CS 0368-4246: Combinatorial Methods in Algorithms (Spring 2025) May 26, 2025

Lecture 9: Explicit Expanders, Expander Decomposition, and Fast Min-Cuts

Instructor: *Or Zamir* Scribes: *Nir Finkelstein*

Reminder from Last Class

- We saw different definitions for expander graphs.
- We looked at their basic properties.
- We showed their existence (by probabilistic construction).
- We saw how to use them to construct error-correcting codes with fast error correction.

In This Class We Will See

- Explicit constructions of expander graphs.
- We will give an example of an expander application that requires an explicit construction.
- We will discuss expander decomposition.

Different Options for Explicit Construction

1. We have already seen an algorithm that runs in polynomial time and returns an expander graph with high probability. We can use this as a basis to create an algorithm with no probability of error, whose expected runtime is polynomial. This would require us to be able to check if a graph is an expander in polynomial time.
2. Deterministic construction - an algorithm with no probabilistic elements.
3. **Strongly explicit construction** - a very fast algorithm (polynomial in $\log n$) that receives the index of a node and returns the indices of its neighbors. (This is possible because each node has a constant number of neighbors, and its index can be represented using log(n) bits).

Building Strongly Explicit Expanders

We will focus on the third option: building strongly explicit expanders.
One way to construct such expanders comes from group theory, specifically from **Cayley graphs**. Given a group H and a subset $S \subseteq H$ (usually a set of generators), we can define a graph where the nodes are the elements of the group. Any two nodes $u, v \in H$ are connected if $u \cdot s = v$ for some $s \in S$.
For example, we can obtain a cycle graph of length n (C_n) by using $\mathbb{Z}_n$ (the cyclic group of size n)

where $S = \{+1, -1\}$. This is an explicit construction because, given an element, it is easy to identify its neighbors simply by multiplying by elements of S. As an example (without proof) of a Cayley graph (Not exactly Cayley) that is an expander [4]: Consider the group $H = \mathbb{F}_p$, and for each element $a \in H$, choose its neighbors to be $a+1, a-1, a^{-1}$. It turns out that this graph is an expander.

Expander Applications: Derandomization

Now we will move on to a classic application of expanders in computer science:
derandomization - reducing or eliminating randomness.

Consider a randomized algorithm A that flips r coins and has one-sided error (can be wrong when saying "no") with an error probability of δ. We want to reduce δ. We can usually do this by running the algorithm k times, which yields:

- Runtime increases by a factor of k.
- Error probability reduces to δ^k.
- Randomness increases by a factor of k (requires $r \cdot k$ coin flips).

We want to proceed in a way that minimizes randomness. Therefore, we will prove the following lemma.

Lemma. *In this setting, using an* ***explicit expander****, we can reduce the error to $O(\frac{\delta}{k})$, the runtime will increase by a factor of k, and the amount of randomness will remain the same (using r coin flips).*

Proof. Let's take an explicit expander graph with $2 \cdot 2^r$ nodes, as defined last week. It's a bipartite graph where the degree of each node in L is d, and the number of nodes on each side is $|L| = |R| = 2^r$. For any $S \subseteq L$ such that $|S| < \gamma \cdot |L|$, it holds that $|N(S)| \geq (1-\epsilon) \cdot d \cdot |S|$.
Each node in the expander represents a choice for all r coin flips. Assume the expander is explicit and $\delta = 0.05$, $\epsilon = 0.1$.
We define the algorithm as follows: Randomly choose one node in L and run the algorithm A on all of its neighbors.
Why does this work? Let $B \subseteq L$ be the set of all "bad" choices from L. That is, B contains nodes whose neighbors are all "bad" choices for the r coin flips (meaning the algorithm A would err for all of them). Assume for contradiction that $|B| \geq 0.1 \cdot \frac{|L|}{d}$. Let $B' \subseteq B$ be a subset of exactly this size. Because the graph is an expander, $N(B')$ has at least: $|B'| \cdot d \cdot 0.9$ neighbors (assuming $\epsilon = 0.1$).
This quantity is greater than $\delta \cdot |R|$, which leads to a contradiction.
Therefore, the probability of choosing a "bad" initial node from L is reduced. This means the error probability is reduced by a factor of d, since $|B| = O(\frac{|L|}{d})$. This method effectively reduces derandomization because now, to choose a vertex in L, one only needs to flip r coins, which is equivalent to selecting a value within the range of 2^r. □

The runtime will be: $dT(n) + \text{poly}(r)$, where:

- $T(n)$ is the runtime of the original algorithm.
- $\text{poly}(r)$ accounts for the time to obtain neighbors: A vertex can be represented with r bits, and by the assumption of a strongly explicit expander, all its neighbors can be obtained in time polynomial in its representation size (i.e., polynomial in r).

Algorithmic Problems on Expanders

Many algorithmic problems become easier if the input graph is an expander. Two examples we discussed that can be solved in sub-linear time are:

- Checking graph connectivity after deletions.
- Finding the shortest path between two nodes.

Now we will see another example: finding a MinCut.

Theorem 1. *Let G be a graph with minimum degree $\delta(G)$ which is also a $\frac{2}{\delta(G)}$-expander. Then its MinCut can be found in $O(n)$ time, and all its MinCuts correspond to separating a single vertex from the rest of the graph.*

Proof. Let's start by showing that there is no MinCut $(C, V \setminus C)$ (where $|C| \leq |V \setminus C|$) such that $|C| \geq \delta(G)/2$. For such a cut $(C, V \setminus C)$, we know that on one hand: $|E(C, V \setminus C)| \leq \delta(G)$ (because the minimum cut must be at most the minimum degree of a vertex). On the other hand, as we saw in the previous class, for an expander graph, it holds that: $|E(C, V \setminus C)| \geq \varphi \cdot Vol(C)$ where ϕ is the expansion constant.
In the current setting, we have $\varphi = 2/\delta(G)$. So, $\varphi \cdot Vol(C) \geq \varphi \cdot |C| \cdot \delta(G) = \frac{2}{\delta(G)} \cdot |C| \cdot \delta(G) = 2|C|$. Thus, combining these inequalities, we get: $2|C| \leq |E(C, V \setminus C)| \leq \delta(G)$ Which implies: $|C| \leq \frac{\delta(G)}{2}$.

Now, let's assume for contradiction that $2 \leq |C| \leq \frac{\delta(G)}{2}$. For any vertex in C, its degree is at least $\delta(G)$. Since $|C| \leq \frac{\delta(G)}{2}$, at most $\frac{\delta(G)}{2} - 1$ edges from any vertex in C can connect to other vertices within C. Therefore, for any vertex in C, at least $\frac{\delta(G)}{2} + 1$ of its edges must connect to vertices in $V \setminus C$.
This means that even if $|C| = 2$, there are more than $\delta(G)$ edges connecting C to $V \setminus C$. $\square$

Beyond the examples we'll see, the use of expanders has become incredibly useful in recent years for many problems related to various types of connectivity issues, as well as in static, dynamic, and distributed algorithms.

Expander Decomposition

We want to transform a general graph into a type of expander.

Definition 2. *Given a graph G and a parameter φ, we say that a partition: $V = V_1 \sqcup V_2 \sqcup \cdots \sqcup V_k$ is a (φ, m')-expander decomposition if the following conditions hold:*

1. *For every i, the graph $G[V_i]$ (the subgraph induced by G on V_i) is a φ-expander.*
2. *The number of edges in G connecting different components is at most m'.*

Theorem 3. *For any graph G and any φ, an expander decomposition exists: (φ, m') for $m' = O(\varphi \cdot m \cdot \log m)$.*

Note: This decomposition is useful when $\varphi \ll \frac{1}{\log m}$.
We'll start by showing its existence and then focus on the runtime for finding the expander decomposition.

Proof. (by algorithm) Given G and φ:

- If G is a φ-expander, return V.
- Otherwise, there exists a cut $(S, V \setminus S)$ such that $\phi(S) < \varphi$ (its conductance is less than φ). Run the algorithm recursively on $G[S]$ and $G[V \setminus S]$ with the parameter φ , and combine the partitions returned.

It's clear that each component returned is a φ-expander since this was the only stopping condition. Therefore, we still need to show that the number of edges between components is less than $O(\varphi \cdot m \cdot \log m)$. We'll use an **amortization argument**. Every time we partition the graph, we'll take the component with the smaller volume (W.L.O.G., S) and distribute the cost of its cut edges $E(S, V \setminus S)$ such that each vertex $v \in S$ pays $\varphi \cdot \deg(v)$.
Let's calculate how much each vertex pays throughout the process. Every time a vertex v pays $\varphi \cdot \deg(v)$, the volume of its component (the one it's currently in) is at least halved. Since the initial maximum volume is $2m$ (sum of all degrees) and the smallest possible volume for a component is at least 1 (for a single vertex with at least degree 1), a vertex can be part of such a "payment event" at most $\log(2m)$ times. Therefore, each vertex v pays at most $\varphi \cdot \deg(v) \cdot \log(2m)$ in total. Summing over all vertices, the total payment is at most:

$$\sum_{v \in V} \varphi \cdot \deg(v) \cdot \log(2m) = \varphi \cdot 2m \cdot \log(2m)$$

□

Runtime Concerns

Two potential issues could adversely affect the runtime:

1. It's unclear how to efficiently check if a graph is a φ-expander or how to find a sparse cut.
2. The cut might be unbalanced, leading to a quadratic runtime for the recursive calls.

The first issue is less concerning, as we can tolerate an approximation to the solution within a factor of polylog(n). To address the second problem, instead of taking just any cut $(S, V \setminus S)$ that satisfies $\phi(S) < \varphi$, we will take the cut where $|S|$ is the largest among all cuts satisfying the condition, where $|S|$ denotes the smaller of the two sets $|S|$ and $|V \setminus S|$

Claim 4. *If φ is the conductance of G and we take the most balanced cut, then the volume will decrease by at least a constant factor every two iterations.*

Proof. Assume $(S_1, V \setminus S_1)$ is the largest cut with conductance less than φ, meaning that $|S_1|$, which is at most $|V \setminus S_1|$, is maximal among all such cuts. Then, assume S_2 is the largest cut in $G[V \setminus S_1]$ with conductance less than φ. Suppose, for contradiction, that $\text{Vol}(S_1) + \text{Vol}(S_2) < \frac{1}{2} \cdot Vol(V)$. Then $S_1 \cup S_2$ would form a cut larger than S_1 with conductance less than φ, contradicting the maximality of S_1. Therefore, $\frac{1}{2} \cdot Vol(V) \leq Vol(S_1) + Vol(S_2)$. Specifically, either S_1 satisfies $\frac{1}{4} \cdot Vol(V) < Vol(S_1) < \frac{1}{2} \cdot Vol(V)$, or $S_1 \cup S_2$ satisfies $\frac{1}{4} \cdot Vol(V) < Vol(S_1 \cup S_2) < \frac{3}{4} \cdot Vol(V)$. In either case, within two iterations, we successfully divided the graph into two or more parts that are smaller by at least a constant factor $(\frac{1}{4})$. □

The issue, however, is that in this proof, we didn't account for finding an approximate cut. In practice, to efficiently find the sparsest balanced cut, we typically need to find an approximation in two senses:

1. We find a cut whose conductance is at most $c_1 \cdot \varphi$.
2. We find a set S that is at least $\frac{1}{c_2} \cdot |S^*|$, where S^* is the maximum-sized cut satisfying the conductance condition.

It's currently known that a fast algorithm exists where $c_1 = c_2 = O(\log^2 n)$. How can we make the algorithm work under the above constraints? We can simply set $\varphi' = \frac{\varphi}{c1}$ since φ is a parameter we are free to choose during the algorithm's execution.

Theorem 5. *For every G and every φ, a $(\varphi, \varphi \cdot m^{1+o(1)})$-expander decomposition is computable in $m^{1+o(1)}$ time. [1]*

In the next lecture, we will see an algorithm for finding an approximate balanced cut. Combined with what we've already seen, this will yield a proof of the theorem. However, there might still be a gap: the algorithm finds an approximation to a balanced cut, rather than to the sparsest balanced cut.
Note. This expander decomposition can be maintained dynamically, even in a distributed computation setting.

Finding MinCut Using Expander Decomposition

Theorem 6. *MinCut can be found in a graph in $m^{1+o(1)}$ time.[3]*

Proof Idea:

1. Use expander decomposition on G with $\varphi \approx \frac{1}{\delta}$.
2. Use what we've learned about finding MinCut in an expander to find the MinCut for the original graph G.

Intuition for the Proof: We take the graph G and decompose it into expanders where $\varphi \approx 100/\delta(G)$. We've already seen that a minimum cut does not cut any expander component in more than one vertex. Therefore, we can hope that it doesn't cross them at all. In such a case, we can contract all components of the expander decomposition and solve MinCut on the remaining graph (including **parallel edges**). In the contracted graph, the number of edges is approximately $m^{1+o(1)}/\delta(G)$, because the decomposition algorithm guarantees $O(\varphi \cdot m \cdot \log m)$ edges between components, and we chose $\varphi = 100/\delta(G)$.

Claim 7. *(Gabow 95') For any graph G, MinCut can be found in time $\tilde{O}(m \cdot \lambda(G))$, where $\lambda(G)$ is the number of edges in the minimum cut. [2]*

We won't prove this claim here. However, in the Algorithms course, we saw a proof of this claim given that we are provided with source (s) and sink (t) vertices that must be on opposite sides of the cut. This uses the Ford-Fulkerson algorithm in flow networks, where each iteration takes $O(m)$ time and each iteration increases the flow (and thus the cut) by one.
Given Gabow's theorem, the runtime for finding the MinCut on the contracted graph from the previous part (after contracting the expander components) is:

$$\tilde{O}\left(\left(\frac{m^{1+o(1)}}{\delta(G)}\right) \cdot \lambda(G)\right)$$

Since $\lambda(G) \leq \delta(G)$ (the minimum cut is always less than or equal to the minimum degree), this simplifies to:

$$\tilde{O}\left(m^{1+o(1)} \cdot \frac{\lambda(G)}{\delta(G)}\right) \leq \tilde{O}(m^{1+o(1)})$$

Next, we'll move to describing the full algorithm that uses expander decomposition and then processes the expander components (meaning it removes certain vertices from them) so we know for certain that no MinCut will split any of the resulting components.

Algorithm

1. Run expander decomposition on G with $\varphi = \frac{40}{\delta(G)}$. Let the resulting components be $V_1, \ldots, V_k$.
2. For each component V_i obtained, perform the following operations:
   (a) **Trim:** For every vertex $v \in V_i$, if $\deg_{G[V_i]}(v) \leq \frac{2}{5}\deg_G(v)$, remove v from V_i. Repeat this operation (re-checking $\deg_{G[V_i]}(v)$ for remaining vertices) until no more vertices satisfy the condition. Let the resulting component be V_i'.
   (b) **Shave:** Define $V_i'' \subseteq V_i'$ as all vertices $v \in V_i'$ for which $\frac{1}{2}\deg_G(v) + 1 < \deg_{G[V_i']}(v)$. This means, in a single iteration, we remove all vertices remaining after step 2.1 for which less than half of their original edges are still within V_i'.
3. Contract the components V_i''.
4. Run Gabow's algorithm on the contracted graph.

Now, we need to prove the following:

1. No minimum cut separates any V_i'' (which justifies contracting them).
2. The number of edges passing between the V_i'' components and the vertices we removed from them is at most $O\left(\frac{m^{1+o(1)}}{\delta(G)}\right)$.

Proof. (part 1: A Minimum Cut Does Not Cross Any V_i'')
We will prove this by showing that for any minimum cut $(C, V \setminus C)$ and for any V_i'', one of the following holds:

$$|V_i'' \cap C| = 0 \quad \text{or} \quad |V_i'' \setminus C| = 0$$

On one hand, the number of edges crossing this internal cut is bounded by the minimum cut value:

$$E(V_i \cap C, V_i \setminus C) \leq \lambda(G) \leq \delta(G)$$

On the other hand, since $G[V_i]$ is a φ-expander, its conductance property gives us a lower bound for any cut within V_i:

$$E(V_i \cap C, V_i \setminus C) \geq \varphi \cdot \min(Vol(V_i \cap C), Vol(V_i \setminus C))$$

Without loss of generality, assume $Vol(V_i \cap C) \leq Vol(V_i \setminus C)$. Thus,

$$Vol(V_i \cap C) = \sum_{v \in V_i \cap C} \deg_G(v) \geq |V_i \cap C| \cdot \delta(G)$$

So,

$$E(V_i \cap C, V_i \setminus C) \geq \varphi \cdot |V_i \cap C| \cdot \delta(G)$$

Combining the two inequalities:

$$\varphi \cdot |V_i \cap C| \cdot \delta(G) \leq \delta(G)$$

Substituting $\varphi = \frac{40}{\delta(G)}$ (as chosen in step 1 of the algorithm):

$$\frac{40}{\delta(G)} \cdot |V_i \cap C| \leq 1$$

Therefore, $|V_i \cap C| \leq \frac{\delta(G)}{40}$

Now we want to show that:

$$\min(|V_i' \cap C|, |V_i' \setminus C|) \leq 2$$

This is true because, again, on one hand:

$$E(V_i' \cap C, V_i' \setminus C) \leq \lambda(G) \leq \delta(G)$$

And on the other hand:

$$E(V_i' \cap C, V_i' \setminus C) = E(V_i' \cap C, V_i') - E(V_i' \cap C, V_i' \cap V_i')$$

Notice that:

$$E(V_i' \cap C, V_i' \cap V_i') < |V_i' \cap C|^2 \leq |V_i' \cap C| \cdot |V_i \cap C|$$

And by the definition of Trim:

$$E(V_i' \cap C, V_i') \leq |V_i' \cap C| \cdot \frac{2}{5}\delta(G)$$

so we get:

$$E(V_i' \cap C, V_i' \setminus C) \geq |V_i' \cap C|\frac{2}{5}\delta(G) - |V_i' \cap C|^2 \geq |V_i' \cap C|\frac{2}{5}\delta(G) - |V_i' \cap C| \cdot |V_i \cap C| \geq |V_i' \cap C|\left(\frac{2}{5}\delta(G) - \frac{\delta(G)}{40}\right)$$

That is:

$$\delta(G) \geq |V_i' \cap C|\left(\frac{2}{5}\delta(G) - \frac{\delta(G)}{40}\right)$$

Hence:

$$|V_i' \cap C| \leq \frac{1}{2/5 - 1/40} = \frac{8}{3}$$

Since $|V_i' \cap C|$ is an integer, we obtain: $|V_i' \cap C| \leq 2$

Next, we need to show that:

$$\min(|V_i'' \cap C|, |V_i'' \setminus C|) = 0$$

Assume for contradiction that $0 < |V_i'' \cap C| \leq 2$. That is, V_i'' intersects a MinCut nontrivially. We know that for any $v \in V_i'' \cap C$, the number of edges from v to V_i'' is greater than $\frac{1}{2}\deg_G(v) + 1$. Since $|V_i'' \cap C| \leq 2$each such vertex v has at most one neighbor in $V_i'' \cap C$ and thus has strictly more than

$\frac{1}{2}\deg_G(v)$ edges to $V_i'' \setminus C$. Therefore, moving v to the other side of the cut, $V_i'' \setminus C$, would reduce the number of crossing edges , which contradicts the minimality of the MinCut. □

Next, we prove part 2.

Claim 8. *The sum of degrees of vertices removed from components during the Trim and Shave steps is at most* $O\left(\frac{m^{1+o(1)}}{\delta(G)}\right)$*, which is also the number of edges that existed between components after the expander decomposition step.*

Proof. **Trim Step**

For every vertex removed during the Trim step, at least $\frac{3}{5}$ of its edges originally connect to vertices outside its resulting component V_i'. Therefore, the number of new edges outside the component is at most $\frac{2}{5} \cdot \deg_G(v)$. On the other hand, the decrease in the number of edges exiting V_i outside the cut is at least $\frac{3}{5} \cdot \deg_G(v) - \frac{2}{5} \cdot \deg_G(v) = \frac{1}{5} \cdot \deg_G(v)$. Thus, in total, the Trim step increases the number of edges between components by at most $2E(V_i, V \setminus V_i)$, meaning the number of new edges between components increases by at most twice the number of edges that exited each component V_i before the Trim operation. Summing over all expander components, we conclude that the Trim step adds at most $O(\frac{m^{1+o(1)}}{\delta(G)})$

Shave Step

Vertices removed from component V_i' during this step are only those for which more than half of their edges previously connected outside the component. Therefore, removing all of them simultaneously costs no more than the number of edges that crossed the component V_i' before the operation. Hence, even after this operation, the number of edges between components remains $O\left(\frac{m^{1+o(1)}}{\delta(G)}\right)$. □

In summary, we have shown that after the expander decomposition steps, followed by the Trim and Shave operations, we can contract the expander components without affecting the minimum cut, thereby reducing the number of edges by a factor related to the minimum degree. Following these steps, by using an algorithm for finding MinCut in $\tilde{O}(m \cdot \lambda(G))$ time, we obtain an algorithm for finding MinCut in $m^{1+o(1)}$ time.

References

[1] 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 (FOCS)*, pages 1158–1167. IEEE, 2020.

[2] HAROLD N GABO. A matroid approach to finding edge connectivity and packing arborescences. *joURNAL OF COMPUTER AND SYSTEM SCIENCES*, 50:259–273, 1995.

[3] Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 384–395, 2021.

[4] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.