Derandomization of undirected s-t connectivity

Ashley Chen and David Wu

July 9, 2022

1 Introduction

The undirected st-connectivity problem, denoted USTCON, asks whether or not, given as input an undirected graph G and two vertices s and t, the two vertices have a path connecting them. The time complexity to solve this problem is well-known to be linear with basic search algorithms such as breadth-first search (BFS) and depth-first search (DFS). However, both of these algorithms require a linear space complexity as well. The question then arises on whether or not it is possible to construct algorithms with more optimized space complexity.

1.1 The Randomized Algorithm

In the second lecture of our very own 18.434 class, we discussed a randomized log-space algorithm for this problem. This is the same algorithm as proposed by Aleliunas, Karp, Lipton, Lovász, and Rackoff in 1979 [1]. The algorithm depends on showing that a random walk constructed by starting from any vertex and choosing the next edge with uniform probability across all edges incident to the last vertex, will visit all the vertices of the connected component of a graph in a polynomial number of steps. Then, the algorithm only needs to record the name of the current vertex and a counter to keep track of the number of steps already taken. To prove that this algorithm has a small one-sided error, we used spectral graph theory to analyze the transition matrix and later the lazy transition matrix to deal with bipartite graphs. We also needed to prove an upper bound for the second largest eigenvalue of the transition matrix; a similar lemma shows up in the derandomized algorithm as well.

The existence of a randomized log-space algorithm then motivates the search for a derandomized algorithm without increasing the space complexity. This problem serves as a case study in understanding the relationship between space and randomness. Furthermore, studying USTCON helps to understand the relationship between the space complexity classes, especially the open question on whether or not RL = L, where RL is the class of problems decidable by randomized log-space algorithms with one-sided error.

1.2 The Derandomized Algorithm

The essence of the derandomized algorithm that we will show in this paper is to make the graph more connected so that it is easier to enumerate through a connected component's paths. Notably, constant-degree expander graphs have logarithmic diameter, so all of their paths can be enumerated in polynomial time and log space. Thus we want to manipulate our graph to become a constant-degree expander. Roughly, the algorithm to solve USTCON can be split into three stages:

1. Given any graph G, turn it into a D-regular graph where each connected component is non-bipartite.
2. **Main Transformation:** Turn each connected component into an expander graph. This is done by iteratively repeating the following two steps a logarithmic number of times:
   (a) Raise the graph to some constant power; then

(b) Reduce the degree of the graph via a graph product with a constant-size expander.

3. Solve the s-t connectivity problem on the resulting graph

Beyond showing that this deterministic algorithm correctly solves USTCON, we need to show that this algorithm runs in polynomial time and logarithmic space. Furthermore, we can even provide a path from s to t given that they are connected; for a discussion on this see the paper by Omer that we followed [3].

2 Background

In this section we aim to individually review and define the techniques and topics used for the main transformation of the algorithm.

2.1 Graph Representations

Given a graph $G = (V, E)$, where V is its vertex set, and E its edge set, we want a more efficient representation of G's connectivity. Most often, **adjacency matrices** are used to describe graphs. We recall that the entry (u, v) of the adjacency matrix A of a graph G is the number of outgoing edges from vertex u to v. We also recall that a graph G is **undirected** if for every outgoing edge from u to v, there is an ingoing edge from v to u; that is, A must be symmetric. Finally, we recall that a graph G is D**-regular** if each vertex has exactly D outgoing edges. In terms of A, this means that each row and column of A sums up to D.

2.1.1 The Rotation Map

Another way of representing graphs is motivated by the inclination to index incident edges and neighboring vertices of a vertex v relative to v itself. This also comes in use when describing walks, since you can describe the vertex and edge you just came from. With this in mind, we provide the following definition of the **rotation map** of a graph:

Definition 2.1. *The* ***rotation map*** Rot_G *of a* D*-regular undirected graph* $G = (V, E)$ *with* $|V| = N$ *is a mapping* $[N] \times [D] \longrightarrow [N] \times [D]$ *such that* $Rot_G(u, i) = (v, j)$ *if the* i^{th} *edge incident to* u *leads to* v*, and from* v*'s perspective, the edge is the* j^{th} *edge incident to* v*.*

Thus, the rotation map helps to keep track of the same edge that may be indexed differently according to which vertex it is recorded with respect to, making it easy to keep track of paths. Rotation maps are necessary in order for our algorithm to be in log-space for that reason.

Another example is that we can define the adjacency matrix A of a graph G with respect to its rotation map, where each entry (u, v) is the magnitude of the set $\{(i, j) \in D^2 | Rot_G(u, i) = (v, j)\}$.

2.2 Expander Graphs and s-t Connectivity

We want to transform all input graphs into expander graphs because they have the unique property of being both sparse and still highly connected. As a result, we can show that the diameter of the graph can be bounded to be logarithmically long with respect to the number of vertices, and we can construct a simple algorithm to determine s-t connectivity for constant-degree expanders. These are the main goals of this subsection.

2.2.1 Spectral and Vertex Expansion

There are three main ways to measure the expansiveness of an expander graph: edge expansion, spectral expansion, and vertex expansion. We will introduce spectral expansion as a way to define the expansiveness of the graphs we work with, but to show logarithmic diameter, we will introduce and use vertex

expansion. Later on, we will still use spectral expansion to analyze the expansiveness of manipulated expander graphs, in particular the graph products of expander graphs in Section 2.4.

To define spectral expansion, we consider the **normalized adjacency matrix** M of a D-regular undirected graph G, which is its adjacency matrix divided by D. In class, we also defined this as the transition matrix of G. Since G is D-regular, we know that the vector $1_N = (1, 1, \ldots, 1) \in \mathbb{R}$ is an eigenvector of M, with an eigenvalue of 1. From our 18.434 class, we also know that all other eigenvalues have absolute magnitude at most 1. It turns out that the eigenvalue with the second largest absolute magnitude is a very useful metric for expansiveness, so we denote it $\lambda(G)$.

Definition 2.2. *A D-regular undirected graph G on N vertices such that $\lambda(G) \leq \lambda$ is an (N, D, λ)-**graph**.*

Another useful metric for spectral expansion is the **spectral gap**, which we describe below:

Definition 2.3. *Given the normalized adjacency matrix of a graph G, the **spectral gap** is the difference between its largest and second largest eigenvalues.*

Since the largest eigenvalue is 1, we can rewrite the expression as $1-\lambda_2$, where λ_2 is the second largest eigenvalue. An important finding relating $\lambda(G)$ to N and D of a D-regular, connected, non-bipartite graph G that is necessary for our main derandomized algorithm is the following:

Lemma 2.4. *(cf. [2]) For every D-regular, connected, non-bipartite graph G on $[N]$, it holds that $\lambda(G) \leq 1 - \frac{1}{DN^2}$.*

Next, we move on to vertex expansion:

Definition 2.5. *A graph G has the property of **vertex expansion** if for every $\lambda < 1$ there exists $\epsilon > 0$ such that for every (N, D, λ)-graph G and for any set S of at most half of the vertices in G, at least $(1 + \epsilon) \cdot |S|$ vertices of G are connected by an edge to some vertex in S. Even stronger, at least $\epsilon \cdot |S|$ of the vertices are outside of S.*

2.2.2 s-t Connectivity

With an understanding of vertex expansion, we then can move on to providing an algorithm for solving the third stage of the derandomized algorithm: solving the s-t connectivity problem on an expander graph.

Lemma 2.6. *Given a (N, D, λ)-graph G and two vertices s and t, there exists a path between these two vertices of length $O(\log N)$.*

Proof. We independently and iteratively apply vertex expansion to a set S and T of vertices, starting with $S = \{s\}$ and $T = \{t\}$. By Definition 2.5, we know there exists an $\epsilon > 0$ such that either $|S| > \frac{N}{2}$ or at least $(1 + \epsilon) \cdot |S|$ vertices of G have an edge to a vertex in S, and that either $|T| > \frac{N}{2}$ or at least $(1 + \epsilon) \cdot |T|$ vertices of G have an edge to a vertex in T. In the first case for S and T, we can stop. In the second case, we take the $(1 + \epsilon) \cdot |S|$ or $(1 + \epsilon) \cdot |T|$ as our new S or T respectively. Since at each step, $|S|$ and $|T|$ grows exponentially, we know it'll take $l_S = O(\log n)$ and $l_T = O(\log n)$ applications of vertex expansion to find two sets containing all vertices connected to s and t respectively such that each set has more than $\frac{N}{2}$ vertices. Then by the Pigeonhole Principle, there exists a vertex v in both S and T that has distance at most l_S to s and l_T to t. Thus by connecting these paths, we have a path from s to t that has length at most $l_S + l_T$, which is $O(\log n)$. □

With this, we can provide an algorithm $\mathcal{A}_{\text{exp}}$ to solve s-t connectivity for expander graphs.

Theorem 2.7. *Given as input a (N, D, λ)-graph G, where D is constant, there exists an algorithm $\mathcal{A}_{\text{exp}}$ that takes $O(\log D \cdot \log N)$ space and polynomial time such that it correctly solves the s-t connectivity problem.*

Proof. Since we know from Lemma 2.6 that the diameter of G is $l = O(\log n)$, we have $\mathcal{A}_{\text{exp}}$ merely check all possible paths of length l from s. Since G is D-regular, there are at most D^l paths. If the algorithm finds a path that ends with t or has t in it, $\mathcal{A}_{\text{exp}}$ returns that s and t are connected. Otherwise, if $\mathcal{A}_{\text{exp}}$ finishes enumerating through all these paths without returning, then it outputs that they are not connected.

This algorithm is correct because if s and t are connected, by Lemma 2.6 we know the length of their path is at most l so $\mathcal{A}_{\text{exp}}$ would check that path and output connected. For the converse, it holds because $\mathcal{A}_{\text{exp}}$ doesn't output until it finds an explicit path connecting s and t.

For runtime analysis, we note that D is constant, so $D^{O(\log n)}$ ends up being polynomial. For space analysis, we note that $\mathcal{A}_{\text{exp}}$ only needs to store the current path that it's checking, as an order for iterating through the paths can be predetermined. Then, we keep track of each edge of the path by indexing it from $[D]$ relative to the vertex it is outgoing from. Thus each edge index takes $O(\log D)$ space, and there are $O(\log N)$ edges in the path, so in total $\mathcal{A}_{\text{exp}}$ takes $O(\log D \cdot \log N)$ space. □

2.3 Graph Powering

The first step of the main transformation of the derandomized algorithm aims to make a graph more like an expander graph through the process of powering. We provide two equivalent definitions:

Definition 2.8. *(Informal) The t^{th} **power** of a graph $G = (V, E)$ is a graph G^t whose vertex set is V, where an edge (u, v) is in its edge set if there exists a walk of length exactly t from u to v.*

Definition 2.9. *(With rotation maps) The t^{th} **power** of a graph $G = (V, E)$ where $|V| = N$ and the vertices are indexed by $[N]$ is the graph G^t whose rotation map is given by $Rot_{G^t}(v_0, (a_1, a_2, \ldots, a_t)) = (v_t, (b_t, b_{t-1}, \ldots, b_1))$, where these values follow the rule $(v_i, b_i) = Rot_G(v_{i-1}, a_i)$ for $1 \leq i \leq t$.*

Intuitively, the rule mentioned by the rotation maps definition enforces that there is an edge between all $t + 1$ vertices, thus a walk of exactly length t as described in the informal definition. Note that self-loops are allowed, and if the walk from u to v is distinct, then that edge is distinct. Thus, for a D-regular graph G, the t^{th} power of G is a D^t-regular graph.

From our understanding of eigenvalues and adjacency matrices, we can see that the following lemma is true:

Lemma 2.10. *Given G an (N, D, λ)-graph and a constant t, G^t is an (N, D^t, λ^t)-graph.*

2.4 Graph Products

The second step of the main transformation is meant to deal with the increased degree that is a by-product of powering a graph. Both provided graph products use a smaller graph H and combines it with the original graph G to maintain the size and connectivity of G but with the degree of H instead. Furthermore, these graph products don't hurt the expansion done by graph powering. For both, we assume that G is a D-regular graph on N vertices, and H is a d-regular graph on D vertices, where $d < D$.

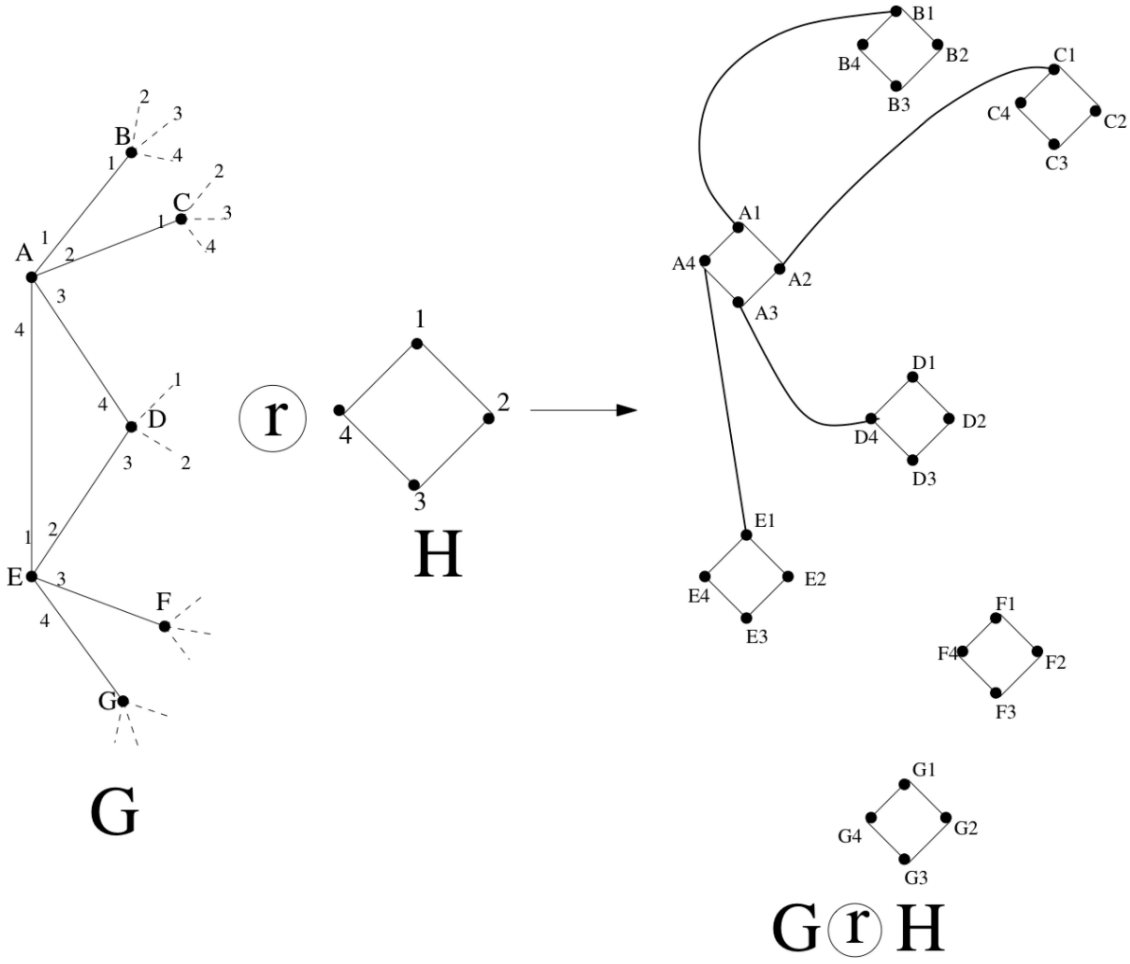
2.4.1 The Replacement Product

We build the replacement product $G \textcircled{r} H$ as follows:

1. For each vertex v in G, replace v with a copy H_v of H, also known as a *cloud*. Each cloud maintains the same edges that H has.

2. For each of the D vertices v_i of each cloud H_v, add another edge from v_i to a vertex in the cloud of the i^{th} neighbor of v in G. Do this in such a way that each vertex of a cloud only has one neighboring vertex from a different cloud.

We provide an example in the image below, taken from [5].



Note that the replacement product now becomes a $(d+1)$-regular graph.

2.4.2 The Zig-Zag Graph Product

This graph product builds upon the replacement product, and is the graph product used in the main transformation. Thus, we will also prove that the expansion of G isn't hurt much after applying the zig-zag graph product.

First, we build the zig-zag graph product $G(z)H$ in the following way:

1. Build the replacement product $G(r)H$.
2. The set of vertices of $G(z)H$ is the same as in $G(r)H$. Thus a vertex in $G(z)H$ would be (u, a) where u is in G and a in H.
3. For each vertex (u, a) within a cloud, $((u, a), (v, b))$ is an edge in $G(z)H$ if there exists a path $((u, a), (u, c), (v, d), (v, b))$ of length 3 in the replacement product such that:
   - (u, c) is a vertex in the same cloud as (u, a), and $((u, a), (u, c))$ is an edge in $G(r)H$. In other words, (a, c) is an edge in H;

- (v, d) is a vertex in a different cloud as (u, a) and (u, c), and $((u, c), (v, d))$ is an edge in $G ⓡ H$; and,
- (v, b) is a vertex in the same cloud as (v, d), and $((v, d), (v, b))$ is an edge in $G ⓡ H$. In other words, (d, b) is an edge in H.

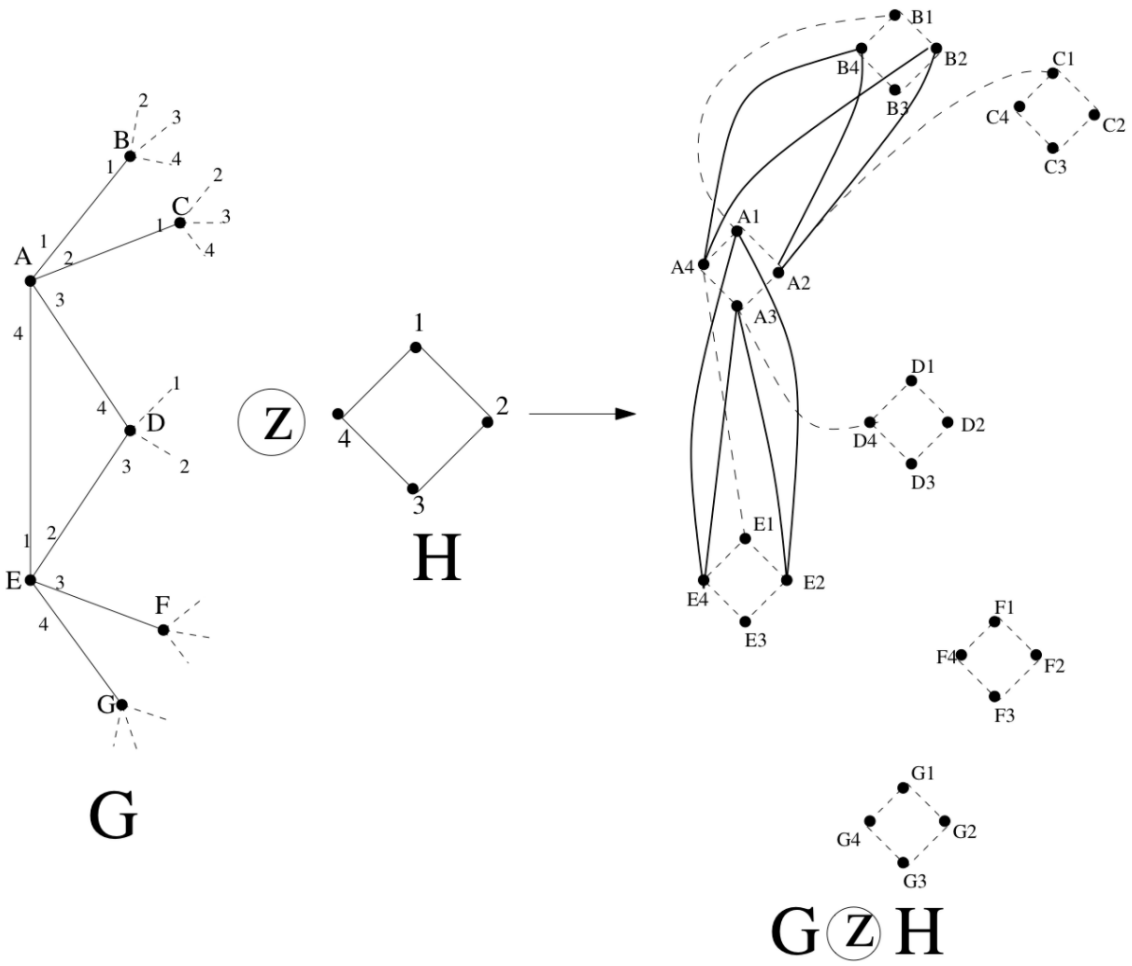
Since this is the graph product that we will be using in our algorithm, we provide the formal rotation map construction as well:

Definition 2.11. *(With rotation maps) If G is a D-regular graph on $[N]$ with rotation map Rot_G and H is a d-regular graph on $[D]$ with rotation map Rot_H, then their **zig-zag product** $G ⓩ H$ is defined to be the d^2-regular graph on $[N] \times [D]$ whose rotation map $Rot_{G ⓩ H}$ is as follows:*

$Rot_{G ⓩ H}((v, a), (i, j))$:

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* $((w, b), (j', i'))$.

Chaining multiple of these paths together from the informal definition makes the path look like a zig-zag going between the different clouds, thus earning its name. [5] extends their replacement product example to provide an example of the zigzag product, which we show below.



Note that each vertex in $G(z)H$ has degree d^2, since there are d options for the (u, x) edge, 1 option for the (x, y) edge, and d options for the (y, v) edge.

Next we want to show that this graph product doesn't hurt the expansion of G, which we can measure by comparing the spectral gap as defined in Definition 2.3 of G and $G(z)H$. While the theorem provided by [4] calculates an exact function of $\lambda(G(z)H)$, what we really need is the following corollary, which uses algebra from the result of the theorem to calculate the spectral gap.

Corollary 2.12. *Given G a (N, D, λ)-graph and H a (D, d, α)-graph, then*

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

3 Transformation into Expander Graphs

Definition 3.1. *Let G be a D^{16} regular graph on N vertices and H be a D-regular graph on D^{16} vertices. We define the following graph transformation $\mathcal{T}(G, H)$.*

- *Set $\ell = 2\lceil \log(DN^2) \rceil$ and $G_0 \leftarrow G$.*
- *For i from $1, \ldots, \ell$, compute*
$$G_i \leftarrow (G_{i-1}(z)H)^8.$$

The intermediate transformations G_i will be denoted as $\mathcal{T}_i(G, H)$, and the final transformation G_ℓ is also denoted as $\mathcal{T}(G, H)$.

We will argue that this transformation $\mathcal{T}$ possesses two crucial properties:

- It turns each connected component into an expander graph without a huge blowup in the size of the graph. In particular, if $\ell = O(\log n)$ then the transformed graph G_ℓ has poly(n) vertices.
- The transformation can be computed in log space.

This graph transformation is precisely the required transformation to solve USTCON using Theorem 2.7, which works for expander graphs that are connected. First, we show that $\mathcal{T}$ preserves connectedness of the input G, by demonstrating that the second largest eigenvalue is bounded away from 1.

Lemma 3.2. *If $\lambda(H) \leq \frac{1}{2}$ and G is a connected, non-bipartite graph, we have $\lambda(\mathcal{T}(G, H)) \leq \frac{1}{2}$.*

Proof. From class we know that for D-regular graphs on N vertices that $\lambda(G) \leq 1 - \frac{1}{DN^2}$. Using the fact that $(1-x)^2 \leq 1 - 1.5x$ for $0 \leq x \leq 0.5$, we obtain that for the choice of ℓ that $(1 - \frac{1}{DN^2})^{2^\ell} \leq \frac{1}{2}$. It suffices to show then that for each i we have $\lambda(G_i) \leq \max\left\{\lambda(G_{i-1}), \frac{1}{2}\right\}$. Using Corollary 2.12, we have

$$\lambda(G_{i-1}(z)H) \leq 1 - \frac{1}{2}\left(1 - \frac{1}{4}\right) \cdot (1 - \lambda(G_{i-1})).$$

Since $G_i = (G_{i-1}(z)H)^8$, Lemma 2.10 then implies that

$$\lambda(G_i) \leq \left[1 - \frac{1 - \lambda(G_{i-1})}{3}\right]^8.$$

If $\lambda(G_{i-1}) < 1/2$ then direct computation shows that $\lambda(G_i) \leq \frac{1}{2}$. Otherwise we can show that $(1 - \frac{1}{3} \cdot (1 - \lambda(G_{i-1})))^4 \leq \lambda(G_{i-1})$, and the conclusion follows. □

Next we will state and quickly give a proof sketch for a simple fact. Essentially, the transformation $\mathcal{T}$ respects the connected components of $G_0 = G$.

Fact 3.3. *Let G, H be inputs to $\mathcal{T}$ and S be a connected component of G. Then*

$$\mathcal{T}(G|_S, H) = \mathcal{T}(G, H)|_{S \times [D^{16}]^\ell}.$$

Proof sketch. The proof goes by induction. The base case is trivial. For the inductive step, we need only unravel the definition of the zig-zag product and graph powering to complete the proof. □

Lemma 3.4. *Let D be a constant. If G is a D^{16} regular graph on N vertices and H is a D regular graph on D^{16} vertices, then the transformation $\mathcal{T}(G, H)$ can be computed in $O(\log N)$ space.*

Rather than provide a rigorous proof of the above lemma, we provide some intuition that can be made precise with additional care. Refer to [3] for more details.

Proof sketch/intuition. When we evaluate $\mathrm{Rot}(G_{i+1})$, we are only required to reuse evaluations of $\mathrm{Rot}(G_i)$ or a constant amount of additional memory (which may also be shared across the layers of the recursion). Another way of phrasing this is that we only need a constant size counter (which takes $O(\log n)$ space). There are some subtleties with accounting here which we brush away for the sake of exposition. □

4 Main Algorithm

In this section we describe a deterministic algorithm for USTCON in logspace.

Theorem 4.1. USTCON $\in$ L.

Proof. The crux of the algorithm, as described earlier, is to transform the input graph G into one where each connected component is an expander. USTCON can be solved in logspace on expanders.

To do so, we want to apply the transformation $\mathcal{T}$ defined in Definition 3.1. But to do so, we need the input G to be a regular graph, so we describe a subroutine for converting G (which is not necessarily regular) into G_{reg}, a D^{16} regular graph whose vertices are pairs (v, w) of vertices in G. Here, D is a large enough constant such that we can pick H to be a $(D^{16}, D, \frac{1}{2})$ expander.

Stage 1 of the algorithm, as in Section 1.2, is as follows. In particular, define $\mathrm{Rot}(G_{\mathrm{reg}}) : ([N] \times [N]) \times [D^{16}]$ as follows:

- $\mathrm{Rot}(G_{\mathrm{reg}})((v, w), 1) = ((v, w'), 2)$ where $w' = w \pmod N + 1$.
- $\mathrm{Rot}(G_{\mathrm{reg}})((v, w), 2) = ((v, w'), 1)$ where $w' = w - 1 \pmod N$ but $w' = N$ if $w = 1$.
- If $(v, w) \in E$ then $\mathrm{Rot}(G_{\mathrm{reg}})((v, w), 3) = ((w, v), 3)$, otherwise it equals $((v, w), 3)$.
- For $3 < i \leq D^{16}$ we have $\mathrm{Rot}(G_{\mathrm{reg}})((v, w), i) = ((v, w), i)$.

The first two bullets establish that the nodes $v \times [N]$ are connected by a cycle of length N. The third establishes that edges only exist between (v, n) and (v', n') for $v \neq v'$ whenever v and v' are adjacent in G. The last bullet are the self loops that saturate the graph to be D^{16} regular.

Note that each transformation only requires a reassignment of a constant number of pointers so is logspace. We check that $\mathrm{Rot}(G_{\mathrm{reg}})$ satisfies the requirements for Definition 3.1. Since it's given by a rotation map, it is in fact D^{16} regular. Due to the self loops, the connected components are also non-bipartite, which is required to apply Lemma 3.2.

Now we are in stage 2 of the algorithm. We need a $(D^{16}, D, \frac{1}{2})$ expander H, which we can construct explicitly (via our discussion in class) or exhaustively evaluated (since the search space is small enough). Next, we define $G_{\mathrm{exp}} \triangleq \mathcal{T}(G_{\mathrm{reg}}, H)$. We need to check that G_{exp} satisfies the hypotheses for the easy algorithm on expander graphs.

Let S be the connected component that s belongs to. We argue that the corresponding part of G_{exp}, defined as $S_{\mathrm{exp}} \triangleq S \times [N] \times [D^{16}]^{\ell}$, is a connected component in G_{exp}. First, we show that S_{exp} is disconnected from the rest of G_{exp}; this demonstrates that it is a union of connected components.

To see this, we first establish that $S_{\mathrm{reg}} \triangleq S \times [N]$ is a connected component of G_{reg}. Each vertex $v \in G$ induces a set of vertices $v \times [N]$ in G_{reg}, which are all connected by the cycle. On the other hand, the construction respects the connectedness of S. So indeed $S \times [N]$ is a connected component of G_{reg}.

Next, we can apply Fact 3.3 with $G = G_{\text{reg}}$ and $S = S_{\text{reg}}$ to conclude that S_{exp} is disconnected from the rest of G_{exp}. Finally, we apply Lemma 3.2 to see that $\lambda(G_{\text{exp}}|_{S_{\text{exp}}}) \leq \frac{1}{2}$, i.e. S_{exp} is connected. Hence the claim is proved.

From here, we are in Stage 3 of the algorithm. We invoke the algorithm $\mathcal{A}_{\text{exp}}$ from Theorem 2.7 as a subroutine, on nodes $s' = (s, 1^{\ell+1})$ and $t' = (t, 1^{\ell+1})$ in G_{exp}. Our algorithm $\mathcal{A}$ outputs the same output as $\mathcal{A}_{\text{exp}}$.

To argue that the algorithm is indeed logspace, it suffices to argue that it is comprised of a constant number of logspace subroutines. In particular,

- The transformation from G to G_{reg} is logspace as argued earlier.
- The transformation of G_{reg} to G_{exp} is logspace due to Lemma 3.4.
- The algorithm $\mathcal{A}_{\text{exp}}$ runs in logspace due to Theorem 2.7

Now, for correctness, since G_{exp} is a valid input to $\mathcal{A}_{\text{exp}}$, it suffices to show that s' and t' live in the same connected component in G_{exp}. Indeed, in G, s and t both belong to S, so both belong to X as defined above. Thus we have provided a logspace algorithm for USTCON. □

References

[1] Romas Aleliunas, Richard M Karp, Richard J Lipton, László Lovász, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 218–223. IEEE Computer Society, 1979.

[2] Noga Alon and Benny Sudakov. Bipartite subgraphs and the smallest eigenvalue. *Combinatorics, Probability and Computing*, 9(1):1–12, 2000.

[3] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), sep 2008.

[4] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 3–13. IEEE, 2000.

[5] Luca Trevisan. Cs359g lecture 17: The zig-zag product, 2011. Last accessed 5 December 2021.