe node: add the proof of this too.] Show that the two concept are equivalent up to logarithmic factor:

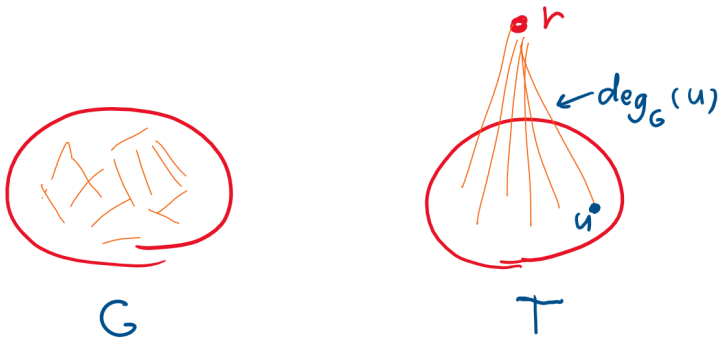
1. if H is a flow sparsifier of G with quality q, then H is a cut sparsifier of G with quality q.
2. if H is a cut sparsifier of G with quality q, then H is a flow sparsifier of G with quality $O(q \log n)$.

2 Flow Sparsifier of Expanders = Star

- Does it make sense to let terminal set be V?
  - We cannot reduce the number of vertices for sure.
  - But what is a non-trivial thing we can do?
- Let's look at this example. For any expander can be simplified to a star!

Example 2.1. Let $G = (V, E)$ be a ϕ-expander.

- Let T be a star where leaves corresponds to V. Let r be the root of the star.
- For each tree edge (v, r), set the capacity in T as $c_T(v, r) = \deg_G(v)$.
- T is a flow sparsifier of G for terminal set V with quality $\frac{\log(n)}{\phi}$.



Proof. Let D be a demand on terminal V. We argue two directions

- If D is routable in G, then D is routable in T.
  - Given D in T, just route everything to the root. The flow paths "match". Done.
  - No congestion in T
    - Note that the flow on (u, r) is exactly $\deg_D(u)$.
    - As D is routable in G, $\deg_D(u) \leq \deg_G(u)$ for all $u \in V$
    - But $\deg_G(u) = c_T(v, r)$ by construction.
- If D is routable in T, then D is routable in G with congestion $O(\log(n)/\phi)$.
  - If D is routable in T, then D is $\deg_G$-restricted. (i.e. $\deg_D(u) \leq \deg_G(u)$).
  - As G is a ϕ-expander, any $\deg_G$-restricted demand is routable with congestion $O(\log(n)/\phi)$.

□

- From now, we just say "H is a flow sparsifier of G" when the terminal set is $V(G)$.

3 Flow Sparsifier from Boundary-linked Expander Decomposition

- Now, let's try to apply this idea for expanders to arbitrary graphs
  - How?
  - As usual, expander decomposition.
  - But we will need boundary-linkedness too.

3.1 Recap: Boundary-linked expander decomposition

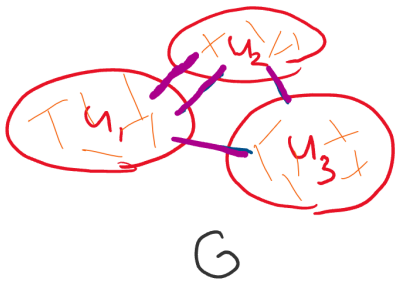
- **Input:**
  - an induced subgraph $G\{U\}$ with boundary vertices
  - a parameter ϕ.
- **Output:** a partition of $U_1, \ldots, U_k$ of U such that
  - U_i is (α, ϕ)-linked where $\alpha = \Theta(1/\log m)$. That is,
    - $G\{U\}$ is α-boundary-linked, and
    - $G\{U\}$ is a ϕ-expander.
  - $\sum_i |\partial_G \langle U_i \rangle \setminus \partial_G \langle U \rangle| = O(|\partial_G \langle U \rangle| + \phi \mathrm{vol}_G(U) \log m)$
- Throughout the lecture, think of $\alpha = \Theta(1/\log m)$ but $\phi = \Theta(1/2^{\sqrt{\log n}}) = 1/n^{o(1)}$. So $\phi \ll \alpha$.

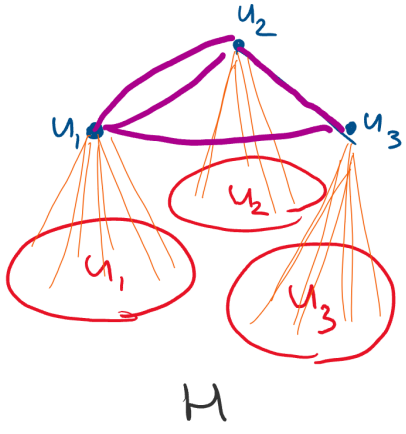
3.2 Sparsifier Construction

Lemma 3.1. *Let $G = (V, E)$ be a graph. Do the following:*

1. *Compute a α-boundary-linked ϕ-expander decomposition $\mathcal{U} = \{U_1, \ldots, U_k\}$ of G.*
2. *Contract each U_i into a vertex u_i. Let $G_{\mathcal{U}}$ be the contracted graph.*
3. *Let H be obtained from $G_{\mathcal{U}}$ as follows. For each $U_i \in \mathcal{U}$, attach a star rooted at u_i in $G_{\mathcal{U}}$ as in Example 2.1. More formally, for each $w \in U_i$, add (w, u_i) with capacity $\deg_G(w)$ into $G_{\mathcal{U}}$.*

Then, we have that H is a flow sparsifier of G with quality $O((\frac{1}{\alpha} + \frac{1}{\phi}) \log m) = O(\frac{1}{\phi} \log m)$.



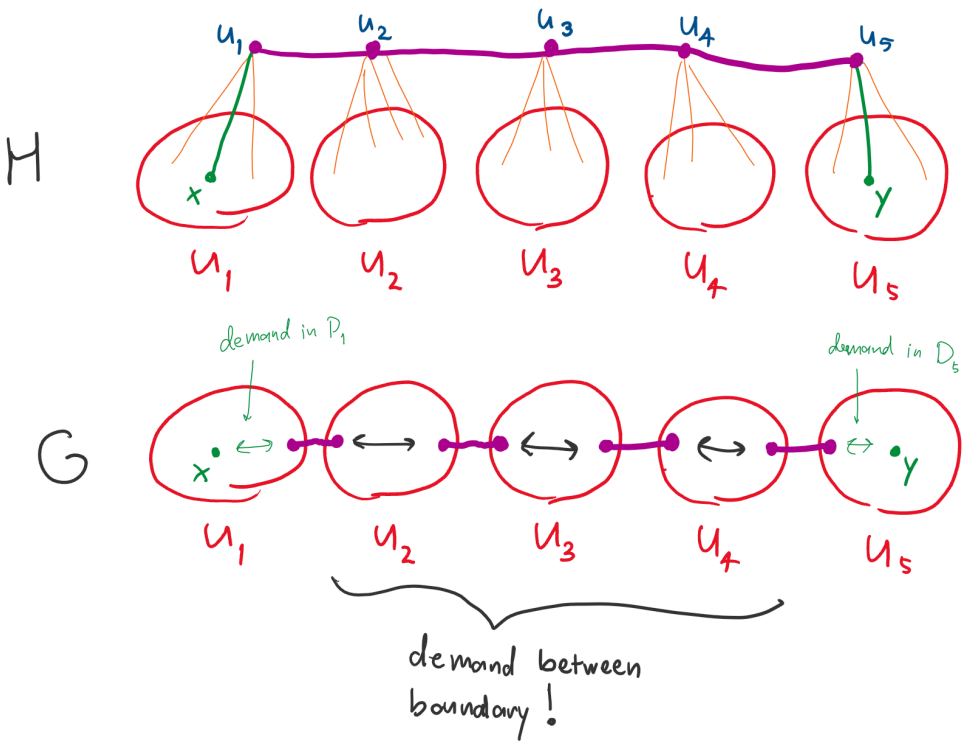


Proof. Let D be a demand on terminal V. Let $D_{\mathcal{U}}$ be a *projection* of D to $V(G_{\mathcal{U}})$, i.e., for $u_i, u_j \in V(G_{\mathcal{U}})$, we define

$$D_{\mathcal{U}}(u_i, u_j) = \sum_{x \in U_i, y \in U_k} D(x, y).$$

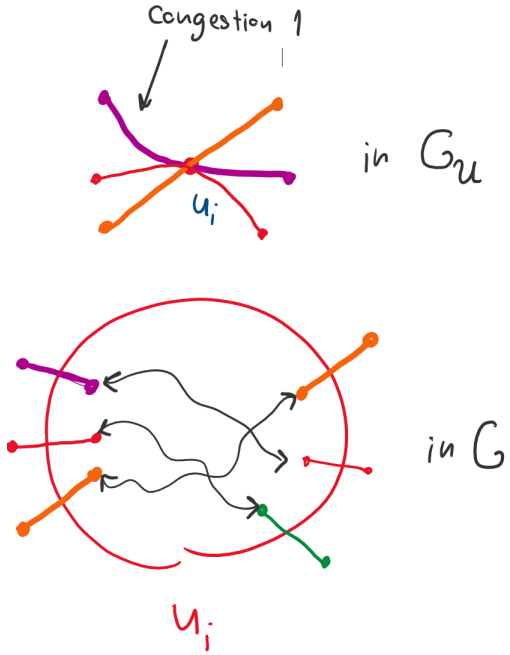
There are two directions:

1. Suppose D is routable in G. Then D is routable in H.
   - For each demand $(x, y) \in E(D)$ where $x \in U_i$ and $y \in U_j$, we split the demand into three parts $x \leftrightarrow u_i$, $u_i \leftrightarrow u_j$, $u_j \leftrightarrow y$.
     (a) The first/third parts, even after summing over all demand pairs, can be routed with congestion using the star edges.
     (b) For the second part, after summing over all demand pairs, this is to route $D_{\mathcal{U}}$ in $G_{\mathcal{U}}$.
       - $D_{\mathcal{U}}$ is routable in $G_{\mathcal{U}}$.
       - **Exercise:** $D_{\mathcal{U}}$ is routable in $G_{\mathcal{U}}$ iff D is routable in G when the edge capacity inside each U_i is infinite.
   - The edges of H used in the two steps above are disjoint. So there is no congestion.
2. Suppose D is routable in H. Then D is routable in G with congestion $O((\frac{1}{\alpha} + \frac{1}{\phi}) \log m)$.
   - We write $D = \sum_{U_i \in \mathcal{U}} D_i + D_{dif}$ where
     - D_i contains all demand pairs (x, y) where both $x, y \in U_i$
     - D^{dif} contains all demand pairs (x, y) where x, y are in different parts.
   - Each D_i is routable in $G[U_i]$ with congestion $O(\frac{\log m}{\phi})$.
     - D_i is $\deg_{G\{U\}}$-restricted. (D_i may not be $\deg_{G[U_i]}$-restricted)
     - $G\{U_i\}$ is a ϕ-expander. So any $\deg_{G\{U\}}$-restricted demand is routable in $G\{U_i\}$ with congestion $O(\frac{\log m}{\phi})$.
     - Actually D_i is routable in $G[U_i]$ with congestion $O(\frac{\log m}{\phi})$.
       * Boundary edges of $G\{U_i\}$ are not used for routing inside $G\{U_i\}$ anyway.
   - Now consider D^{dif}.
     - Let F_H be the flow that routes D^{dif} in H.
     - Let $F_{G_{\mathcal{U}}}$ be the flow in $G_{\mathcal{U}}$ is obtained from F_H by restricting to edges in $G_{\mathcal{U}}$ (note that $H = G_{\mathcal{U}}$ + stars).
     - Observe that $F_{G_{\mathcal{U}}}$ routes $D^{dif}_{\mathcal{U}}$ in $G_{\mathcal{U}}$.
     - $F_{G_{\mathcal{U}}}$ can be viewed a "broken flow" in G.
       * Purple flow in picture below is the flow path of $F_{G_{\mathcal{U}}}$.
       * The green edge extends the flow-path to endpoints in $V(G)$



– To complete the broken paths from $F_{G_{\mathcal{U}}}$ in G, this induces a new demand.

∗ The total black demand (from the whole flow-path except at the initial/last cluster) induces the 1-restricted demand between boundary vertices of each $G\{U_i\}$.



· Can route this black part using $O(\frac{\log n}{\alpha})$ inside each $G[U_i]$

· because $G\{U_i\}$ is α-boundary-linked.

∗ The total green demand (from the flow-path at the initial/last cluster) induces the $\deg_{G\{U_i\}}$-restricted demand between vertices inside $G\{U_i\}$.

- Can route this green part using $O(\frac{\log n}{\phi})$ inside each $G[U_i]$.
- This is just exactly situation as when we want to route demand D_i inside $G[U_i]$.

* Done by concatenation flow paths.

□

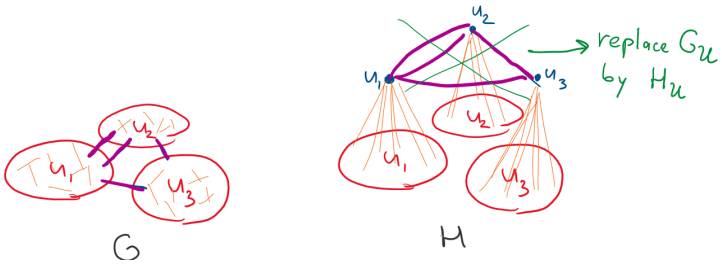
4 Recurse!

- Summary: Given any graph G and its boundary-linked ϕ-expander decomposition $\mathcal{U}$: we can get flow sparsifier of G as follows:
  - Contract graph G into $G_{\mathcal{U}}$ (the part is small $|E(G_{\mathcal{U}})| = \tilde{O}(\phi|E(G)|)$).
  - Attach the star on each expander $U \in \mathcal{U}$ (this part is very simple).
- This should inspire you to recurse on $G_{\mathcal{U}}$ to simplify this smaller graph $G_{\mathcal{U}}$ further.
- The following theorem is the key tool:

Theorem 4.1. *Let $G = (V, E)$ be a graph. Do the following:*

1. *Compute a α-boundary-linked ϕ-expander decomposition $\mathcal{U} = \{U_1, \ldots, U_k\}$ of G.*
2. *Contract each U_i into a vertex u_i. Let $G_{\mathcal{U}}$ be the contracted graph.*
3. *Let $H_{\mathcal{U}}$ be a flow sparsifier of $G_{\mathcal{U}}$ with quality q.*
4. *Let H be obtained from $H_{\mathcal{U}}$ as follows. For each $U_i \in \mathcal{U}$, attach a star rooted at u_i in $H_{\mathcal{U}}$ as in Example 2.1. More formally, for each $w \in U_i$, add (w, u_i) with capacity $\deg_G(w)$ into $H_{\mathcal{U}}$.*

Then, we have that H is a flow sparsifier of G with quality $O((\frac{q}{\alpha} + \frac{1}{\phi}) \log m)$.



- It will be very crucial that the factor q appears only at the $\frac{1}{\alpha}$ term and not $\frac{1}{\phi}$.
- Our plan:
  1. Specify the recursive algorithm more precisely. When we get is something called the **expander hierarchy**.
  2. Assuming Theorem 4.1, show that expander hierarchy is a flow sparsifier with good quality.
  3. Proof Theorem 4.1.

4.1 The recursive structure: Expander Hierarchy

- Let's be more specific what we mean by recursive algorithm.
  1. Initialize $G_0 = G$.
  2. For $i = 0, 1, \ldots$
     - Compute the decomposition $\mathcal{U}_i$ of G_i.
     - Set $G_{i+1} \leftarrow G_{\mathcal{U}_i}$.
     - If G_{i+1} has a single vertex, break.
  3. Let h be the maximum level i. Let $T_h = G_h$ be a trivial flow sparsifier of G_h.
  4. For $i = h-1, \ldots, 1$
     - Given $G_{\mathcal{U}_i}$ and a flow sparsifier T_{i+1} of $G_{\mathcal{U}_i}$, compute a flow sparsifier T_i of G_i using Theorem 4.1.
- At the end, T_0 is a flow sparsifier of G.
  - T_0 is a tree! This looks great. It is very simple.
- We call this tree T_0 **the expander hierarchy** of G.
  - If each $\mathcal{U}_i$ is an α-boundary-linked ϕ-expander decomposition of G_i.
  - Then we say that T_0 is a (α, ϕ)-expander hierarchy of G.
- In other words, expander hierarchy obtained by recursively computing α-boundary-linked ϕ-expander decomposition in the contracted graph.

4.2 Quality of Expander Hierarchy

- First, we point out why it is so crucial that
  - The quality in Theorem 4.1 is $O((\frac{q}{\alpha}+\frac{1}{\phi})\log m)$ not just $O(q(\frac{1}{\alpha}+\frac{1}{\phi})\log m) = O(\frac{q}{\phi}\log m)$.
  - Note that: $O(q(\frac{1}{\alpha}+\frac{1}{\phi})\log m)$ is quite natural to expect.
    - The non-recursive version in Lemma 3.1 gives $O((\frac{1}{\alpha}+\frac{1}{\phi})\log m)$ quality.
    - We recurse on the sparsifier of quality q. So we should pay an extra factor of q.
- Why $O(\frac{q}{\phi}\log m)$ is not good enough?
  - By induction, for each i, T_i would be a flow sparsifier of G_i with quality $O(\frac{\log m}{\phi})^{h-i}$.
  - $h = \log_{\tilde{O}(\phi)} m \geq \log_{\phi} m$ because the graph size reduces by $\tilde{O}(\phi)$ factor for each level.
  - So the quality of T_0 is $\Theta(\frac{\log m}{\phi})^h \geq \Theta(m)$... very bad.

Exercise 4.2. Given a unit-capacity graph G, show that any spanning tree of G is a flow sparsifier of G with quality $O(m)$ or $O(m \log m)$. Given a capacitated graph, show that any maximum spanning tree of G is a flow sparsifier of G with quality $O(m)$ or $O(m \log m)$.

- Why $O((\frac{q}{\alpha}+\frac{1}{\phi})\log m)$ is good enough?
  - By induction, we can prove that T_0 has quality $O((\frac{\log m}{\alpha})^h \frac{1}{\phi})$.

* Basically, the quality loss per level is now $O(\frac{\log m}{\alpha}) \ll O(\frac{\log m}{\phi})$.

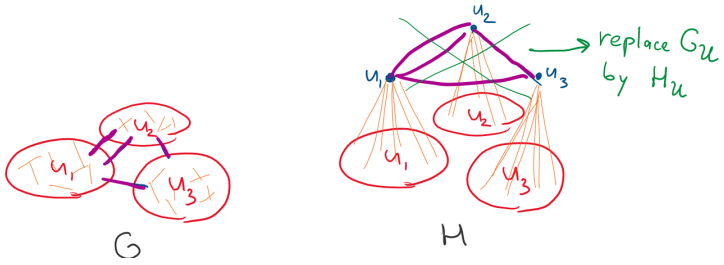
– By setting $\phi = 1/2^{\sqrt{\log m}}$ we have $h = \log_{\tilde{O}(\phi)} m = O(\sqrt{\log m} \log\log m)$.

– So the quality of T_0 is $2^{O(\sqrt{\log m}\log\log m)} = n^{o(1)}$.

- Now, we see how crucial it is that factor q appears only at the $\frac{1}{\alpha}$ term and not $\frac{1}{\phi}$.
- Let's prove it.

4.3 Proof of Theorem 4.1

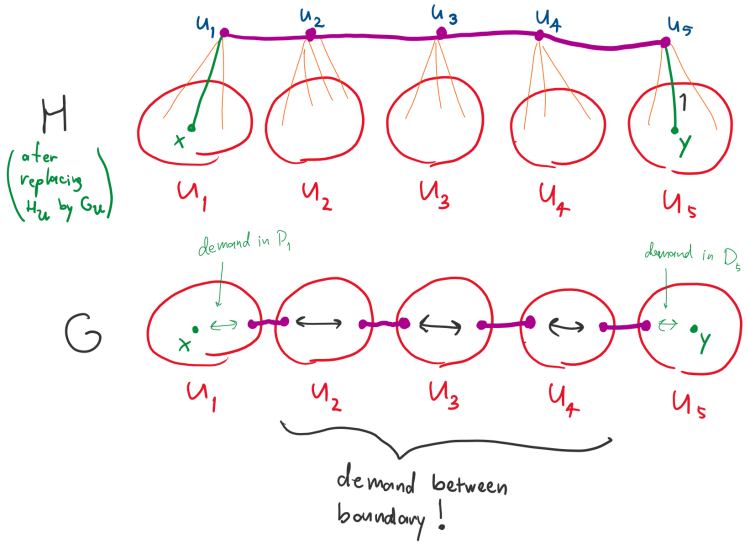
- The prove actually is quite simple. Just need to inspect the proof of the non-recursive Lemma 3.1.



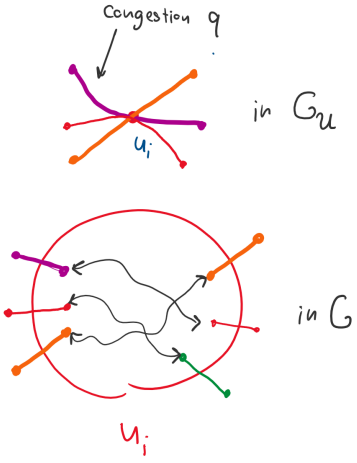
I copied the proof below and high-light the change. There are two directions:

1. Suppose D is routable in G. Then D is routable in H.
   - For each demand $(x, y) \in E(D)$ where $x \in U_i$ and $y \in U_j$, we split the demand into three parts $x \leftrightarrow u_i$, $u_i \leftrightarrow u_j$, $u_j \leftrightarrow y$.
     (a) The first/third parts, even after summing over all demand pairs, can be routed with congestion using the star edges.
     (b) For the second part, after summing over all demand pairs, this is to route $D_{\mathcal{U}}$ in $H_{\mathcal{U}}$.
       – $D_{\mathcal{U}}$ is routable in $G_{\mathcal{U}}$.
       – As $H_{\mathcal{U}}$ is a flow sparsifier of $G_{\mathcal{U}}$, $D_{\mathcal{U}}$ is routable in $H_{\mathcal{U}}$
   - The edges of H used in the two steps above are disjoint. So there is no congestion.
2. Suppose D is routable in H. Then D is routable in G with congestion $O((\frac{q}{\alpha} + \frac{1}{\phi}) \log m)$.
   - We write $D = \sum_{U_i \in \mathcal{U}} D_i + D^{dif}$ where
     – D_i contains all demand pairs (x, y) where both $x, y \in U_i$
     – D^{dif} contains all demand pairs (x, y) where x, y are in different parts.
   - Each D_i is routable in $G[U_i]$ with congestion $O(\frac{\log m}{\phi})$.
     – D_i is $\deg_{G\{U\}}$-restricted. (D_i may not be $\deg_{G[U_i]}$-restricted)
     – $G\{U_i\}$ is a ϕ-expander. So any $\deg_{G\{U\}}$-restricted demand is routable in $G\{U_i\}$ with congestion $O(\frac{\log m}{\phi})$.
     – Actually D_i is routable in $G[U_i]$ with congestion $O(\frac{\log m}{\phi})$.
       * Boundary edges of $G\{U_i\}$ are not used for routing inside $G\{U_i\}$ anyway.
   - Now consider D^{dif}.

– Let F_H be the flow that routes D^{dif} in H.
– Let $F_{H_{\mathcal{U}}}$ be the flow in $H_{\mathcal{U}}$ is obtained from F_H by restricting to edges in $H_{\mathcal{U}}$
– Observe that $F_{H_{\mathcal{U}}}$ routes $D^{dif}_{\mathcal{U}}$ in $H_{\mathcal{U}}$.
– As $H_{\mathcal{U}}$ is a flow sparsifier of $G_{\mathcal{U}}$ with quality q, there is a flow $F_{G_{\mathcal{U}}}$in $G_{\mathcal{U}}$ that routes $D^{dif}_{\mathcal{U}}$ with congestion q.
– $F_{G_{\mathcal{U}}}$ can be viewed a "broken flow" in G.
  * Purple flow in picture below is the flow path of $F_{G_{\mathcal{U}}}$.
  * The green edge extends the flow-path to endpoints in $V(G)$



– To complete the broken paths from $F_{G_{\mathcal{U}}}$ in G, this induces a new demand.
  * The total black demand (from the whole flow-path except at the initial/last cluster) induces the q-restricted demand between boundary vertices of each $G\{U_i\}$.



    · The demand is q-restricted because $F_{G_{\mathcal{U}}}$ has congestion q
    · Can route this black demand using $O(q \cdot \frac{\log m}{\alpha})$ congestion inside each $G[U_i]$
    · because $G\{U_i\}$ is α-boundary-linked.
  * The total green demand (from the flow-path at the initial/last cluster) induces the $\deg_{G\{U_i\}}$-restricted demand between vertices inside $G\{U_i\}$.

- This is not a $q \cdot \deg_{G\{U_i\}}$-restricted demand because the endpoints u_i of the flow-path in $G_{\mathcal{U}}$ receives flow equals to its demand. There is no blow-up factor of q here.
- Can route this green part using $O(\frac{\log n}{\phi})$ inside each $G[U_i]$.
- This is just exactly situation as when we want to route demand D_i inside $G[U_i]$.

* Done by concatenation flow paths.

5 Tree Flow Sparsifier

- When a cut/flow sparsifier of G is a tree, then we call it a tree cut/flow sparsifier.
- This is also called **Räcke Tree** because its first construction with polylog(n) quality was discovered by Harald Räcke[1]
- We saw that the expander hierarchy gives $n^{o(1)}$-quality tree flow sparsifier.

5.1 State of the Art

- Polynomial time (slow)
  - Tree cut sparsifier with quality $O(\log^{1.5} n \log\log n)$ (or $O(\log n \log\log n)$ for existence only)[2]
  - Tree flow sparsifier with quality $O(\log^2 n \log\log n)$[3]
- Almost linear time
  - Tree flow sparsifier with quality $O(\log^4 n)$[4]
  - Tree flow sparsifier with quality $n^{o(1)}$.[5]
    * Today: Expander hierarchy.
    * Simplest construction. Can be made dynamic.
- All known constructions guarantee that T has low depth $O(\log n)$. This is useful.
- Open:
  - $O(\log n)$ quality is possible?
  - Simple algorithm with polylog(n) quality.

[1] https://home.ttic.edu/~harry/pdf/min_congestion.pdf
[2] https://link.springer.com/chapter/10.1007%2F978-3-662-44777-2_64
[3] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.640.8407&rep=rep1&type=pdf
[4] https://epubs.siam.org/doi/pdf/10.1137/1.9781611973402.17
[5] https://arxiv.org/abs/2005.02369

6 Perspective: Trees are good

- Trees are very computational friendly.
  - Think of top tree: we can compute and maintain almost everything on trees very quickly.
- A theme in graph algorithms: compute tree representations that faithfully preserve fundamental properties of a given graph.
  - spanning forests (preserving connectivity)
  - shortest path trees (preserving distances from a source)
  - Gomory-Hu trees (preserving pairwise minimum cuts)
  - low stretch spanning trees (preserving average distances between pairs of vertices)
  - treewidth decomposition (preserving "tree-like" structure)
- Tree flow sparsifier is an astonishingly strong tree.
  - It approximately preserves the values of *all cuts*.
  - Amazing that it exists at all.
- Today, we just saw a simplest known way to compute tree flow sparsifier.

Approximate Max Flow in Near-linear Time

December 11, 2025

1 Recall some terminology

- Let $G = (V, E)$ be an undirected graph with edge capacities $c \in \mathbb{R}^E_{\geq 0}$ (with n vertices and m edges).
- A **flow** $f : V \times V \to \mathbb{R}$ satisfies $f(u, v) = -f(v, u)$ and $f(u, v) = 0$ for $\{u, v\} \notin E$.
  - The notation $f(u, v) > 0$ means that **mass** is routed in the direction from u to v.
  - The **congestion** of f is $\max_{\{u,v\} \in E} \frac{|f(u,v)|}{c(u,v)}$. If the congestion is at most 1, we say that f is **feasible** or **respects capacities.**
  - The **net flow going out of** u is $f_{out}(u) = \sum_{v \in V} f(u, v)$.
- Let $\mathsf{d} : V \to \mathbb{R}$ be a **demand function**.
  - We say that flow f **satisfies demand** d if $\mathsf{d}(u) = f_{out}(u)$ for all $u \in V$.
  - For any $S \subseteq V$, let $\mathsf{d}(S) = \sum_{v \in S} \mathsf{d}(v)$ be the **total demand** on S.
  - Assume $\mathsf{d}(V) = 0$.

Fact 1.1. *$|\mathsf{d}(S)| \leq \epsilon \cdot \delta(S)$ for all $S \subseteq V$ iff there is a flow with congestion ϵ satisfying d.*

- We say d is **feasible** if $|\mathsf{d}(S)| \leq \delta(S)$ for all $S \subseteq V$ (i.e. there is a feasible flow satisfying the demand).

2 The Approximate Max Flow Problem

- **The $(1 + \epsilon)$-approximate max flow problem**:
  - given a capacitated graph $G = (V, E, c)$ and a demand d,
  - output either
    - a cut S where $\delta(S) < \mathsf{d}(S)$
    - a flow $f \in \mathbb{R}^E$ with congestion $(1 + \epsilon)$ satisfying d

Exercise 2.1 (Routing through maximum spanning tree)**.** Show how to solve the $O(m)$-approximate max flow problem in $O(m \log n)$ time using maximum spanning trees.

- One of the key idea for solving this problem is to change the problem a bit...

2.1 "Almost Route" demand instead

- For a flow f and a demand function d, define **excess** or **residual demand** $\mathsf{d}^f(v) = \mathsf{d}(v) - f_{out}(v)$ for every $v \in V$.
- We say that f **ϵ-satisfies** d if $|\mathsf{d}^f(S)| \leq \epsilon\delta(S)$ for all $S \subseteq V$
  - That is, there exists f_{aug} with congestion ϵ where $f + f_{aug}$satisfies d.
- **The ϵ-almost-route problem**: output either
  - a cut S where $\delta(S) < |\mathsf{d}(S)|$
  - a feasible flow $f \in \mathbb{R}^E$ that ϵ-satisfies d.

Lemma 2.2. *We can solve the approximate max flow problem by solving the almost-route problem $O(\log n)$ time plus $O(m)$ additional time.*

Proof. Given graph G and demand d_0, call almost-route on (G, d_0).

- If we get a cut S where $\delta(S) < |\mathsf{d}_0(S)|$, done.
- If we get a feasible flow f_0 ϵ-satisfying d_0, we will show how to construct f satisfying d_0 with congestion $(1 + O(\epsilon))$.
  - Let $\mathsf{d}_1 \leftarrow \mathsf{d}_0^{f_0}$ be the residual demand. Call almost-route on $(\epsilon \cdot G, \mathsf{d}_1)$.
  - Can we get a cut? No.
    - We knew that $|\mathsf{d}_1(S)| \leq \epsilon \cdot \delta(S)$ for all $S \subseteq V$ (as f_0 ϵ-satisfies d_0).
  - We must get a feasible flow f_1 on $\epsilon \cdot G$ that ϵ-satisfies d_1.
    - f_1 has congestion ϵ on G.
    - $|\mathsf{d}_1^{f_1}(S)| \leq \epsilon^2\delta(S)$ for all $S \subseteq V$.
  - Let $\mathsf{d}_2 \leftarrow \mathsf{d}_1^{f_1}$. Call almost-route on $(\epsilon^2 \cdot G, \mathsf{d}_2)$ and get a feasible flow f_2 on $\epsilon^2 \cdot G$ that ϵ-satisfies d_2.
    - f_2 has congestion ϵ^2 on G.
    - $|\mathsf{d}_2^{f_2}(S)| \leq \epsilon^3\delta(S)$ for all $S \subseteq V$.
  - Repeat until we call almost-route on $(\epsilon^L \cdot G, \mathsf{d}_L)$ we get f_L where $L = O(\log m)$.

Lemma 2.3. *Consider $f = f_0 + f_1 + \cdots + f_L$. We have*

1. *f has congestion at most $1 + \epsilon + \cdots + \epsilon^L = 1 + O(\epsilon)$, and*
2. *f (ϵ^{L+1})-satisfies d_0.*

Proof. For the first statement, this is because each f_i has congestion at most ϵ^i on G.

For the second statement, for any S, we have

$$\begin{aligned}\mathsf{d}_0^f(S) &= \mathsf{d}_0(S) - f_0(S) - f_1(S) - \cdots - f_L(S)\\ &= \mathsf{d}_1(S) - f_1(S) - \cdots - f_L(S)\\ &= \mathsf{d}_L(S) - f_L(S)\\ &= \mathsf{d}_L^{f_L}(S)\end{aligned}$$

but $|\mathsf{d}_L^{f_L}(S)| \leq \epsilon^{L+1}\delta(S)$ □

□

- The residual demand d_0^f can be satisfied with a flow with congestion ϵ^{L+1}.
- We can find a flow f_{final} satisfying d_0^f with congestion $O(m) \cdot \epsilon^{L+1} \leq 1/\text{poly}(m)$.
  - How? Route through maximum spanning tree, see Exercise 2.1.
- We just return $f + f_{final}$ which has $1 + O(\epsilon)$ congestion and exactly satisfies d_0

3 First Ingredient: Congestion Approximator

- Motivation:
  - How can we guarantee that f ϵ-satisfies d?
  - There are 2^n many constraints $|\mathsf{d}^f(S)| \leq \epsilon\delta(S)$ for all $S \subseteq V$.

Definition 3.1 (Congestion Approximator)**.** An **congestion approximator of G with quality q** is a family of cuts $\mathcal{C}$ such that, for any demand vector $\mathsf{d} \in \mathbb{R}^V$ we have that if $|\mathsf{d}(S)| \leq \delta(S)$ for all $S \in \mathcal{C}$, then $|\mathsf{d}(S)| \leq q \cdot \delta(S)$ for all $S \subseteq V$.

Exercise 3.2. Let T be a tree. Let $\mathcal{C}$ contains all $n-1$ cuts defined by each tree edge. Show that $\mathcal{C}$ is a congestion approximator of T with quality 1.

Exercise 3.3. Let G be a ϕ-expander. Let $\mathcal{C} = \{\{v\}\}_{v \in V}$ contains all n singleton cuts. Show that $\mathcal{C}$ is a congestion approximator of T with quality $1/\phi$.

3.1 Construction via Tree Flow Sparsifier

Lemma 3.4. *Let T be a tree flow sparsifier of G with quality q. For every $u \in V(T)$, let $S_u \subseteq V$ denote the set of leaves in the subtree rooted at u. Let*

$$\mathcal{C}_T = \{S_u \mid u \in V(T)\}.$$

Then $\mathcal{C}_T$ is a congestion approximator of G with quality q.

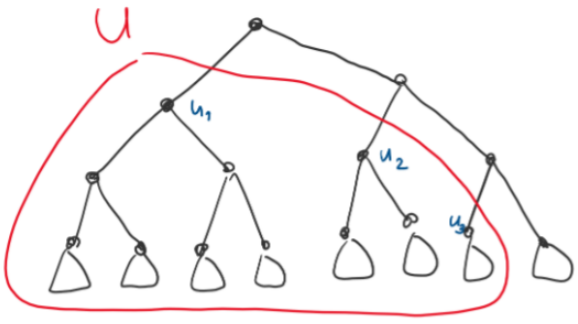
Proof. Given a demand $\mathsf{d} \in \mathbb{R}^{V(G)}$ on $V(G)$, we will prove two things

1. If $|\mathsf{d}(S)| \leq \delta(S)$ for all $S \in \mathcal{C}_T$, then d is feasible in T.
2. If d is feasible in T, then d can be routed with congestion q in G (and so $|\mathsf{d}(S)| \leq q \cdot \delta(S)$ for all $S \subseteq V$).

□

For the first step, suppose for contradiction that there is d is infeasible.

- There is a cut $U \subseteq V(T)$ where $|\mathsf{d}(U)| > \delta_T(U)$.



- But then there must exists $u \in V(T)$ where $|\mathsf{d}(S_u)| > c_T(u, parant(u)) \geq \delta(S_u)$

For the second step, suppose d is feasible in T.

- Let f_T be the feasible flow in T that routes d.
- Let F_T be a set of flow-paths in the decomposition of f_T. Think of F_T as a multi-commodity flow.
- Let D be the multi-commodity demand that F_T routes. In particular, D is routable in T.
- But then D is routable in G with congestion q, because T has is a flow sparsifier with quality q.
- So d is routable in G with congestion q.
  - For more details, let F_G be a multi-commodity flow that routes D in G.
  - Let f_G be obtained from F_G by treating F_G as a single-commodity flow (i.e. allow flow cancellation).
  - We have that f_G routes d in G.

Using the state-of-the-art algorithm for tree-flow sparsifiers, we have:

Corollary 3.5. *A congestion approximator $\mathcal{C}$ of G with quality* polylog(n) *and $|\mathcal{C}| \leq 2n$ can be constructed in $\tilde{O}(m)$ time.*[1] *Moreover, each vertex is contained in at most $O(\log n)$ sets from $\mathcal{C}$.*

We can also use the expander hierarchy from the previous class. But the quality would be $n^{o(1)}$ and the construction time would be $m^{1+o(1)}$.

3.2 Reduce the problem further...

- So we now reduce our problem to the following:
- Given a congestion approximator $\mathcal{C}$ with quality q and $|\mathcal{C}| = O(n)$, output either
  - a cut S where $\delta(S) < |\mathsf{d}(S)|$
  - a feasible flow $f \in \mathbb{R}^E$ such that for all $S \in \mathcal{C}$

$$|\mathsf{d}^f(S)| \leq \frac{\epsilon}{q}\delta(S)$$

    because this implies that f ϵ-satisfies d.

- In other words, either
  - find a violating cut, or
  - make sure that the total excess on every set $S \in \mathcal{C}$ is small (instead of all sets $S \subseteq V$).

[1] https://arxiv.org/pdf/1411.7631.pdf

4 Second Ingredient: Multiplicative Weights Update

Here, we state how the multiplicative weight update (MWU) framework can be used for solving *general* LPs.

- Given a following linear program (LP):

$$(LP) \quad \begin{aligned} &\text{Find } x \text{ s.t.} \\ &Ax \geq b \\ &x \in \Delta \end{aligned}$$

  where $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^n$, $x \in \mathbb{R}^m$, Δ is convex (e.g. $x \geq 0$).

- Define:
  - $Ax \geq b$ as a set of "hard" constraints
  - $x \in \Delta$ as an "easy" constraint

- Our goal is to find $\overline{x}$ that approximately satisfies the LP, i.e.,

$$\begin{aligned} A\overline{x} &\geq b - \epsilon \mathbf{1} \\ \overline{x} &\in \Delta \end{aligned}$$

- The MWU allows us to find $\overline{x}$ if we can solve the problem "average" problem for a few rounds.

- More precisely,
  - In each round $t = 1, 2, \ldots$, the framework will fix some $p_t \in \mathbb{R}^n_{\geq 0}$ (i.e. "weights" of hard constraints).
    - p_1 is initially the all-one vector.
  - We need to solve the following "average" LP:

$$\begin{aligned} &\text{Find } x \text{ s.t.} \\ &p_t^\top Ax \geq p_t^\top b \\ &x \in \Delta \end{aligned}$$

    Note that, the hard constraint are not average into just one constraint!
  - Suppose we can compute a feasible solution x_t of the average problem (i.e. $p_t^\top Ax_t \geq p_t^\top b$ and $x_t \in \Delta$) such that x_t has **width** at most ρ, i.e.

$$|A_i x_t - b_i| \leq \rho \forall i \in [n]$$

    That is, x_t does not violate or over-satisfy each constraint by more than ρ.
  - Given x_t, the MWU framework will adaptively update p_{t+1} based on x_t as follows:

$$(p_{t+1})_i = (p_t)_i \cdot \exp(\frac{\epsilon}{\rho} \cdot (b_i - A_i x_t))$$

    and proceed to the next round.

    - Intuition: If the i-th constraint is violated a lot by x_t ($A_i x_t \ll b_i$), then the weight $(p_{t+1})_i$ increases a lot. So in the next round, the i-th constraint should be satisfied.
    - From the way we update weights, this give the name *multiplicative weight update* framework.

- After $T = O(\rho^2 \log n/\epsilon^2)$ rounds, the MWU framework guarantees that $\overline{x} = (\sum_{t=1}^{T} x_t)/T$ satisfies

$$A\overline{x} \geq b - \epsilon \mathbf{1}$$
$$\overline{x} \in \Delta$$

- It is quite an amazing framework: reducing a worst-case problem to an average problem.

5 The Algorithm

Recall our goal: either find

- a violating cut S where $\delta(S) < |\mathsf{d}(S)|$, or
- a feasible f where $|\mathsf{d}^f(S)| \leq \epsilon/q \cdot \delta(S)$ for all $S \in \mathcal{C}$.

To formulate this goal as an LP (so that we can apply MWU), let's define some notations.

- Let $\epsilon' = \epsilon/q$.
- Let $B \in \{0, -1, 1\}^{V \times E}$ be the incidence matrix of G.
    - Note that, for every v, we have $f_{out}(v) = (Bf)_v$
- For any $S \subseteq V$, let $1_S \in \mathbb{R}^V$ be the indicator vector of S.
- We have

$$\begin{aligned}
&|\mathsf{d}^f(S)| \leq \epsilon' \cdot \delta(S) && \iff \\
&1_S^\top(\mathsf{d} - Bf) \leq \epsilon' \cdot \delta(S) \text{ and } 1_S^\top(\mathsf{d} - Bf) \geq -\epsilon' \cdot \delta(S) && \iff \\
&\frac{1}{\delta(S)}\mathbf{1}_S^\top Bf \geq \frac{1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} - \epsilon' \text{ and } -\frac{1}{\delta(S)}1_S^\top Bf \geq -\frac{1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} - \epsilon' &&
\end{aligned}$$

We can rewrite our goal: either find

- a violating cut S where $\delta(S) < \mathsf{d}(S)$, or
- a flow f where
    1. for all e, $|f_e|/c_e \leq 1$
    2. for all $S \in \mathcal{C}$, $\frac{1}{\delta(S)}\mathbf{1}_S^\top Bf \geq \frac{1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} - \epsilon'$ and $-\frac{1}{\delta(S)}1_S^\top Bf \geq -\frac{1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} - \epsilon'$ for all $S \in \mathcal{C}$.

So now, let's write an LP

$$\begin{aligned}
&\text{Find } f \text{ s.t.} \\
&\frac{1}{\delta(S)}\mathbf{1}_S^\top Bf \geq \frac{1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} && \text{for all } S \in \mathcal{C} && (1)\\
&\frac{-1}{\delta(S)}\mathbf{1}_S^\top Bf \geq \frac{-1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} && \text{for all } S \in \mathcal{C} && (2)\\
&|f_e|/c_e \leq 1 && \text{for all } e \in E && (3)
\end{aligned}$$

We want to find a violating cut S or find f that satisfies the above LP upto ϵ' additive factor. We are now ready to apply MWU:

- 1 and 2 are "hard" constraints.
- 3 are "easy" constraints.

In round t of MWU, the framework sets the weights $\{p_{S,\circ}\}_{S\in\mathcal{C},\circ\in\{+,-\}}$ of each hard constraint where $p_{S,\circ} \geq 0$.

- Given the weights, we have need to solve the following average LP

$$\begin{aligned}
&\text{Find } f \text{ s.t.} \\
&\sum_{S\in\mathcal{C}} \frac{p_{S,+}}{\delta(S)}\mathbf{1}_S^\top Bf - \sum_{S\in\mathcal{C}} \frac{p_{S,-}}{\delta(S)}\mathbf{1}_S^\top Bf \geq \sum_{S\in\mathcal{C}} \frac{p_{S,+}}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} - \sum_{S\in\mathcal{C}} \frac{p_{S,-}}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} \\
&|f_e|/c_e \leq 1 && \text{for all } e \in E
\end{aligned}$$

- Define potentials on vertices as

$$\phi = \sum_{S\in\mathcal{C}} \frac{p_{S,+}}{\delta(S)}\mathbf{1}_S - \sum_{S\in\mathcal{C}} \frac{p_{S,-}}{\delta(S)}\mathbf{1}_S \in \mathbb{R}^V.$$

- We need to solve

$$\begin{aligned}
&\text{Find } f \text{ s.t.} \\
&\phi^\top Bf \geq \phi^\top \mathsf{d} \\
&|f_e|/c_e \leq 1 && \text{for all } e \in E
\end{aligned}$$

  - To understand ϕ more intuitively, observe that

$$\phi_v = \sum_{S\ni v} \frac{1}{\delta(S)}(p_{S,+} - p_{S,-})$$

  - Each v is contained in $O(\log n)$ sets from $\mathcal{C}$, so ϕ can be computed $O(n \log n)$ time.
    - Actually, we can compute ϕ in $O(n)$ time when $\mathcal{C}$ is defined from a tree flow sparsifier. **(Exercise)**

- To solve this, it suffices to find f^t that maximizes $\phi^\top Bf^t$ where $\max_e |f_e^t|/c_e \leq 1$. (Then, check if $\phi^\top Bf^t \geq \phi^\top \mathsf{d}$).

– For any f, we can expand the expression as

$$\phi^\top Bf = \sum_{e=(u,v)} (\phi_v - \phi_u) f_{(u,v)}$$

– To maximize each term, just set $f^t_{(u,v)} = c_{(u,v)}\text{sgn}(\phi_v - \phi_u)$.

– We can find the optimal f^t where $\phi^\top Bf^t = \sum_{e=(u,v)} c_e|\phi_u - \phi_v|$.

- There are two cases
  - If $\phi^\top Bf^t < \phi^\top d$, we claim that we can find a violating cut. Done (to be proved below).
  - If $\phi^\top Bf^t \geq \phi^\top d$, then the flow f^t satisfies the average problem in round t. We can feed f^t into the MWU framework and proceed to the next round.
  - How many round? This depends on the width of f^t. (How much f^t violates or over-satisfies the original constraints?)
  - **Claim**: the width of f^t is at most 2.
    * We want to prove that for all $S \in \mathcal{C}$

$$|\frac{1}{\delta(S)}\mathbf{1}_S^\top Bf - \frac{1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d}| \leq 2.$$

    * We have $\mathbf{1}_S^\top Bf = \sum_{e\in E(S,V-S)} f_e \leq \sum_{e\in E(S,V-S)} c_e = \delta(S)$. So $|\frac{1}{\delta(S)}\mathbf{1}_S^\top Bf| \leq 1$.
    * Also, we have $|\frac{1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d}| = \frac{\mathsf{d}(S)}{\delta(S)} \leq 1$ because we can actually assume that $d_S \leq \delta(S)$ for each $S \in \mathcal{C}$ (this can be easily checked fast from the very beginning), otherwise we obtain a violating cut.
  - So there are $T = O(\log n/\epsilon'^2) = O(\frac{q^2}{\epsilon^2}\log n) = \tilde{O}(1/\epsilon^2)$ rounds.
  - After T rounds, we get $\overline{f} = \frac{f_1+\cdots+f_T}{T}$ where

$$\begin{aligned}
\frac{1}{\delta(S)}\mathbf{1}_S^\top B\overline{f} &\geq \frac{1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} - \epsilon' && \text{for all } S \in \mathcal{C}\\
\frac{-1}{\delta(S)}\mathbf{1}_S^\top B\overline{f} &\geq \frac{-1}{\delta(S)}\mathbf{1}_S^\top \mathsf{d} - \epsilon' && \text{for all } S \in \mathcal{C}\\
|\overline{f}_e|/c_e &\leq 1 && \text{for all } e \in E
\end{aligned}$$

  - So $\overline{f}$ is a feasible flow that ϵ-satisfies d. Done.

- It remains to prove that $\phi^\top Bf^t < \phi^\top d$, then there is a violating cut.

Lemma 5.1. *If there is $\phi \in \mathbb{R}^V$ where $\sum_{e=(u,v)} c_e|\phi_u - \phi_v| < \phi^\top \mathsf{d}$, then we can find a threshold cut $S_\tau = \{u \mid \phi_u < \tau\}$ such that $\delta(S_\tau) < \mathsf{d}(S_\tau)$ in $O(n)$ time.*

Proof. Rearrange the indices so that $\phi_1 \leq \phi_2 \leq \cdots \leq \phi_n$. □

- We can translate $\phi_i \leftarrow \phi_i - \phi_1$ so $\phi_1 = 0$. Why?
  - $\sum_{e=(u,v)} c_e|\phi_u - \phi_v|$ is translation invariant.
  - $\phi^\top \mathsf{d}$ is changed to $\phi^\top \mathsf{d} - \phi_0 \mathsf{d}(V) = \phi^\top \mathsf{d}$.

- We can assume $\phi_n = 1$ by scaling.
- Now, choose a random threshold $\tau \in [0, 1]$. Let $S_\tau = \{u \mid \phi_u \geq \tau\}$.

$$\begin{aligned}\mathbb{E}_\tau[\delta(S_\tau)] &= \sum_{e=(u,v)\in E} c_e \Pr[\phi_u \leq \tau < \phi_v] \\ &= \sum_{e=(u,v)} c_e |\phi_u - \phi_v|\end{aligned}$$

  and

$$\begin{aligned}\mathbb{E}_\tau[\mathsf{d}(S_\tau)] &= \sum_{v\in V} \phi_v \mathsf{d}(v) \\ &= \phi^\top \mathsf{d}\end{aligned}$$

- So we have

$$\mathbb{E}_\tau[\delta(S_\tau)] < \mathbb{E}_\tau[\mathsf{d}(S_\tau)]$$

  and so there exists τ where $\delta(S_\tau) < \mathsf{d}(S_\tau)$.
- Let's conclude the algorithm
  - In each round, given $\{p_{S,\circ}\}_{S\in\mathcal{C},\circ\in\{+,-\}}$, compute ϕ in $O(n)$ time.
  - for each edge $(u, v) \in E$, send flow at full capacity from v to u if $\phi_v > \phi_u$, and vice versa. Total time: $O(m)$.
  - There are $O(\frac{q^2}{\epsilon^2} \log n)$ rounds. So total time is $\tilde{O}(m/\epsilon^2)$.
- Output: either
  - a violating cut S where $\delta(S) < |\mathsf{d}(S)|$, or
  - a feasible f where $|\mathsf{d}^f(S)| \leq \epsilon/q \cdot \delta(S)$ for all $S \in \mathcal{C}$.
    - This implies that f ϵ-satisfies d.

6 Intuition: What happen in Expanders?

- Try to interpret this algorithm on ϕ-expanders.
- Here, the congestion approximator is very simple: all singleton cuts give a congestion approximator with quality $\frac{1}{\phi}$.
  - Why? See 3.3.
- When all the cuts from the congestion approximator are singletons, this allows us to interpret the algorithm more easily.
- From how MWU works, observe that $p^{t+1}_{v,+} = p^t_{v,+} \cdot \exp(\frac{\epsilon}{\rho} \cdot \frac{\mathsf{d}^{f^t}(v)}{\deg(v)})$ and $p^{t+1}_{v,-} = p^t_{v,-} \cdot \exp(\frac{\epsilon}{\rho} \cdot \frac{-\mathsf{d}^{f^t}(v)}{\deg(v)})$. So

$$p^t_{v,+} = \exp(\frac{\epsilon}{\rho} \cdot \sum_{t=1}^{t} \frac{\mathsf{d}^{f^i}(v)}{\deg(v)})$$

$$p^t_{v,-} = \exp(\frac{\epsilon}{\rho} \cdot \sum_{t=1}^{t} \frac{-\mathsf{d}^{f^i}(v)}{\deg(v)})$$

- Define $\overline{f}^t = \frac{f_1+\cdots+f_t}{t}$ as the average flow so far up to time t. Recall that our algorithm returns $\overline{f} = \overline{f}^T = \frac{f_1+\cdots+f_T}{T}$ where $T = O(\frac{\rho^2}{\epsilon^2}\log n)$.
  - By definition, we have $\sum_{i=1}^t \mathsf{d}^{f^i}(v) = t \cdot \mathsf{d}^{\overline{f}^t}(v)$.
  - $\mathsf{d}^{\overline{f}^t}(v)$ has a natural interpretation: it is the average excess at vertex v up to time t.
  - Our goal is that the average excess should be close to 0 by time T. That is, $\mathsf{d}^{\overline{f}^T}(v) \approx 0$.
- Now, the potential of v at round t is

$$\phi_v^t = \frac{p_{v,+}^t - p_{v,-}^t}{\deg(v)} = \frac{\exp(\frac{\epsilon}{\rho} \cdot t \cdot \frac{\mathsf{d}^{\overline{f}^t}(v)}{\deg(v)}) - \exp(\frac{\epsilon}{\rho} \cdot t \cdot \frac{-\mathsf{d}^{\overline{f}^t}(v)}{\deg(v)})}{\deg(v)}$$

- Up to scaling (also in the exponent)

$$\phi_v^t \sim e^{\mathsf{d}^{\overline{f}^t}(v)} - e^{-\mathsf{d}^{\overline{f}^t}(v)}$$

- Interpretation:
  - When $\mathsf{d}^{\overline{f}^t}(v) > 0$, then ϕ_v^t goes up positively exponentially fast.
  - When $\mathsf{d}^{\overline{f}^t}(v) < 0$, then ϕ_v^t goes down negatively exponentially fast.
- This make senses. Why?
  - Recall our we construct the flow in each round. For every edge (u, v), if $\phi_u > \phi_v$, then send flow at full capacity from u to v. Otherwise, do the opposite direction.
  - So if v has excess ($\mathsf{d}^{\overline{f}^t}(v) > 0$), then ϕ_v is big. So the algorithm would likely send flow out of v in the next round.
  - So if v has deficit ($\mathsf{d}^{\overline{f}^t}(v) < 0$), then ϕ_v is small. So the algorithm would likely send flow into v in the next round.
- At the end, we expect $\mathsf{d}^{\overline{f}}(v) \approx 0$ for all v.
  - More precisely, $|\mathsf{d}^{\overline{f}}(v)| \le \epsilon \deg(v)$ for all $v \in V$.
  - This implies $|\mathsf{d}^{\overline{f}}(S)| \le \frac{\epsilon}{\phi}\delta(S)$ for all $S \subseteq V$. So $\overline{f}$ $\frac{\epsilon}{\phi}$-satisfies d.
- This intuitively is similar to push-relabel algorithms.
  - We maintain levels and flow from higher vertices to lower vertices.

7 History and State of the art

- Nice book by David Williamson: http://www.networkflowalgs.com/
- 60-70's
  - Ford-Fulkerson
  - $O(mn^2)$ Blocking flow (Dinic)

  - $O(m \min\{m^{1/2}, n^{2/3}\})$ time in unit capacity (Even-Tarjan, Karzanov)

- 80's
  - $O(mn \log n)$ Blocking flow + dynamic tree (Sleator Tarjan)
  - $O(mn \log n)$ Push relabel (Goldberg Tarjan)

- 90's
  - $\tilde{O}(m \min\{m^{1/2}, n^{2/3}\})$ in general graph (Goldbeg Rao)

- In sparse graphs, $n^{1.5}$ is the best since 70's... but there was a breakthrough in approximation algorithms in

- Approximation in edge-capacitated graphs
  - $\tilde{O}(mn^{1/3})$ Electrical flow + MWU + Arc Boosting (Christiano et al '10 [2])
  - $\tilde{O}(m)$ Congestion approximator + MWU (Sherman'13[3], KLOS'13[4], Peng'16[5])

- Approximation in vertex-capacitated graphs
  - $O(m^{1+o(1)})$ Dynamic shortest path + MWU (Bernstein Gutenberg **S**'21[6])

- Exact (based on Interior Point Method + Dynamic algorithms)
  - $\tilde{O}(m + n^{1.5})$ (BLLSSSW'21[7])
  - $O(m^{4/3+o(1)})$ unit-capacity [Kathuria'20] [Lui Sidford'20] https://www.youtube.com/watch?v=VF3EbC

- My belief:
  - Exact max flow in $\tilde{O}(m)$ time (in 5-10 years).
  - Maybe in your PhD thesis.

[2] https://arxiv.org/abs/1010.2921
[3] https://arxiv.org/pdf/1304.2338.pdf
[4] https://arxiv.org/pdf/1304.2077.pdf
[5] https://arxiv.org/abs/1411.7631
[6] https://arxiv.org/pdf/2101.07149.pdf
[7] https://arxiv.org/pdf/2101.05719.pdf

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 6: Cut/Flow Equivalences Among Expanders

September 16, 2021

Instructor: Thatchaphol Saranurak

Scribe: Tian Zhang

1 Overview and Preliminaries

As is discussed in the previous lectures, an expander behaves like complete graph (the most well-connected graph) such that it is **robust against deletions**. This lecture is going to show expanders are the most well-connected graphs in the senses of **cut and flow**.

We first introduce some definitions.

Definition 1.1. Given a graph $G = \{V, E\}$ and a function $\boldsymbol{d} : V \to \mathbb{R}$, we say G has **degree profile** $\boldsymbol{d}$ if $\boldsymbol{d}(u) = \deg_G(u), \forall u \in V$. And graph is $\boldsymbol{d}$**-restricted** if $\deg_G(u) \leq \boldsymbol{d}(u), \forall u \in V$.

Definition 1.2. The $\boldsymbol{d}$**-product graph** $K = \{E, V\}$ is a complete graph such that the capacity/weight of each edge (u, v) is $\kappa_K(u, v) = \frac{\boldsymbol{d}(u)\boldsymbol{d}(v)}{\boldsymbol{d}(V)}$.

- Note that function $\boldsymbol{d}$ is exactly the degree profile of K, since $\forall u \in V, deg_G(u) = \sum_{v \in V} \boldsymbol{d}(u)\frac{\boldsymbol{d}(v)}{\boldsymbol{d}(V)} = \boldsymbol{d}(u)$.
- when $\forall u \in V$, $\mathbf{d}(u) = 1$, the graph is a normalized complete graph where each edge has weight $\frac{1}{n}$.

Definition 1.3. Given graphs $G = \{V, E\}$ and $H = \{V, E'\}$, $H \preccurlyeq^{\mathbf{cut}} G$ if $\kappa_H(\partial_H S) \leq \kappa_G(\partial_G S), \forall S \subseteq V$. $H \preccurlyeq^{\mathbf{flow}} G$ if H is embeddable into G with no congestion. And $H \preccurlyeq^{\mathbf{deg}} G$ if $\deg_H(u) \leq \deg_G(u)$ for all u.

2 Product Graphs and Their Cut/Flow Connectivity

Our goal of this section is to show that among all $\boldsymbol{d}$-restricted graph, the $\boldsymbol{d}$-product graph K is the most well-connected graph w.r.t. both $\preccurlyeq^{\text{cut}}$ and $\preccurlyeq^{\text{flow}}$.

2.1 Cut Connectivity of Product Graphs

Every cut of product graph is almost maximally big. Formally, we have the following lemma.

Lemma 2.1. *For all $\boldsymbol{d}$-restricted graphs H and the $\boldsymbol{d}$-product graph K, we have $H \preccurlyeq^{\text{cut}} 2K$.*

Proof. For any cut S where $\boldsymbol{d}(S) \le \boldsymbol{d}(V \setminus S)$, we have $\kappa_H(\partial_H S) \le \text{vol}_H(S) = \sum_{u\in S} \deg_H(u) \le \sum_{u\in S} \deg_K(u) = \boldsymbol{d}(S)$ since H is $\boldsymbol{d}-$restricted.

For each $u \in S$, we have $\kappa_K(E_K(u, V\setminus S)) = \sum_{v\notin S} \frac{\boldsymbol{d}(u)\boldsymbol{d}(v)}{\boldsymbol{d}(V)} = \boldsymbol{d}(u)\sum_{v\notin S} \frac{\boldsymbol{d}(v)}{\boldsymbol{d}(V)} = \boldsymbol{d}(u)\frac{\boldsymbol{d}(V\setminus S)}{\boldsymbol{d}(V)} \ge \boldsymbol{d}(u)/2$. So $\kappa_K(\partial_K S) \ge \boldsymbol{d}(S)/2$. □

- Note: this is the same as proving that $\Phi(K, H) \ge 1/2$.

2.2 Flow Connectivity of Product Graphs

All $\boldsymbol{d}$-restricted demands are routable on the $\boldsymbol{d}$-product graph. To see that, let us first get some intuition and discuss about routing demands with small congestion on arbitrary graphs.

Given any $\boldsymbol{d}$-restricted demand H and an arbitrary graph G,

- If there is a node u in G where $\deg_G(u) \ll \boldsymbol{d}(u)$, then clearly routing some H in G must cause large congestion.
- What if $\deg_G(u) \ge \boldsymbol{d}(u)$ for all u? Is this enough to route H in G in general? No, for example...

However, if G is a $\boldsymbol{d}$-product graph, condition "$\deg_G(u) \ge \boldsymbol{d}(u), \forall u$" is enough to make demand H routable in G.

Lemma 2.2. *All $\boldsymbol{d}$-restricted demands H are routable in the $\boldsymbol{d}$-product graph K with congestion 2, i.e. $H \preccurlyeq^{\text{flow}} 2K$.*

Proof. To embed H into K, we construct the following flow $\mathcal{F} = \{f_{(s,t)}\}_{(s,t)\in E(H)}$: For each $(s,t) \in E(H)$, the flow $f_{(s,t)}$ send flow through the two-hop paths $P_{svt} = (s, v, t)$ with value $f(P_{svt}) = \kappa_H(s,t) \cdot \frac{\boldsymbol{d}(v)}{\boldsymbol{d}(V)}$. Note that $\|f_{(s,t)}\| = \sum_v \kappa_H(s,t) \cdot \frac{\boldsymbol{d}(v)}{\boldsymbol{d}(V)} = \kappa_H(s,t)$. So, $\mathcal{F}$ indeed embeds H into K.

Now, we bound the congestion when we implement flow $\mathcal{F}$ in K. For each edge $(u, v) \in E(K)$, it is contained in flow paths $\{P_{suv}\}_{s\in V}$ and $\{P_{uvt}\}_{t\in V}$ with total flow value

$$\sum_{s\in V} \kappa_H(s,v)\cdot\frac{\boldsymbol{d}(u)}{\boldsymbol{d}(V)} + \sum_{t\in V} \kappa_H(u,t)\cdot\frac{\boldsymbol{d}(v)}{\boldsymbol{d}(V)} = \frac{\deg_H(v)\boldsymbol{d}(u)}{\boldsymbol{d}(V)} + \frac{\deg_H(u)\boldsymbol{d}(v)}{\boldsymbol{d}(V)} \le \frac{\boldsymbol{d}(v)\boldsymbol{d}(u)}{\boldsymbol{d}(V)} + \frac{\boldsymbol{d}(u)\boldsymbol{d}(v)}{\boldsymbol{d}(V)} \le 2\kappa_K(u,v)$$

□

Therefore, $\mathcal{F}$ embeds H into K with congestion 2.

3 Cut/Flow Connectivity of Expanders

Given an expander G, if you want to prove that $H \preccurlyeq^{\text{cut}} G$ or $H \preccurlyeq^{\text{flow}} G$ (within some constant factor), you can just show that **degree of G is big enough**! Formally, we have the following lemma.

Lemma 3.1. *Let G be ϕ-expander. Let H be any graph where $H \preccurlyeq^{\deg} G$. Then, $H \preccurlyeq^{\text{cut}} \frac{1}{\phi} G$.*

Proof. For any $S \subset V$ where $\text{vol}_G(S) \leq \text{vol}_G(V \setminus S)$, we have that

$$\kappa_H(\partial_H S) \leq \text{vol}_H(S) \leq \text{vol}_G(S)$$

because $H \preccurlyeq^{\deg} G$. At the same time we have

$$\kappa_G(\partial_G S) \geq \phi \text{vol}_G(S).$$

So $\kappa_H(\partial_H S) \leq \frac{1}{\phi}\kappa_G(\partial_G S)$ and so $H \preccurlyeq^{\text{cut}} \frac{1}{\phi} G$. □

By the approximate max-flow min-cut theorem, we know that $H \preccurlyeq^{\text{cut}} \frac{1}{\phi} G$ implies $H \preccurlyeq^{\text{flow}} \frac{O(\log n)}{\phi} G$. So we have

Corollary 3.2. *Let G be ϕ-expander. Let H be any graph where $H \preccurlyeq^{\deg} G$. Then, $H \preccurlyeq^{\text{flow}} \frac{O(\log n)}{\phi} G$.*

4 Cut/Flow Connectivity Equivalence of Expanders

We are going to show that expanders with same degree profile are approximately equivalent in some sense. We first formalize notations of equivalence.

4.1 Notations of Equivalence

Definition 4.1. G_1 and G_2 are **α-cut-equivalent** if there are $\alpha_1, \alpha_2 > 0$ where $\alpha_1 \cdot \alpha_2 \leq \alpha$ such that $G_2 \preccurlyeq^{\text{cut}} \alpha_1 G_1$ and $G_1 \preccurlyeq^{\text{cut}} \alpha_2 G_2$. Denote this by $G_1 \approx_\alpha^{\text{cut}} G_2$.

- Our definition is designed such that it measures how much the cut size of G_1 and G_2 are similar *modulo some scaling*. See the example below.

Example 4.2. G_1 and $G_2 = 100 \cdot G_1$ are 1-cut-equivalent, because $G_2 \preccurlyeq^{\text{cut}} 100 G_1$ but $G_1 \preccurlyeq^{\text{cut}} \frac{1}{100} G_2$.

Definition 4.3. G_1 and G_2 are **α-flow-equivalent** if there are $\alpha_1, \alpha_2 > 0$ where $\alpha_1 \cdot \alpha_2 \leq \alpha$ such that $G_2 \preccurlyeq^{\text{flow}} \alpha_1 G_1$ and $G_1 \preccurlyeq^{\text{flow}} \alpha_2 G_2$. Denote this by $G_1 \approx_\alpha^{\text{flow}} G_2$.

- Our definition is designed such that it measures the congestion when we try to embed G_1 and G_2 into each other.

Exercise 4.4. Show that:

- If $G_1 \approx_\alpha^{\text{flow}} G_2$, then $G_1 \approx_\alpha^{\text{cut}} G_2$.
- If $G_1 \approx_\alpha^{\text{cut}} G_2$, then $G_1 \approx_{O(\alpha \log^2 n)}^{\text{flow}} G_2$.

4.2 Equivalence and Well-Connectivity of Expanders

Based on previous conclusions given by simple proofs, We can obtain statements of strong conceptual messages.

In a sentence, **Expanders with Degree Profile d are all equivalent in the sense of cut/flow connectivity and they are the most well-connected d-restricted graphs.**

First, we have that $\tilde{\Omega}(1)$-expanders with the same degree profile are approximately equivalent to each other w.r.t. both $\preccurlyeq^{\text{cut}}$ and $\preccurlyeq^{\text{flow}}$.

Corollary 4.5. *Let G and G' be ϕ-expanders with degree profile $\boldsymbol{d}$. We have*

- *$G \preccurlyeq^{\text{cut}} \frac{1}{\phi} G'$ and $G' \preccurlyeq^{\text{cut}} \frac{1}{\phi} G$. So G and G' are $O(\frac{1}{\phi^2})$-cut-equivalent.*
- *$G \preccurlyeq^{\text{flow}} O(\frac{\log n}{\phi}) G'$ and $G' \preccurlyeq^{\text{flow}} O(\frac{\log n}{\phi}) G$. So G and G' are $O(\frac{\log^2 n}{\phi^2})$-flow-equivalent.*

Notice first that this statement is robust against perturbation of $\boldsymbol{d}-$ even if G and G' have different degree profile within a factor of some constant, which is going to add some extra factor on the equivalence factor, they are still $O\left(\frac{1}{\phi^2}\right)$-cut- and $O\left(\frac{\log^2 n}{\phi^2}\right)$-flow-equivalent.

This Corollary shows that, in other words, $\tilde{\Omega}(1)-$expanders with the same degree profile form a $\tilde{O}(1)$-cut- and -flow-equivalence class, where everything within the class have a similar cut and flow structure. Moreover, the most well-connected $\boldsymbol{d}$-restricted graph, i.e. the $\boldsymbol{d}$-product graph K, is also in this equivalence class, since it is an $\Omega(1)-$expander with degree profile $\boldsymbol{d}$. The approximation factor is a bit better too.

Corollary 4.6. *Let G be ϕ-expander with degree profile $\boldsymbol{d}$. Let K be the $\boldsymbol{d}$-product graph. We have*

- *$G \preccurlyeq^{\text{cut}} 2K$ and $K \preccurlyeq^{\text{cut}} \frac{1}{\phi} G$. So G and K are $O(\frac{1}{\phi})$-cut-equivalent.*
- *$G \preccurlyeq^{\text{flow}} 2K$ and $K \preccurlyeq^{\text{flow}} O(\frac{\log n}{\phi}) G$. So G and K are $O(\frac{\log n}{\phi})$-flow-equivalent.*

This is nice because G can have much fewer edges than K, yet G is as "good" as K.

Because of this, any expander with degree profile $\boldsymbol{d}$ is approximately the most well-connected $\boldsymbol{d}$-restricted graph w.r.t. both $\preccurlyeq^{\text{cut}}$ and $\preccurlyeq^{\text{flow}}$.

Corollary 4.7. *Let G be a ϕ-expander with degree profile $\boldsymbol{d}$.*

- *For all $\boldsymbol{d}$-restricted graphs H, we have $H \preccurlyeq^{\text{cut}} O(\frac{1}{\phi}) G$.*
- *For all $\boldsymbol{d}$-restricted graphs H, we have $H \preccurlyeq^{\text{flow}} O(\frac{\log n}{\phi}) G$.*

4.3 Certificate For Well-Connectivity via Expanders

- Lastly, suppose you want to show that some arbitrary $\boldsymbol{d}$-restricted graph G' is the most well-connected among all $\boldsymbol{d}$-restricted graphs.
- What is the *certificate* for this?

Corollary 4.8. *Let G be a ϕ-expander with degree profile $\boldsymbol{d}$. Let G' be any $\boldsymbol{d}-$restricted graph. If $G \preccurlyeq^{\text{flow}} G'$, then $H \preccurlyeq^{\text{flow}} O(\log(n)/\phi) \cdot G'$ for all $\boldsymbol{d}$-restricted demand H.*

- That is, a feasible flow embedding that embeds any expander G into G' is the certificate.

5 Summary

Let's fix the degree profile $\boldsymbol{d}$. We want to compare "well-connectivity" of graphs with this degree profile $\boldsymbol{d}$.

- First, we prove that the $\boldsymbol{d}$-product graph K is basically the most well-connected graph. In both cut and flow perspective, we have

  $$H \preccurlyeq^{\text{cut}} 2K \text{ and } H \preccurlyeq^{\text{flow}} 2K$$

  for any graph H where $\deg_H \leq \boldsymbol{d}$ (i.e. $\boldsymbol{d}$-restricted graph).

- Second, we show that if $H \preccurlyeq^{\deg} G$ and G is a ϕ-expander, then $H \preccurlyeq^{\text{cut}} \frac{1}{\phi} G$.

  - Therefore, for any two $\tilde{\Omega}(1)$-expanders G_1 and G_2 with the same degree profile $\boldsymbol{d}$ (i.e. $G_1 =^{\deg} G_2$), we have

    $$G_1 \approx^{\text{cut}} G_2$$

  - Combining with the approximate max-flow min-cut theorem (i.e. $H \preccurlyeq^{\text{cut}} G \overset{\approx}{\iff} H \preccurlyeq^{\text{flow}} G$), we have

    $$G_1 \approx^{\text{flow}} G_2$$

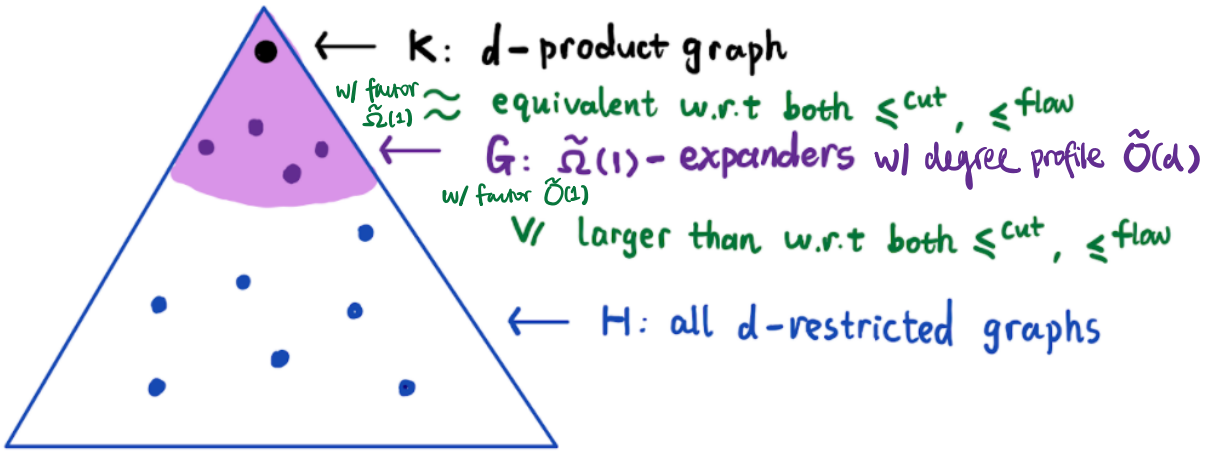
  - That is, expanders with the same degree profile are all approximately equivalent from both cut and flow perspectives.
  - Note that K is in the equivalent class too (as K is a $\Omega(1)$-expander).

- Now, as K is the most well-connected graph w.r.t $\preccurlyeq^{\text{cut}}$ and $\preccurlyeq^{\text{flow}}$, the whole class of expanders are basically most well-connected as well.

  - That is, for any $\tilde{\Omega}(1)$-expander G and a $\boldsymbol{d}$-restricted graph H, we have

    $$H \lessapprox^{\text{cut}} G \text{ and } H \lessapprox^{\text{flow}} G$$

- This is the picture you should have in mind



- To summarize, if you know $H \preccurlyeq^{\deg} G$ and G is an expander, then $H \lessapprox^{\text{cut}} G$ and $H \lessapprox^{\text{flow}} G$.
  - This is important.
  - For both flow and cut, all that "matters" for expanders is to compare degree cuts (when we allow some approximate)
- Lastly, suppose you want to show that some graph G' is the most well-connected among graphs with the same degree profile $\boldsymbol{d}$.
  - It is enough to show $K \lessapprox^{\text{cut}} G'$ or $K \lessapprox^{\text{flow}} G'$.
  - But it is enough to show $G \lessapprox^{\text{cut}} G'$ or $G \lessapprox^{\text{flow}} G'$ for **any** expander G too.
  - That is, if you can embed an expander G into G', then you are done.

6 The "Right" Way to Think of Expanders

- Let $G = (V, E)$ be a ϕ-expander.
  - By the previous discussion, if G has uniform degree, then G is cut/flow-equivalent to the corresponding $\boldsymbol{d}$-product graph, which is essentially the complete graph on V with unit weight on each edge (after scaling by some constant.) Hence we can think of G as an unweighted complete graph on V, and this is a nice picture to have in mind.
  - But, otherwise, it is equivalent to the $\deg_G$-product graph, which is quite hard to think about since it has different weights on different nodes.
- I find it much more convenient to think of an expander in the following way.
- Suppose that $G = (V, E)$ is unweighted for now (otherwise make parallel edges).
- Let G' be obtained from G by subdividing each edge $e = (u, v)$ into (u, x_e) and (x_e, v).
  - We call x_e the *split node* of e.
  - Let $X_E = \{x_e \mid e \in E\}$.
  - So $V(G') = V(G) \cup X_E$.
- Here is the "better" way to think about an expander. Let K_E be a **unweighted clique** on X_E (no edges on V).
  - Note: every node in $\frac{1}{m-1} K_E$ has degree 1.
- Essentially, G **is an expander iff an unweighted clique on split-nodes** X_E **(after scaling by** $\frac{1}{m}$**) is embeddable into** G'**.** Namely, we can think of G as an unweighted complete graph on X_E.

Intuitively this is saying that for a graph to be an expander, all of its edges should be pairwise "well connected", in the sense that there should be a collection of flows on G' that allows any edge in G to send a unit of flow to any other edge in G while this collection has small congestion. In this case X_E is representing all edges in G, and embedding K_E into G' is finding the collection of flows that we described above. To show this perspective is true, we state the following lemma formally,

Lemma 6.1. *We have:*

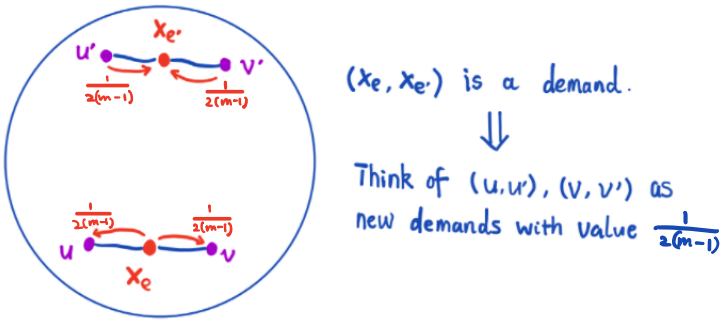
1. *If G is a ϕ-expander, then $\frac{1}{m-1}K_E \preccurlyeq^{\text{flow}} O(\frac{\log n}{\phi}) \cdot G'$.*
2. *If G is not a ϕ-expander, then $\frac{1}{m-1}K_E \not\preccurlyeq^{\text{flow}} \frac{1}{100\phi} \cdot G'$.*

Corollary 6.2. *For any 1-restricted graph H_E where $V(H_E) = X_E$, we have $H_E \preccurlyeq^{\text{flow}} O(\frac{\log n}{\phi}) \cdot G'$. That is, every edge can exchange 1 unit of flow between any other edges.*

Now, we prove the lemma.

Proof. There are two directions.

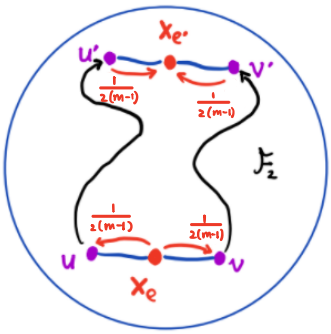
1. Suppose G is a ϕ-expander.
   - You just to construct a $\frac{1}{m-1}K_E$-concurrent flow $\mathcal{F}$ with small congestion.
   - We will construct $\mathcal{F}$ in several steps.
     (a) Fix a demand $(x_e, x_{e'})$ where $e = (u, v)$ and $e' = (u', v')$,
     (b) Think of this demand as directed demand so that we can draw the flow, but all we need is just a undirected flow.
     (c) x_e sends flow to u, v (each of value $1/2(m-1)$)
     (d) $x_{e'}$ receive flow from u', v' (each of value $1/2(m-1)$)



     (e) This induces new demand. Indeed, consider the demand graph $H = (V, E_H, \kappa_H)$ where $E_H = \{(u, v)\}_{u,v \in V(E)}$. Essentially H is a clique on V, and we will discuss the construction of κ_H in the next step.
     (f) We still look at the fixed demand $(x_e, x_{e'})$ first. For this demand $(x_e, x_{e'})$, we can treat sending $\frac{1}{m-1}$ unit of demand between x_e and $x_{e'}$ as a two-step process where firstly (1)x_e sending $\frac{1}{2(m-1)}$ unit of demand between $x_e$and u, x_e and v respectively; $x_{e'}$ sending $\frac{1}{2(m-1)}$ unit of demand between $x_{e'}$and u', $x_{e'}$ and v' respectively, then (2)sending $\frac{1}{2(m-1)}$ unit of demand between u, u' and v, v' respectively. In this way, we separate the demand sending process for all possible pairs $(x_{e_1}, x_{e_2}), x_{e_1}, x_{e_2} \in X_E$, and accumulate the unit of demand on each edge in E_H to form κ_H.
     Notice that the total demand each original node $u \in V$ needs to send/receive is $\deg_G(u)/2$, i.e. $\deg_H(u) = \deg_G(u)/2$. This is because u is adjacent to $\deg_G(u)$ vertices in X_E where each vertex is involved in $(m-1)$ edges in demand graph K_E, and each edge(representing $1/(m-1)$ unit of demand need to pass through) induces $\frac{1}{2(m-1)}$ unit of demand got send between u and some other vertex.

(g) Therefore, H is a $\frac{\deg_G}{2}$-restricted demand graph. We know $H \preccurlyeq^{\text{flow}} O(\frac{\log n}{\phi})G$. As $G \preccurlyeq^{\text{flow}} G'$, we have $H \preccurlyeq^{\text{flow}} O(\frac{\log n}{\phi}) \cdot G'$.

(h) So there is an H-concurrent flow $\mathcal{F}_2$ that embeds H into G' with congestion $q_2 = O(\frac{\log n}{\phi})$.



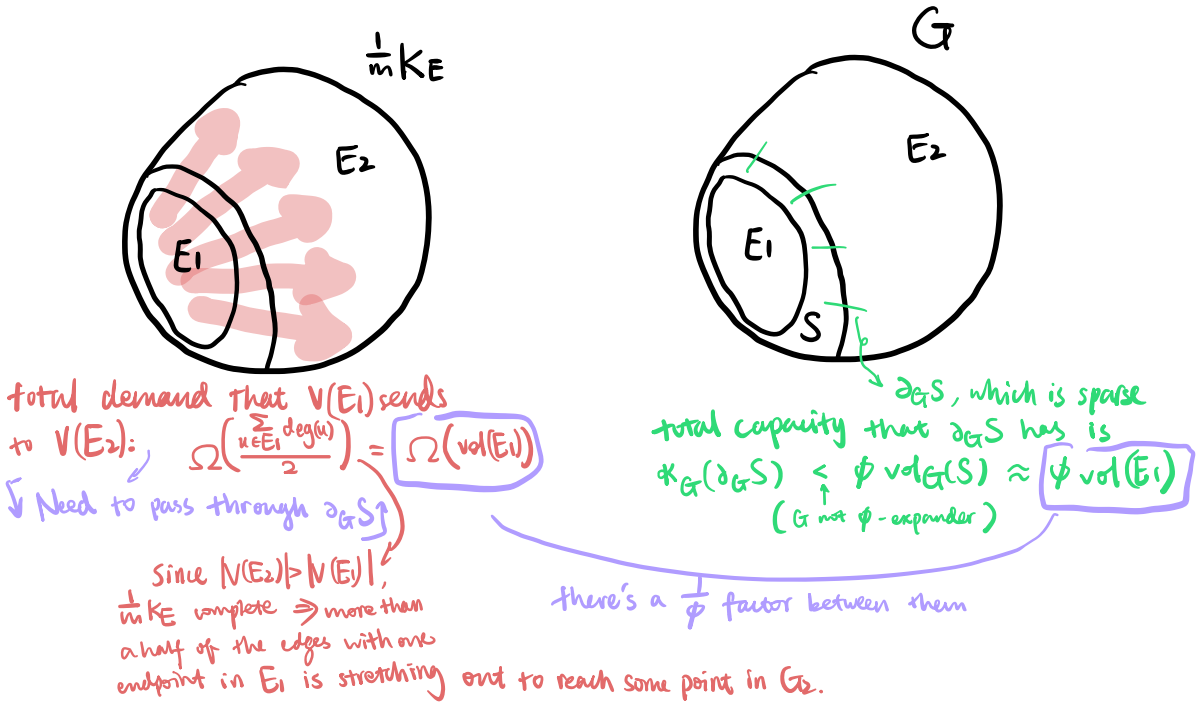
(i) Concatenate the $\mathcal{F}_2$ with the "red" flow for every demand. That would satisfy all demand of $\frac{1}{m-1}K_E$.

(j) What is the congestion? It is $1/2 + q_2 = O(\frac{\log n}{\phi})$. It is not just q_2! Why? This is because it's possible that there might be a flow in $\mathcal{F}_2$ with the largest congestion overlap with the flow between an endpoint in $V(G)$ and a split point in X_E on the edge with congestion q_2. In that case, the congestion on that edge should be $q_2 + 1/2$ where $1/2$ is the congestion generated by the flow between one endpoint to its adjacent split node.

(k) We have a $\frac{1}{m-1}K_E$-concurrent flow with congestion $O(\frac{\log n}{\phi})$ in G'.

2. Suppose G is not a ϕ-expander, where $\phi < 1/10$.

- Let $(S, \overline{S})$ be a cut such that $\text{vol}_G(S) \leq \text{vol}_G(\overline{S})$ and $\kappa_G(\partial_G S) < \phi \text{vol}_G(S)$.
- Consider $E_1 = E_G(S, S)$ and $E_2 = E_G(\overline{S}, \overline{S})$. Note that $|E_1| = \Theta(\text{vol}_G(S))$.
- The total demand that E_1 need to send to E_2 is $\Theta(|E_1|)$. Any concurrent flow for this demand must go through the cut S which has capacity $\kappa_G(\partial_G S)$.
- So the congestion must be least $\Theta(|E_1|)/\kappa_G(\partial_G S) = \Omega(1/\phi)$ (or $1/100\phi$ if we compute the constant explicitly).



□

- The proof above is quite simple for flow.
- You can give a tighter bound for cut. But it is a bit more tedious. This is an example where it can be easier to think about flow instead of cut (although they are essentially equivalent).

Exercise 6.3. Show that $\Phi(G) = \Theta(\Phi(G'))$.

Exercise 6.4. We have:

- If G is a ϕ-expander, then $K_E \preccurlyeq^{\text{cut}} \frac{100}{\phi} \cdot G'$.
- If G is not a ϕ-expander, then $K_E \not\preccurlyeq^{\text{cut}} \frac{1}{100\phi} \cdot G'$.

7 Optional: Computing Quality of Cut/Flow Equivalence

- Question: Given G and H, what is the smallest α where H and G are α-cut/flow-equivalent?

7.1 Cut

For any graph G and H,

- H and G are α-cut-equivalent where $\alpha = 1/(\Phi(G,H)\cdot\Phi(H,G)) \ge 1$.
- H and G are not α'-cut-equivalent for any $\alpha' < \alpha$.

Proof. Verify that $\Phi(G,H) \cdot H \preccurlyeq^{\text{cut}} G$ and $\Phi(H,G) \cdot G \preccurlyeq^{\text{cut}} H$. □

- As we know how to $O(\log n)$-approximate both $\Phi(G,H)$ and $\Phi(H,G)$ (even $O(\sqrt{\log n}\log\log n)$-approximation algorithm is known), we have the following:

Corollary 7.1. *Given graphs G and H, there is a polynomial time algorithm that* $\text{poly}\log(n)$*-approximates the minimum α cut-equivalence factor of G and H.*

Question 7.2 (Very Interesting Open Problem)**.** *Is there a polynomial time or even subexponential time algorithm that $O(1)$-approximates the minimum α cut-equivalence factor between G and H?*

7.2 Flow

For any graph G and H,

- H is a α-flow sparsifier of G where $\alpha = 1/(\mathsf{mcf}(G,H) \cdot \mathsf{mcf}(H,G)) \geq 1$.
- Moreover, H is not a α'-flow sparsifier of G for any $\alpha' < \alpha$.

Proof. Verify that $\mathsf{mcf}(G,H) \cdot H \preccurlyeq^{\text{cut}} G$ and $\mathsf{mcf}(H,G) \cdot G \preccurlyeq^{\text{cut}} H$. □

- In contract to the cut case, we can compute the embeddability factor α exactly because $\mathsf{mcf}(G,H)$ and $\mathsf{mcf}(H,G)$ are computable via LPs.

Corollary 7.3. *Given graphs G and H, there is an polynomial algorithm that exactly computes the minimum α flow-equivalence factor between G and H.*

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 12: Local Min Cuts and Isolating Cuts

October 7, 2021

Instructor: Thatchaphol Saranurak

Scribe: Yaowei Long

1 Overview

In this lecture, we are going to introduce two new techniques for minimum cut problems. The first technique is called *local min cuts*, aiming at finding some small cut without reading the whole graph. The second technique is called *isolating cuts*. Given a set of terminals in a graph, the algorithm for isolating cuts will, for each terminal, find a cut separate it from the other terminals. These two techniques can lead to efficient algorithms for some graph problems. For example, global mincut in directed graphs can be found efficiently using local min cuts, and isolating cuts can be applied to minimum Steiner cuts problem in undirected graphs.

2 Local min cuts

Given a large directed graph $G = (V, E)$ and a vertex $x \in V$ called seed, with access to the adjacency list of each vertex, a local min cut algorithm can determine if there is a small cluster containing x in time roughly proportional to the size of this cluster. In this note, a cluster is a sparse cut, which is essentially a vertex set L s.t. there are few edges going from L to $V \setminus L$.

Formally speaking, for a seed x, let $k, \nu < m/k$ be given parameters about the cut. The local min cut algorithm $\mathsf{Local}(x, \nu, k)$ will either

- declare that no cut L s.t. $x \in L$, $\delta_{\text{out}}(L) < k$ and $\text{vol}(L) \leq \nu$, where $\delta_{\text{out}}(L)$ denotes the number of edges going out L and $\text{vol}(L)$ is the volume[1] of L, or
- return a cut L' s.t. $x \in L'$, $\delta_{\text{out}}(L') < k$ and $\text{vol}(L') \leq O(\nu k)$.

The running time of $\mathsf{Local}(x, \nu, k)$ is $O(\nu k^2)$, and it will give a correct result with constant probability[2].

[1] The volume of a vertex set L is the total degree (ignoring directions of edges) of vertices in L.

[2] We can amplify the correctness in $\log(\nu, k)$ time to make this algorithm correct with high probability w.r.t. ν and k.

2.1 Warm-up

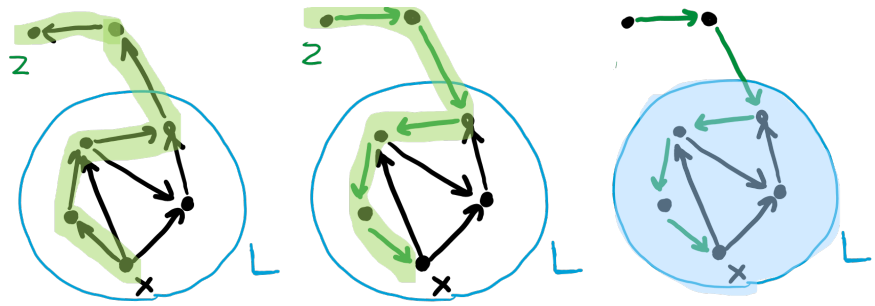
First, we look at a toy example. We assume that there is a cut L that includes x has $\mathrm{vol}(L) \leq v$, $\delta_{\mathrm{out}}(L) = 1$ and $\delta_{\mathrm{in}}(L) = 0$. In other words, there is only one edge going out from L to $V \setminus L$ and no edge going into L from $V \setminus L$.

Roughly speaking, the idea is as follows. See figure 1.

(1) We start a DFS from x. Suppose that we find a simple path P starting from x to z with length $v + 1$ in this DFS.

(2) Because $\delta_{\mathrm{in}}(L) = 0$ and the length of P is larger than $\mathrm{vol}(L)$, we know z is outside L. By reversing all edges in P, we have $\delta_{\mathrm{out}}(L) = 0$ in the new graph.

(3) Start another DFS from x in the new graph and let's say these explored vertices make up a cut L'. This DFS will not explored more than v vertices and $\mathrm{vol}(L') \leq v$, because at the worst case we will get stuck after exploring the whole L. Because we only reverse edges on a simple path, we can also guarantee $\delta_{\mathrm{out}}(L') \leq 1$.



(a) Step (1) (b) Step (2) (c) Step (3)

Figure 1: The algorithm for the toy example

The intuition behind this example is that, for a cut L and a seed $x \in L$, if we can find a path P from x to some $z \notin L$, reversing edges in P will decrease $\delta_{\mathrm{out}}(L)$ by 1 but make $\delta_{\mathrm{in}}(L)$ unchanged. Once $\delta_{\mathrm{out}}(L)$ becomes 0, a DFS from x will find the cut L (or even a smaller one).

2.2 The algorithm

In this subsection we will introduce the whole algorithm. Roughly speaking, if we want to detect a cut L with $\delta_{\mathrm{out}}(L) < k$ and $\mathrm{vol}(L) \leq v$, we will repeat the above subroutine k times. Suppose that such a cut L exists. In each of the first $k - 1$ iterations, we will find a path from x to some $z \notin L$ with some good probability. Then in the kth round, we will have $\delta_{\mathrm{out}}(L) = 0$ with some constant probability and L can be detected by a DFS.

The formal description is as follows. The algorithm $\mathsf{Local}(x, v, k)$ will repeat the following k times. See figure 2

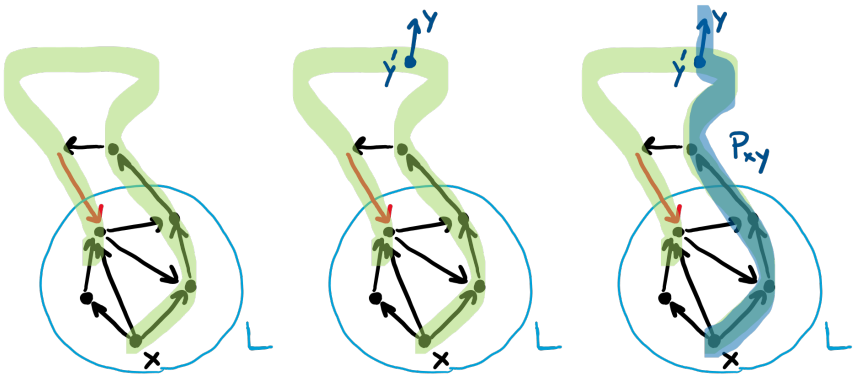
(1) Grow a DFS tree T from x by exploring exactly $kv + 1$ edges. Let L' be all explored vertices.

(2) If the DFS can only explore less then $kv + 1$ edges, we return L' and the whole algorithm terminates.

(3) Sample an edge (y', y) uniformly from the DFS tree.

(4) Reverse the path from x to y in T.

If the algorithm didn't terminate in all k iterations, it outputs there is no cut L s.t. $x \in L$, $\delta_{\text{out}}(L) < k$ and $\text{vol}(L) \leq v$.



(a) Step (1) (b) Step (3) (c) Step (4)

Figure 2: The local min cut algorithm

Lemma 2.1 (Time complexity). *The running time of* Local(x, v, k) *is* $O(vk^2)$.

Proof. Observe that in each iteration, the DFS takes $O(vk)$ time. Therefore, the total running time of k iterations is $O(vk^2)$. □

Lemma 2.2 (Soundness). *If* Local(x, v, k) *return a cut* L'*, then* $\delta_{\text{out}}(L') < k$ *and* $\text{vol}(L') = O(vk)$.

Proof. We can immediately get $\text{vol}(L') \leq 2vk = O(vk)$ from the algorithm. Also observe that $\delta_{\text{out}}(L') = 0$ at the moment when L' is returned. Since there are less then k paths are reversed before and each reversed path will decrease $\delta_{\text{out}}(L')$ by at most 1, we get $\delta_{\text{out}}(L') < k$ initially. □

Lemma 2.3 (Completeness). *If there exists* L *s.t.* $\delta_{\text{out}}(L) < k$ *and* $\text{vol}(L) \leq v$*, then the algorithm will return a cut* L' *with probability at least* $1/10$.

Proof. Let say $\delta_{\text{out}}(L) = k' < k$ initially. In each of the first k' iterations, the edge (y', y) sampled in step (3) satisfies $y', y \notin L$ with probability $\frac{vk+1-v}{vk+1} \geq 1 - 1/k$, because there are $vk + 1$ edges in the DFS tree but only at most v of them will have endpoints in L. Therefore, the probability of $y \notin L$ for all the first k' iterations is at least

$$(1 - 1/k)^{k'} \geq (1 - 1/k)^k \geq 1/10.$$

since $k' < k$. In this case, $\delta_{\text{out}}(L) = 0$ at the $(k' + 1)$th iteration and the algorithm must return some L' since $\text{vol}(L) \leq v \leq vk + 1$. □

In summary, the algorithm introduced in this subsection can solve the local min cut problem efficiently. It is a randomized algorithm with errors only on one side, namely, it always guarantees the correctness of the returned cut L'. Therefore, the probability in lemma 2.3 can be amplified by repeating this algorithm. Essentially, the local min cut algorithm is a localized and randomized version of the Ford-Fulkerson algorithm.

2.3 Applications: Global Mincut in Directed Graphs

In the global mincut problem, given a directed graph $G = (V, E)$ with m edges and n vertices and a parameter k, we need to output a cut $(S, V \setminus S)$ where $|E(S, V \setminus S)| < k$ (if exists). Here $E(S, V \setminus S)$ denote edges from S to $V \setminus S$.

A following trivial algorithm can solve this problem. First we fix some source s and then compute an (s, t)-mincut in G and the reverse graph G^R for all $t \in V$. One of such (s, t)-mincuts should be a global mincut. This algorithm needs $\Theta(n)$ calls to max flows, which is $\Omega(mn)$ time. In what follows, we will introduce a efficient new algorithm for small k using local min cut technique. Precisely, the running time will be $\tilde{O}(mk^2)$.

We consider two main cases and solve them by different algorithms.

2.3.1 Balanced Case

In this case, we assume that $\text{vol}(S), \text{vol}(V \setminus S) \geq m/k$, i.e. the cut $(S, V \setminus S)$ is balanced.

We repeat the following $T = O(k \log n)$ times.

- Sample s and t with probability proportional to its degree (i.e. v is chosen with probability $\deg(v)/(2m)$).
- If (s, t)-mincut (A, B) has size less than k, return (A, B). Note that we only decide whether the size of (s, t)-mincut is less then k rather than the exact size of (s, t)-mincut. Therefore, a typical Ford-Fulkerson algorithm can handle this step in $O(mk)$ time.

The total time for this case is $O(mkT) = \tilde{O}(mk^2)$.

Lemma 2.4. *Suppose there exists a cut $(S, V \setminus S)$ of size less than k where $\text{vol}(S), \text{vol}(V \setminus S) \geq m/k$, then the above algorithm returns a cut (A, B) of size less than k with probability $1 - 1/n^{10}$.*

Proof. Without loss of generality, we assume S is the smaller side. We have

$$\Pr[s \in S \text{ and } t \in V \setminus S] = \Pr[s \in S] \cdot \Pr[t \in V \setminus S] = m/k/(2m) \cdot 1/2 \geq 1/(4k).$$

When $s \in S$ and $t \in V \setminus S$, the algorithm will return a cut of size less then k. After repeating $T = O(k \log n)$ times, a cut of size less than k will be returned with probability $1 - 1/n^{10}$. □

2.3.2 Unbalanced Cases

In this case, there exists a cut $(S, V \setminus S)$ of size less than k where $\text{vol}(S) < m/k$ or $\text{vol}(V \setminus S) < m/k$. Without loss of generality, we assume that S is the smaller side. We can run the same algorithm in the reverse graph G^R to handle the case that $V \setminus S$ is smaller. Let's say $\text{vol}(S) = v < m/k$.

We are going to detect S using the local min cut algorithm. Note that the local min cut algorithm works only when $v < m/k$. That's the reason why we consider these two cases.

Assume that we know v in advance. The algorithm will repeat the following $T = O(\frac{m}{v} \log n)$ times.

- Sample x probability proportional to its degree (i.e. x is chosen with probability $\deg(x)/(2m)$).
- If $\mathsf{Local}(x, v, k)$ return a cut (A, B) has size less than k, return (A, B). Note that here we hope that $\mathsf{Local}(x, v, k)$ is correct with high probability (such as $1 - 1/n^{100}$), so the running time of this step is $O(vk^2 \log n)$ by repeating the local min cut algorithm above $O(\log n)$ times.

The total running time is $O(vk^2 \log nT) = \tilde{O}(mk^2)$

Lemma 2.5. *Suppose there exists a cut $(S, V \setminus S)$ of size less than k where $\mathrm{vol}(S) = v < m/k$, then the above algorithm returns a cut (A, B) of size less than k with probability $1 - 1/n^{10}$.*

Proof. For each iteration, $\Pr[x \in S] = \mathrm{vol}(S)/\mathrm{vol}(V) = v/2m$. By repeating $T = O(\frac{m}{v} \log n)$ times, the probability that $x \in S$ in at least one iteration is $1 - 1/n^{100}$. The local min cut algorithm in this iteration will returns a cut of size less than k with probability $1 - 1/n^{100}$. Therefore, the probability of this algorithm being correct is $(1 - 1/n^{100})^2 > 1 - 1/n^{10}$. □

The last problem is that we don't know $v = \mathrm{vol}(S)$. However, we don't need to know the exact v, we can try $v = 2^i$ for all $1 \le i \le \log m$. As long as $v/2 \le \mathrm{vol}(S) \le v$, the above algorithm will still work by adjusting some constant.

2.3.3 Conclusions

In summary, let $(S, V \setminus S)$ be the cut with size less than k. If $\mathrm{vol}(S), \mathrm{vol}(V \setminus S) \ge m/k$, the algorithm for the balanced case will detect a cut with size less than k with high probability. If $\mathrm{vol}(S) < m/k$, then the algorithm for the unbalanced case will detect a desired cut with high probability. Therefore, if no sparse cut is detected, then with high probability there is no sparse cut in G.

3 Isolating Cuts

Consider an unweighted **undirected** graph $G = (V, E)$ with n vertices and m edges.

Definition 3.1. Let $A, B \subseteq V$ be some vertex sets. A vertex set S is an (A, B)**-cut** if $A \subseteq S$ and $B \subseteq V \setminus S$. A (A, B)**-mincut** is an (A, B)-cut with the cut size $\delta(S)$ minimized.

Given a vertex set $T \subseteq V$ (T is called a terminal set and vertices in T are called terminals), an isolating cuts algorithm will return, for each terminal $v \in T$, a cut S_v which is a $(v, T \setminus v)$-mincut. We also call S_v a **minimum** v**-isolating cut**. Any $(v, T \setminus v)$-cut is a v**-isolating cut**.

We can design a trivial algorithm for this problem by directly computing $(v, T \setminus v)$-mincut for each terminal v, which will invoke $|T|$ max flows. In what follows, we will introduce an algorithm only invoke $O(\log |T|)$ max flows. Strictly speaking, the following algorithm will run max flows on lots of small graphs and the total number of edges in these graph is $O(\log |T| m)$.

3.1 The algorithm

Initially we name vertices in T from $1, \ldots, |T|$ using binary strings of $\lceil \log |T| \rceil$.

The algorithm is as follows. First, for each $i = 1, \ldots, \lceil \log |T| \rceil$

- For each terminal v, color v **red** if the i-th bit of v is 0. Otherwise, color it **blue**.
- Let R_i and B_i denote the set of red and blue nodes.
- Compute the edge set $E_i \subseteq E$ crossing (R_i, B_i)-mincut C_i. See figure 3.

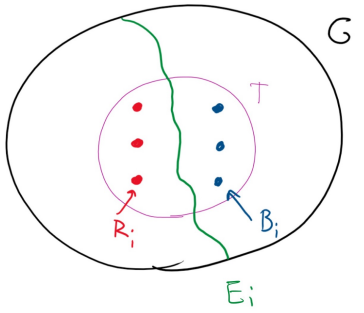


Figure 3: (R_i, B_i)-mincut

Note that deleting $\cup_i E_i$ "partitions" G into connected components where each component contains at most one terminal.

Indeed, for each $v \in T$, let $U_v := \cap_{i:\text{the } i\text{-th bit of } v \text{ is } 0} C_i \cap_{j:\text{the } j\text{-th bit of } v \text{ is } 1} V \setminus C_j$, then U_v is the set of vertices of the connected component in $G \setminus (\cup_i E_i)$ containing v, and formally,

Lemma 3.2. *$U_v \cap T = \{v\}$. That is, U_v isolates v from $T \setminus v$.*

Proof. By definition 3.1, $R_i \subseteq C_i$ and $B_i \subseteq V \setminus C_i$ for all i.

First, because $v \in R_i$ for all i s.t. the i-th bit of v is 0 and $v \in B_i$ for all i s.t. the i-th bit of v is 1, by the defition of U_v we immediately get $v \in U_v$.

Second, there is no another $v' \neq v$ in U_v. For each $v' \neq v$, v' and v will have a different bit. That is $\exists i \in [\lceil \log |T| \rceil]$,without loss of generality, assume $v \in R_i$ and $v' \in B_i$. We have $U_v \subseteq C_i$ and $v' \in B_i \subseteq V \setminus C_i$. Therefore, $v' \notin U_v$.

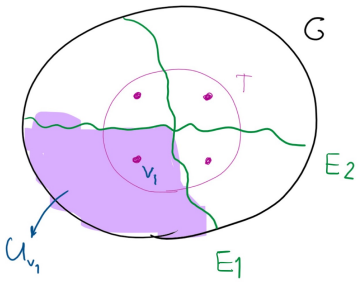


Figure 4: U_{v_1} for $v_1 \in T$, where T is a terminal set with 4 terminals

□

Before we continue introducing the algorithm, we first talk about the submodularity of cuts, which is useful to prove some properties of isolating mincuts.

Lemma 3.3 (Submodularity of cuts)**.** *Let $A, B \subseteq V$,*

$$\delta(A \cap B) + \delta(A \cup B) \leq \delta(A) + \delta(B).$$

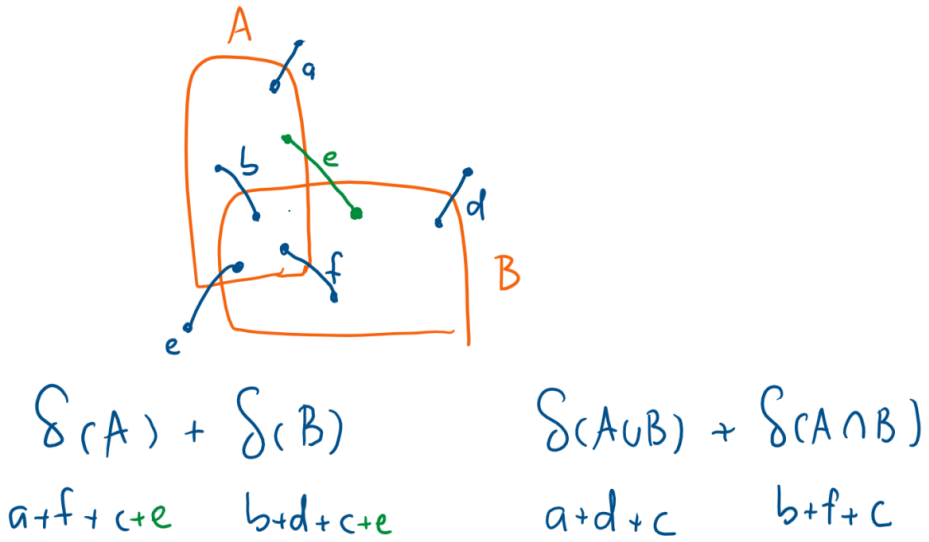


Figure 5: Proof of lemma 3.3

Proof. See figure 5. We can partition V into 4 disjoint regions $A \cap B$, $A \setminus B$, $B \setminus A$, $V \setminus (A \cup B)$, and partition edges into 6 types by considering regions of endpoints. Observe that only edges with two endpoints in $A \setminus B$ and $B \setminus A$ respectively will contribute 1 to $\delta(A)$ and $\delta(B)$ but nothing to $\delta(A \cup B)$ and $\delta(A \cap B)$. Other types of edges will contribute to two sides equally. □

Remark 3.4. If $E(A \setminus B, B \setminus A) = \emptyset$, $\delta(A \cap B) + \delta(A \cup B) = \delta(A) + \delta(B)$.

Corollary 3.5. *If S_1 and S_2 are (A, B)-mincuts, then $S_1 \cap S_2$ and $S_1 \cup S_2$ are also (A, B)-mincuts.*

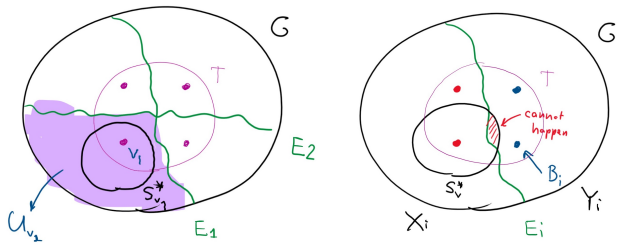
Proof. First observe that if S_1 and S_2 are (A, B)-cuts, then $S_1 \cap S_2$ and $S_1 \cup S_2$ are also (A, B)-cuts. Therefore, $\delta(S_1 \cap S_2), \delta(S_1 \cup S_2) \geq \delta(S_1) = \delta(S_2)$. By lemma 3.3, we immediately have $S_1 \cap S_2$ and $S_1 \cup S_2$ are (A, B)-mincuts. □

Definition 3.6. S_v^* is a inclusion-wise minimal $(v, T \setminus v)$-mincut if each $(v, T \setminus v)$-mincut S_v has $S_v^* \subseteq S_v$.

Lemma 3.7. *There exists a unique inclusion-wise minimal $(v, T \setminus v)$-mincut S_v^*.*

Proof. If there are two $(v, T \setminus v)$-mincut S_v' and S_v'' where $v \in S_v', S_v''$ and neither $S_v' \subseteq S_v''$ nor $S_v'' \subseteq S_v'$, then $S_v' \cap S_v''$ will be a $(v, T \setminus v)$-mincut by the submodularity of cuts (see lemma 3.3 and corollary 3.5). Therefore, let S_v^* be the intersection of all $(v, T \setminus v)$-mincuts and it will be the unique inclusion-wise minimal $(v, T \setminus v)$-mincut. □

Lemma 3.8. $S_v^* \subseteq U_v$.



(a) $S_v^* \subseteq U_v$ (b) $S_v^* \subseteq X$

Figure 6: Proof of lemma 3.8

Proof. See figure 6. $\forall i \in [\lceil \log |T| \rceil]$, let $X_i = C_i$ if $v \in R_i$ and let $X_i = V \setminus C_i$ if $v \in B_i$. Then $U_v = \bigcup_i X_i$. We are going to prove $S_v^* \subseteq X_i$ for all i, which immediately implies $S_v^* \subseteq U_v$.

Suppose for contradiction that $\exists i \in [\lceil \log |T| \rceil]$ s.t. S_v^* is not a subset of X_i. By the submodularity, we have

$$\delta(S_v^* \cap X_i) + \delta(S_v^* \cup X_i) \leq \delta(S_v^*) + \delta(X_i)$$

Since $S_v^* \cap X_i$ is a $(v, T \setminus v)$-cut and S_v^* is the inclusion-wise minimal $(v, T \setminus v)$-mincut, we have $\delta(S_v^* \cap X_i) > \delta(S_v^*)$.

Therefore, $\delta(S_v^* \cup X_i) < \delta(X_i)$. Because $S_v^* \cup X_i$ is a (R_i, B_i)-cut (or (B_i, R_i)-cut), this gives a contradiction.

□

By lemma 3.8, we only need to find a $(v, T \setminus v)$-mincut inside U_v. Formally speaking, for each terminal $v \in T$, we construct a graph G_v by contracting $V \setminus U_v$ into a single vertex t_v, and then compute a (v, t_v)-mincut in G_v as S_v.

Lemma 3.9. *Each S_v is a minimum v-isolating cut.*

Proof. S_v is a $(v, T \setminus V)$-cut because $S_v \subseteq U_v$. Since S_v^* is a (v, t_v)-mincut in G_v, so $\delta(S_v) \leq \delta(S_v^*)$ and S_v is a $(v, T \setminus v)$-mincut in G. □

Lemma 3.10. *The above algorithm of isolating cuts will run max flows on graphs with totally $O(m \log |T|)$ edges.*

Proof. In the first step, we need to compute $\log |T|$ red-blue mincuts in G, which means we run max flows on graphs with $O(m \log |T|)$ edges.

In the second step, we will compute a (v, t_v)-mincut for each $v \in T$. Since $\sum_{v \in T} |E(G_v)| \leq 2|E(G)| = O(m)$, this step we will run max flows on graphs with $O(m)$ edges. □

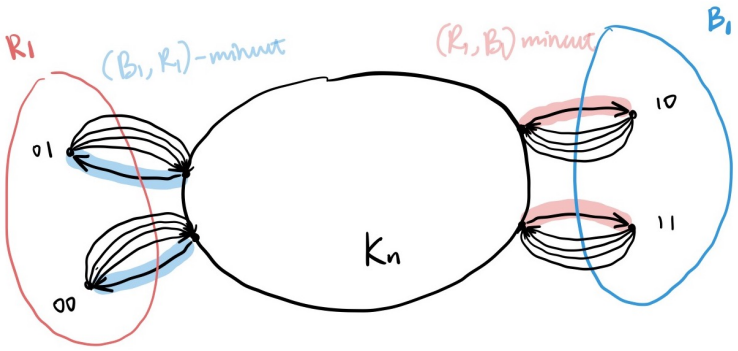
3.2 What goes wrong in directed graphs?

Let S_v^* be the inclusion-wise minimal $(v, T \setminus v)$-mincut, that is $\forall S_v$ where $v \in S_v$ and $\delta_{out}(S_v)$ is minimized, we have $S_v^* \subseteq S_v$. Then, by submodularity of δ_{out}, we can show the existence of S_v^*.

However, for each $i \in [\lceil \log |T| \rceil]$, given R_i and B_i, (R_i, B_i)-mincut and (B_i, R_i)-mincut might be different cuts even with different size and it's not clear which one we should take. Indeed,

- If we compute both of them and take for each v, U_v to be the weakly connected component containing v in G after deleting both cuts' edges, then it's possible that $\sum_{v\in T} U_v = \Omega(mn)$ whence even though the algorithm works ($\forall v \in T$, $S_v^* \subseteq U_v$) the speed of the algorithm is not guaranteed.

  Indeed, consider the graph as below where each terminal is connected to a vertex in a clique with $|T|$ outgoing edges and one ingoing edges, then deleting edges from $\forall i \in [\lceil \log |T| \rceil]$ (R_i, B_i)-mincut and (B_i, R_i)-mincut would be deleting all ingoing edges attach to each terminal, which means $\forall v \in T, \mathrm{vol}(U_v) = \mathrm{vol}(G[V(K_n) \cup \{v\}]) - 2 = \Omega(m)$. Below in the graph is an example when $|T| = 4$.



- If we only compute one of them, then it might be the case that U_v will contain more than one terminal for some v, which means the algorithm won't work at all.

  Indeed, wlog if we only delete (R_i, B_i)-mincut, still consider the example in the previous case, then $\forall i \in [\lceil \log |T| \rceil], v \in B_i$, U_v contains v and all vertices in R_i.

3.3 Application: Steiner Mincut in Undirected Weighted Graphs

In a Steiner mincut problem, we are given an undirected weighted graph $G = (V, E, \omega)$ and a terminal set $T \subseteq V$. The desired output is a T-Steiner mincut, which is a minimum cut S separating some parts of terminals (i.e. $S \cap T \neq \emptyset$ and $S \setminus T \neq \emptyset$).

This problem is a generalization of global mincut and (s,t)-mincut. When $T = V$, it turns to a global mincut problem. When $T = \{s,t\}$, it turns to an (s,t)-mincut problem. Trivially, an algorithm using $|T|-1$ max flow can solve this problem. We just need to fix an $s \in T$ and compute an (s,t)-mincut for each $t \in T$.

In what follows, we will introduce an algorithm using only $\mathrm{poly}\log(n)$ max flows.

3.3.1 The algorithm

Let's assume S^* is a T-Steiner mincut and $|S^* \cap T| \leq |T|/2$.

Lemma 3.11. *For $T' \subseteq T$ s.t. $T' \cap S^* = \{v\}$ and $|T' \setminus S^*| \geq 1$, S^* is a minimum v-isolating cut, namely, a $(v, T' \setminus v)$-mincut.*

Proof. First S^* is a $(v, T' \setminus v)$-cut since S^* separates v and $T' \setminus v$.

Second, S^* is a minimal one. Otherwise there is a smaller cut S' that separates T'. Thus S' also separates T and S^* cannot be a T-Steiner mincut. □

By lemma 3.11, we just need to construct a $T' \subseteq T$ s.t. $|T' \cap S^*| = 1$ and $|T' \setminus S^*| \geq 1$. Then we can find S^* by running an isolating cut algorithm on T'. We will construct T' by random sampling.

First we guess $|S^* \cap T|$. By iterating i from 0 to $\lfloor \log(|T|/2) \rfloor$, in one iteration we have $2^i \leq |S^* \cap T| \leq 2^{i+1}$. We construct T' by sampling each vertex in T independently with probability $p = 1/2^i$. Let $k = |S^* \cap T|$. We have $1/p \leq k \leq 2/p$ and $1/p \leq |T|/2$. Further,

$$\begin{aligned}
\Pr[|S^* \cap T'| = 1] &= kp(1-p)^{k-1} \\
&\quad \text{since exactly one vertex is chosen into } T' \text{ among } k \text{ vertices} \\
&\geq (1-p)^{2/p} \\
&\quad \text{by } k \leq 2/p \\
&= \Theta(1).
\end{aligned}$$

and

$$\begin{aligned}
\Pr[|T' \setminus S^*| \geq 1] &= \Pr[|T' \cap (T \setminus S^*)| \geq 1] \\
&\quad \text{since at least one vertex in } T \setminus S^* \text{ is chosen} \\
&= 1 - (1-p)^{|T \setminus S^*|} \\
&\geq 1 - (1-p)^{|T|/2} \\
&\quad \text{by } |T \setminus S^*| \geq |T|/2 \text{ since } |T \cap S^*| \leq |T|/2 \\
&\geq 1 - (1-p)^{1/p} \\
&\quad \text{by } 1/p \leq |T|/2 \\
&= \Theta(1).
\end{aligned}$$

Therefore,

$$\Pr[|S^* \cap T'| = 1 \text{ and } |T' \setminus S^*| \geq 1] = \Theta(1).$$

By running isolating cuts on $O(\log n)$ independent samples $T'_1, \ldots, T'_{O(\log n)}$, we can find S^* with high probability.

The above algorithm invokes isolating cuts algorithm $O(\log^2 n)$ times summing over $O(\log n)$ levels of i and $O(\log n)$ samples for each i, so it needs $O(\log^3 n)$ max flows.

4 Extensions and more applications

Both local min cuts and isolating cuts have versions about vertex-cuts.

For local vertex-mincut problem $\mathsf{Local}(x, \nu, k)$, the key idea is that we can convert a directed graph G into another directed graph G_{split} by

- splitting each vertex v into two vertex v_{in} and v_{out} (except x and we let $x_{\text{in}} = x_{\text{out}} = x$) and
- converting any edge $(u, v) \in E(G)$ into edge $(u_{\text{out}}, v_{\text{in}})$ and adding an edge $(v_{\text{in}}, v_{\text{out}})$ for each $v \neq x$.

Observe that a vertex-mincut S s.t. $x \in S$ in G is kind of one-to-one corresponding to an edge-mincut S' s.t. $x \in S'$ in G_{split}. We can solve the local vertex-mincut problem on G by running the local edge-mincut problem on G_{split}.

For isolating vertex-cuts, we assume that the given terminal T is an independent set in G. The key point is that the submodularity still holds on vertex cuts, so we can still get the v-isolating vertex-mincut by only considering the small graph U_v.

There are some more applications of these two techniques. For local min cut technique, we can use it to solve k-vertex connectivity[3] in $\tilde{O}(mk^2)$ time which is the fastest algorithm when k is small. Some other applications are property testing and vertex sparsifiers.

For isolating cuts, there are some applications on Gomory-Hu trees, general symmetric submodular function minimization and connectivity augmentation.

[3]See paper [FNSYY'19] *Computing and Testing Small Connectivity in Near-Linear Time and Queries via Fast Local Cut Algorithms* and video https://www.youtube.com/watch?v=V1kq1filhjk

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 19: Deterministic Minimum Cuts via Expander Decomposition

November 16, 2021

Instructor: Thatchaphol Saranurak

Scribe: Chaitanya Nalam

We have seen from previous lecture that we can use expander decomposition to get spanners and cut-sparsifiers which reduce the number of edges in the graph but preserving the important information from the graph like shortest distance between vertices (or) cut edges of any subset of vertices in graph etc., although at a loss in factor of approximation.

Today, we will see how to use expander decomposition help us solve some problem exactly.

The spanners and cut-sparsifiers seen in previous class reduce the number of edges in the graph maintaining the same vertex set which comes under a broad regime of Edge Sparsification. There is a similar notion with respect to vertics called Vertex Sparsification where we preserve some properties of the original graph by reducing the number of vertices. In this lecture we will see how we can preserve every min-cut of the graph exactly in a smaller graph with less number of vertices called **Exact Mincut Sparsifier**.

1 Introduction and Problem

One of the very well studied problems in graph theory is the problem of "Max-flow min-cut" which is as follows:

- Input: a graph $G = (V, E)$ with n vertices and m edges
- Output: a set $S \neq \emptyset$ and $S \subset V$ where $|E(S, V \setminus S)|$ is minimized

The set S is called min-cut of the graph G and $|E(S, V \setminus S)|$ is the size of the minimum cut denoted by $\lambda(G)$ also called edge connectivity of the graph G.

In this lecture we restrict ourselves to the case where G is unweighted and undirected. We only the find the value of $\lambda(G)$ however we can also find the min-cut easily. Below table summarizes the state of the art for calculating edge connectivity $\lambda(G)$ of the graph G.

Reference	Time	Det/Rand	Note
Kawarabayashi Thorup [STOC'15]	$m \log^{12} n$	Det	Complicated (Page Rank)
Henzinger Rao Wang [SODA'17]	$m \log^2 n \log\log n$	Det	Complicated (Local Flow)
Ghaffari Nowicki Thorup [SODA'20]	$m \log n, m + n \log^3 n$	Rand	Simple
This lecture	$m^{1+o(1)}$	Det	Simple, conceptually

All the algorithms listed in above table follow the same framework of Mincut-preserving contractions where vertices are contracted which fall on completely one side of any min-cut, so not contracting any cut edges of mincut thus preserving every mincut of the graph.

2 Overview and Runtime analysis

Given an input graph G we create a contracted graph G' from G such that $|E(G')| = O(m/\delta)$ and $|V(G')| = O(n/\delta)$ where δ is the minimum degree of the graph G with the guarantee that every non-trivial mincut of the graph G is preserved in G'. Non-trivial min-cuts of the graph are cuts such that both $|S|, |V \setminus S| \geq 2$ (i.e. every cut other than the singleton cuts are non-trivial).

From this we can see that

Lemma 2.1. $\lambda(G) = \min(\lambda(G'), \delta)$

Proof. The minimum cut in graph G can be either trivial cut or non-trivial cut. If it is trivial cut then $\lambda(G) = \delta$.

If it is non-trivial minimum cut then according to the guarantee it is preserved in G' thus we have $\lambda(G) \geq \lambda(G')$.

Every cut in G' is mapped to a cut in G hence minimum cut in G' cannot be smaller than minimum cut in G so $\lambda(G') \geq \lambda(G)$ so $\lambda(G) = \lambda(G')$. □

From the work of [Gab95] we have the following theorem.

Theorem 2.2. *Edge connectivity $\lambda(G)$ of a graph can be computed in $O(\lambda m \log(n^2/m))$.*

Hence given graph G' we can compute $\lambda(G')$ in $\tilde{O}(\lambda(G')|E(G')|)$ where we know that $\lambda(G') = \lambda(G)$ is at most $\delta(G)$ (the min degree), hence we can compute $\lambda(G')$ in $\tilde{O}(m/\delta(G) \cdot \delta(G)) = \tilde{O}(m)$. So we are done once we compute G'.

We use *Expander decomposition* from the work of [CGL+19] to construct G'. Formally we use the following theorem from their work.

Theorem 2.3. *Given a graph $G = (V, E)$ and expansion parameter α we can return a partition $\mathcal{P} = \{X_1, X_2, \cdots X_k\}$ of vertices V in $O(m\gamma)$ time where $\gamma = n^{o(1)}$, such that each induced sub graph on X_i i.e. $G[X_i]$ is an $\alpha-$expansion expander and inter partition edges in the graph are small i.e. $O(\alpha n\gamma)$.*

Note: This variant of decomposition is slightly different from the one given in the paper but it follows from the work with slight modifications.

Using expander decomposition stated above we construct G' in $O(m\gamma)$ time with number of edges $|E(G')| = O(m\gamma/\delta)$ where $\gamma = n^{o(1)}$. Then using [Gabow] we get $\lambda(G)$ in time $\tilde{O}(m\gamma)$ instead of $\tilde{O}(m)$ as stated.

3 Computing G'

As we have stated above we will use expander decomposition to construct G' from $G = (V, E)$. Let C be a non-trivial min-cut of graph G and we will use λ to denote $\lambda(G)$ from now.

Computing G' can be summarised in the following simple steps.

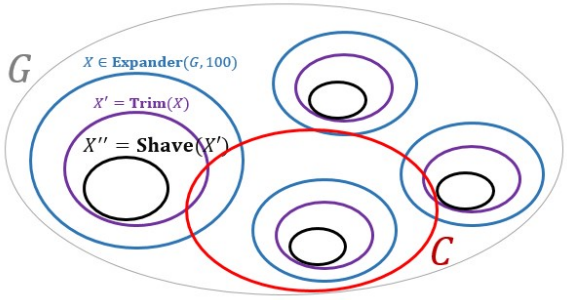


Figure 1: Computing G' (overview)

1. Compute expander decomposition of G with parameter $\alpha = 100$ giving us the partition $\mathcal{P}$ of vertices V.
$$\mathcal{P} = Expander(G, 100)$$

2. Apply a routine called $Trim$ to each of the partition $X \in \mathcal{P}$ in expander decomposition to give a smaller vertex set X'.
$$\mathcal{P}' = \{X' = Trim(X) | X \in \mathcal{P}\}$$

3. Apply a routine called $Shave$ to each set of vertices $X' \in \mathcal{P}'$ to give a still smaller vertex set X''.
$$\mathcal{P}'' = \{X'' = Shave(X') | X' \in \mathcal{P}'\}$$

4. Finally contract each of the vertex sets $X'' \in \mathcal{P}''$ to obtain graph G'.
$$G' = G/\mathcal{P}''$$
(contract each $X'' \in \mathcal{P}''$).

From above process to ensure that G' preserves all min-cuts of the graph G, it should be that for every min-cut C of graph G we should have for all vertex sets $X'' \in \mathcal{P}''$ should either be totally contained in min-cut (or) have no overlap with min-cut so that contracting these sets will not affect any min-cut of G thus preserving all of them.

Key Property:
$$\text{For all } X'' \in \mathcal{P}'', X'' \subseteq C \text{ (or) } X'' \cap C = \varnothing$$

In the remainder of this section we will define and show why applying such routines will help us satisfy the key property required above.

3.1 Overlap with Expander is small

Since, first step is to use the expander decomposition let us formally define the expander decomposition.

Expander Decomposition:

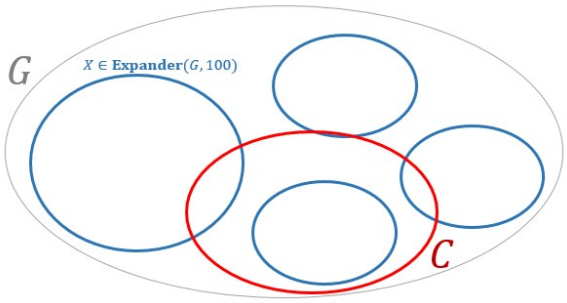


Figure 2: Expander decomposition of G

- Input: $G = (V, E)$, expansion parameter α
- Output: partition $\mathcal{P} = \{X_1, X_2, \cdots X_k\}$ of V such that
  - For all $S \neq \emptyset$ and $S \subset X_i$: $|E(S, X_i \setminus S)| \geq \alpha \min(|S|, |X_i \setminus S|)$ i.e. $G[X_i]$ is an $\alpha-$expansion expander.
  - $\sum_{i=1}^{k} |E(X_i, V \setminus X_i)| = O(\alpha n \gamma)$
- Time: $O(m\gamma)$ where $\gamma = n^{o(1)}$

Let us consider any min-cut C of the graph G we will show that it either contains a whole expander of expander decomposition (or) has minimal overlap with an expander. If min-cut contains the whole expander then contracting any subset of that expander will not affect the min-cut C. Otherwise we should have minimal overlap.

This is intuitively true from the definition of expander because whenever the overlap with expander is more because of the expansion property of expander number of edges going out of the overlapped part is more but limited by size of the minimum cut which will limit the number of vertices in the overlap. Formally let $S = C \cap X$ be the intersection of the min-cut C and expander X and assume without loss of generality that S is smaller side in X if not we can consider $\overline{C}$ in G which is also a min-cut and then take the intersection with X.

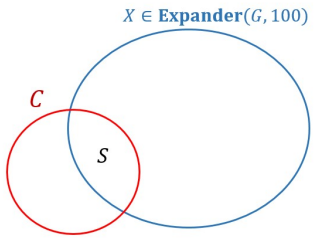


Figure 3: Mincut has small overlap with expander

Lemma 3.1. $|S| \leq \lambda/100$.

Proof.

$$\begin{aligned} \lambda &\geq E(S, X \setminus S) && (E(C, V \setminus C) \supseteq E(S, X \setminus S)) \\ &\geq 100|S| && (\text{As } S \text{ is on the smaller side and } X \text{ is an 100-expander}) \end{aligned}$$

□

3.2 Trimming an Expander reduces the overlap further

Above we showed that the number of vertices in expander that a min-cut has overlap with are small and now we want to reduce that further. We see the motivation to come up with a process like $Trim$ that will help us progress towards our goal.

Observe that any vertex in min-cut C will have at most $1/2$ of it's incident edges going out because if its not the case then we can remove it from the cut C to get a smaller cut. This is the basic clue for defining a process like $Trim$. Since any vertex has at most half of it's edges going out so any vertex $v \in S$ would have at most $1/2$ of its edges coming inside of expander X (as some edges can go out of $C \cup X$). So, intuitively we can delete any vertex in X that has less than $1/2$ of its degree inside X.

We now formally define the process of $Trim$.

Algorithm 1: $Trim(X)$

Initialize: $X' = X$
while $\exists v \in X$ *such that* $|E(v, X)| < 2\delta/5$ **do**
  $X' = X' \setminus \{v\}$
return X'

After the process from the guarantee of the algorithm that every $v \in X'$ has $|E(v, X')| \geq 2\delta/5$.

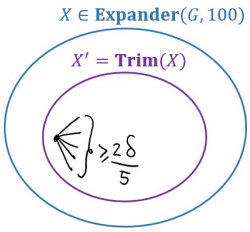


Figure 4: Guarantee after $Trim$ routine

Note that at first look it might seem that the vertices that are deleted depend on the order of deletions. However, it is not true and the set of vertices deleted does not depend on the order. This is because deletion of one of the vertex will only decrease the number of edges incident in X for the other vertex so it will be deleted sooner or later and hence any vertex that would have been deleted will be deleted and cannot escape from deletion because of a choice between vertices. It can be thought of as a dependency graph where deletion of a vertex reduces the degree of other vertex.

It can also be that the whole vertex set X is trimmed making X' empty thus giving no reduction in number of vertices using contraction. However we will later show that the number of vertices in the final graph G' would still be bounded by n/δ.

Now we show a stronger bound on the number of vertices that any min-cut C has overlap with any of the trimmed sets. Let $X' = Trim(X)$ and $S' = (X' \cap C) \subseteq S$.

Lemma 3.2. $|S'| \leq 2$.

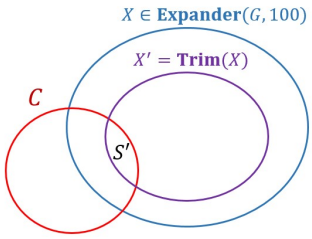


Figure 5: Overlap of Mincut after $Trim$

Why should S' be so small?

If S' is not small, because each and every vertex in S' emits at least $2\delta/5$ edges out of them and only small fraction of them can go inside S' (as $|S'|$ is bounded trivially by size of S). So, many edges go out of S'. But many edges cannot go out of S' as it is contained in min-cut C whose size can be only λ. This is formalized in the proof below.

Proof. Assume $|S'| \geq 3$ for contradiction. We also have that $|S'| \leq |S| \leq \frac{\lambda}{100} \leq \frac{\delta}{100}$. Since edges going out must be at most λ and edges emitting out from vertices in S' are at least $|S'| \cdot \frac{2\delta}{5}$ and edges that can go inside S' can be at most $\binom{|S'|}{2} \leq |S'|^2$. Here we need the requirement that G is simple graph, if not number of edges cannot be bounded as shown.

So we have contradiction as follows:

$$\begin{aligned}
\lambda &\geq |E(S', X' \setminus S')| && (S' \text{ contained in min-cut } C) \\
&\geq |S'| \cdot \frac{2\delta}{5} - 2|S'|^2 \\
&\geq \delta && (\text{when } |S'| \in [3, \tfrac{\delta}{100}] \text{ for large enough } \delta \gtrsim 265)
\end{aligned}$$

□

Note that the constant 265 is just the result of solving the inequality with appropriate bounds. What happens if δ is not large enough. Then we are done right!!! Why?

Because when δ is small λ is small too and we can use [Gab95] directly to find edge connectivity in $\tilde{O}(\lambda \cdot m) = \tilde{O}(m)$.

3.3 Shaving an Expander ensures no overlap

Trimming is done now lets $Shave$. Observe that in trimming as we have discussed that removing any vertex that has smaller degree inside expander X will remove the vertices that might be inside the min-cut C. However we have used a weaker requirement of deleting vertices v, which have

fewer than $\frac{2\delta}{5}(< \frac{deg(v)}{2})$ degree inside expander X. This relaxation will help us to reduce the number of vertices that need to be deleted as we do not want to end up pruning the whole expander. And as we saw above that the $Trim$ is good enough, but does not completely eliminate vertices inside min-cut C.

This necessitates the need for $Shave$ process which as we will show completely removes the vertices inside C. Let X' be any set we have

Algorithm 2: $Shave(X')$

$X'' = \{v \in X' : |E(v, X')| > \frac{deg(v)}{2} + 1\}$
return X''

Figure 6: Mincut does not overlap after $Shave$

Let $X'' = Shave(X')$ and $S'' = (X'' \cap C) \subseteq S'$.

Lemma 3.3. $S'' = \emptyset$

Proof. Assume S'' has a vertex v and since C is a non-trivial min-cut there is at least another vertex in C. Since $|S'| \leq 2$ there can be at most another 1 vertex in S'' other than v. Since $v \in X''$ we have

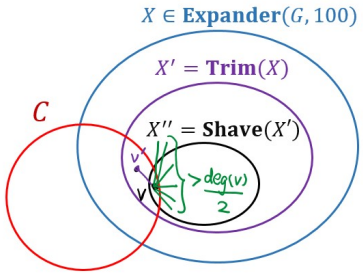


Figure 7: Non empty intersection implies a smaller min-cut

$|E(v, X')| > \frac{deg(v)}{2} + 1$ and so number of edges going out of C is at least $> (\frac{deg(v)}{2} + 1) - 1 = \frac{deg(v)}{2}$ and so we can remove v out of C still leaving behind a valid cut (because C is non-trivial min-cut) that has size smaller than min-cut which is a contradiction. □

Note it is very crucial that we need C to be a non-trivial min-cut otherwise removing v will make it empty and would not lead to contradiction.

3.4 Final steps

Hence we have shown that applying $Trim, Shave$ routines to each of expander sets in expander decomposition will ensure that the remaining sets $X'' \in P''$ will not have intersection with any non-trivial min-cut C. Since they are either completely inside (or) outside a min-cut we can safely contract them to give a smaller graph G' which preserves all non-trivial min-cuts as required.

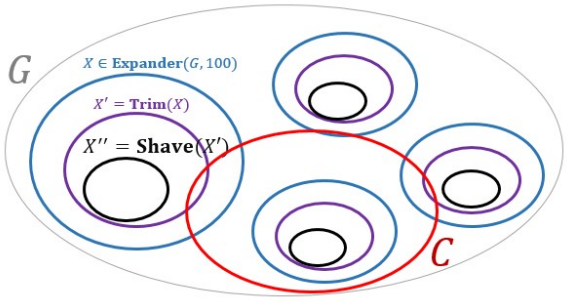


Figure 8: Shaved sets can be safely contracted

4 G' is small

As we have seen in previous section we have computed the graph G' however we still have to prove that the number of edges and vertices of G' are few so that we can use [Gab95] on this graph and still achieve the required time bound.

To start with, note that the number of inter expander edges in expander decomposition is $O(\alpha n\gamma) = O(n\gamma)$. If we have contracted all vertex sets $\mathcal{P}$ in G i.e. $G/\mathcal{P}$ then we would have got $O(n\gamma)$ edges in G'. However, we have added more edges and vertices to G' by $Trim, Shave$ routines as shown in figure below.

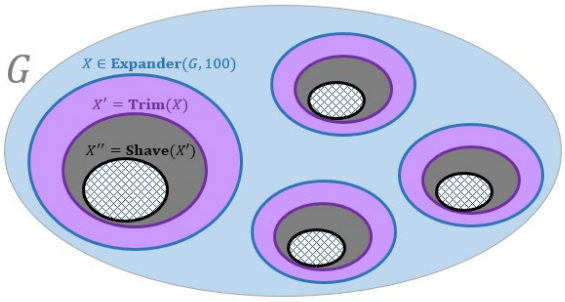


Figure 9: Edges in G' are $O(n\gamma)$

We bound the number of edges (and vertices) in the regions produced by these routines using careful charging argument and show that up to constant factors number of edges are still $O(n\gamma)$.

Then we have G' with $O(n\gamma) \leq O(\frac{m\gamma}{\delta})$ edges as δ is min degree.

4.1 Why we don't trim too much?

As we proceed with deleting vertices from an expander $X \in \mathcal{P}$ which means we are reducing the number of vertices that are going to be contracted thus increasing both number of edges,vertices that are going to be finally in G'.

To bound the number of edges that are added we define a state tuple that represents the information about number of edges.

Let (t, e) represent the state of the process $X' = Trim(X)$ where X' is a dynamic set that changes with each deletion where as $X \in \mathcal{P}$ is the constant set that is part of expander decomposition. We define t as the number of boundary edges that are going out of X' and e is the number of edges that are present outside all dyanmic sets $X' \in \mathcal{P}'$ that is number of edges in $e = |E(G/\mathcal{P}')|$ as X' is dynamic so is $\mathcal{P}', e$.

Initially $X' = X$ so total count of boundary edges coming out of $X' = X \in \mathcal{P} = \mathcal{P}'$ is just the count of inter expander edges in expander decomposition. Observe that it is also the value of e which is the number of edges in $G/\mathcal{P}'$.

Observe that every vertex $v \in X'$ that is being removed has $\leq \frac{2}{5}\delta \leq \frac{2}{5}deg(v)$ edges inside X' whenever we delete it from X'. Also there at least $\frac{3}{5}deg(v)$ edges going out of X' from this vertex. Here comes the charging argument from the two key observations stated below.

1. Once the vertex is removed the number of boundary edges going out of X' is reduced by at least $\frac{3}{5}deg(v) - \frac{2}{5}deg(v)$ as there are at most $\frac{2}{5}deg(v)$ new edges are becoming boundary edges as now v is outside X'. Hence the number of boundary edges that are decreased are at least $\frac{deg(v)}{5}$ due to removal of v.

2. Also whenever a vertex is removed from X' there are at most $\frac{2}{5}deg(v)$ which are being added to G'. So e increases by $\frac{2}{5}deg(v)$.

From the above two observations every time a vertex is removed t decreases by at least $\frac{deg(v)}{5}$ and e increases by at most $2\frac{deg(v)}{5}$. Hence the increase in e can be charged to decrease in t and since t cannot decrease beyond zero becoming negative hence e can increase at most by 2 times the initial value of t which is $O(n\gamma)$. So total number of edges increases due to trimming

$$|E(G/\mathcal{P}') \setminus E(G/\mathcal{P})| = O(n\gamma).$$

4.2 Why we don't shave too much?

Here, we bound the size of $E(G/\mathcal{P}'') \setminus E(G/\mathcal{P}')$.

For each $X' \in \mathcal{P}'$, we add more edges to G' when we remove vertices from X' just like in trimming. Number of edges that are added to G' are at most $\sum_{v \in X' \setminus Shave(X')} |E(v, X')|$.

For each $v \in X' \setminus Shave(X')$, we have $|E(v, X')| \leq \deg(v)/2 + 1$ so $|E(v, V \setminus X')| \geq \deg(v)/2 - 1$. Assuming $\delta \geq 4$, we have

$$|E(v, X')| \leq 4|E(v, V \setminus X')|$$

and so

$$\sum_{v \in X' \setminus Shave(X')} |E(v, X')| \leq 4|E(X', V \setminus X')|$$

$$|E(G/\mathcal{P}'') \setminus E(G/\mathcal{P}')| \leq \sum_{X' \in \mathcal{P}'} 4|E(X', V \setminus X')|$$
$$\leq \sum_{X \in \mathcal{P}} 4|E(X, V \setminus X)| = O(n\gamma)$$

where the last inequality is because $Trim$ only decreases $|E(X, V \setminus X)|$ and so $|E(X', V \setminus X')| \leq |E(X, V \setminus X)|$ for each $X' = Trim(X)$.

4.3 Size of G'

So total number of edges in G' after $Trim$ and $Shave$ are

Lemma 4.1. $|E(G')| = O(m\gamma/\delta)$.

Proof.

$$\begin{aligned} E(G') &= E(G/\mathcal{P}) \\ &\quad + E(G/\mathcal{P}') \setminus E(G/\mathcal{P}) \\ &\quad + E(G/\mathcal{P}'') \setminus E(G/\mathcal{P}') \\ |E(G')| &= O(n\gamma) && \text{(Each set has } O(n\gamma) \text{ edges)} \\ &\leq O(m\gamma/\delta) && (m \geq n\delta) \end{aligned}$$

□

Lemma 4.2. $|V(G')| = O(n\gamma/\delta)$.

Proof. It is enough to show that each and every vertex in G' has degree at least δ. Vertices which are part of the original graph have degree at least δ. Vertices that are formed as a result of contraction can be categorized into two cases based on number of vertices that are contracted.

1. Number of vertices contracted $\leq \frac{\delta}{2}$. Since each vertex inside contracted set has at least δ edges incident on it and at most $\frac{\delta}{2}$ remain inside the contracted set. So at least $\frac{\delta}{2}$ edges go out of the set. Since there are at least two vertices we have at least δ edges going out of the contracted set.
2. Number of vertices contracted $> \frac{\delta}{2}$ and number of such sets can be at most $O(n/\delta)$.

So total number of vertices in G' are at most $\frac{O(n\gamma)}{\delta} + O(\frac{n}{\delta}) = O(\frac{n\gamma}{\delta})$. □

References

[CGL+19] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019.

[Gab95] H.N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.

University of Michigan–Ann Arbor

Department of Electrical Engineering and Computer Science

EECS 498 004 **Advanced Graph Algorithms**, Fall 2021

Lecture 28: Summary and Beyond

December 9, 2021

Instructor: Thatchaphol Saranurak

Scribe: Gary Hoppenworth

1 What have we learned?

1.1 5 Equivalent Views of Expanders (w.r.t Conductance)

1. **Robust against deletions.** We have seen that expanders are robust against deletions in the sense that when you remove a 'small' number of edges from an expander, it will only disconnect a 'small' volume of edges from the graph. More formally, let G be a ϕ-expander undergoing d insertions/deletions. Then there is a $P \subset V(G)$ such that P has small volume ($\text{vol}(P) \leq O(d/\phi)$) and $G \setminus P$ is a $\Omega(\phi)$-expander.

2. **Cut view.** Expanders are well-connected graphs with respect to cuts. Formally, if graphs G and H share the same vertex set, and every vertex in H has smaller degree than the corresponding vertex in G, then we say that $H \preccurlyeq^{\text{deg}} G$. Likewise, we say that $H \preccurlyeq^{\text{cut}} G$ if every cut $(S, \overline{S})$ in H has fewer edges than the corresponding cut in G. We have seen that

$$G \text{ is } \phi\text{-expander and } H \preccurlyeq^{\text{deg}} G \implies H \preccurlyeq^{\text{cut}} \frac{1}{\phi} G$$

   So G is the most well-connected graph (within a $1/\phi$ factor) with respect to cuts among all graphs with the same vertex degree profile.

3. **Flow view.** Expanders are well-connected graphs with respect to flow. We say that $H \preccurlyeq^{\text{flow}} G$ if there is a flow in G that can route the demand H with no congestion. The problem of determining whether $H \preccurlyeq^{\text{flow}} G$ is called the maximum concurrent flow problem. We saw a generalization of the max-flow min-cut theorem, which stated that

$$H \preccurlyeq^{\text{flow}} G \implies H \preccurlyeq^{\text{cut}} G \implies H \preccurlyeq^{\text{flow}} O(\log n) G$$

   It immediately follows from the cut view of expanders that ϕ-expanders are the most well-connected graphs with respect to flow (within a $O(\log n/\phi)$ factor) among all graphs with the same vertex degree profile.

4. **Eigenvalue view.** Given a graph G, we saw how to define the Laplacian $\boldsymbol{L}_G$ and normalized Laplacian $\boldsymbol{N}_G$ matrices of G. We saw that the second eigenvalues of these matrices are closely related to the conductance $\Phi(G)$ of G. We proved Cheeger's inequality which states that

$$\frac{\lambda_2(\boldsymbol{N}_G)}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2(\boldsymbol{N}_G)}.$$

   Because ϕ-expanders are the graphs with $\Phi(G) \geq \phi$, this gives another view of expanders from a spectral perspective.

5. **Random walks.** We defined a lazy random walk on graphs and showed that no matter where you start in the graph, the probability distribution of your location in the graph at time t will eventually converge to a stationary distribution. In this stationary distribution, the probability you are at specific vertex at time t is proportional to the degree of that vertex. Given a graph G, we can define the mixing time, $\tau_{\text{mix}}(G, \epsilon)$, as the number of steps it takes a random walk in G to 'mix' with G so that the probability distribution of its location will be close to the stationary distribution within an ϵ amount. We saw that

$$\tau_{\text{mix}}(G, \epsilon) \approx 1/\lambda_2(\boldsymbol{N}_G)$$

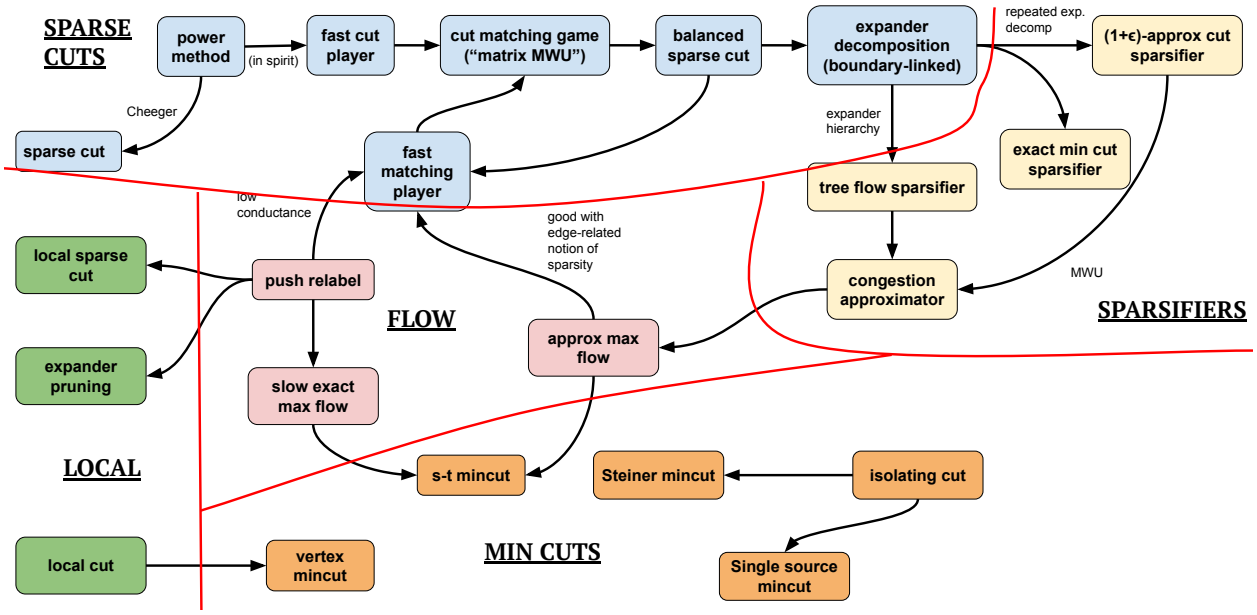
   We know good expanders have large $\lambda_2(\boldsymbol{N}_G)$ from Cheeger's inequality. This tells us random walks on expanders will have a small mixing time.

1.2 Graph Algorithm Toolkit / Concepts

We saw many algorithmic tools and concepts that are useful for solving problems related to graphs and expanders. Here are some of them.

- Metric embedding
- Power method
- Isolating cuts
- Local cuts
- Cut-matching games
- Balanced cuts
- Expander decomposition and its variants
- Sparsifiers in various forms
- Push relabel
- MWU
- Approximate max flows

These ideas are all connected.



2 More Problems that can be solved using Related Techniques

We have learned many important techniques from this class, and we have seen many state-of-the-art algorithms. Below, I included additional important problems that we haven't learn about, but material from our class will help you understand/improve the state-of-the-art.

2.1 Minimum cuts

1. Global minimum cuts in $\tilde{O}(m)$ randomized time[1]
   - In class: max flow time (isolating cuts), $\tilde{O}(mk^2)$ time (local cuts)
   - Ingredient: Graph sparsification, MWU
2. Global minimum cuts in $m^{1+o(1)}$ deterministic time[2]
   - Ingredient: Deterministic graph sparsification (Expander hierarchy, Pessimistic estimator method for derandomization)
3. Vertex connectivity in max flow time[3]
   - In class: $\tilde{O}(mk^2)$ time (local cuts)
   - Ingredient: Isolating cuts, graph sketching
   - Exciting open questions:
     - $\tilde{O}(m)$ time (even for $(1+\epsilon)$-approximation)
     - $o(mn)$ for weighted case

[1] https://arxiv.org/pdf/cs/9812007.pdf
[2] https://arxiv.org/abs/2106.05513
[3] https://arxiv.org/pdf/2104.00104.pdf

2.2 Sparse cuts

1. Expander decomposition in $\tilde{O}(m/\phi)$ time[4]
   - In class: $m^{1+o(1)}$ time
   - Ingredient: *non-stop* version of cut-matching game
2. Deterministic expander decomposition in $m^{1+o(1)}$ time
   (a) Ingredient: multi-commodity cut-matching game, dynamic shortest path
   (b) Exciting open problem: $\tilde{O}(m)$ time
3. Expander decomposition in distributed networks in polylog(n/ϕ) rounds[5]
   - Ingredient: Basics on the distributed computation model, *non-stop* cut-matching game
   - Many applications in distributed computations: finding cliques in distributed network and so on
4. Local sparse cuts starting from a single node inside a sparse cut.
   - In class: given a set A with a sparse cut S where overlapping a lot with A (i.e. $\text{vol}(S \cap A) \geq \sigma \text{vol}(S)$). Time: $O(\frac{\text{vol}(A)}{\phi\sigma})$
   - Ingredient: Local PageRank[6], Local random walk[7]

2.3 Graph sparsification

1. Deterministic $(1+\epsilon)$-approximate cut/spectral sparsifier in poly(n) time
   - In class: randomized (we only describe the algorithm, no proof)
   - Ingredient: more spectral graph theory
2. Vertex sparsifier for cuts with quality $O(\frac{\log|T|}{\log\log|T|})$ in poly(n) time [8]
   - Problem: Given a graph $G=(V,E)$ and a terminal T, find a graph H where $V(H)=T$ and, for all $A, B \subseteq T$, we have
   $$\text{mincut}_G(A,B) \leq \text{mincut}_H(A,B) \leq O(\frac{\log|T|}{\log\log|T|}) \cdot \text{mincut}_G(A,B).$$
   - In class: using boundary-linked decomposition: $O(\log n)$ quality, $V(H) \supset T$, and only for unweighted graphs
3. Vertex sparsifier for cut of size at most c in time $\tilde{O}(mc^{O(c)})$[9] or poly(n)[10]

[4] https://arxiv.org/pdf/2007.14898.pdf
[5] https://arxiv.org/pdf/2007.14898.pdf
[6] http://www.math.ucsd.edu/~fan/wp/localpartition.pdf
[7] Chapter 22 of http://cs-www.cs.yale.edu/homes/spielman/sagt/sagt.pdf
[8] https://epubs.siam.org/doi/pdf/10.1137/130908440
[9] https://arxiv.org/abs/2007.07862
[10] https://arxiv.org/abs/2011.15101

- Problem: Given a graph $G = (V, E)$ and a terminal T, find a graph H where $V(H) \supseteq T$ and, for all $A, B \subseteq T$, we have

$$\min\{c, \text{mincut}_G(A, B)\} = \min\{c, \text{mincut}_H(A, B)\}.$$

- In class: not exact (boundary-linked decomposition)
- Ingredient: boundary-linked decomposition, representative sets based on matroid

4. Tree flow sparsifier with quality polylog(n)[11]
   - In class: quality $n^{o(1)}$ via expander hierarchy
   - Ingredient: *non-stop* cut-matching game

2.4 Flow

Let $G = (V, E)$ be an undirected graph and let $\mathsf{d} \in \mathbb{R}^V$ be a demand.

In the max flow problem (a.k.a. ℓ_∞-flow problem), find a flow f satisfying d such that $\max_{e \in E} |f(e)|$ is minimized.

1. (ℓ_2-flow): Electrical flow and Laplacian solvers in $\tilde{O}(m)$ time
   - Problem: find a flow f satisfying d such that $\sum_{e \in E} f(e)^2$ is minimized
     - Equivalent problem: find a potential vector x where $\boldsymbol{L}_G x = \mathsf{d}$.
   - Ingredient: Graph sparsification, Conjugate gradient descent (many other interesting ways)
2. (ℓ_1-flow): Transshipment and shortest paths in $\tilde{O}(m)$ work and $\tilde{O}(1)$ depth (parallel time)[12]
   - Problem: find a flow f satisfying d such that $\sum_{e \in E} |f(e)|$ is minimized
   - Ingredient: MWU, metric embedding
3. Multi-commodity flow on expanders in $m^{1+o(1)}$ time.
   - Ingredient: dynamic shortest paths[13] or random-walk based algorithm[14]
   - Exciting open problem: $\tilde{O}(m)$ time
4. Exact max flow in $\tilde{O}(m^{1.5-\epsilon})$
   - Ingredient: dynamic spectral sparsification, interior point method
   - Exciting open problem: $\tilde{O}(m)$ time

[11] https://www.youtube.com/watch?v=XAdM0CqrxOI https://epubs.siam.org/doi/pdf/10.1137/1.9781611973402.17
[12] https://arxiv.org/pdf/1911.01626.pdf
[13] Section 3.2 in https://arxiv.org/pdf/2009.08479.pdf
[14] https://groups.csail.mit.edu/tds/papers/Ghaffari/podc117.pdf

2.5 Other Problems

There are other problems not so related to techniques in this class (at least not yet). But I want to mention them anyway.

1. All-pair shortest paths in $O(n^3/2^{O(\sqrt{\log n})})$ time.
    - Exciting question: $O(n^{2.99})$ time.
2. Parallel s-t reachability in $m^{1+o(1)}$ work and $n^{0.5+o(1)}$ depth
    - Exciting question: $\tilde{O}(m)$ work and polylog(n) depth.

3 Related Fields

There are many other exciting questions beyond this. For example, we can try to make everything taught in this class...

- dynamic (my field)
- distributed/parallel
- sub-linear space/sub-linear time.

4 Final words

- I hope you learn a lot and push the state-of-the-art further.
- Thank you for your hard work throughout the course.