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:

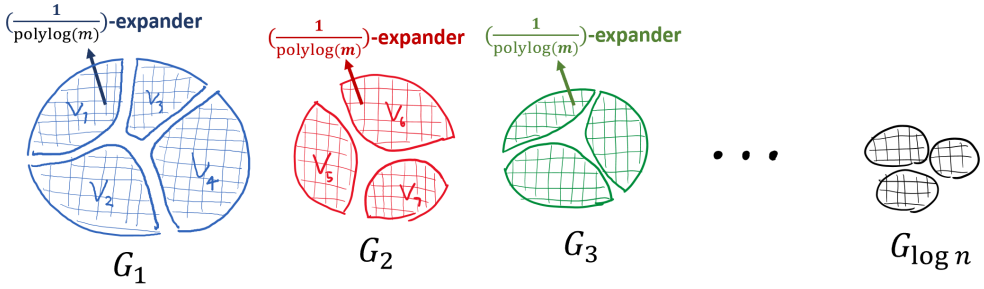
Theorem 2.1 (Repeated Expander Decomposition)**.** *For any unweighted graph $G = (V, E)$ with n vertices and m edges, there is a partition of* ***edges*** *E into $G_1, \ldots, G_{\log m}$ such that G_i contains disjoint union of $\Omega(\frac{1}{\log m})$-expanders.*

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.*

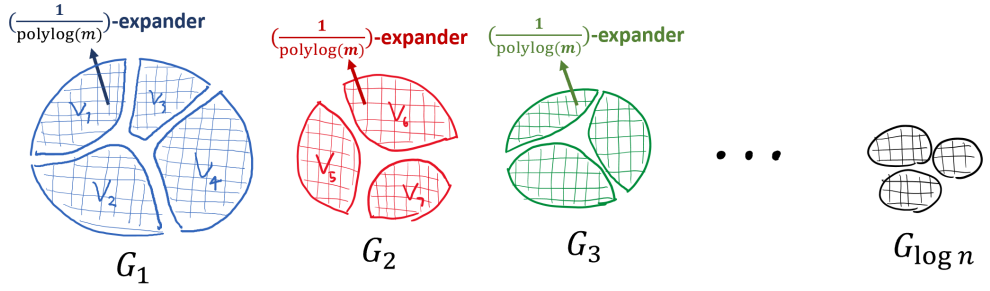
Proof. Recall that any ϕ-expander has diameter at most $D = O(\log(m)/\phi)$. Therefore, for any edge $(u, v) \in G$, we have $\text{dist}_H(u, v) \leq D$. □

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

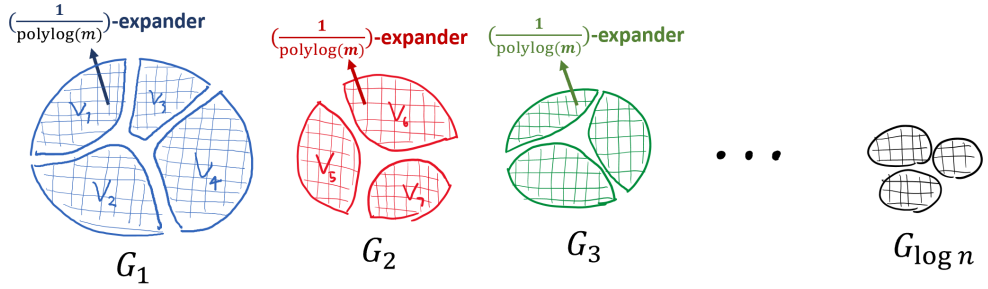
$$\sum_{e \in E} \frac{1}{\tilde{\lambda}_e} = O(\frac{1}{\phi} \sum_{(u,v) \in E} \frac{1}{\min\{\deg(u), \deg(v)\}}) = O(\frac{1}{\phi} n)$$

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