CS 278 Final Project – Undirected st-Connectivity in Log-Space

William Lin, Jack Spilecki

1 Introduction

Does randomness save memory? Or are we able to derandomize space-bounded computation? While it is believed that $\mathsf{L} = \mathsf{RL}$, a proof has so far eluded the community. Here, L is the class of problems solved using log-space deterministically, and RL is the class of problems with randomized algorithms with one-sided bounded error using log-space.

A natural problem in RL is USTCONN, or *undirected st-connectivity*: given an undirected graph G, and two vertices s, t, decide if s and t are in the same connected component. We previously saw that the directed version, STCONN, is complete for NL. However, USTCONN is in RL due to a simple and natural algorithm [AKL+79]: if G is undirected and s and t are in the same connected component, then polynomial-length paths starting at s are provably likely with at least constant probability to reach t. The deterministic algorithm for USTCONN thus reduces the need of randomly picking polynomially many edges by deterministically constructing an expander, giving hope that all problems in RL may be derandomized. In fact, USTCONN is complete for the complexity class SL, the class of problems solved by symmetric, log-space Turing machines [LP82]. These are Turing machines with a limited form of non-determinism: the configuration graph is undirected. Intuitively, this class contains problems with some sort of symmetry, and complete problems include determining whether a given graph is bipartite, if a given edge is contained in a cycle, or if a graph has an even number of connected components [AG].

Previously it was known that $\mathsf{L} \subseteq \mathsf{SL} \subseteq \mathsf{RL} \subseteq \mathsf{NL} \subseteq \mathsf{L}^{3/2}$, where the final inclusion is due to Saks and Zhou [SZ99]. In 2005, Omer Reingold made significant progress in our understanding of L versus RL by showing that USTCONN is in fact in L [Rei08], and hence also, as a corollary, that $\mathsf{L} = \mathsf{SL}$.

In this report, we describe Reingold's breakthrough log-space algorithm for USTCONN. We then conclude by describing research extending Reingold's work in the hopes of showing $\mathsf{L} = \mathsf{RL}$, and possible directions related to this algorithm.

2 Background

We begin the technical part of the paper by briefly covering some preliminary, background material.

2.1 Graph Representations

In this paper, we consider undirected graphs $G = (V, E)$, and allow for self-loops and multiple edges between the same vertices. We utilize two main graph representations: the adjacency matrix and the rotation map.

Given a graph G on N vertices, the adjacency matrix A is the $N \times N$ matrix whose entry A_{ij} counts the number of edges from vertex i to vertex j. For undirected graphs, A is a symmetric matrix.

An undirected, D-regular graph may also be represented by a *rotation map.* For this, assume the edges at each vertex are labelled 1 through D.

Definition 1. *An undirected D-regular graph G is represented by a* rotation map $\mathrm{Rot}_G : [N] \times [D] \to [N] \times [D]$*, where $\mathrm{Rot}_G(v, i) = (w, j)$ if the ith edge from v leads to w, and is the jth edge of w.*

2.2 Expanders

Informally, expanders are graphs that are sparse, but well-connected. There are multiple formal definitions of expansion, and two are relevant for us.

First is the notion of *spectral expansion.* The adjacency matrix of an undirected graph A is symmetric, and can be diagonalized. If we normalize the adjacency matrix of a D-regular graph G, obtaining $\frac{1}{D}A$, then the eigenvalues may be listed $1 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N \geq -1$. The *spectral expansion* is $\lambda(G) = \max(|\lambda_2|, |\lambda_N|)$. The smaller this quantity, the more connected the graph.

Definition 2. *An (N, D, λ)-graph is an undirected graph G such that G has N vertices, is D-regular, and has $\lambda(G) \leq \lambda$.*

We also have a notion of *vertex expansion*, which is more intuitively tied to connectivity; here larger expansion means more connectivity:

Definition 3. *The* vertex expansion *of $h_{out}(G)$ of a graph G is $h_{out}(G) = \min_{S:|S|\leq \frac{n}{2}} (|\delta_{out}(S)|/|S|)$ where δ_{out} denotes the set of all neighbors of S that are* not *in S*

We note that for undirected, D-regular graphs, with $D = O(1)$, the well-known Cheeger inequalities relate the spectral expansion to the vertex expansion. Specifically, good spectral expanders (i.e. $\lambda(G) < 1$, with $\lambda(G) = O(1)$) will be good vertex expanders (i.e. $h_{\text{out}}(G) > 0$, also $O(1)$).

3 Reingold's Algorithm for USTCONN

In this section, we describe ideas and constructions that go into Reingold's algorithm for USTCONN, outline the algorithm itself, and explain why it can be implemented using only log-space. Where unlisted, all results are from [Rei08].

3.1 The Starting Observation: st-Connectivity in Good Expanders

We begin with an observation: if an undirected, D-regular graph G is a good expander, with $D = O(1)$, then it has logarithmic diameter.

Proposition 4. *Let $\lambda < 1$ be a constant, and let G be any (N, D, λ)-graph, where $D = O(1)$. For all $s, t \in V$, there exists a path of length $O(\log N)$ connecting s to t.*

Proof. For $k \in \mathbb{N}$, consider the set $S_k \subset V$ consisting of all vertices that can be connected to s with a walk of length at most k. Since G is a good spectral expander, it is also a good vertex expander, and hence there exists a constant $\epsilon > 0$ such that if $|S_k| \leq N/2$, then $|S_{k+1}| \geq (1+\epsilon)|S_k|$. Since $|S_0| = |\{s\}| = 1$, it follows that $|S_k| \geq \min(N/2 + 1, (1+\epsilon)^k)$, and hence for some $k = O(\log N)$, $|S_k| > N/2$. Applying the same logic to sets $T_k \subset V$, defined analogously to S_k but for t, shows that for some $k = O(\log N)$, we have $|S_k|, |T_k| > N/2$. Hence, there exists a $u \in S_k \cap T_k$. Then the path $s \to \cdots \to u \to \cdots \to t$ is of length at most $2k = O(\log N)$. $\square$

However, if a graph has logarithmic diameter, then we can solve st-connectivity simply by enumerating all paths of length $O(\log N)$ starting at s, and returning **TRUE** iff we encounter t. If we are using the rotation map to represent our graph, then this can be done by storing only the current vertex, which requires $O(\log N)$ space, and a stack of size $O(\log N)$ of edge labels so we can backtrack. Since each label requires $O(\log D)$ space, we will require $O(\log D \log N) = O(\log N)$ space.

So if our starting graph were a good expander with constant degree, then we can solve st-connectivity in log-space.

3.2 A High-Level Overview

The overall idea of Reingold's algorithm is to somehow reduce the problem for a general graph to the special case of a graph in which every connected component is a good expander. Provided we do so in a way so that the final graph has polynomially many vertices and constant degree, we can leverage the observation of the previous section. Before delving into specifics, we give the following high-level overview of the algorithm:

1. First, convert the graph G into a regular graph G_0 of constant degree with the property that each connected component is non-bipartite. This step is not difficult, but makes possible the analysis involved with the next step.
2. By a sequence of transformations $G_0 \to G_1 \to \cdots \to G_\ell$, convert G_0 to a graph G_ℓ with the property that each connected component of G_ℓ has good expansion. These transformations utilize a combination of graph products we develop in the following section.
3. Now that each connected component is a good expander, enumerate all logarithmic-length paths from a specific vertex s' in G_ℓ and see if any of them reach another specific vertex t'. The vertices s' and t' will be connected iff s and t are themselves connected, due to our constructions, and so this suffices to check if $\langle G, s, t\rangle \in$ USTCONN.

However, we do not actually construct these graphs, as this would require too much space – indeed, each graph G_i is bigger than the last. What we will actually show is that we can simulate walks in the final graph G_ℓ in log-space.

3.3 Graph Products

In this subsection, we develop graph products which play a central role in the second step of the algorithm.

3.3.1 Powering

A straightforward way to amplify the expansion of a graph is to take *powers* of that graph. Let G be a graph on vertices V. Then informally, the *the kth power of* G, G^k, is the graph on V which has an edge (u, v) for each distinct walk of length k in G starting at u and ending at v. An alternate description is that if G has adjacency matrix A, then G^k is the graph whose adjacency matrix is A^k. We have the following fact:

Proposition 5. *If G is an (N, D, λ)-graph, then G^k is an (N, D^k, λ^k)-graph.*

Proof. If G is D-regular, then there are D^k walks of length k starting at any vertex, so that G^k is D^k-regular. The eigenvalues of A^k are the kth powers of the eigenvalues of A, so that $\lambda(G^k) = \lambda(G)^k$. □

It will not suffice to simply take a large power of the graph G_0. The reason is that the final graph must have constant degree. It will turn out that we would have to take a power $k = O(\log N)$ so that $\lambda(G^k) = O(1)$, but in doing so we have raised the degree to $D^k = \mathsf{poly}(N)$, which we cannot tolerate if restricted to log-space.

3.3.2 The Replacement Product

Let G be a (N, D, λ_G)-graph, and let H be a (D, d, λ_H)-graph, where we emphasize that the degree of G is the same as the number of vertices of H. We will think of d as being much smaller than D, as this will be the case for our applications.

We can define the *replacement product* of such a pair, $G \circledr H$. Informally, we replace each vertex v of G with a copy of H, H_v. We can denote the vertices by (v, a), where $v \in V(G)$, and $a \in V(H)$. Then a vertex (v, a) is in the copy H_v, and so is connected by some edges "from H", i.e. to vertices (v, b) if $(a, b) \in E(H)$. It is also connected by a unique edge "from G": if $(u, v) \in E(G)$, so that $\mathrm{Rot}_G(u, a) = (v, b)$, then (u, a) is connected to (v, b) as well. Here we use the fact that the number of vertices of H is the same as the degree of G, as we have identified vertex labels in H with edge labels in G, both elements of $[D]$.

3.3.3 The Zig-Zag Product

The zig-zag product, introduced in [RVW04], builds on the replacement product. Let G and H be as in the previous section. Informally, the zig-zag product, $G \textcircled{z} H$, is the graph with the vertices of the replacement product (i.e. $V(G) \times V(H) = [N] \times [D]$) and with an edge $((v, a), (w, b))$ whenever there is a walk of length *three* in $G \textcircled{r} H$ of the specific form: $(v, a) \to (v, a') \to (w, b') \to (w, b)$. That is, the walks must first take one step in H_v, then take the unique out of H_v into some other copy H_w, and then take one step in H_w. A specific edge may then be represented by the pair of elements of $[d]$: the first and third edge labels (the second edge is unique, and does not need to be specified). The situation is depicted in Figure 1.

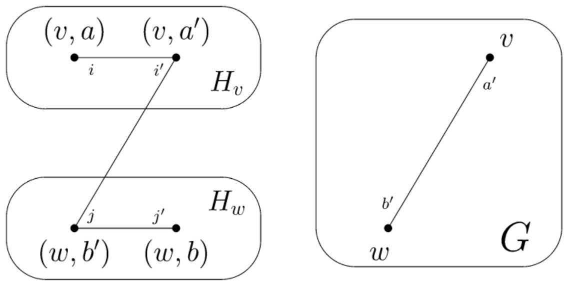

Figure 1: On the left, a walk of length three in the replacement product $G \textcircled{r} H$ corresponding to an edge in $G \textcircled{z} H$ between (v, a) and (w, b). The first and third steps in this walk are along edges within a copy of H. The second edge, between copies, connects (v, a') to (w, b'), since $\mathrm{Rot}_G(v, a') = (w, b')$, depicted on the right.

How do the parameters of G compare to those of $G \textcircled{z} H$? The number of vertices goes from N to ND, and the degree goes from D to d^2. For the spectral expansion, Reingold, Vadhan and Wigderson proved the following bound in with an intricate analysis of the Rayleigh quotient:

Theorem 6 ([RVW04]). *If G is an (N, D, λ_G)-graph and H is a (D, d, λ_H)-graph, then $G \textcircled{z} H$ is an $(ND, d^2, f(\lambda_G, \lambda_H))$ graph, where*

$$f(\lambda_G, \lambda_H) = \frac{1}{2}\left(1 - \lambda_H^2\right)\lambda_G + \frac{1}{2}\sqrt{\left(1 - \lambda_H^2\right)^2 \lambda_G^2 + 4\lambda_H^2}.$$

For our purposes, we can relax this result to give a lower bound on the spectral gap of $G \textcircled{z} H$:

Corollary 7. *If G is an (N, D, λ_G)-graph and H is a (D, d, λ_H)-graph, then*

$$1 - \lambda(G \textcircled{z} H) \geq \frac{1}{2}(1 - \lambda_H^2)(1 - \lambda_G).$$

Proof. Since $\lambda_G \leq 1$, we have

$$\frac{1}{2}\sqrt{\left(1 - \lambda_H^2\right)^2 \lambda_G^2 + 4\lambda_H^2} \leq \frac{1}{2}\sqrt{\left(1 - \lambda_H^2\right)^2 + 4\lambda_H^2} = \frac{1}{2}\left(1 + \lambda_H^2\right) = 1 - \frac{1}{2}\left(1 - \lambda_H^2\right).$$

Then

$$1 - \lambda(G \textcircled{z} H) \geq 1 - \left(\frac{1}{2}(1 - \lambda_H^2)\lambda_G + 1 - \frac{1}{2}\left(1 - \lambda_H^2\right)\right) = \frac{1}{2}(1 - \lambda_H^2)(1 - \lambda_G).$$

□

We will be thinking of H as a fixed graph with $D = O(1)$ and $\lambda = O(1)$. Therefore, when taking the zig-zag product with H, we will only be decreasing the spectral gap by at most a constant factor.

3.4 Transforming Graphs into Expanders

The complementary properties of graph powering and taking the zig-zag product with a fixed low-degree good expander H allow for the following possibility: that some combination of taking powers and zig-zag products will allow us to decrease λ while keeping the degree of the graph low, and not increasing the number of vertices by too much.

Indeed, this will be the case. For now, we will assume that we have a connected, non-bipartite graph G_0 on $[N]$ which is D_H^{16}-regular, and a fixed D_H-regular expander H on $[D_H^{16}]$, with $\lambda_H \leq 1/2$ and $D_H > 3$. We will show in the next section how to construct an appropriate such graph G_0 from G. Then each component of G_0 already has weak expansion, by the following lemma:

Lemma 8 ([AS00]). *For every d-regular, connected, non-bipartite graph G on* $[N]$*, we have*

$$\lambda(G) \leq 1 - 1/dN^2.$$

It is important to begin with weak expansion, so that we can increase this expansion by powering. Define graphs G_i, for $i = 1, 2, \ldots$, by

$$G_i = (G_{i-1} ⓩ H)^8.$$

This product is well-defined, since each G_i is D_H^{16}-regular. Indeed, if G_{i-1} is D_H^{16}-regular, then $G_{i-1} ⓩ H$ is defined, and is D_H^2-regular, and taking the eighth power then shows G_i is D_H^{16}-regular as well. Reingold shows that, as long as $\lambda(G_i) > 1/2$, this alternation of zig-zagging and powering is as good as squaring, as far as the spectral expansion is concerned, and if $\lambda(G_i) \leq 1/2$, then $\lambda(G_{i+1}) \leq 1/2$. Therefore, we can repeatedly alternate this process to obtain graphs with good expansion:

Lemma 9. *Suppose* $\lambda(H) \leq 1/2$*. If* $\lambda(G_{i-1}) \leq 1/2$*, then* $\lambda(G_i) \leq 1/2$*. Otherwise,* $\lambda(G_i) \leq \lambda(G_{i-1})^2$*.*

Proof. By Corollary 10, $\lambda(G_{i-1} ⓩ H) \leq 1 - \frac{3}{8}(1 - \lambda(G_{i-1})) < 1 - \frac{1}{3}(1 - \lambda(G_{i-1}))$. If $\lambda(G_{i-1}) \leq 1/2$, then $\lambda(G_{i-1} ⓩ H) \leq 5/6$, so that $\lambda(G_i) \leq (5/6)^8 < 1/2$. Otherwise, we may apply the inequality $(1 - \frac{1}{3}(1-x))^4 \leq x$ which holds for $x \geq 1/2$, to find $\lambda(G_i) = \lambda(G_i ⓩ H)^8 \leq \lambda(G_{i-1})^2$. □

Now, applying this alternation will yield a good expander after sufficiently many trials. Indeed, since each connected component will begin by satisfying, by Lemma 9, $\lambda(G_0) \leq 1 - \frac{1}{D_H^{16} N^2}$, applying Lemma 10 gives,

$$\lambda(G_i) \leq \max\left(\frac{1}{2}, \left(1 - \frac{1}{D_H^{16} N^2}\right)^{2^i}\right).$$

Therefore, for $\ell = O(\log D_H^{16} N^2) = O(\log N)$, we have $\lambda(G_\ell) \leq 1/2$, and so:

Proposition 10. *Let* G_i *and* H *be defined as above. If* $\lambda(H) \leq 1/2$*, and* G_0 *is connected and non-bipartite, then* $\lambda(G_\ell) \leq 1/2$*, for* $\ell = O(\log N)$*.*

What if the graph G_0 is not connected? In this case, the transformations defined above act on each connected component separately, and each connected component of G_ℓ satisfies $\lambda(G_\ell) \leq 1/2$. Note also that the image of any connected component of G_0 under this transformation is itself a connected component (for example, the image has good expansion, and must therefore be connected).

3.5 Constructing G_0

In the previous part, we have assumed a graph G_0 that is D_H^{16}-regular and non-bipartite. But how do we construct G_0 from our given graph G? Here we describe a simple possibility.

Let G be a graph on N vertices. Then, to form G_0, replace every vertex of G with a cycle of N vertices. The vertices of the ith cycle are labelled (i, j), for $j \in [N]$. In addition to the cycle edges, we also have edges "from G", so that (i,j) is an edge of G iff $((i,j),(j,i))$ is an edge of G_0. Now every vertex (i,j) is adjacent to two "cycle edges", and possibly one "inter-cycle edge", and so has degree either 2 or 3. We finish the construction of G_0 by placing $D_H^{16}-2$ or $D_H^{16}-3$ self-edges on each vertex, so that G_0 is D_H^{16}-regular.

Since D_H is some constant larger than 3, every vertex has at least one self-loop, and the graph is non-bipartite as well. Note also that under this transformation, connected components get mapped to connected components.

We may represent G_0 via a rotation map. Specifically, we may label the edges as:

$$\mathrm{Rot}_{G_0}((i,j),k) = \begin{cases} ((i,j+1 \mod N),2) & k=1, \\ ((i,j-1 \mod N),1) & k=2, \\ ((j,i),3) & k=3 \text{ and } (i,j)\in E(G) \\ ((i,j),3) & k=3 \text{ and } (i,j)\notin E(G) \\ ((i,j),k) & k>3. \end{cases}$$

Thus, the cycle edges have labels 1 at one end, and 2 at the other, the inter-cycle edges are labelled 3 at each end, and the self-loops are labelled k at each end, for $k>3$ (and possibly $k=3$, if there is no inter-cycle edge at that vertex).

3.6 Reingold's Algorithm: Putting it All Together

We have seen we can construct G_0, a non-bipartite, D_H^{16}-regular graph. From here, we can construct $G_1,\ldots,G_\ell$ using powering and the zig-zag product with a fixed expander H. At the end of this process, if we take $\ell = O(\log N)$, we obtain a graph G_ℓ on $N^2(D_H^{16})^\ell = \mathsf{poly}(N)$ vertices, with degree $D_H^{16}=O(1)$, and with $\lambda(G_\ell)\le 1/2$. The transformations operate in such a way so that s and t are connected iff any images of s and t in G_ℓ are connected. For simplicity, we may choose $s' := (s,1^{\ell+1})$ and $t' := (t,1^{\ell+1})$. In order to see if $\langle G,s,t\rangle \in \mathrm{USTCONN}$, we may then simply enumerate all paths of length $O(\log \mathsf{poly} N) = O(\log N)$ in G_ℓ starting at s' and return **TRUE** iff we ever encounter t'.

It remains to show then that we can actually simulate a walk in G_ℓ in log-space, given access to G. To do so, it suffices to show we can compute the rotation map in G_ℓ in log-space. We will do so recursively.

First, we can compute the rotation map $\mathrm{Rot}_{G_0}((i,j),k)$ in log-space, given access to G, since we can simply check the value of k, and then: if $k=1$ or $k=2$ update j appropriately, and swap to 2 or 1 respectively; if $k=3$, check if there is an edge (i,j) in G, and if so, return $((j,i),k)$; and otherwise, return the input.

For $i>0$, we have to show we can recursively compute Rot_{G_i} from $\mathrm{Rot}_{G_{i-1}}$. Recall that vertices of G_i can be represented as tuples $(v,a_0,\ldots,a_{i-1})$, where $v\in G_0$ and each $a_i\in[D_H^{16}]$ indicates a vertex of H (recall, in the zig-zag product $G_{i-1} \circledz H$, we replace every vertex of G_{i-1} with a vertex of H). Edge labels are given by a tuple $a_i = (k_{i,1},\ldots,k_{i,16})$, where $k_{i,j}\in[D_H]$. This is because a single edge in $G_{i-1} \circledz H$ is labelled by two edges in H (the second edge, from G_{i-1} does not need a label), and since in $G_i = (G_{i-1} \circledz H)^8$, edges correspond to eight edges in $G_{i-1} \circledz H$. To compute $\mathrm{Rot}_{G_i}((v,a_0,\ldots,a_{i-1}),a_i)$, we have the following algorithm:

```
For $j = 1$ to 16:
1. Set $a_{i-1}, k_{i,j} \leftarrow \mathrm{Rot}_H(a_{i-1}, k_{i,j})$
2. If $j$ is odd, recursively set $(v,a_0,\ldots,a_{i-1}) \leftarrow \mathrm{Rot}_{G_{i-1}}((v,a_0,\ldots,a_{i-2}),a_{i-1})$
3. If $j = 16$, reverse $a_i$, i.e. $(k_{i,1},\ldots,k_{i,16}) \leftarrow (k_{i,16},\ldots,k_{i,1})$
```

We briefly explain why this works. The first step is responsible for making steps inside a "copy of H". The copy of H is labelled by $(v,a_0,\ldots,a_{i-2})$, and we are at the vertex a_{i-1} in this copy. We take the edge labelled by $k_{i,1}$, and to get the new vertex and return edge label, we compute the rotation map in H. The second step then makes a step "between copies of H". The unique edge out of the current copy of H corresponds to the a_{i-1}th edge out of the vertex $(v,a_0,\ldots,a_{i-2})$ in G_{i-1}. Computing the rotation map in G_{i-1} gives us both the new copy of H and the vertex in that copy, and therefore gives us the vertex in G_i we end up at. We then repeat these steps appropriately. At the end, we have to reverse a_i since our return edge labels are out of order.

We need to store $(v, a_1, \ldots, a_i)$, which is operated upon at all levels of the recursion. Since $v \in [N^2]$, $[a_j] \in [D_H^{16}]$, and $i \leq \ell = O(\log N)$, we require $O(\log N)$ space to store this data. We also need to store a stack of the integers j, and since this stack may have depth ℓ, we need $O(\log N)$ memory for this too. Thus, the rotation map may recursively be computed in $O(\log N)$ space.

Therefore, since we can decide USTCONN by enumerating walks of length $O(\log N)$ in log-space given the ability to calculate the rotation map, and since we can compute the rotation map in log-space as well, Reingold's algorithm runs in log-space. Finally, we conclude:

Theorem 11. USTCONN $\in \mathsf{L}$

Corollary 12. $\mathsf{L} = \mathsf{SL}$

4 Further Research and Future Directions

In this final section, we describe some further research related to USTCONN and possible future directions. The major avenue of research following Reingold's paper addresses the question of whether L equals RL. Much progress has been made in this direction, though we still do not know the answer. We also describe two topics in the direction of simplifying Reingold's algorithm: first, an alternate transformation to that used in Reingold's algorithm, which raises the possibility of further improvements; and second, an interesting and approachable direction: whether we might be able to solve USTCONN more simply using a catalytic tape as an additional resource.

4.1 Progress on $\mathsf{L} \overset{?}{=} \mathsf{RL}$

A significant follow-up paper by Reingold, Trevisan and Vadhan [RTV06] extends the techniques in Reingold's paper to a restricted class of *directed* graphs. Specifically, it solves STCONN in deterministic log-space provided the graph is *D-regular* (each vertex has D incoming, and D outgoing edges) and *consistently labelled* (at each vertex, each incoming edge has a unique labelling). While STCONN is NL-complete in general, they also show STCONN with the promise that random walks starting at s have $\mathsf{poly}(N)$ mixing time is RL-complete, so this constitutes progress towards L vs. RL. To this end, the paper also showed that techniques used in Reingold's proof could be used to create pseudorandom walks for regular, consistently labelled directed graphs. These walks can be generated with a seed requiring log-space, and starting at s should end uniformly at vertices reachable by s, which would resolve the problem when enumerating the possible seeds. In this, the RTV proof showed that if the given pseudorandom generator could work for *all* regular directed graphs, it would imply $\mathsf{L} = \mathsf{RL}$.

Another follow-up by Chung, Reingold and Vadhan [CRV11] considers again directed graphs with polynomial mixing time, but also assumes we are given access to a stationary distribution. They show that, given this restriction and additional resource, *st*-connectivity can be solved in log-space. Since the stationary distributions of the consistently labelled, regular, directed graphs of the previous work are able to be calculated easily, this follow-up shows that the implicit knowledge of the stationary distribution is sufficient to solve the problem, and identifies this knowledge as the main obstacle in the general case.

4.1.1 Recent Progress with Pseudorandom Generators

Recent progress in work to derandomize RL focuses on creating pseudorandom generators that look to fool polynomial-width bounded read once branching programs. A pseudorandom generator G is a function from $\{0,1\}^l \rightarrow \{0,1\}^n$ taking in a seed that outputs "approximately random bits". Specifically, G *ϵ-fools* a model of computation F if for all $f \in F$, we have $|\mathbf{Pr}_{x\in\{0,1\}^l}[C(g(x)) = 1] - \mathbf{Pr}_{x\in\{0,1\}^n}[C(x)]| \leq \epsilon$. We can think of this as trying to model random bits with a small seed length, and if we can do this we can effectively bring randomness down in space complexity. In the context of $\mathsf{L} \overset{?}{=} \mathsf{RL}$, works seek to improve upon Nisan's PRG [Nis92], which gives a pseudorandom generator such that for any size S, it 2^{-S} fools $\mathsf{SPACE}(S)$ with a seed length of $O(S \log n)$. In this case, the Nisan PRG can fool log space machines with a seed length of $\log^2 n$. Improving upon this result has been difficult, and most work focuses on more restricted cases.

Most recent work is organized around finding pseudorandom generators to fool *read-once branching programs* (ROBPs). Read-once branching programs are a non-uniform model of computation consisting of nodes arranged in layers, and on input moves to the next layer, eventually reaching an accept or reject state. The model is a DAG with a single input vertex, and can be seen as a decision tree. Formally, it takes in binary strings x of input size n as input, and transitions based on the values of x_i, in either arbitrary order or fixed order. An ROBP has width w if every layer has at most w states, and has $n+1$ layers. A result by Barrington from 1989 ([Bar89]) showed that constant-width branching programs are equivalent to circuits in NC^1. Turing machines of space S and R random bits can be seen to be modeled by branching programs of width 2^S and length R if viewed as a function of randomness and transitioning between configurations, so log space Turing machines have polynomial width ROBPs. Thus PRGs fooling ROBPs can be seen as derandomizing RL. The techniques involved in recent work vary significantly, with work on analyzing fooling different types of ROBPs with different restrictions, for example analyzing the Impagliazzao-Nisan-Widgerson ([INW94]) PRG for unbounded width ROBPs ([HPV21]), PRGs for monotone ROBPs ([DMR+21]), or hitting set generators for ROBPs([HZ20]).

The main open problem related to Reingold's work is continuing the highly-technical and deep line of research in this direction.

4.2 Possibilities for Simpler Algorithms

We lastly consider two directions for simpler algorithms for USTCONN in the low-space regime. We first describe a graph operation which performs the same role as the alternation between squaring and zig-zagging, but is much simpler. We then describe an interesting model of computation, whose additional power may lead to simplifications as well.

4.2.1 Derandomized Squaring

In [RV05], Rozenman and Vadhan give a simpler transformation of G_0 to a graph G_ℓ with good expansion properties, involving a graph operation called *derandomized squaring*.

We can view standard graph squaring as an operation which takes the vertex set of a graph G, and for all vertices v, places a clique on the neighbors of v. The upside of squaring is that $\lambda(G^2) = \lambda(G)^2$, but the downside is that it also squares the degree, and this is the reason we could not use it alone in Reingold's algorithm. In *derandomized squaring*, instead of placing a clique on the neighbors of a D-regular graph G, we place H, a $(D, d, \lambda(H))$-graph, where λ_H is small. Rozenman and Vadhan show that this operation is almost as good as squaring, specifically, that $\lambda(G') \leq \lambda(G)^2 + \lambda(H)$. By arguments similar to Reingold's, we can then use this repeatedly to form the basis of a log-space algorithm for USTCONN.

The main advantages is that the vertex set remains the same throughout the sequence of transformations, and the reduced complexity: we only have to apply derandomized squaring once for every zig-zag and eighth power Reingold applies. Could Reingold's algorithm be even further improved by considering other kinds of graph transformations?

4.2.2 Catalytic Computing

Catalytic computing provides an interesting avenue for exploring algorithms based in space-bounded computation [Mer23]. Here we briefly explain the idea of catalytic computing.

Definition 13. *A* catalytic Turing machine *with space $s(n)$ and catalytic space $c(n)$ is a Turing machine M with a work tape of length $s(n)$ and an additional catalytic tape of length $c(n)$. For any $\tau \in \{0,1\}^{c(n)}$, with the catalytic tape initialized to τ, the catalytic tape must contain τ when halting.*

Informally, our Turing machine is given an extra tape, initialized arbitrarily. It seems we have access to additional space for free, but there's a catch: we must return this tape to its initial configuration! We say $\mathsf{CSPACE}(s, c)$ is the set of problems solvable by a catalytic Turing machine with space s and catalytic space c. We see that $\mathsf{SPACE}(s) \subseteq \mathsf{CSPACE}(s, c) \subseteq \mathsf{SPACE}(s + c)$, and define CL as $\mathsf{CSPACE}(\log n, n^{O(1)})$, so that $\mathsf{L} \subseteq \mathsf{CL} \subseteq \mathsf{PSPACE}$. The additional catalytic space does not initially seem beneficial, as any contents of catalytic space used must be "remembered" in order to rewrite it at the end.

Here is one interesting way in which catalytic space can prove valuable, which forms the basis for a proof that $\mathsf{BPL} \subseteq \mathsf{CL}$ (here, BPL is the two-sided bounded error probabilistic version of L). The broad idea is to check, in log-space, whether our catalytic tape is able to be compressed or not: if so, we use the freed-up space to iterate through all possible "random seeds", and finally decompress; if not, the bits in the catalytic tape are "sufficiently random" to use as input to our BPL machine. This dual approach is suggestive of the intriguing possibilities available to us with a catalytic tape.

We believe that catalytic computing offers simple, natural, and accessible follow-up questions as well. Of course, since $\mathsf{RL} \subseteq \mathsf{BPL} \subseteq \mathsf{CL}$, we know that there exist space-bounded catalytic algorithms for USTCONN. One natural question we might consider is whether there is a *simple* algorithm using this extra additional resource in an interesting way. In particular, while Reingold's algorithm for USTCONN involves a complicated construction and significant analysis, it might be possible that clever usage of additional resources can lead to significant simplification.

References

[AG] Carme Àlvarez and Raymond Greenlaw. A compendium of problems complete for symmetric logarithmic space.

[AKL+79] Romas Aleliunas, Richard Karp, Richard Lipton, László Lovász, and Charles Rackoff. Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems. pages 218–223, 11 1979.

[AS00] Noga Alon and Benny Sudakov. Bipartite Subgraphs and the Smallest Eigenvalue. *Combinatorics Probability and Computing*, 9(1):1–12, 2000.

[Bar89] David A. Mix Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[CRV11] Kai-Min Chung, Omer Reingold, and Salil Vadhan. S-t connectivity on digraphs with a known stationary distribution. 7(3), 2011.

[DMR+21] Dean Doron, Raghu Meka, Omer Reingold, Avishay Tal, and Salil P. Vadhan. Pseudorandom Generators for Read-Once Monotone Branching Programs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 58:1–58:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[HPV21] William M. Hoza, Edward Pyne, and Salil P. Vadhan. Pseudorandom Generators for Unbounded-Width Permutation Branching Programs. In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[HZ20] William M. Hoza and David Zuckerman. Simple Optimal Hitting Sets for Small-Success RL. *SIAM J. Comput.*, 49(4):811–820, 2020.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 356–364. ACM, 1994.

[LP82] Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2):161–187, 1982.

[Mer23] Ian Mertz. Reusing Space: Techniques and Open Problems. *Bull. EATCS*, 141, 2023.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Comb.*, 12(4):449–461, 1992.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

[RTV06] Omer Reingold, Luca Trevisan, and Salil Vadhan. Pseudorandom walks on regular digraphs and the rl vs. l problem. STOC '06, New York, NY, USA, 2006. Association for Computing Machinery.

[RV05] Eyal Rozenman and Salil Vadhan. Derandomized squaring of graphs. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 436–447, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[RVW04] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree, 2004.

[SZ99] Michael Saks and Shiyu Zhou. $\mathsf{BP}_h\mathsf{SPACE}(s) \subseteq \mathsf{DSPACE}(s^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.