Undirected Connectivity in Log-Space*

Omer Reingold†

May 3, 2008

Abstract

We present a *deterministic*, log-space algorithm that solves st-connectivity in undirected graphs. The previous bound on the space complexity of undirected st-connectivity was $\log^{4/3}(\cdot)$ obtained by Armoni, Ta-Shma, Wigderson and Zhou [ATSWZ00]. As undirected st-connectivity is complete for the class of problems solvable by symmetric, non-deterministic, log-space computations (the class SL), this algorithm implies that $\mathrm{SL} = \mathrm{L}$ (where L is the class of problems solvable by deterministic log-space computations). Independent of our work (and using different techniques), Trifonov [Tri05] has presented an $O(\log n \log\log n)$-space, deterministic algorithm for undirected st-connectivity.

Our algorithm also implies a way to construct in log-space a *fixed* sequence of directions that guides a deterministic walk through all of the vertices of any connected graph. Specifically, we give log-space constructible universal-traversal sequences for graphs with restricted labeling and log-space constructible universal-exploration sequences for general graphs.

1 Introduction

We resolve the space complexity of undirected st-connectivity (denoted USTCON), up to a constant factor, by presenting a log-space (polynomial-time) algorithm for solving it. Given as input an undirected graph G and two vertices s and t, the USTCON problem is to decide whether or not the two vertices are connected by a path in G (our algorithm will also solve the corresponding search problem, of finding a path from s to t if such a path exists). This fundamental combinatorial problem has received a lot of attention in the last few decades and was studied in a large variety of computational models. It is a basic building block for more complex graph algorithms and is complete for the class SL of problems solvable by symmetric, non-deterministic, log-space computations [LP82] (see [AG96] for a recent study of SL and quite a few of its complete problems).

The time complexity of USTCON is well understood as basic search algorithms, particularly breadth-first search (BFS) and depth-first search (DFS), are capable of solving USTCON in linear time. In fact, these algorithms apply to the more complex problem of st-connectivity in directed graphs (denoted STCON), which is complete for NL (non-deterministic log-space computations). Unfortunately, the space required to run these algorithms is linear as well. A much more space efficient algorithm is Savitch's [Sav70], which solves STCON in space $\log^2(\cdot)$ (and super-polynomial time).

Major progress in understanding the space complexity of USTCON was made by Aleliunas, Karp, Lipton, Lovász, and Rackoff [AKL+79], who gave a *randomized* log-space algorithm for the problem. Specifically, they showed that a random walk (a path that selects a uniform edge at each step) starting from

*Preliminary version appeared in STOC 2005 [Rei05]

†Department of Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel omer.reingold@weizmann.ac.il Research supported by US-Israel Binational Science Foundation Grants 2002246 and 2006060.

an arbitrary vertex of any connected undirected graph will visit all the vertices of the graph in polynomial number of steps. Therefore, the algorithm can perform a random walk starting from s and verify that it reaches t within the specified polynomial number of steps. Essentially all that the algorithm needs to remember is the name of the current vertex and a counter for the number of steps already taken. With this result we get the following view of space complexity classes: $\mathrm{L} \subseteq \mathrm{SL} \subseteq \mathrm{RL} \subseteq \mathrm{NL} \subseteq \mathrm{L}^2$ (where RL is the class of problems that can be decided by randomized log-space algorithms with one-sided error and L^c is the class of problems that can be decided deterministically in space $\log^c(\cdot)$).

The existence of a randomized log-space algorithm for USTCON puts this problem in the context of derandomization. Can this randomized algorithm be derandomized without substantial increase in space? Furthermore, the study of the space complexity of USTCON has gained additional motivation as an important test case for understanding the tradeoff between two central resources of computations, namely between memory (space) and randomness. Particularly, a natural goal on the way to proving $\mathrm{RL} = \mathrm{L}$ is to prove that $\mathrm{USTCON} \in \mathrm{L}$, as USTCON is undoubtedly one of the most interesting problems in RL.

Following [AKL+79], most of the progress on the space complexity of USTCON indeed relied on the tools of derandomization. In particular, this line of work greatly benefited from the development of pseudorandom generators that fool space-bounded algorithms [AKS87, BNS89, Nis92a, INW94] and it progressed concurrently with the study of the L vs. RL problem. Another very influential notion, introduced by Stephen Cook in the late 70's, is that of a universal-traversal sequence. Loosely, this is a fixed sequence of directions that guides a *deterministic* walk through all of the vertices of any connected graph of the appropriate size (see further discussion below).

While Nisan's space-bounded generator [Nis92a], did not directly imply a more space efficient USTCON algorithm it did imply quasi-polynomially-long, universal-traversal sequences, constructible in space $\log^2(\cdot)$. These were extremely instrumental in the work of Nisan, Szemeredi and Wigderson [NSW89] who showed that $\mathrm{USTCON} \in \mathrm{L}^{3/2}$ – The first improvement over Savitch's algorithm in terms of space (limited of course to the case of undirected graphs). Using different methods, but still heavily relying on [Nis92a], Saks and Zhou [SZ99] showed that *every* RL *problem* is also in $\mathrm{L}^{3/2}$ (their result in fact generalizes to randomized algorithms with two-sided error). Relying on the techniques of both [NSW89] and [SZ99], Armoni, et. al. [ATSWZ00] showed that $\mathrm{USTCON} \in \mathrm{L}^{4/3}$. Their USTCON algorithm was the most space-efficient one previous to this work. We note that the most space-efficient *polynomial-time* algorithm for USTCON previously known was Nisan's [Nis92b], which still required space $\log^2(\cdot)$. Independent of our work (and using different techniques), Trifonov [Tri05] has presented an $O(\log n \log\log n)$-space, deterministic algorithm for USTCON.

Our approach

In retrospect, the essence of our algorithm is very natural: If you want to solve a connectivity problem on your input graph, first *improve its connectivity*. In other words, transform your input graph (or rather, each one of its connected components), into an expander.[1] We will also insist on the final graph being constant degree. Once the connected component of s is a constant-degree expander, then it is trivial to decide if s and t are connected: Since expander graphs have logarithmic diameter, it is enough to enumerate all logarithmically long paths starting with s and to see if one of these paths visits t. Since the degree is constant, the number of such paths is polynomial and they can easily be enumerated in log space.

[1]Loosely, expanders are graphs with very strong connectivity properties. There are several possible ways to define expanders, and in the following informal description we will (somewhat cheatingly) elude to a definition based on vertex expansion - namely that every set of vertices have "many" neighbors. For the knowledgable reader, we point out that the particular measure of expansion that seems the most convenient to work with is the second eigenvalue (in absolute value) of the adjacency matrix of the graph (we will only need to work with regular graphs). It may however be that other, more combinatorial, measures, such as edge expansion, will also do (see [RTV06] for a more detailed discussion).

How can we turn an arbitrary graph into an expander? First, we note that every connected, non-bipartite, graph can be thought of as an expander with very small (but non-negligible) expansion. Consider for example an arbitrary connected graph with self-loops added to each one of its vertices. The number of neighbors of every strict subset of the vertices is larger than its size by at least one. In this respect, the graph can be thought of as expanding by a factor $1 + 1/N$ (where N is the total number of vertices in the graph). Now, a very natural operation that improves the expansion of the graph is powering. The k^{th} power of G contains an edge between two vertices v and w for every path of length k between v and w in G. Formally, it can be shown that by taking some polynomial power of any connected non-bipartite graph (equivalently, by repeatedly squaring the graph logarithmic number of times), it will indeed turn into an expander.

The down side of powering is of course that it increases the degree of the graph. Taking a polynomial or any non-constant power is prohibited if we want to maintain constant degree. Fortunately, there exist operations that can counter this problem. Consider for example, the replacement product of a D-regular graph G with a d-regular graph H on D vertices (with $d \ll D$). This can be loosely defined as follows: Each vertex v of G is replaced with a "copy" H_v of H. Each of the D vertices of H_v is connected to its neighbors in H_v but also to one vertex in H_w, where (v, w) is one of the D edges going out of v in G. The degree in the product graph is $d + 1$ (which is smaller than D). Therefore, this operation can transform a graph G into a new graph (the product of G and H) of smaller degree. It turns out that if H is a "good enough" expander, the expansion of the resulting graph is "not worse by much" than the expansion of G. Formal statements to this effect were proven by Reingold, Vadhan and Wigderson [RVW02] for both the replacement product and the zig-zag product, introduced there. Independently, Martin and Randall [MR00], building on previous work of Madras and Randall [MR96], proved a decomposition theorem for Markov chains that also implies that the replacement product preserves expansion.

Given the discussion above, we are ready to informally describe our USTCON algorithm. First, turn the input graph into a constant-degree, regular graph with each connected component being non-bipartite (this step is very easy). Then, the main transformation turns each connected component of the graph, in logarithmic number of phases, into an expander. Each phase starts by raising the current graph to some constant power and then reducing the degree back via a replacement or a zig-zag product with a constant-size expander. We argue that each phase enhances the expansion at least as well as squaring the graph would, and *without the disadvantage of increasing the degree.* (An undesirable side effect of each phase is increasing the size of the graph. Nevertheless, as the increase will only be by a constant factor, this is tolerable.) Finally, all that is left is to solve USTCON on the resulting graph (which is easy as the diameter of each connected component is only logarithmic).

To conclude that USTCON $\in$ L, we need to argue that all of the above can be done in logarithmic space, which easily reduces to showing that the main transformation can be carried out in logarithmic space. For that, consider the graph G_i obtained after i phases of the transformation. We note that a step on G_i (i.e., evaluating the j^{th} neighbor of some vertex v in G_i) is composed of a constant number of operations that are either a step on the graph G_{i-1} from the previous phase or an operation that only requires a constant amount of memory. As the memory for each of these operations can be freed after it is performed, the memory for carrying out a step on G_i is only larger by an additive constant than the memory for carrying out a step on G_{i-1}. This implies that the entire transformation is indeed log space.

The RVW Combinatorial Construction of Expanders As discussed, we borrow our main technical tool (a bound on the expansion of the zig-zag or replacement product), from [RVW02]. More interestingly, our main transformation repeats exactly the same sequence of operations as in their combinatorial construction of expander graphs. Namely, both transformations iterate graph powering and a zig-zag product with a constant size expander. Somewhat surprisingly, the goals of the transformations are quite different: In [RVW02], they start with a constant size expander and in this sequence of operations make it into an arbitrarily large expander. Here we transform *any* connected graph (which is already large but is not an expander) into an expander. On a technical level, this means that the zig-zag product needs to be applied when the larger graph has extremely weak expansion properties. Still we require that the product essentially preserves this (weak but valuable) expansion. In contrast, in [RVW02] the zig-zag product is applied to two expanders. Very fortunately, the zig-zag product (as well as the replacement product) work quite well in this unusual setting of parameters.

Viewing the aforementioned transformations a bit more abstractly, we observe that in both cases the desired parameter (the size of the graph in [RVW02], and its expansion here) are improved in a slow and iterative manner while maintaining a careful balance between competing parameters (specifically, between expansion and degree in both these transformations). A similar structure is shared by Dinur's beautiful recent proof of the PCP Theorem. See [Gol05] for an insightful perspective of this approach, as exemplified by [Din07, JSV04, RVW02], and our own work.

Universal traversal sequences While universal-traversal sequences were introduced as a way for proving $\mathrm{USTCON} \in \mathrm{L}$, these are interesting combinatorial objects in their own right. A universal-traversal sequence for D-regular graphs on N-vertices, is a sequence of edge labels in $\{1, \ldots, D\}$ such that for every such graph, for every labeling of its edges, and for every start vertex, the *deterministic* walk defined by these labels (where in the i^{th} step we take the edge labeled by the i^{th} element of the sequence), visits all of the vertices of the graph. Aleliunas et. al. [AKL$^+$79] showed that a polynomial-length universal-traversal sequence exists, and in fact almost every sequence of the appropriate length will do. We are interested in obtaining a polynomially-long, universal-traversal sequence that is *constructible in logartihmic space* (even less explicit sequences may still be very interesting). This is again a derandomization problem. Namely, can we derandomize the probabilistic construction of universal-traversal sequences?

Explicit constructions of polynomially-long universal-traversal sequences are only known for extremely limited classes of graphs. Even for expander graphs, such sequences are only known when the edges are "consistently labelled" [HW93] (this means that the labels of all edges that lead to any particular vertex are distinct). It is therefore not very surprising that our algorithm on its own does not imply full fledged universal-traversal sequences. Still, our algorithm can be shown to imply a very local, and quite oblivious, deterministic procedure for exploring a graph. We can think of our algorithm as maintaining a single pebble, that is placed on the *edges* of the graph. The pebble is moved either from one side of the edge to another, or between different edges that are adjacent to the same vertex (say to the next or to the previous edge). As with universal-traversal sequences, the fixed sequence of instructions is good for every graph, for every labeling of its edges, and for any starting point on the graph. The only difference from universal-traversal sequences is that the pebble here is placed on the edges rather than on the vertices of the graph. In particular, we get polynomially-long, universal-*exploration* sequences for all undirected graphs. In universal-exploration sequences, introduced by Koucky [Kou01], the elements of the sequence are not interpreted as absolute edge-labels but rather as offsets from the previous edge that was traversed. In terms of traversal sequences, our algorithm implies a polynomially-long, universal-traversal sequence that is *constructible in logartihmic space* under some restrictions on the labeling. These restrictions were relaxed in a subsequent work [RTV06] to be identical to those of [HW93] (for universal-traversal sequences on expander graphs). For more details see Section 5.

More on previous work

Graph connectivity problems and space-bounded derandomization are the focus of a vast and diverse body of research. The scope of this paper only allows for an extremely partial discussion of this area. Some very beautiful and influential research (as many of the papers already mentioned above) is only briefly touched upon, other areas will not be discussed at all (examples include, time-space tradeoffs for deterministic and randomized connectivity algorithms, restricted constructions of universal traversal sequences, and analysis of connectivity in many other computational models). Insightful, though somewhat outdated, surveys on these topics were given by Wigderson [Wig92] and by Saks [Sak96]. Useful discussion and pointers were also given by Koucky [Kou03]. We continue here by mentioning a few of the most related previous results (most of which are subsumed by the results of this paper). A more technical comparison with some previous work appears in Section 6.

Following Aleliunas et. al. [AKL$^+$79], Borodin et. al. [BCD$^+$89] gave a *zero-error*, randomized, log-space algorithm for USTCON. An upper bound of different nature on SL was given by Karchmer and Wigderson [KW93], who showed $\mathrm{SL} \subseteq \oplus\mathrm{L}$.

Nisan and Ta-Shma [NTS95] showed that SL is closed under complement, thus collapsing the "symmetric log-space hierarchies" of both Reif [Rei84] and Ben Asher et. al. [YBAS95], and putting some very interesting problems into SL. To give just one example, the planarity of bounded-degree undirected graphs was placed in SL as a corollary (we refer again to [AG96] for a list of SL-complete problems).

A research direction initiated by Ajtai et. al. [AKS87], and continued with Nisan and Zuckerman [NZ96] is to fully derandomize (i.e., to put in L) $\log n$-space computations that use fewer than n random bits ($\mathrm{poly}\log n$ bits in the case of [NZ96]). Raz and Reingold [RR99] showed how to derandomize $2^{\sqrt{\log n}}$ bits for subclasses of RL. One of their main applications can be viewed as derandomizing $2^{\sqrt{\log n}}$ bits for SL. It is interesting to note (and personally gratifying to the author) that the techniques of [RR99] played a major role in the definition of the zig-zag product and with this work found their way back to the study of space-bounded derandomization.

Goldreich and Wigderson [GW02] gave an algorithm that on all but a tiny fraction of the graphs, evaluates USTCON correctly (and on the rest of the graphs outputs an error message).

Based on rather relaxed *computational hardness assumptions*, Klivans and van Melkebeek [KvM02] proved both that $\mathrm{RL} = \mathrm{L}$ and that efficiently constructible, polynomial length, universal traversal sequences exist.

2 Preliminaries

This section discusses various aspects of graphs: their representation, eigenvalue expansion, graph powering, and two graph products (the replacement product and the zig-zag product). The definitions and notation used here are borrowed directly from [RVW02].

2.1 Graphs representations

There are several standard representations of graphs. Fortunately, there exist log-space transformations between natural representations. Thus, the space complexity of USTCON is to a large extent independent of the representation of the input graph.

When discussing the eigenvalue expansion of a graph, we will consider its adjacency matrix. That is, the matrix whose (nonnegative, integral) entry (u, v) equals to the number of edges that go from vertex u to vertex v. Note that this representation allows graphs with self loops and parallel edges (and indeed such graphs may be generated by our algorithm). A graph is **undirected** iff its adjacency matrix is symmetric (implying that for every edge from u to v there is an edge from v to u). It is D**-regular** if the sum of entries in each row (and column) is D (so exactly D edges are incident to every vertex).

Let G be a D-regular undirected graph on N vertices. When considering a walk on G, we would like to assume that the edges leaving each vertex of G are labeled from 1 to D in some arbitrary, but fixed, way. We can then talk about the i'th edge incident to a vertex v, and similarly about the i'th neighbor of v. A central insight of [RVW02] is that when taking a step on a graph from vertex v to vertex w, it may be useful to keep track of the edge traversed to get to w (rather than just remembering that we are now at w). This gave rise to a new representation of graphs through the following *permutation* on pairs of vertex name and edge label:

Definition 2.1 *For a D-regular undirected graph G, the* **rotation map** $\mathrm{Rot}_G : [N] \times [D] \rightarrow [N] \times [D]$ *is defined as follows:* $\mathrm{Rot}_G(v, i) = (w, j)$ *if the i'th edge incident to v leads to w, and this edge is the j'th edge incident to w. (Recall that for every integer k we denote by $[k]$ the set $\{1, 2, \ldots, k\}$.)*

Rotation maps will indeed be the representation of choice for this work. Specifically, the first step of our algorithm will be to transform the input graph into a regular one specified by its rotation map (in particular, this step will give labels to the edges of the graph).

2.2 Eigenvalue expansion and st-connectivity for expanders

Expanders are sparse graphs which are nevertheless highly connected. The strong connectivity properties of expanders make them very desirable in our context. Specifically, since the diameter of expander graphs is only logarithmically long, there is a trivial log-space algorithm for finding paths between vertices in constant-degree expanders. The particular formalization of expanders used in this paper is the (algebraic) characterization based on the spectral gap of their adjacency matrix. Namely, the gap between the first and second eigenvalues of the (normalized) adjacency matrix.

The **normalized adjacency matrix** M of a D-regular undirected graph G, is the adjacency matrix of G divided by D. In terms of the rotation map, we have:

$$M_{u,v} = \frac{1}{D} \cdot \left| \{(i,j) \in [D]^2 : \mathrm{Rot}_G(u,i) = (v,j)\} \right| .$$

M is simply the transition probability matrix of a random walk on G. By the D-regularity of G, the all-1's vector $1_N = (1, 1, \ldots, 1) \in \mathbb{R}^N$ is an eigenvector of M of eigenvalue 1. It turns out that all the other eigenvalues of M have absolute value at most 1. We denote by $\lambda(G)$, the **second largest eigenvalue** (in absolute value) of G's normalized adjacency matrix. We refer to a D-regular undirected graph G on N vertices such that $\lambda(G) \leq \lambda$ as an (N, D, λ)**-graph**. It is well-known that the second largest eigenvalue of G is a good measure of G's expansion properties. In particular, it was shown by Tanner [Tan84] and Alon and Milman [AM85] that second-eigenvalue expansion implies (and is in fact equivalent [Alo86]) to the standard notion of **vertex expansion**. In particular, for every $\lambda < 1$ there exists $\varepsilon > 0$ such that for every (N, D, λ)-graph G and for any set S of at most half the vertices in G, at least $(1 + \varepsilon) \cdot |S|$ vertices of G are connected by an edge to some vertex in S (and in particular the neighborhood of S contains at least $\varepsilon|S|$ vertices outside of S). This immediately implies that G has a logarithmic diameter:

Proposition 2.2 *Let $\lambda < 1$ be some constant. Then for every (N, D, λ)-graph G and any two vertices s and t in G, there exists a path of length $O(\log N)$ that connects s to t.*

Proof: By the vertex expansion of G, for some $\ell = O(\log N)$ both s and t have more than $N/2$ vertices of distance at most ℓ from them in G. Therefore, there exists a vertex v that is of distance at most ℓ from both s and t. ∎

We can therefore conclude that st-connectivity in constant-degree expanders can be solved in log-space:

Proposition 2.3 *Let $\lambda < 1$ be some constant. Then there exists a space $O(\log D \cdot \log N)$ algorithm $\mathcal{A}_{exp}$ such that when a D-regular undirected graph G on N vertices is given to $\mathcal{A}_{exp}$ as input, the following hold:*

1. *If s and t are in the same connected component and this component is an (N', D, λ)-graph then $\mathcal{A}_{exp}$ outputs 'connected'.*

2. *If $\mathcal{A}_{exp}$ outputs 'connected' then s and t are indeed in the same connected component.*

Proof: The algorithm $\mathcal{A}_{\text{exp}}$ simply enumerates all D^ℓ paths of length $\ell = O(\log N)$ from s. (Where the leading constant in the big-O notation depends on λ as in Proposition 2.2.) The algorithm $\mathcal{A}_{\text{exp}}$ outputs 'connected' if and only if at least one of these paths encounters t.

Following any particular path from s of length ℓ requires space $O(\log N)$, (when given as input the sequence of ℓ edge labels in $[D] = \{1, 2, \ldots D\}$ traversed by this path). Enumerating all these D^ℓ paths requires space $O(\log D \cdot \log N)$. By Proposition 2.2, in case (1), s and t are of distance at most ℓ of each other and $\mathcal{A}_{\text{exp}}$ will indeed find a path from s to t and will output 'connected'. On the other hand, $\mathcal{A}_{\text{exp}}$ never outputs 'connected' unless it finds a path from s to t, implying (2). ■

Using the Probabilistic Method, Pinsker [Pin73] showed that most 3-regular graphs are expanders (in the sense of vertex expansion), and this result was extended to eigenvalue bounds in [Alo86, BS87, FKS89, Fri91]. Various *explicit* families of constant-degree expanders, some with optimal tradeoff between degree and expansion, were given in literature (cf. [Mar73, GG81, JM87, AM85, AGM87, LPS88, Mar88, Mor94, RVW02]). Our algorithm will employ a single constant size expander with rather weak parameters. This expander can be obtained by exhaustive search or by any of the explicit constructions mentioned above. In fact, one can use simpler explicit constructions than the ones given above, as we can afford a rather large degree (with respect to the number of vertices), rather than a constant degree. An example of a simpler construction that would suffice is the one given by Alon and Roichman [AR94], (see also related discussions in [RVW02] regarding their "base graph").

Proposition 2.4 *There exists some constant D_e and a $((D_e)^{16}, D_e, 1/2)$-graph.*

Finally, a key fact for our algorithm is that every connected, non-bipartite graph has a spectral gap which is at least inverse polynomial in the size of the graph (recall that a graph is non-bipartite if there is no partition of the vertices such that all the edges go between the two sides of the partition).

Lemma 2.5 (cf. [AS00]) *For every D-regular, connected, non-bipartite graph G on $[N]$ it holds that $\lambda(G) \leq 1 - 1/DN^2$.*

2.3 Powering

Our main transformation will take a graph and transform each one of its connected components (that in itself will be a connected, non-bipartite graph), into a constant degree expander. If we ignore the requirement that the graph remains constant degree, a simple way of amplifying the (inverse polynomial) spectral gap of a graph is by powering.

Definition 2.6 *Let G be a D-regular multigraph on $[N]$ given by rotation map Rot_G. The t'**th power** of G is the D^t-regular graph G^t whose rotation map is given by $\mathrm{Rot}_{G^t}(v_0, (a_1, a_2, \ldots, a_t)) = (v_t, (b_t, b_{t-1}, \ldots, b_1))$, where these values are computed via the rule $(v_i, b_i) = \mathrm{Rot}_G(v_{i-1}, a_i), i = 1, 2, \ldots, t$.*

Proposition 2.7 *If G is an (N, D, λ)-graph, then G^t is an (N, D^t, λ^t)-graph.*

Proof: The normalized adjacency matrix of G^t is the t'th power of the normalized adjacency matrix of G, so all the eigenvalues also get raised to the t'th power. ■

2.4 Two graph products

While taking a power of a graph reduces its second eigenvalue, it also increases its degree. As we are interested in producing constant-degree graphs, we need a complementing operation that reduces the degree of a graph without harming its expansion by too much. We now discuss two graph products that are capable of doing exactly that.

The first is the very natural product, known as the **replacement product**. Assume that G is a D-regular graph on $[N]$ and H is a d-regular graph on $[D]$ (where d is significantly smaller than D). Very intuitively, the replacement product of the two graphs is defined as follows: Each vertex v of G is replaced with a "copy" H_v of H. Each of the D vertices of H_v is connected to its neighbors in H_v but also to one vertex in H_w, where (v, w) is one of the D edges going out of v in G. The degree in the product graph is $d + 1$ (which is smaller than D).[2] A second, slightly more involved, product introduced by Reingold, Vadhan and Wigderson [RVW02], is the **zig-zag graph product**. Here too we replace each vertex v of G with a "copy" H_v of H. However, the edges of the zig-zag product of G and H correspond to a subset of the paths of length three in the replacement product of these graphs[3] (see formal definition below). The degree of the product graph here is d^2 (which should still be thought of as significantly smaller that D).

It is immediate from their definition, that both products can transform a graph G to a new graph (the product of G and H) of smaller degree. As discussed in the introduction, it was previously shown [RVW02, MR00] that if H is a "good enough" expander, then the expansion of the resulting graph is "not worse by much" than the expansion of G (see formal statement below for the zig-zag product). Either one of these products can be used in our USTCON algorithm (with some variation in the parameters). We find it more convenient to work here with the zig-zag product even though it is a bit more involved. More specifically, we find it less cumbersome to argue that our algorithm can be run in log space when using the zig-zag product. Hence we proceed by formally defining this product.

Definition 2.8 ([RVW02]) *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 $G \textcircled{z} H$ *is defined to be the d^2-regular graph on $[N] \times [D]$ whose rotation map $\mathrm{Rot}_{G \textcircled{z} H}$ is as follows (see Figure 1 for an illustration):*

$\mathrm{Rot}_{G \textcircled{z} H}((v, a), (i, j))$**:**

1. *Let* $(a', i') = \mathrm{Rot}_H(a, i)$.
2. *Let* $(w, b') = \mathrm{Rot}_G(v, a')$.
3. *Let* $(b, j') = \mathrm{Rot}_H(b', j)$.
4. *Output* $((w, b), (j', i'))$.

In [RVW02], $\lambda(G \textcircled{z} H)$ was bounded as a function of $\lambda(G)$ and $\lambda(H)$. The interesting case there was when both $\lambda(G)$ and $\lambda(H)$ were small constants (and in fact, $\lambda(G)$ is significantly smaller than $\lambda(H)$). In our context, $\lambda(H)$ will indeed be a small constant but G may have an extremely small spectral gap (recall that the spectral gap of G is $1 - \lambda(G)$). In this case, we want the spectral gap of $G \textcircled{z} H$ to be roughly the same as that of G (i.e., smaller by at most a constant factor). It turns out that the stronger bound on $\lambda(G \textcircled{z} H)$, given in [RVW02] implies a useful bound also in this case. We note that a much simpler proof

[2] Sometimes it is better to consider the *balanced* replacement product, where for every edge (v, w) in G the corresponding edge between H_v and H_w is taken d times in parallel. The degree of the product graph in this case is $2d$ instead of $d + 1$.

[3] Those length three paths that are composed of a "short edge" (an edge inside one of the copies H_v), a "long edge" (one that corresponds to an edge of G), and finally one additional "short edge".

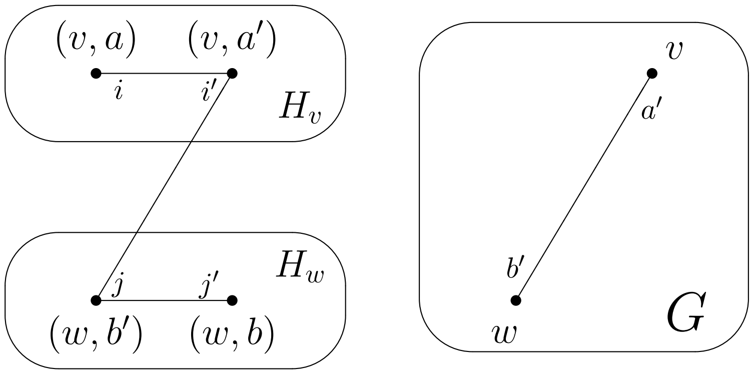

Figure 1: On the left – an edge of the zig-zag product is composed of three steps: a "short step" (in H_v), a "big step" (between H_v and H_w which corresponds to an edge of G between v and w), and a final "small step" (in H_w). The values i, i', j and j' are labels of edges of H (going out of the H vertices a, a', b' and b respectively). On the right – the projection of these steps on the graph G (which corresponds to the middle step specified by $(w, b') = \mathrm{Rot}_G(v, a')$).

for the sort of bound on the zig-zag product we need is given in [RTV06, RV05] (in a more general setting than the one considered in [RVW02]).

Theorem 2.9 ([RVW02]) *If G is an (N, D, λ)-graph and H is a (D, d, α)-graph, then $G Ⓩ H$ is a $(N \cdot D, d^2, f(\lambda, \alpha))$-graph, where*

$$f(\lambda,\alpha) = \frac{1}{2}(1-\alpha^2)\lambda + \frac{1}{2}\sqrt{(1-\alpha^2)^2\lambda^2 + 4\alpha^2}.$$

As a simple corollary, we have that the spectral gap of $G Ⓩ H$ is smaller than that of G by a factor that only depends on $\lambda(H)$.

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

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

Proof: Since $\lambda \leq 1$ we have that

$$\frac{1}{2}\sqrt{(1-\alpha^2)^2\lambda^2 + 4\alpha^2} \leq \frac{1}{2}\sqrt{(1-\alpha^2)^2 + 4\alpha^2} = \frac{1}{2}(1+\alpha^2) = 1 - \frac{1}{2}(1-\alpha^2).$$

Therefore, $f(\lambda, \alpha)$ from Theorem 2.9 satisfies $f(\lambda, \alpha) \leq 1 - \frac{1}{2}(1-\alpha^2)(1-\lambda)$. ■

3 Transforming graphs into expanders

This section gives a log-space transformation that essentially turns each one of the connected components of a graph into an expander. This is the main part of our USTCON algorithm.

Definition 3.1 (Main Transformation) *On input G and H, where G is a D^{16}-regular graph on $[N]$ and H is a D-regular graph on $[D^{16}]$, both given by their rotation maps, the transformation $\mathcal{T}$ outputs the rotation map of a graph G_ℓ defined as follows:*

- *Set $\ell = 2\lceil \log DN^2 \rceil$.*
- *Set G_0 to equal G, and for $i > 0$ define G_i recursively by the rule:*

$$G_i = (G_{i-1} \textcircled{z} H)^8.$$

Denote by $\mathcal{T}_i(G, H)$ the graph G_i, and $\mathcal{T}(G, H) = G_\ell$

Note that by the basic properties of powering and the zig-zag product, it follows inductively that each G_i is a D^{16}-regular graph over $[N] \times ([D^{16}])^i$. In particular, the zig-zag product of G_i and H is well defined. In addition, if D is a constant, then $\ell = O(\log N)$ and G_ℓ has $poly(N)$ vertices. Our first lemma shows that $\mathcal{T}$ is capable of turning an input graph G into an expander G_ℓ (as long as H is in itself an expander).

Lemma 3.2 *Let G and H be the inputs of $\mathcal{T}$ as in Definition 3.1. If $\lambda(H) \leq 1/2$ and G is connected and non-bipartite then $\lambda(\mathcal{T}(G, H)) \leq 1/2$.*

Proof: Since $G = G_0$ is connected and non-bipartite we have by Lemma 2.5 that $\lambda(G_0) \leq 1 - 1/DN^2$. By the choice of ℓ , a simple calculation shows that $(1 - 1/DN^2)^{2^\ell} < 1/2$ (using for example that for $x \leq 0.5$ it holds that $(1 - x)^2 \leq 1 - 1.5x$). It is therefore enough to prove that for every $i > 0$, it holds that $\lambda(G_i) \leq \max\{\lambda(G_{i-1})^2, 1/2\}$. Denote $\lambda = \lambda(G_{i-1})$. Since $\lambda(H) \leq 1/2$, we have by Corollary 2.10 that $\lambda(G_{i-1} \textcircled{z} H) \leq 1 - 3/8(1 - \lambda) < 1 - 1/3(1 - \lambda)$. By the definition of G_i and by Proposition 2.7 we have that $\lambda(G_i) < [1 - 1/3(1 - \lambda)]^8$. We now consider two cases. First, if $\lambda < 1/2$ then $\lambda(G_i) < (5/6)^8 < 1/2$. Otherwise, elementary calculation shows that $[1 - 1/3(1 - \lambda)]^4 \leq \lambda$ and therefore $\lambda(G_i) < \lambda^2$. The lemma follows. ■

As we are working our way to solving st-connectivity, rather than solving connectivity (the problem of deciding if the input graph is connected or not), our transformation should be meaningful even for graphs that are not connected (as even in this case the two input vertices s and t may still be in the same connected component). For that, we will argue that $\mathcal{T}$ operates separately on each connected component of G. The reason is that $\mathcal{T}$ is composed of two operations (the zig-zag product and powering), that also operate separately on each connected component. We will need some additional notation: For any graph G and subset of its vertices S, denote by $G|_S$ the subgraph of G induced by S (i.e., the graph on S which contains all of the edges in G between vertices in S). A set S is a connected component of G if $G|_S$ is connected and the set S is disconnected from the rest of G (i.e., there are no edges in G between vertices in S and vertices outside of S).

Lemma 3.3 *Let G and H be the inputs of $\mathcal{T}$ as in Definition 3.1. If $S \subseteq [N]$ is a connected component of G then*

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

Proof: We will only rely on S being disconnected from the rest of G, and will prove inductively that $\mathcal{T}_i(G|_S, H) = \mathcal{T}_i(G, H)|_{S \times ([D^{16}])^i}$. Note that for $i > 0$ this directly implies that $S \times ([D^{16}])^i$ is disconnected from the rest of $\mathcal{T}_i(G, H)$ (since both $\mathcal{T}_i(G|_S, H)$ and $\mathcal{T}_i(G, H)$ are D^{16}-regular, and thus all of the D^{16} edges incident to a vertex in $S \times ([D^{16}])^i$ reside inside $\mathcal{T}_i(G, H)|_{S \times ([D^{16}])^i}$). The base case $i = 0$ is trivial, and here too $S \times ([D^{16}])^i = S$ is disconnected from the rest of $\mathcal{T}_i(G, H) = G$, by assumption.

Assume by induction that $\mathcal{T}_i(G|_S, H) = \mathcal{T}_i(G, H)|_{S\times([D^{16}])^i}$. Set $G_i = \mathcal{T}_i(G, H)$ and $S_i = S\times([D^{16}])^i$ (and recall that S_i is disconnected from the rest of G_i). Then, by the definition of the zig-zag product, $S_i \times [D^{16}]$ is disconnected from the rest of $G_i \text{ⓩ} H$ and the edges incident to $S_i \times [D^{16}]$ in $G_i \text{ⓩ} H$ are exactly as in $G_i|_{S_i} \text{ⓩ} H$. By the definition of powering we now have that $S_i \times [D^{16}]$ is disconnected from the rest of $(G_i \text{ⓩ} H)^8$ and the edges incident to $S_i \times [D^{16}]$ in $(G_i \text{ⓩ} H)^8$ are exactly as in $(G_i|_{S_i} \text{ⓩ} H)^8$. This proves the induction hypothesis for $i + 1$ and completes the proof. ■

Finally, we need to argue that $\mathcal{T}$ is a log-space transformation (when D is a constant). The reason is that the evaluation of the rotation map $\mathrm{Rot}_{G_{i+1}}$ of each graph G_{i+1} in the definition of $\mathcal{T}$ requires just a constant additional amount of memory over the evaluation of Rot_{G_i}. Simply, the evaluation of $\mathrm{Rot}_{G_{i+1}}$ is composed of a constant number of operations, where each operation is either an evaluation of Rot_{G_i} or it requires constant amount of memory (and the same memory can be used for each one of these operations). So the additional memory needed for evaluating $\mathrm{Rot}_{G_{i+1}}$ is essentially a constant size counter (keeping track of which operation we are currently performing). Formalizing the above intuition is somewhat tricky, as careless composition of small space transformations will incur additional low term costs (which will result in an $O(\log N \log\log N)$-space algorithm). Nevertheless, this intuition can still be closely followed by a proof as shown by Goldreich [Gol08]. We choose a somewhat different approach of opening the recursive structure of our algorithm and bounding its use of memory directly.

Lemma 3.4 *For every constant D the transformation $\mathcal{T}$ of Definition 3.1 can be computed in space $O(\log N)$ on inputs G and H, where G is a D^{16}-regular graph on $[N]$ and H is a D-regular graph on $[D^{16}]$.*

Proof: We describe an algorithm $\mathcal{A}_\mathcal{T}$ that on inputs G and H computes the rotation map Rot_{G_ℓ} of $G_\ell = \mathcal{T}(G, H)$. Namely, given G and H (written on the read-only input tape), it enumerates all values $(\bar{v}, \bar{a})$ in the domain of Rot_{G_ℓ} and outputs $[(\bar{v}, \bar{a}), \mathrm{Rot}_{G_\ell}(\bar{v}, \bar{a})]$. Recall that a value $(\bar{v}, \bar{a})$ in the domain of Rot_{G_ℓ} consists of $\bar{v} \in [N] \times ([D^{16}])^\ell$ which is the name of a G_ℓ vertex, and $\bar{a} \in [D^{16}]$, which is the label of a G_ℓ edge. Since $\ell = O(\log N)$ and D is a constant, the length of each value $(\bar{v}, \bar{a})$ is $O(\log N)$ and therefore enumerating all of these values can be done in space $O(\log N)$. It remains to show that for any particular value $(\bar{v}, \bar{a})$, evaluating $\mathrm{Rot}_{G_\ell}(\bar{v}, \bar{a})$ can also be done in the required space.

The algorithm $\mathcal{A}_\mathcal{T}$ will first allocate the following variables: v which will take value in $[N]$ (specifying a vertex of G), and $\ell + 1$ variables $a_0, a_1 \dots a_\ell$ each taking value in $[D^{16}]$ (and each specifying a vertex name of H; In addition, a_0 may specify an edge label of G). It is sometimes convenient to view each one of $a_1 \dots, a_\ell$ as specifying a sequence of 16 edge labels of H. In this case we denote $a_i = k_{i,1} \dots k_{i,16}$. Now, $\mathcal{A}_\mathcal{T}$ will copy the value $(\bar{v}, \bar{a})$ into the above mentioned variables: $\bar{v}$ into $v, a_0, \dots, a_{\ell-1}$ and $\bar{a}$ into a_ℓ. Throughout the execution of $\mathcal{A}_\mathcal{T}$, the values of these variables will slowly evolve such that when $\mathcal{A}_\mathcal{T}$ finishes (for this particular $(\bar{v}, \bar{a})$), *the same variables* will contain the desired output $\mathrm{Rot}_{G_\ell}(\bar{v}, \bar{a})$ (which is of the same range as the input $(\bar{v}, \bar{a})$).

We describe the operation of $\mathcal{A}_\mathcal{T}$ in a recursive manner that closely follows the definition of $\mathcal{T}$. Particularly, at each level of the recursion, $\mathcal{A}_\mathcal{T}$ will evaluate Rot_{G_i} for some i on the appropriate prefix $v, a_0, \dots, a_i$ of the variables defined above. For the base case $i = 0$, $\mathrm{Rot}_{G_0} = \mathrm{Rot}_G$ is written on the input tape, and can therefore be evaluated in space $O(\log N)$ by simply searching the input tape for the desired entry. For larger i, the evaluation of Rot_{G_i} is as follows:

For $j = 1$ **to** 16

- Set $a_{i-1}, k_{i,j} \leftarrow \mathrm{Rot}_H(a_{i-1}, k_{i,j})$.
- If j is odd, recursively set $v, a_0 \dots a_{i-1} \leftarrow \mathrm{Rot}_{G_{i-1}}((v, a_0 \dots a_{i-2}), a_{i-1})$.

- If $j = 16$, reverse the order of the individual labels in a_i: Set $k_{i,1}, \ldots, k_{i,16} \leftarrow k_{i,16}, \ldots, k_{i,1}$.

The correctness of $\mathcal{A}_\mathcal{T}$ immediately follows from the definition of $\mathcal{T}$ and from the operations of which it consists (powering and the zig-zag product). Essentially, going over the operations (in the first two bullets) for any two consecutive values of j corresponds to one step on $(G_{i-1} ⓩ H)$. Repeating eight times implies a path of length eight on $(G_{i-1} ⓩ H)$, or alternatively one step on $(G_{i-1} ⓩ H)^8$. The third bullet reverses the order of labels to fit the definition of zig-zag and powering.

We therefore concentrate on the space complexity of $\mathcal{A}_\mathcal{T}$. Note that each node of the recursion tree performs a constant number of operations and makes a constant number of recursive calls. In addition the depth of the recursion is $\ell + 1 = O(\log N)$. Therefore, maintaining the recursion can be done in space $O(\log N)$. Furthermore, each one of the basic operations (evaluating Rot_G, evaluating Rot_H, and reversing the order of labels in the last step) can be performed in space $O(\log N)$. Finally, the only memory that needs to be kept after a basic operation is performed, is the memory holding the variables $v, a_0, \ldots, a_\ell$ (that are shared by all of these operations), and the memory for maintaining the recursion. For completeness, we give in Appendix A an implementation of $\mathcal{A}_\mathcal{T}$ which includes low level details such as an exact manner one may maintain the recursion. We therefore conclude that the space complexity of $\mathcal{A}_\mathcal{T}$ is $O(\log N)$ which completes the proof. ■

4 A log-space algorithm for USTCON

This section puts together the tools developed above into a deterministic log-space algorithm that decides undirected st-connectivity. As will be discussed in Section 5, the algorithm can also output a path from s to t if such a path exists.

Theorem 4.1 $\mathrm{USTCON} \in \mathrm{L}$

As undirected USTCON is complete for SL [LP82], Theorem 4.1 can be rephrased as follows.

Theorem 4.2 $\mathrm{SL} = \mathrm{L}$

Proof: [of Theorem 4.1] We give an algorithm $\mathcal{A}_{\mathrm{con}}$ that gets as input a graph G over the set of vertices $[N]$, and two vertices s and t in $[N]$. For concreteness, we assume that the graph is given via the adjacency matrix representation. $\mathcal{A}_{\mathrm{con}}$ will answer 'connected' if and only if there exists a path in G between s and t (i.e., s and t are in the same connected component). Furthermore, G will use space which is logarithmic in its input size.

The algorithm $\mathcal{A}_{\mathrm{con}}$ will need to evaluate the rotation map of a $((D_{\mathrm{e}})^{16}, D_{\mathrm{e}}, 1/2)$-graph H, where D_{e} is some constant. By Proposition 2.4, there exists such a graph and therefore $\mathcal{A}_{\mathrm{con}}$ can obtain it by exhaustive search using constant amount of memory (a more efficient alternative is of course to obtain H by any of the explicit constructions of expanders mentioned in Section 2.2).

Let $\mathcal{T}$ be the transformation given by Definition 3.1. We would like to apply $\mathcal{T}$ to G and H in order to obtain a graph where each connected component is an expander. For such graphs, st-connectivity can be solved in logarithmic space by Proposition 2.3. However, we will first need to preprocess G in order to get a new graph G_{reg} such that (G_{reg}, H) is a correct input to $\mathcal{T}$. In particular, we need G_{reg} to be a D_{e}^{16}-regular graph given by its rotation map. There are various ways of transforming G to G_{reg}. The one given here was selected for its simplicity even though it is not the most efficient one possible (in terms of the size of G_{reg}). Essentially, we replace every vertex of G with a cycle of length N and each of the vertices (v, w), where there is an edge between v and w in G, is also connected to (w, v) (the rest of the edges are self loops). The rotation map $\mathrm{Rot}_{G_{\mathrm{reg}}} : ([N] \times [N]) \times [D_{\mathrm{e}}^{16}] \mapsto ([N] \times [N]) \times [D_{\mathrm{e}}^{16}]$ of G_{reg} is formally defined as follows:

- $\mathrm{Rot}_{G_{\mathrm{reg}}}((v,w),1) = ((v,w'),2)$, where $w' = w+1$ if $w < N$ and $w' = 1$ otherwise.

- $\mathrm{Rot}_{G_{\mathrm{reg}}}((v,w),2) = ((v,w'),1)$, where $w' = w-1$ if $w > 1$ and $w' = N$ otherwise.

- In case there is an edge between v and w in G then $\mathrm{Rot}_{G_{\mathrm{reg}}}((v,w),3) = ((w,v),3)$. Otherwise, $\mathrm{Rot}_{G_{\mathrm{reg}}}((v,w),3) = ((v,w),3)$.

- For $i > 3$, $\mathrm{Rot}_{G_{\mathrm{reg}}}((v,w),i) = ((v,w),i)$.

The transformation from G (given by its adjacency matrix) to G_{reg} (given by its rotation map) is clearly computable in logarithmic space. Furthermore, G_{reg} is D_{e}^{16}-regular by definition and all its connected components are non-bipartite (as every vertex in G_{reg} has self loops). Finally, for every connected component $S \subseteq [N]$ of G we have that $S \times [N]$ is a connected component in G_{reg}. To see that, we first note that for every vertex $v \in [N]$ the set of vertices $v \times [N]$ is in the same connected component of G_{reg} (as this set is connected by a cycle). Furthermore, there is an edge in G_{reg} between some vertex in $v \times [N]$ and some vertex in $w \times [N]$ if and only if v and w are connected by an edge in G (the only possible edge that can connect these subsets is an edge between (v,w) and (w,v) which only exists in G_{reg} if there is an edge between v and w in G).

Now define $G_{\mathrm{exp}} = \mathcal{T}(G_{\mathrm{reg}}, H)$, and $\ell = O(\log N)$ is the corresponding value as in Definition 3.1. Let S be the connected component of G, such that $s \in S$. By the arguments above, $S \times [N]$ is a connected component of G_{reg}, and $G_{\mathrm{reg}}|_{S\times[N]}$ is non-bipartite. By Lemma 3.3, $S \times [N] \times ([D_{\mathrm{e}}^{16}])^{\ell}$ is disconnected from the rest of G_{exp} (as both G_{exp} and $G_{\mathrm{exp}}|_{S\times[N]\times([D_{\mathrm{e}}^{16}])^{\ell}} = \mathcal{T}(G_{\mathrm{reg}}|_{S\times[N]}, H)$ are D_{e}^{16}-regular). By Lemma 3.2 and Lemma 3.3, we have that $\lambda(G_{\mathrm{exp}}|_{S\times[N]\times([D_{\mathrm{e}}^{16}])^{\ell}}) \leq 1/2$. In particular, we have that $S \times [N] \times ([D_{\mathrm{e}}^{16}])^{\ell}$ is a connected component of G_{exp}.

Let $\mathcal{A}_{\mathrm{exp}}$ be the the algorithm guaranteed by Proposition 2.3 (which decides undirected st-connectivity correctly in graphs where the connected component of the starting vertex is an expanders). The algorithm $\mathcal{A}_{\mathrm{con}}$ will now invoke $\mathcal{A}_{\mathrm{exp}}$, on the graph G_{exp} and the vertices $s' = (s, 1^{\ell+1})$ and $t' = (t, 1^{\ell+1})$. If $\mathcal{A}_{\mathrm{exp}}$ outputs that s' and t' are connected in G_{exp} then $\mathcal{A}_{\mathrm{con}}$ will output that s and t are connected in G. Otherwise, $\mathcal{A}_{\mathrm{con}}$ will output that s and t are not connected.

The algorithm $\mathcal{A}_{\mathrm{con}}$ is log-space since it is composed of a constant number of log-space procedures: (1) The transformation from G to G_{reg}. (2) The transformation from G_{reg} to G_{exp}, which is computable by a log-space algorithm $\mathcal{A}_{\mathcal{T}}$ by Lemma 3.4. (3) The algorithm $\mathcal{A}_{\mathrm{exp}}$ which is log-space by Proposition 2.3. Correctness of $\mathcal{A}_{\mathrm{con}}$ is argued as follows. First, s' and t' are connected in G_{exp} if and only if s and t are connected in G (since $S \times [N] \times ([D_{\mathrm{e}}^{16}])^{\ell}$ is a connected component of G_{exp}, where S is the connected component of G that contains s). The correctness of $\mathcal{A}_{\mathrm{con}}$ now follows since Proposition 2.3 implies that $\mathcal{A}_{\mathrm{exp}}$ will output 'connected' if and only if s' and t' are indeed connected in G_{exp} (as $\lambda(G_{\mathrm{exp}}|_{S\times[N]\times([D_{\mathrm{e}}^{16}])^{\ell}}) \leq 1/2$). ■

5 Universal traversal and exploration sequences

In this section, we look closer into our USTCON algorithm and conclude that it also solves the corresponding search problem (i.e., finding the path from s to t if such a path exist). In addition, it implies efficiently-constructible universal-traversal sequences for graphs with restricted labeling, and universal exploration sequences for general graphs. The sort of restriction we pose on the labeling of graphs is a strengthening of the "consistent labeling" used in [HW93]. In a subsequent work [RTV06], our restriction is relaxed back to "consistent labeling", and is therefore identical to the restriction of [HW93] for universal-traversal sequences on expander graphs.

We start by analyzing $\mathcal{T}$, the main transformation of the algorithm, given by Definition 3.1. We show that every edge in $\mathcal{T}(G, H)$ translates to a path in G between the appropriate vertices, and that this path is log-space constructible (as this path is indeed computed during the log-space evaluation of $\mathcal{T}$). Looking ahead to the universal-traversal sequences, we note that if we restrict the labeling of G, then the labels of edges, traversed along this path, are independent of G.

Definition 5.1 *Let π be a permutation over $[D]$ and Rot_G the rotation map of a D-regular graph G. Then Rot_G is π-consistent if for every v, i, w and j such that $\mathrm{Rot}_G(v,i) = (w,j)$, it holds that $j = \pi(i)$. In such a case we may also say that the labeling of G is π-consistent.*

An example of a π-consistent labeling is symmetric labeling where π is simply the identity. Namely, every edge is labelled in the same way from both its end points. However, other kinds of π-consistent labeling come up naturally. An example for that is the labeling of G_{reg} in the proof of Theorem 4.1. We can now state the appropriate technical lemma regarding the transformation $\mathcal{T}$.

Lemma 5.2 *Let D be some constant. Let G be a D^{16}-regular graph on $[N]$ and let H be a D-regular graph on $[D^{16}]$, both given by their rotation maps. Let $G_\ell = \mathcal{T}(G, H)$, where $\mathcal{T}$ and ℓ are given by Definition 3.1.*

There exists a log-space algorithm $\mathcal{A}_{e2p}$ such that given Rot_G, Rot_H and $(\bar{v}, \bar{a})$ in the domain of Rot_{G_ℓ}, it outputs a sequence of labels in $[D^{16}]$ with the following property: If the first element of $\bar{v}$ is a vertex $u \in [N]$ and the the first element of $\mathrm{Rot}_{G_\ell}(\bar{v}, \bar{a})$ is a vertex $w \in [N]$, then the walk on G from u using the labels that the algorithm outputs leads to w.

Furthermore, for every fixed permutation π on $[D^{16}]$, if the labeling of G is π-consistent, the log-space algorithm can produce the sequence of labels without access to Rot_G.

Proof: Consider the log-space algorithm $\mathcal{A}_\mathcal{T}$ in the proof of Theorem 3.4, as it evaluates $\mathrm{Rot}_{G_\ell}(\bar{v}, \bar{a})$. Consider in particular the two variables v and a_0 used by $\mathcal{A}_\mathcal{T}$. To begin with, v is initialized to the value u (the first element of $\bar{v}$). At the end, v is guaranteed to contain the value w. Throughout the run of $\mathcal{A}_\mathcal{T}$, the variable v is only updated by the rule $v, a_0 \leftarrow \mathrm{Rot}_G(v, a_0)$ (used at the bottom of the recursion). We enhance $\mathcal{A}_\mathcal{T}$ a bit, to define an algorithm $\mathcal{A}_{\mathrm{e2p}}$ as claimed by the lemma. By the above discussion, all that $\mathcal{A}_{\mathrm{e2p}}$ needs to do is to output the value of a_0 just before each time $\mathcal{A}_\mathcal{T}$ updates v.

Regarding the second part of the lemma. We note that the the only way Rot_G influences the value of a_0 is through the evaluations $v, a_0 \leftarrow \mathrm{Rot}_G(v, a_0)$. If G is π-consistent, then $\mathcal{A}_{\mathrm{e2p}}$ can completely ignore the variable v and the rotation map of G. To simulate $\mathcal{A}_\mathcal{T}$, it is sufficient that whenever $\mathcal{A}_\mathcal{T}$ evaluates $v, a_0 \leftarrow \mathrm{Rot}_G(v, a_0)$, then $\mathcal{A}_{\mathrm{e2p}}$ will evaluate $a_0 \leftarrow \pi(a_0)$. ■

Using Lemma 5.2, it is not hard to obtain the algorithm that finds paths in undirected graphs.

Theorem 5.3 *There exists a log-space algorithm $\mathcal{A}_{srch}$ that gets as input a graph G over the set of vertices $[N]$, and two vertices s and t in $[N]$, and outputs a path from s to t if such a path exists (otherwise it outputs 'not connected').*

Proof: Consider the algorithm $\mathcal{A}_{\mathrm{con}}$ from the proof of Theorem 4.1. We revise it to an algorithm $\mathcal{A}_{\mathrm{srch}}$ as required by the theorem. First, we note that it is enough for $\mathcal{A}_{\mathrm{srch}}$ to output a path from $(s, 1)$ to $(t, 1)$ in G_{reg} if such a path exists, as it is easy to transform (in log-space) such a path to a path from s to t in G (and the existence of the two paths is equivalent).

Next we note that $\mathcal{A}_{\mathrm{con}}$ enumerates all logarithmically-long paths from $s' = (s, 1^{\ell+1})$ in G_{exp}. If it does not find a path that visits $t' = (t, 1^{\ell+1})$, it concludes that s and t are not connected in G. Therefore, in such a case, $\mathcal{A}_{\mathrm{srch}}$ can output 'not connected'. Otherwise $\mathcal{A}_{\mathrm{con}}$ found a short path from s' to t'. Apply

$\mathcal{A}_{\text{e2p}}$ guaranteed by Lemma 5.2 on each edge on the path from s' to t'. Each time $\mathcal{A}_{\text{e2p}}$ outputs a sequence of edge-labels in G_{reg}. Let $\vec{a}$ be the concatenation of these sequences. It follows from Lemma 5.2 that the path in G_{reg} starting from $(s,1)$ and following the edges according to the labels in $\vec{a}$ leads to $(t,1)$. The theorem now follows. ■

To give our result regarding universal-traversal sequences, we need some notation. Let $\vec{a} = \{a_1, ..., a_m\}$ be a sequence of values in $[D]$ (these are interpreted as edge labels). $\vec{a}$ is an (N, D)-universal traversal sequence, if for every connected D-regular, labelled graph G on N vertices, and every start vertex $s \in [N]$, the walk that starts at s and follows the edges labelled $a_1, ..., a_m$, visits every vertex in the graph. For a permutation π over $[D]$, we say that $\vec{a}$ is an (N, D) π-universal traversal sequence, if the above property holds for every connected D-regular graph on N vertices, *that has a π-consistent labeling*, (rather than for all such graphs).

Theorem 5.4 *There exists a log-space algorithm that takes as input 1^N and a permutation π over $[D]$ and outputs an (N, D) π-universal traversal sequence.*

Proof: First we argue that it is enough to construct an $(N \cdot D, D_{\text{e}}^{16})$ π'-universal sequence for the following simple permutation: $\pi'(1) = 2, \pi'(2) = 1$ and for every $i > 2$ $\pi'(i) = i$. Furthermore, all we need is that the sequence will traverse non-bipartite graphs. Consider a (connected) D-regular graph G on N vertices that has a π-consistent labeling. This graph can be transformed into a D_{e}^{16}-regular (connected and non-bipartite) graph G' on $N \cdot D$ vertices that has a π'-consistent labeling. Each vertex $v \in N$ is transformed into a cycle over D vertices $(v, 1), \ldots, (v, D)$, the edges of the cycle are labelled 1 and 2 (just as in the definition of G_{reg} in the proof of Theorem 4.1). The edge labelled 3 going out of (v, i) will lead to $\mathrm{Rot}_G(v, i)$ (and will be labelled 3 from that end as well). All other edges are self loops.

Assume that a sequence of labels $a_1, ..., a_m$, visits every vertex of G' starting from every vertex $(v, 1)$. We can translate this (in log space) into a sequence of labels $b_1, \ldots, b_{m'}$ that traverses G from every vertex v. To do that, we simulate the walk on G' from an arbitrary vertex $(v, 1)$. As v is unknown and our simulation does not rely on G, it will only know at each point the value b such that the walk at this point visits some vertex (w, b) of G' (where w is unknown). First b is set to 1. Then, during the simulation, labels $a_i > 3$ can be ignored (as they are self loops). Given labels 1 and 2, b can easily be updated (these are edges on the cycle). Finally, when encountering $a_i = 3$ the walk moves from a vertex (w, b) to a vertex $(w', \pi(b))$ (as the labeling of G is π-consistent), and so it is easy to update the value of b (given access to π). The projection of the walk on G is exactly the edges labelled 3 that are taken by the walk on G'. Therefore, to transform the sequence of a_i's to the sequence of b_i's we can simply output (throughout the above simulation) the current value of b, whenever we encounter a label $a_i = 3$.

Now we consider a D_{e}^{16}-regular (connected and non-bipartite) graph G' on $N \cdot D$ vertices that has a π'-consistent labeling. Let H be a $((D_{\text{e}})^{16}, D_{\text{e}}, 1/2)$-graph. Finally let $G_\ell = \mathcal{T}(G, H)$, where $\mathcal{T}$ and ℓ are given by Definition 3.1. By Lemma 3.2, $\lambda(G_\ell) \leq 1/2$ and therefore its diameter is logarithmic. Therefore, for every two vertices v and u of G' one of the polynomially many sequences of labels (of the appropriate logarithmic length) will visit $(u, 1^\ell)$, starting at $(v, 1^\ell)$. Let B be the set of all these sequences of labels. Lemma 5.2 gives a way to translate in log-space each one of the sequences in B into a corresponding sequence of edge-labels of G'. Let B' be the set of translated sequences. By Lemma 5.2 and the above argument, for every two vertices v and u of G' one of the sequences in B' will lead a walk in G' that starts in v through the vertex u. We should also note that given a sequence $\vec{a} = a_1, \ldots, a_m$ that leads from a vertex v to a vertex u, we have that the sequence $\pi'^{-1}(a_m), \ldots, \pi'^{-1}(a_1)$ leads from u to v (this operation simply reverses the walk). We refer to this latter sequence as the reverse of $\vec{a}$ (note that given $\vec{a}$ as input, the reverse of $\vec{a}$ can easily be computed in logarithmic space - to output the ith symbol look for a_{m-i+1}, and apply $\pi'^{-1} \equiv \pi'$ to it). Finally, we can define a sequence that traverses all of the vertices of G' regardless of the starting vertex. Simply, we concatenate for each sequences in B' its reversed sequence and concatenate all of these sequences one after the other. By the arguments above, for every vertex v, the sequence we obtain will visit v after every pair of a sequence and its reversed sequence. Furthermore, for every vertex u, one of these sequences will lead to u. As the log-space construction of this sequence ignores the graph G' (and only relies on π'), we obtained the desired $(N \cdot D, D_{\mathsf{e}}^{16})$ π'-universal sequence for non-bipartite graphs. The lemma follows. ■

In an (N, D)-universal *exploration* sequence, the sequence of labels is interpreted as offsets rather than absolute labels. This means that if we entered a vertex v on an edge labelled a (from v's view point), and we are reading the label b, then we will leave v on the edge labelled $a + b$ (or $a + b - D$ if $a + b > D$). In fact this notion can apply to graphs that are not-regular (it then makes sense to allow negative elements in the sequence). Universal-exploration sequences have more flexibility than universal-traversal sequences. For example, it is not clear how to transform a universal-traversal sequence for degree-3 graphs to one for higher-degree graphs. This is easy for universal-exploration sequences (and seems desirable as USTCON can easily be reduced to USTCON for regular-graphs of any degree larger than 2). Koucky [Kou03, Theorem 85] showed how to transform a universal-traversal sequence to a universal-exploration sequence. His transformation (which relies on a transformation of graphs that is essentially the same as the one from G to G' in the proof of Theorem 5.4), only needs the universal-sequence to work for graphs with π-consistent labeling for some simple permutation π. We can therefore conclude from Theorem 5.4 a log-space construction for general universal-exploration sequences.

Corollary 5.5 *There exists a log-space algorithm that takes as input $(1^N, 1^D)$ and produces an (N, D)-universal exploration sequence.*

6 Discussion and further research

We start by comparing the techniques of this paper with some previous ones, with the goal of shedding some light on the source of our improvements. We continue by discussing some open problems and the results of a subsequent work.

Comparison with previous techniques The USTCON algorithms of [Sav70, NSW89, ATSWZ00] also operate by transforming, in phases, the input graph into a more accommodating one. In each one of these algorithms, each phase "charges" logarithmic amount to the space complexity of the algorithm. The improvement in the space complexity is directly correlated with reducing the number of phases needed for the transformation. With this approach, the only way to obtain a log-space algorithm is to reduce the number of phases to a constant. We deviate from this direction, as we use a logarithmic number of phases (just as in Savitch's algorithm), to gradually improve the connectivity of the input graph. The space efficiency of our algorithm stems from each transformation being significantly less costly in space.

The parameter being improved by [NSW89, ATSWZ00], is the size of the graph (each transformation shrinks the graph by collapsing it to a "representative" subset of the vertices). In contrast, our transformation will in fact expand the graph by a polynomial factor (as each phase, enlarges our graph by a constant factor).[4] The parameter Savitch's transformation improves is the diameter of the graph, which is much closer to the parameter we improve (the expansion). In fact, each phase of Savitch's algorithm can be described very similarly to our algorithm. Each one of these phases consists of squaring the graph and then removing parallel edges (which may reduce the degree). Although all that is eventually needed by our algorithm is

[4]It is interesting to note that in the alternative proof that SL=L given in [RV05], the size of the graph remains the same and it is the degree that moderately enlarges.

indeed that the *diameter* of the resulting graph will be small, our analysis relies on bounding the *expansion* of intermediate graphs – a stronger notion of connectivity than the diameter. This allows our transformation to preserve constant degree of the graph (rather than linear degree in Savitch's algorithm), which is crucial for our analysis of the space complexity.

Further Research

There are many open problems and new research directions brought up by this work, we discuss just a few of those. A very natural question is whether the techniques of this paper can be used towards a proof of $\mathrm{RL} = \mathrm{L}$. While progress in the context of RL does not seem immediate (as the case of symmetric computations does seem significantly easier), we feel that it is still quite plausible. A more ambitious research direction is to reevaluate the common conjecture that Savitch's algorithm is optimal for STCON. While this conjecture may very well be correct, we feel that there is still not enough evidence supporting it. Another open problem is to come up with full-fledged, efficiently-constructible, universal-traversal sequences (see Section 5). Interestingly, it seems that this problem shares some of the obstacles that one encounters when trying to generalize the USTCON algorithm to solving RL (this is formalized to some extent in the work of [RTV06]; see discussion below).

Finally, we have made no attempt to optimize our algorithm in terms of running time (or the constant in the space complexity). Major improvements in efficiency can come about by better analysis of the zig-zag and replacement products. These may also determine which one of these products yields a more efficient algorithm. Important progress in this direction was done by Rozenman and Vadhan. They give an alternative, though related, deterministic log-space algorithm for USTCON. Their analysis relies on a new, very natural, graph operation which they call derandomized squaring. This operation improves the expansion of a graph comparably to squaring but with a significantly smaller increase in the degree. They give a *tight* bound on the expansion of the resulting graph using a beautiful and enlightening new analysis.[5] Correspondingly, their algorithm obtains much better performance (though there is still much room for improvement).

Current boundaries of our approach towards derandomizing RL In a subsequent work with Trevisan and Vadhan [RTV06], we have made some progress on extending our techniques to dealing with the general RL case. We obtained the following results:

1. Generalizing our techniques to directed graphs (digraphs), we presented a deterministic, log-space algorithm that given a *regular* digraph G (or, more generally, a digraph with Eulerian connected components) and two vertices s and t, finds a path from s to t if one exists.

2. For digraphs that are regular and *consistently labelled*, we were able to produce *pseudorandom walks* (and universal-traversal sequences) in logarithmic space.

3. We have proved that if the pseudorandom walks of item (2) could be generalized to all regular digraphs (including ones that are not consistently labelled) then $\mathrm{L} = \mathrm{RL}$. This was done so by exhibiting a new complete promise problem for RL, and showing that such a problem can be solved in deterministic logarithmic space given a log-space pseudorandom walk generator for regular digraphs. The complete promise problem is essentially STCON restricted to digraphs for which the random walk is promised to have polynomial mixing time (such a problem indeed seems more amenable to our techniques).

In another subsequent work with Chung and Vadhan [CRV07], we have shown how to solve STCON in deterministic log-space in digraphs if (i) we are given a stationary distribution of the random walk on the

[5]Their analysis also translates to a simple new analysis of the zig-zag and replacement products. Unfortunately, the analysis for these products is probably still not tight.

graph in which both of the input vertices s and t have nonnegligible probability mass and (ii) the random walk which starts at the source vertex s has polynomial mixing time.

Summing up the results of [RTV06, CRV07], we can identify different obstacles in extending our techniques to solving RL when we consider *oblivious* and *explicit* derandomization. Loosely, the setting of explicit derandomization is one where we are given the RL machine and directly try to derandomize this particular machine. In this case we learn from [CRV07] that the obstacle is knowing (or being able to approximate) the probabilities of intermediate configurations. An example of oblivious derandomiztion is derandomiztion by pseudorandom generators (as such generators work for the entire class of problems rather than for specific problem). We learn from [RTV06] that to derandomize RL it is sufficient to concentrate on walks on *regular* digraphs (for which the stationary distribution is known to be uniform). For such derandomization the challenge revolves around the *labeling* of edges (in the explicit derandomization case, labeling is never a problem as it is easy to turn a graph into a consistently labeled one).

Acknowledgments

This work came about during a delightful visit to UC Berkeley. I am most grateful to Irit Dinur and Luca Trevisan for many hours of stimulating discussions on closely related topics and for creating the most conducive research environment possible for me. I would like to thank Moni Naor, Ran Raz, Salil Vadhan and Avi Wigderson for many discussions that helped me form my intuitions on the derandomization of space bounded computations. Among other contributions, I want to thank Moni for steering me towards this topic early on during my PhD studies, and to thank Ran, Salil and Avi for intuitions formed during our joint work on [RR99, RVW00]. Finally, I would like to thank the anonymous reviewers for their careful reading and for many useful comments.

References

[AKS87] Miklós Ajtai, János Komlós, and E. Szemerédi. Deterministic simulation in LOGSPACE. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 132–140, New York City, 25–27 May 1987.

[AKL+79] 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 (FOCS)*, pages 218–223, San Juan, Puerto Rico, 29–31 October 1979. IEEE.

[Alo86] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[AGM87] Noga Alon, Zvi Galil, and Vitali D. Milman. Better expanders and superconcentrators. *Journal of Algorithms*, 8(3):337–347, 1987.

[AM85] Noga Alon and Vitali D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory. Series B*, 38(1):73–88, 1985.

[AR94] Noga Alon and Yuval Roichman. Random Cayley graphs and expanders. *Random Structures & Algorithms*, 5(2):271–284, 1994.

[AS00] Noga Alon and Benny Sudakov. Bipartite subgraphs and the smallest eigenvalue. *Combinatorics, Probability & Computing*, 9(1), 2000.

[AG96] Carme Alvarez and Raymond Greenlaw. A compendium of problems complete for symmetric logarithmic space. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(039), 1996.

[ATSWZ00] Roy Armoni, Amnon Ta-Shma, Avi Wigderson, and Shiyu Zhou. An $o(log(n)^{4/3})$ space algorithm for (s,t) connectivity in undirected graphs. *Journal of the ACM*, 47(2):294–311, 2000.

[BNS89] László Babai, Noam Nisan, and Mario Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *Journal of Computer and System Sciences*, pages 204–232, 15–17 May 1989.

[BCD$^+$89] Allan Borodin, Stephen A. Cook, Patrick W. Dymond, Walter L. Ruzzo, and Martin Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing (SICOMP)*, 18(3):559–578, 1989.

[BS87] Andrei Broder and Eli Shamir. On the second eigenvalue of random regular graphs. In *28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 286–294, Los Angeles, California, 12–14 October 1987. IEEE.

[CRV07] Kai-Min Chung, Omer Reingold, and Salil Vadhan. S-T connectivity on digraphs with known stationary distribution. In *Proceedings of the 22nd Annual IEEE Conference on Computational Complexity (CCC)*, pages 236–249, 12–16 June 2007. Full version posted as *ECCC* TR07-030.

[Din07] Irit Dinur. The PCP Theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

[Fri91] Joel Friedman. On the second eigenvalue and random walks in random d-regular graphs. *Combinatorica*, 11(4):331–362, 1991.

[FKS89] Joel Friedman, Jeff Kahn, and Endre Szemerédi. On the second eigenvalue in random regular graphs. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing (STOC)*, pages 587–598, Seattle, Washington, 15–17 May 1989.

[GG81] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, June 1981.

[Gol05] Oded Goldreich. Bravely, moderately: A common theme in four recent results. *Electronic Colloquium on Computational Complexity (ECCC)*, (098), 2005. Also appeared as part of SIGACT news complexity theory column 51.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. Online drafts at http://www.wisdom.weizmann.ac.il/ oded/cc-drafts.html.

[GW02] Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. In *Proceedings of the 6th International Workshop on Randomization and Computation (RANDOM)*, pages 209–223, 2002.

[HW93] Shlomo Hoory and Avi Wigderson. Universal traversal sequences for expander graphs. *Inf. Process. Lett.*, 46(2):67–69, 1993.

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

[JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, 2004.

[JM87] Shuji Jimbo and Akira Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7(4):343–355, 1987.

[KW93] Mauricio Karchmer and Avi Wigderson. On span programs. In *Proc. of the 8th Structures in Complexity conference*, pages 102–111, 1993.

[KvM02] Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing (SICOMP)*, 31(5):1501–1526, 2002.

[Kou01] Michal Koucky. Universal traversal sequences with backtracking. In *IEEE Conference on Computational Complexity (CCC)*, pages 21–27, 2001.

[Kou03] Michal Koucky. *On traversal sequences, exploration sequences and completeness of Kolmogorov random strings*. PhD thesis, Rutgers University, 2003.

[LP82] Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computation. *Theor. Comput. Sci.*, 19:161–187, 1982.

[LPS88] Alex Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[MR96] Neal Madras and Dana Randall. Factoring markov chains to bound mixing rates. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 194–203, 1996.

[Mar73] Gregory A. Margulis. Explicit constructions of expanders. *Problemy Peredachi Informatsii*, 9(4):71–80, 1973.

[Mar88] Gregory A. Margulis. Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. *Problemy Peredachi Informatsii*, 24(1):51–60, 1988.

[MR00] Russell A. Martin and Dana Randall. Sampling adsorbing staircase walks using a new markov chain decomposition method. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 492–502, Redondo Beach, CA, 17–19 October 2000. IEEE.

[Mor94] Moshe Morgenstern. Existence and explicit constructions of $q+1$ regular Ramanujan graphs for every prime power q. *Journal of Combinatorial Theory. Series B*, 62(1):44–62, 1994.

[Nis92a] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Nis92b] Noam Nisan. RL $\subseteq$ SC. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 619–623, 1992.

[NSW89] Noam Nisan, Endre Szemerédi, and Avi Wigderson. Undirected connectivity in $o(log^{1.5}n)$ space. In *Proceedings of the 30th FOCS*, pages 24–29, Research Triangle Park, North Carolina, 30 October–1 November 1989. IEEE.

[NTS95] Noam Nisan and Amnon Ta-Shma. Symmetric logspace is closed under complement. *Chicago J. Theor. Comput. Sci.*, 1995.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, February 1996.

[Pin73] Mark S. Pinsker. On the complexity of a concentrator. In *7th Annual Teletraffic Conference*, pages 318/1–318/4, Stockholm, 1973.

[RR99] Ran Raz and Omer Reingold. On recycling the randomness of the states in space bounded computation. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing (STOC)*, Atlanta, GA, May 1999.

[Rei84] John H. Reif. Symmetric complementation. *Journal of the ACM*, 31(2):401–421, 1984.

[Rei05] Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, pages 376–385, 2005.

[RTV06] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L Problem. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 457–466. ACM, 2006.

[RVW00] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 3–13, Redondo Beach, CA, 17–19 October 2000. IEEE.

[RVW02] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155(1), January 2002. Extended abstract in Proc. of *FOCS '00*.

[RV05] Eyal Rozenman and Salil Vadhan. Derandomized squaring of graphs. In *Proceedings of the 8th International Workshop on Randomization and Computation (RANDOM)*, number 3624 in Lecture Notes in Computer Science, pages 436–447, Berkeley, CA, August 2005. Springer.

[Sak96] Michael Saks. Randomization and derandomization in space-bounded computation. In *IEEE 11th Annual Conference on Structure in Complexity Theory*, 1996.

[SZ99] Michael Saks and Shiyu Zhou. $\mathrm{bp_h space}(S) \subseteq \mathrm{dspace}(S^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999. 36th IEEE Symposium on the Foundations of Computer Science (Milwaukee, WI, 1995).

[Sav70] J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Tan84] Michael R. Tanner. Explicit concentrators from generalized n-gons. *SIAM Journal on Algebraic Discrete Methods*, 5(3):287–293, 1984.

[Tri05] Vladimir Trifonov. An o(log n log log n) space algorithm for undirected s,t-connectivity. In *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, 2005.

[Wig92] Avi Wigderson. The complexity of graph connectivity. In *In Proceedings of the 17th Mathematical Foundations of Computer Science*, pages 112–132, 1992.

[YBAS95] D. Peleg Y. Ben-Asher, K. Lange and A. Schuster. The complexity of reconfiguring network model. *Information and Computation*, 21(1):41–58, 1995.

A Low Level Implementation

In the proof of Lemma 3.4 we describe an algorithm $\mathcal{A}_\mathcal{T}$ that on inputs G and H computes the rotation map Rot_{G_ℓ} of $G_\ell = \mathcal{T}(G,H)$. More specifically, given $(\bar{v},\bar{a})$ in the domain of Rot_{G_ℓ} it outputs $\mathrm{Rot}_{G_\ell}(\bar{v},\bar{a})$. For completeness, we give here an implementation of $\mathcal{A}_\mathcal{T}$ which includes low level details such as an exact manner one may maintain the recursion in the definition of $\mathcal{A}_\mathcal{T}$. There are several ways of completing such details and there is nothing particularly interesting or challenging in that. That the following is a faithful log-space implementation of $\mathcal{A}_\mathcal{T}$ can be verified by (somewhat tedious) inspection.

The algorithm $\mathcal{A}_\mathcal{T}$ will first allocate the following variables:

- v - takes value in $[N]$ and specifying a vertex of G.
- $\ell+1$ variables $a_0, a_1 \ldots a_\ell$ - each taking value in $[D^{16}]$. Each specifying a vertex name of H; In addition, a_0 may specify an edge label of G. It is sometimes convenient to view each one of $a_1 \ldots, a_\ell$ as specifying a sequence of 16 edge labels of H. In this case we denote $a_i = k_{i,1} \ldots k_{i,16}$.[6]
- I - takes value in $[\ell]$ and specifying the current height in the recursion level.
- ℓ variables $j_1 \ldots j_\ell$ - each taking value in $[16]$ and together specifying the recursion path.
- $basic$ - logarithmically-long space which is sufficient to carry out the basic operations as will be defined below.

The algorithm $\mathcal{A}_\mathcal{T}$ will initialize the above variables as follows:

- Copy the input $(\bar{v},\bar{a})$ into $v, a_0, a_1 \ldots a_\ell$: $\bar{v}$ into $v, a_0, \ldots, a_{\ell-1}$ and $\bar{a}$ into a_ℓ.
- I is set to ℓ.
- $j_1 \ldots j_\ell$ are each set to one.
- $basic$ is initialized to an arbitrary default setting (e.g. all zeros).

The algorithm $\mathcal{A}_\mathcal{T}$ operates as follows:

1. Set $a_{I-1}, k_{I,j_I} \leftarrow \mathrm{Rot}_H(a_{I-1}, k_{I,j_I})$ (using the memory in $basic$).
2. If j_I is odd, and $I = 1$ set $v, a_0 \leftarrow \mathrm{Rot}_G(v, a_0)$ (using the memory in $basic$). Set $j_I \leftarrow j_I + 1$ and go to Step (1).
3. If j_I is odd and $I > 1$, set $j_{I-1} \leftarrow 1$ and $I \leftarrow I - 1$. Go to Step (1).

[6]Note that since D is a constant, decomposing a value in $[D^{16}]$ to the 16 corresponding edge labels can be done in constant time. It will be however more elegant to assume that throughout the run of the algorithm values in $[D^{16}]$ are represented as sequences in $[D]^{16}$ (this is naturally the case when D is a power of two).

4. If $j_I = 16$, reverse the order of the individual labels in a_I: Set $k_{I,1}, \ldots, k_{I,16} \leftarrow k_{I,16}, \ldots, k_{I,1}$ (using the memory in $basic$).

5. If $j_I = 16$ and $I = \ell$, output the content $v, a_0, a_1 \ldots a_\ell$ (as $\mathrm{Rot}_{G_\ell}(\bar{v}, \bar{a})$) and halt.

6. If $j_I = 16$ and $I < \ell$, set $j_{I+1} \leftarrow j_{I+1} + 1$ and $I \leftarrow I + 1$. Go to Step (1).