CS640 **Computational Complexity**

Constructing Expanders: The Zig-Zag Product

Instructor: Manindra Agrawal *Scribe: Ramprasad Saptharishi*

Over the last few classes we were introduced to expander graphs and we saw how they could be used to reduce the number of random bits required for amplification.

The crucial part in that is that, given a vertex v of the expander, we should be able to pick a random neighbour of that quickly. Though the graph is of huge size (2^m vertices), we want our neighbour computation to be fast. This is captured by *rotation maps.*

1 Rotation Maps

Definition 1. *Let G be a d-regular n vertex graph. The rotation map, denoted by Rot_G, is a map $Rot_G : V \times \{1, 2, \cdots, d\} \longrightarrow V \times \{1, 2, \cdots d\}$ such that $Rot_G(u, i) = (v, j)$ if the i-th neighbour of u is vertex v and the j-th neighbour of v is u.*

From the definition it is clear that this is a permutation on $V \times \{1, 2, \cdots, d\}$ and it is also an involution; that is Rot_G applied twice successively is the identity. Therefore, this map can be represented as a symmetric permutation matrix of dimension $nd \times nd$.

We would want our expander graphs to have rotation maps that are computable in time $\mathsf{poly}(\log n, \log d)$.

2 Constructing Expander Graphs

Expander graphs are available in plenty. Infact, the following result states that almost all graphs are expanders

Theorem 1. *A random d-regular graph on n vertices has its second largest eigenvalue less than $9/10$ with high probability.*

Therefore, the existence of such graphs is clearly not an issue. But we would want to explicitly construct them.

A few such constructions were discovered earlier and all of them had a similar flavour - simple to visualize but really hard to show that they are indeed good expanders or even find efficient rotation maps. To illustrate that, consider the following graph. Let p be a prime and consider a graph on p vertices. For each i, connect it to $i-1, i+1$ and $i^{-1} \bmod p$ (put a self loop on 0). This actually forms an expander graph!

People were looking for explicit constructions of a family of constant degree expander graphs for a long time. The general idea was to take a small expander graph, and somehow blow it up or enlarge it to give a larger expander graph. We shall now look at certain *graph products.*

3 Graph Products

We shall look at some ways by which we can take two graphs G and H and try and get a bigger graph by taking some product between them.

3.1 Powering

Suppose G and H are (n, d, λ) and (n, d', λ') expanders. Define the adjacency matrix of the product graph GH as the product of the adjacency matrices. This may not be a $0, 1$ matrix but think of this as a multigraph where $A_{ij} = k$ means that there are k edges between vertex i and j.

The resulting graph will also be an n vertex graph and have degree dd'. What about the eigenvalue?

Notice that this new normalized adjacency matrix also has $\left|\hat{1}\right\rangle$ as an eigenvector with eigenvalue 1. Take any other eigenvector orthogonal to this. Then the adjacency matrix of H shrinks it by λ' and then G shrinks it by λ. Therefore, the second largest eigenvalue is $\lambda\lambda'$.

Therefore the resulting graph GH is an $(n, dd', \lambda\lambda')$ expander. We shall denote a product of G with itself k times as G^k.

3.2 Tensor Product

Let G be an (n, d, λ) expander and H be a (n', d', λ') expander. The tensor product $G \otimes H$ is defined as follows:

- The vertex set of $G \otimes H$ is $V_G \times V_H$. Vertices can be thought of as (u, u') such that $u \in G$ and $u' \in H$.
- Vertex (u, u') is connected to (v, v') if and only if $(u, v) \in G$ and $(u', v') \in H$.

It is clear from the definition that the number of vertices is nn' and the degree is dd'. And it is not hard to check that the adjacency matrix of this new graph will now be the tensor product of the adjacency matrices of the old graphs. And therefore, the related eigenvalues and eigenvectors are also corresponding products. Therefore, the second largest eigenvalue of $G \otimes H$ is $\max\{\lambda, \lambda'\}$.

Therefore, the resulting graph $G \otimes H$ is an $(nn', dd', \max\{\lambda, \lambda'\})$ expander.

Let us now look at what the two products give us. The powering is good in the sense that it reduces λ, which is good for us. The number of vertices however remain the same. The tensor product on the other hand increases the number of vertices and doesn't change the eigenvalue much since it remains the max of the 2 graphs and is therefore under control.

The problem with both the operations is that the degree of the graph keeps increasing. And with just these two products there is no hope of getting a family of constant degree expanders.

The degree blows up because of the freedom that both the products allows. In case of powering, if you look at GH, it is equivalent to taking one edge in H and then one in G. So in essence you have complete freedom of choosing any edge in G as long as it is incident in the vertex you are on after the H-edge. In case of tensoring, they are essentially two parallel moves, one on G and one on H and therefore allows tremendous freedom.

What we need to do is reduce this freedom; somehow make the edge in one of the graphs "influence" the other edge. This is exactly what happens in the *zig zag* product which finally solves the problem of degree blow-up.

4 The Zig-Zag Product

Suppose G is an (N, D, λ_1) expander and H is an (D, d, λ_2) expander. We can then define what the zig-zag product of the two graphs are. It is denoted by $G ⓩ H$.

We shall first define it informally so that the readers get a good picture of what it is. Take the graph G. Each vertex u has degree D and therefore has D edges going out. Now replace each vertex u by a *cloud* of D vertices such that each new vertex u_i represents the i-th edge going out of u. Therefore, the vertex set of $G ⓩ H$ is of size ND since each vertex in G is now blown up by a D vertex cloud. To make the understanding, we shall think of the new vertices as pairs (u, i) which corresponds to the i-th vertex in the

cloud of u. We hence have ND vertices, identified as clouds, present with absolutely no edges between them. We need to now define the edges.

Any edge in the zig-zag product will correspond to a 3-step walk which is as follows. You start at vertex (u, i) which is the i-th vertex in the cloud of u. Now think of the vertices in this cloud as vertices of H. Take one edge in H to go from (u, i) to (u, j) where j is a neighbour of i in H. Now you are at vertex (u, j). The vertex j corresponds to the j-th edge going out of u; take that edge in G to go to vertex v in G and thereby going from a vertex in cloud u to a vertex in cloud v. Now we would go from u to v through some edge of v, say the k-th edge. This corresponds to (v, k) in the cloud. So we end up there. Now think of the cloud of v as the graph H and go to some neighbour of k, say l. Therefore, you finally end up in (v, l). This three step walk defines a single edge in $G(\mathbb{Z})H$; you then connect (u, i) and (v, l) by an edge.

OK, let us first find out what the degree of the new graph is. How many neighbours does (u, i) have? Let us see what the freedom is. From (u, i) we can take any edge in H for the first step of the walk; that gives us d choices. Then the new vertex defines the intercloud edges so there is no choice there. After that, we take one more edge in H in the new cloud which again gives us d choices. Therefore, the degree of this new graph is d^2.

Infact we can say that (u, i)'s (a, b)-th neighbour is (v, k) if your first walk took the a-th neighbour of a and then in the last step too the b-th neighbour to k. Therefore, the edge labels can also be thought of as tuples (a, b) where $1 \leq a, b \leq d$.

Let us now formally define the edges. Recall that the rotation map $\text{Rot}_{G(\mathbb{Z})H}$ is a map that takes a vertex and edge number and returns the destination vertex and its edge number that we took to get there.

To compute $\text{Rot}_{G(\mathbb{Z})H}((u, i), (a, b))$, let $\text{Rot}_H(i, a) = (j, a')$ and let $\text{Rot}_G(u, j) = (v, j')$ and then let $\text{Rot}_G(j, b) = (k, b')$. Then, define $\text{Rot}_{G(\mathbb{Z})H}((u, i), (a, b))$ as $((v, k), (b', a'))$.

This rotation map therefore defines all the edges of $G(\mathbb{Z})H$. Now for the bounds on eigenvalues.

4.1 Eigenvalue Bounds

We now need to argue that he second largest eigenvalue isn't too large. Note that we don't need the eigenvalue to actually decrease. The graph powering reduces the eigenvalue a lot. Therefore if we reduce it enough

through powering and then use the zig-zag product to get the degree down and a slight increase in eigenvalue will not cause too much of trouble.

The following theorem gives a bound on the eigenvalue of the new graph.

Theorem 2. *The graph $G(z)H$ is an (ND, d^2, λ) expander where $\lambda = 1 - (1 - \lambda_1)(1 - \lambda_2)^2$.*

Proof. The adjacency matrix of $G(z)H$ isn't as difficult as it seems. Every edge in the new graph corresponds to a 3 step walk. You start at (u, i) and go to a neighbour j of i in cloud u and therefore to (u, j). This is like the adjacency matrix of H acting on the second coordinate and keeping the first coordinate fixed; this is what is captured by the matrix $I \otimes H$ where H is the normalized adjacency matrix of the graph H.

From (u, j), you go the j-th neighbour, say v, of u. You would be taking v's k-th edge into v and hence you end up at (v, k). This means that u's j-th neighbour is v and as a k-th neighbour of v. This is exactly the rotation map of G, takes (u, j) and returns (v, k).. Let us call the matrix representing the rotation map as R_G. Then this matrix captures the second step.

The third step is again keeping the first coordinate fixed and the second coordinate changing based on H. Therefore, the third step is $I \otimes H$ as well. Hence, the normalized adjacency of $G(z)H$ is $(I \otimes H)R_G(I \otimes H)$.

We need to compute the 2nd largest eigenvalue of $Z = (I \otimes H)R_G(I \otimes H)$. Write $H = (1 - \lambda_2)J + \lambda_2 C$ where J is the vector with all $1/D$s.

$$\begin{aligned} Z &= (I \otimes H)R_G(I \otimes H) \\ &= (I \otimes ((1 - \lambda_2)J + \lambda_2 C))R_G(I \otimes ((1 - \lambda_2)J + \lambda_2 C)) \\ &= (1 - \lambda_2)^2(I \otimes J)R_G(I \otimes J) + (1 - \lambda_2)\lambda_2(I \otimes C)R_G(I \otimes J) \\ &\quad +(1 - \lambda_2)\lambda_2(I \otimes J)R_G(I \otimes C) + \lambda_2(I \otimes C)R_G(I \otimes C) \\ &= (1 - \lambda_2)^2(I \otimes J)R_G(I \otimes J) + (1 - (1 - \lambda_2))^2 E \end{aligned}$$

where E is the rest of the matrices. After this, it's just some verification using the eigenvectors. It can be shown that $\|E\| \leq 1$. We leave it to the reader to think about it.

As for the other term, look at the matrix $(I \otimes J)R_G(I \otimes J)$. Thinking of it graphically, the first $(I \otimes J)$ preserves the first coordinate and sends the second coordinate to "any" vertex in H. Then R_G then goes to "any" neighbour of G and then the final $I \otimes J$ fixes the first component again. Hence, it can be formally shown as well, $(I \otimes J)R_G(I \otimes J) = G \otimes J$.

Now the only eigenvalues of J are 1 and 0 (0 has multiplicity $D-1$) and therefore the second largest eigenvalue of J is 0. Therefore, the second largest eigenvalue of $G \otimes J$ is λ_1.

Therefore,

$$\lambda \leq (1-\lambda_2)^2\lambda_1 + 1 - (1-\lambda_2)^2 = 1 - (1-\lambda_1)(1-\lambda_2)^2$$

□

4.2 Revisiting Probability Amplification

Recall the part when we discussed amplifying the success probability in an RP algorithm. We had an algorithm that used m random bits and whose error was less than say δ. We used $m + k\log D$ bits to get the error down to $2^{-\Omega k}$. Think of this as a first block of m bits to figure out the first vertex, and then k chunks of $\log D$ bits. In this k chunks of $\log D$ bits, you can recursively apply the procedure. Think of $\log D = m'$ and this looks like k repeated applications of the naive algorithm that uses m' random bits. Therefore, you can think of an expander on $2^{m'} = D$ vertices say of degree d and do a random walk there to reduce the random bits again.

Therefore we would now be using $\log m + \log D + \log d + \log d + \cdots$. One could group the first two chunks to get a chunk of $\log mD$ random bits and then followed by a lot of $\log d$ chunks. This is like a random walk on an expander on mD vertices and degree $2^{O(\log d)}$ which say is d^2. And this is essentially what we get in the zig-zag product!

Thus, we can think of the probability amplification in two steps: start with an expander on 2^m vertices but not necessarily of constant degree D. Now use the smaller expander on D vertices to reduce the random bits even further. This corresponds to using the expander which is the zigzag product of the two expanders.

5 Efficient Computable Family of Constant Degree Expanders

Using the 3 graph products that we discussed, we can now go ahead and construct a family of efficiently computable expander graphs with constant degree. To start with, let us pick a small constant sized graph with $\lambda < 1/2$. This can be picked by bruteforce.

Let H be a $(d^8, d, 1/2)$ expander. Define $G_1 = HH$. Therefore, G_1 will be an d^{16} vertex d^2 regular graph. Now interatively, define G_k for $k \geq 2$ as

$$G_k = (G_{k-1} \otimes G_{k-1})^2 \textcircled{z} H$$

The following claim is very straightforward to check.

Claim 3. *The family of graphs $\{G_i\}$ form a family of expander graphs each of degree d^2. And G_k is a $(8(2^k - 1), d^2, 1/4)$ expander. And further, the rotation maps of G_k can be computed in time $O(k)$ which is logarithmic in the size of the vertex set of the graph and is hence efficient.*