Undirected Connectivity in Log-Space

Miranda Christ Rohan Jha Shivam Nadimpalli

October 16, 2019

ft. Omer Reingold

What is this talk about?

- You're in a new city, and want to get home.
- But, you have terrible memory!

What is this talk about?

- You're in a new city, and want to get home.
- But, you have terrible memory!

A possible solution:

- Walk around randomly
- Either reach home, or run out of patience and give up

What is this talk about?

- You're in a new city, and want to get home.
- But, you have terrible memory!

A possible solution:

- Walk around randomly
- Either reach home, or run out of patience and give up

Q. How much memory does this approach require?

What is this talk about?

- You're in a new city, and want to get home.
- But, you have terrible memory!

A possible solution:

- Walk around randomly
- Either reach home, or run out of patience and give up

Q. How much memory does this approach require?
A. You only need to know where you are. DFS would have to know where you came from.

What is this talk about?

More formally...

USTCON: Undirected s-t Connectivity
Input: $\langle G, s, t \rangle$ where G graph, $s, t \in V(G)$
Output: T or F

- Complete for SL
- STCONN complete for NL

Log-space algorithm $\implies$ L = NL

But why should you care?

$$L \subset SL \subset RL \subset NL \subset P \subset \dots$$

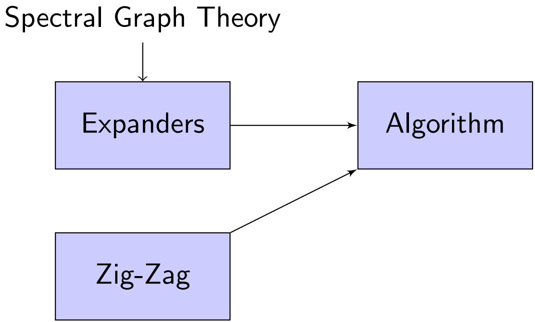
- This algorithm makes the state of complexity theory less pathetic ☺
- Strong hint that randomness isn't needed when space is limited

Outline of Talk

Now, suppose there exist magical graphs on which you can solve USTCON in log space. . .

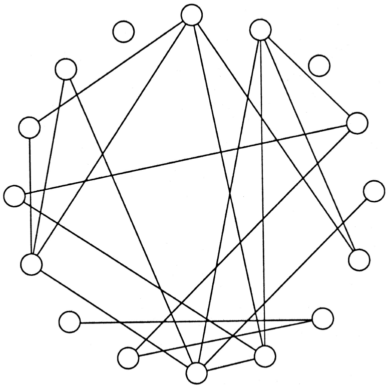
What's a natural thing to do?

Outline of Talk



Expanders

We want a measure of “connectedness” of a graph.



Any guesses?

Edge Expansion

What about this measure?

Definition. The *edge expansion* of a graph $G = (V, E)$ is

$$h(G) = \min_{S \subset V, |S| \leq \frac{n}{2}} \frac{|E(S, \overline{S})|}{|S|}$$

where $E(S, \overline{S}) = \{(u, v) \in E \mid u \in S, v \in V \backslash S\}$.

Edge Expansion

Definition. The *edge expansion* of a graph $G = (V, E)$ is

$$h(G) = \min_{S \subset V, |S| \leq \frac{n}{2}} \frac{|E(S, \overline{S})|}{|S|}$$

where $E(S, \overline{S}) = \{(u, v) \in E \mid u \in S, v \in V \setminus S\}$.

Q. Why $|S| \leq \frac{n}{2}$?

Edge Expansion

Definition. The *edge expansion* of a graph $G = (V, E)$ is

$$h(G) = \min_{S \subset V, |S| \leq \frac{n}{2}} \frac{|E(S, \overline{S})|}{|S|}$$

where $E(S, \overline{S}) = \{(u, v) \in E \mid u \in S, v \in V \setminus S\}$.

Q. Why $|S| \leq \frac{n}{2}$?

Q. Why is $|E(S, \overline{S})|$ in the numerator?

Edge Expansion

Definition. The *edge expansion* of a graph $G = (V, E)$ is

$$h(G) = \min_{S \subset V, |S| \leq \frac{n}{2}} \frac{|E(S, \overline{S})|}{|S|}$$

where $E(S, \overline{S}) = \{(u, v) \in E \mid u \in S, v \in V \setminus S\}$.

Q. Why $|S| \leq \frac{n}{2}$?

Q. Why is $|E(S, \overline{S})|$ in the numerator?

Q. Why is $|S|$ in the denominator?

Examples!

Definition. The *edge expansion* of a graph $G = (V, E)$ is

$$h(G) = \min_{S \subset V, |S| \leq \frac{n}{2}} \frac{|E(S, \overline{S})|}{|S|}$$

where $E(S, \overline{S}) = \{(u, v) \in E \mid u \in S, v \in V \backslash S\}$.

Ex. Compute $h(K_n)$.

Examples!

Definition. The *edge expansion* of a graph $G = (V, E)$ is

$$h(G) = \min_{S \subset V, |S| \leq \frac{n}{2}} \frac{|E(S, \overline{S})|}{|S|}$$

where $E(S, \overline{S}) = \{(u, v) \in E \mid u \in S, v \in V \backslash S\}$.

Ex. Compute $h(K_n)$.

Ex. Compute $h(G)$ where G is a disconnected graph.

What is an Expander?

Informally, an *expander* is a graph with high expansion.

What is an Expander?

Informally, an *expander* is a graph with high expansion.

Q. Is K_n a good expander?

What is an Expander?

Informally, an *expander* is a graph with high expansion.

Q. Is K_n a good expander?

For most practical purposes, we want graphs with low degree.

Some more intuition:

"Sparse but well connected" graph
or
Look like random graphs

Computing $h(G)$

Given a graph G, how would you compute $h(G)$?

Computing $h(G)$

Given a graph G, how would you compute $h(G)$?

- You want the *sparsest* cut.
- Shown to be NP-hard, and best known approximation is $\mathcal{O}(\log n)$ due to Arora, Rao, and Vazirani (2009)

Linear Algebra Review

Spectral graph theory offers a solution! Recall...

Given a $n \times n$ matrix T:

Definition. Non-zero vectors v such that $T \cdot v = \lambda \cdot v$ for some λ are called *eigenvectors*, and the corresponding scalar λ is said to be an *eigenvalue*.

The eigenvalues are given by the roots to $(T - x \cdot I) = 0$ where I is the $n \times n$ identity matrix.

Linear Algebra Review

Recall. . .

Theorem. (Spectral theorem) If M is a $n \times n$ real, symmetric matrix, then there exist real numbers $\lambda_1, \ldots, \lambda_n$ and mutually orthogonal unit vectors $\psi_1, \ldots, \psi_n$ such that for each i, ψ_i is an eigenvector of M with eigenvalue λ_i.

Graphs & Matrices

Let $G = (V, E)$ be a d-regular graph on n vertices.

- The adjacency matrix is A_G is given by

$$A_G(i,j) = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}.$$

- The normalized adjacency matrix M_G is given by $\frac{1}{d} \cdot A_G$.

Both of these are $n \times n$ real, symmetric matrices, but we'll only care about M_G.

Graphs & Matrices

By the spectral theorem, there exist real numbers $\lambda_1, \ldots, \lambda_n$ and mutually orthogonal unit vectors $\psi_1, \ldots, \psi_n$ such that for each i, ψ_i is an eigenvector of M_G with eigenvalue λ_i.

Order these! Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$.

We will call $\{\lambda_1, \ldots, \lambda_n\}$ the *spectrum* of the graph G.

Let $\lambda(G)$ be the second-largest eigenvalue in absolute value.

Graphs & Matrices

Q. Isomorphic graphs will have identical spectra. Is the converse true?

Graphs & Matrices

Q. Isomorphic graphs will have identical spectra. Is the converse true?

Figure: $K_{1,4}$ and $C_4 \cup K_1$ are isospectral but non-isomorphic.

Graphs & Matrices

But what can the spectrum tell us?

Spectral Graph Theory

Claim. If G is d-regular, then $\lambda_1 = 1$.

Proof. Let $\vec{v} = (v_1, \ldots, v_n)$ be a non-zero eigenvector of M_G with eigenvalue λ. WLOG suppose v_1 maximizes $|v_i|$ over all i. As $\vec{v}$ was assumed to be non-zero, $|v_1| > 0$. For arbitrary $i \in V$, let $\Gamma(i) = \{j \in V \mid (i,j) \in G\}$. Then we have

$$|\lambda_1 v_1| = \frac{1}{d} \cdot |(A_G \vec{v})_1| = \frac{1}{d} \cdot \left| \sum_{i \in \Gamma(1)} v_i \right| \leq \frac{1}{d} \cdot \sum_{i \in \Gamma(1)} |v_i| \leq |v_1|.$$

Now, $\vec{v} = (1, \ldots, 1)$ is an eigenvector of M_G with eigenvalue 1, and so $\lambda_1 \geq 1$. We conclude that $\lambda_1 = 1$. $\square$

Spectral Graph Theory

Many other cool results! Here's some that you can try your hand at:

- The multiplicity of λ_1 is equal to the number of connected components of G.
- If G is bipartite, then λ_i and $-\lambda_i$ have identical multiplicities in the spectrum of G for any real number λ_i.

Spectral Graph Theory

Back to business: We wanted to compute $h(G)$...

Theorem. (Cheeger's Inequality) Let $G = (V, E)$ be a finite, connected, d-regular graph and let $\lambda = \lambda(G)$. Then we have

$$\frac{1-\lambda}{2} \leq \frac{h(G)}{d} \leq \sqrt{2(1-\lambda)}.$$

Spectral Graph Theory

Given a connected graph G, we will call $1 - \lambda(G)$ the *spectral gap* of G.

- If large spectral gap, $h(G)$ is greater
- Greater $h(G)$, better connected

Applications of Expanders

- Error correcting codes
- Derandomization and pseudorandomness
- MCMC
- Metric embeddings

See Hoory-Linial-Wigderson's survey for more. Let's get back to USTCON. . .

USTCON on Expanders

Claim. If G is D-regular, connected, non-bipartite then

$$\lambda(G) \leq 1 - \frac{1}{DN^2}.$$

Proof can be found in Alon-Sudakov (2000).

USTCON on Expanders

Claim. Expanders have diameter $\mathcal{O}(\log n)$ where n is the number of vertices.

Proof. Let s, t be two nodes in G. We want to show that $d(s,t) \leq \mathcal{O}(\log n)$. Consider the following procedure:

- Initialize $S, T = \emptyset$ and $i = 0$.
- While $|S| \leq n/2$:
  - Add all vertices connected to any $v \in S$ to S
  - $i = i + 1$
- Same for T

Note that diameter of G is i.

USTCON on Expanders

Claim. Expanders have diameter $\mathcal{O}(\log n)$ where n is the number of vertices.

Proof. (contd.) Now, during each step of adding to S, we add at least $\frac{h(G)}{3}$ vertices, i.e. the size of S grows by at least $c = \left(1 + \frac{h(G)}{3}\right)$.

So inner each loop runs for at most

$$\log_c \frac{n}{2} = \log \frac{n}{2} \times \frac{1}{\log\left(1 + \frac{h(G)}{d}\right)}$$

steps, i.e. $i \leq 2 \log \frac{n}{2} \times \frac{1}{\log\left(1+\frac{h(G)}{d}\right)} = \mathcal{O}(\log n)$. $\square$

Powering

Q. How do we make an expander while preserving connectivity?

Powering

Q. How do we make an expander while preserving connectivity?

A. Add edges within connected components (powering!)

Powering

Given a graph G on nodes $[N]$, G^t is a graph on $[N]$ with an edge from u to v for *every* path from u to v in G of length t.

This is equivalent to taking the t^{th} power of the adjacency matrix.

Since we include self-loops for every node, this preserves connectivity.

Rotation Map

Note that we don't have enough space to take powers of the adjacency matrix. This leads us to a new graph representation:

Recall that G is a graph of degree D on N nodes.
$Rot_G : [N] \times [D] \to [N] \times [D]$

$Rot_G(v, i) = (w, j)$
Meaning the i^{th} edge leaving node v is the j^{th} edge leaving node w

Rotation Map

In the context of powering:

Recall that G is a graph of degree D on N nodes.
$Rot_{G^t} : [N] \times [D]^t \to [N] \times [D]^t$

$Rot_{G^t}(v, (a_1, ..., a_t)) = (w, (b_1, ..., b_t))$
$(a_1, ..., a_t)$ represents a sequence of edge numbers to take starting from node v, and w is where we end up.

Powering

Powering gives us several nice properties:

- $\lambda(G^t) = \lambda(G)^t$
- If G is D-regular, G^t is D^t-regular

Powering

Q: Why isn't powering good enough?

- Obvious approach is to take G^N; does this work?
- Savitch's Algorithm uses powering in $O(\log^2 N)$ space

Savitch 1970

Savitch's algorithm solves STCON in $O(\log^2 N)$ space

Defines G^{sq} as the graph with *one* edge from u to v if connected by a path of at most length 2 in G

Computes $(G^{sq})^{\log N}$, then checks if there is an edge from s to t

Products

Goal: Reduce degree without hurting expansion too much

Replacement product $\rightarrow$ zig-zag product

Replacement product

Denoted by $G \circledR H$

Think of H as much smaller than G

Intuitively, we replace every node of G with a copy of H (a cloud). Then, if two nodes shared an edge in G, we add an edge between their clouds.

Replacement product

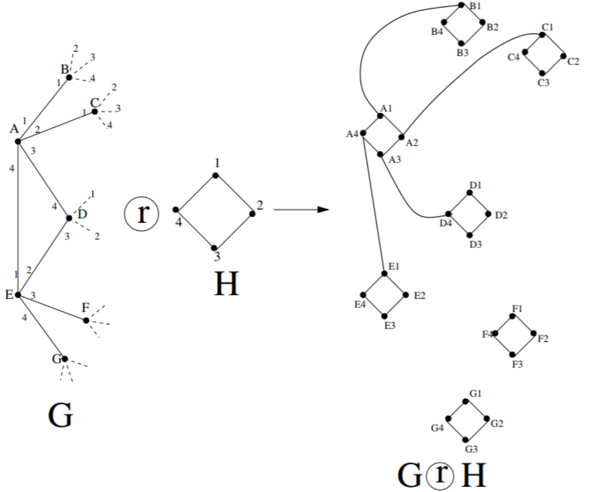


Figure 1: The replacement product of G and H (not all edges shown)

Replacement product

Inputs: G, D-regular on N nodes, and H, d-regular on D nodes

Output: $G \textcircled{r} H$, d+1-regular on $N \cdot D$ nodes

Note the reduction in degree!

Zig-zag product

Q: Why construct this product?

Clean bounds on its 'damage' to the spectral gap!

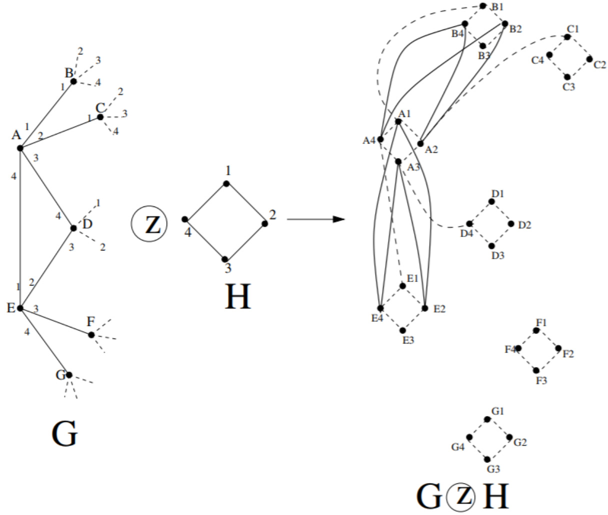
Zig-zag product

Again, think of H as much smaller than G

Intuitively, take the replacement product of H and G, keep all of the nodes, remove all the edges, and only add an edge between u and v if v could have been reached in the replacement product by:

1. Taking a small step within u's cloud
2. Taking a big step between u's cloud and v's cloud
3. Taking a small step within v's cloud

Zig-zag product



Zig-zag product

Inputs: G, D-regular on N nodes, and H, d-regular on D nodes

Output: $G \circledR H$, d^2-regular on $N \cdot D$ nodes

Again, a reduction in degree!

Zig-zag product

Recall that if G is an (N, D, λ)-graph, it has degree D, N nodes, and $\lambda(G) = \lambda$

Corollary 2.10: If G is an (N, D, λ)-graph and H is a (D, d, α)-graph, then

$$1 - \lambda(G \textcircled{z} H) \geq \frac{1}{2}(1 - \alpha^2) \cdot (1 - \lambda)$$

Our tools

Powering: improves connectivity but degree blows up

Zig-zag: reduces degree to a constant, without a terrible reduction in connectivity

Main idea: alternately power and zig-zag to improve connectivity while keeping the degree constant

Main transformation

Inputs: G, a D^{16}-regular graph on $[N]$ and H, a D-regular graph on $[D^{16}]$

Transformation:

1. Set l as the smallest number such that $(1 - \frac{1}{D^{16}N^2})^{2^l} < \frac{1}{2}$ (this is $O(\log N)$)
2. Set $G_0 = G$, and for $i > 0$, define G_i recursively by $G_i = (G_{i-1} \text{ⓩ} H)^8$
3. Let $T(G, H) = G_l$

Output: G_l, a D^{16}-regular graph on $[N] \cdot ([D^{16}])^l$

Main transformation

Will show two facts on the board:

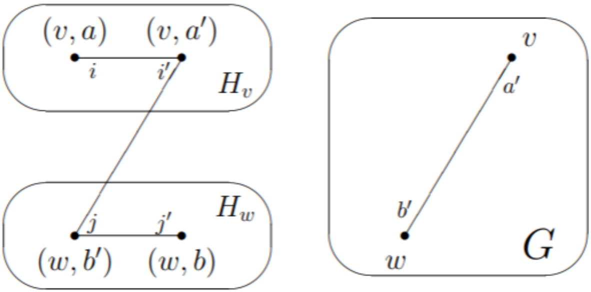
1. If $\lambda(H) \leq \frac{1}{2}$ and G is connected and non-bipartite then $\lambda(T(G, H)) \leq \frac{1}{2}$
2. The transformation can be run in log-space

Back to zig-zag

Formal definition of $Rot_{G \textcircled{z} H}((v, a), (i, j))$

1. Let $(a', i') = Rot_H(a, i)$
2. Let $(w, b') = Rot_G(v, a')$
3. Let $(b, j') = Rot_H(b', j)$
4. Output $((w, b), (j', i'))$

Back to zig-zag



Algorithm

Given a graph G and nodes s, t:

- make $H = (D_e^{16}, D_e, 1/2)$-graph
- preprocess G to get G_{reg}, a D_e^{16}-regular graph
- compute $G_{exp} = \mathcal{T}(G_{reg}, H)$
- enumerate all $O(\log n)$ length paths from s; check if any end up at t

Making H

Recall H is a $(D_e^{16}, D_e, 1/2)$-graph, where D_e is some constant.

Easy: precompute H and store in constant space.

Preprocessing G

1. Replace every vertex of degree $d > 3$ by a cycle of length d
2. Connect each of the d new vertices to a distinct neighbor of the old vertex
3. Create enough self loops that each vertex has degree D_e^{16}

Now we apply the main transformation.

Algorithm

We have a graph G_{exp} with diameter $k = O(\log N)$ and degree d. Now what?

Observe there are d^k paths originating from node s. We check each and see if we end up at t.

We need only know which path we're currently traversing, and where in that path we are

- current path: $O(\log N)$ bits
- current position in path: $O(\log N)$ bits

Algorithm

But wait... we can't actually store G_{exp}

Recall that at the end of the main transformation, we'll have a D_e^{16}-regular graph on $[N] \times ([D_e^{16}])^{O(\log N)}$ nodes. That's a lot!

But that's ok! We can compute edges only as we need them.

Algorithm

This is exactly what the rotation map is good for!

Recall that we input a node number and edge number, and the rotation map tells us which node we end up at.

We can compute the rotation map of the main transformation recursively with $O(\log N)$ space.

Summing Up

So, to see if s and t are connected:

We enumerate all $O(\log N)$ paths originating from s in the transformed graph.

We traverse each path, and for each edge, compute where it leads us (via the transformation).

We check if any of these paths lead us to t. Done!