

$$SL \subseteq \underline{L}^{4/3}$$

Roy Armoni* Amnon Ta-Shma† Avi Wigderson‡ Shiyu Zhou§

Abstract

We present a deterministic algorithm that computes st -connectivity in undirected graphs using $O(\log^{4/3} n)$ space. This improves the previous $O(\log^{3/2} n)$ bound of Nisan, Szemerédi and Wigderson [NSW92].

*Institute of Computer Science, The Hebrew University of Jerusalem, Israel. E-mail: aroy@cs.huji.ac.il

†Weizmann Institute, Israel. E-mail: amnont@wisdom.weizmann.ac.il. The work was supported by a Phil Zacharia Postdoctoral fellowship.

‡Institute of Computer Science, The Hebrew University of Jerusalem, Israel. E-mail: avi@cs.huji.ac.il

§Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389. E-mail shiyu@cis.upenn.edu. The work was mainly done while visiting the Institute of Computer Science, The Hebrew University of Jerusalem, Israel. Part of the work was done while at Bell Laboratories, Murray Hill, New Jersey, USA.

1 Introduction

Undirected st -connectivity ($USTCON$) is a fundamental computational problem, and algorithms for it serve as basic subroutines for more complex graph problems. It is complete for the class SL of symmetric nondeterministic log-space computations [LP82], and is a subproblem of Directed st -connectivity, which captures the class NL of general nondeterministic computation. The combinatorics of $USTCON$, as well as its time complexity, are extremely well understood. However, its space complexity is still a mystery, which was a source of some beautiful discoveries in complexity theory. We adopt NC -style notations and let $L^\alpha = DSPACE((\log n)^\alpha)$.

Savitch's result [Sav70] from 1970, $NL \subseteq L^2$ implies a deterministic $(\log n)^2$ space bound for SL directly. Remarkably, since then, all progress went via probabilistic algorithms for $USTCON$ and their derandomization.

In the late 70's Cook suggested universal traversal sequences (UTS) as a basis for log-space algorithms for $USTCON$. A traversal sequence is a (deterministic) instruction for a pebble moving on the vertices of a graph, in much the same way as the random coin provides such instructions in a random walk. Such a sequence is universal if it eventually leads the pebble to visit all nodes in every connected graph (of a given size).

Alleliunas et al [AKLLR79] proved not only the *existence* of such a UTS of polynomial length, but did it via the probabilistic method, giving in particular a probabilistic log-space (RL) algorithm for $USTCON$. Thus they established $SL \subseteq RL$. Unfortunately, it did not provide deterministic space-efficient algorithms to generate such short UTS .

In a seminal paper, Nisan [Nis90] proved that a UTS can be constructed in L^2 . This construction was based on a pseudo-random generator that fools RL machines. In particular, it can be used to derandomize the above probabilistic algorithm. This hierarchical generator requires $\log n$ universal hash functions of $\log n$ bits each. While not directly improving the deterministic space bound for $USTCON$, Nisan's techniques were the basis of all subsequent progress, starting with his own paper [Nis92] proving $RL \subseteq SC$ (which in particular gives a $(\log n)^2$ -space, polynomial time algorithm for $USTCON$).

The first reduction in space for this problem was achieved by Nisan, Szemerédi and Wigderson [NSW92] who proved $SL \in L^{3/2}$. The key idea is to scale down Nisan's UTS . They use short $((\log k)^2$ bits) UTS 's from every vertex of the graph to visit large (size k) neighborhoods. Then a pairwise independent sample of the vertices (which is easily derandomized) is used to create a new graph which is much smaller (by a factor of k), but still captures the connectivity essence of the original. Iterating this process eventually leads to solving $USTCON$ on a 2-node graph. The bottleneck for improving this bound was that log-space UTS can only guarantee neighborhoods of size $\exp(\sqrt{\log n})$, implying a similar shrinking in size per iteration, which implies $\sqrt{\log n}$ iterations.

While it seems that the symmetric structure was essential for the above improvement, Saks and Zhou [SZ95] found a completely different way to obtain the stronger result $RL \subseteq L^{3/2}$. They showed that Nisan's generator can be replaced by a weaker process, an *off-line randomized algorithm*, which uses only $\sqrt{\log n}$ hash functions, that are used repeatedly in $\sqrt{\log n}$ iterations. This required a mechanism for removing the dependencies of the outcome of each iteration on the choice of hash functions, which was achieved by (easily derandomized)

perturbations and rounding.

The bound of this paper, $SL \subseteq L^{4/3}$, requires a careful combination of the ingredients of both papers above, with some new ideas. We will follow the shrinking scheme of [NSW92]. However, we replace the *UTS* of [NSW92] by a pseudo-random walk using Nisan’s generator with *short* hash functions. We show that such a walk of length $k^{O(1)}$, just like a random one, will visit (the neighbors of) $O(k)$ vertices with constant probability on every graph of any size! Note that since only $(\log k)^2$ bits are used, Nisan’s analysis does not apply, and the analysis we provide is the main technical contribution of the paper.

Next we would like to repeatedly use the same hash functions of this pseudo-random walk in many iterations, in [SZ95] style. To remove dependencies we approximate the average behavior of this set of functions (an object independent of any particular one function) with sufficient accuracy and high probability. This is achieved without space and random bit penalty via the recent extractor constructions (and their use in oblivious sampling) of [Zuc96]. This technique for removing dependencies has potential for generalization, and may become a useful derandomization tool.

The rest of the paper is organized as follows. In section 2 we give basic notation, definitions and preliminary tools. In section 3, we give the motivation and overview of our construction and state the main claims. The details of the construction and the correctness proofs will be shown in the later sections.

2 Preliminaries

Let n be a positive integer. We use $[n]$ to denote the set of integers $\{1, 2, \dots, n\}$. If M is an $n \times n$ matrix, then the rows and columns of M are indexed by $[n]$. n is called the *dimension* of the matrix M , and is denoted $\dim(M)$. We denote by $M[i, j]$ the (i, j) -th entry of M .

For an $n \times n$ matrix M over reals, we define the *norm* of M , denoted $\|M\|$, to be the maximum over all i and j of the absolute value of $M[i, j]$, i.e., $\|M\|$ is the L_∞ norm of M .

We will need the following two approximation operators, perturbation and truncation, defined in [SZ95]. Let δ be a nonnegative real number. The *perturbation operator* Σ_δ is a function mapping any nonnegative real number $z \in [0, 1]$ to $\Sigma_\delta(z) = \max\{z - \delta, 0\}$. Let t be a positive integer. The *truncation operator* $\lfloor \cdot \rfloor_t$ is a function mapping any nonnegative real number $z \in [0, 1]$ to $\lfloor z \rfloor_t$ obtained by truncating the binary expansion of z after t binary digits. Thus $\lfloor z \rfloor_t = 2^{-t} \lfloor 2^t z \rfloor$. These operators are extended to matrices by simply applying them entry by entry to the matrix.

If x is an input to an algorithm then we use $|x|$ to denote the length of x , i.e., the number of bits needed to represent x .

From now on, unless otherwise specified, a matrix is a square matrix with nonnegative real entries and an integer is a nonnegative integer.

2.1 Graphs and Matrices

We will be considering undirected graphs. Without loss of generality, we will identify a graph G with its adjacency matrix, denoted also by G , in which $G[i, j]$ is 1 if (i, j) is an edge in G and is 0 otherwise. Thus the number of vertices in G , which we call the *size* of G , is $\dim(G)$.

Let G be a graph. A *neighborhood matrix* N of G is a real matrix of dimension $\dim(G)$ such that all the entries are in the range $[0, 1]$ and if $N[i, j] \neq 0$ then vertex i is connected to vertex j in G . A *boolean neighborhood matrix* of G is a neighborhood matrix of G with boolean entries.

Let m be a positive integer and α be a nonnegative real number. Suppose M is a nonnegative real matrix. The i -th row of M is said to be (m, α) -rich if the number of entries that are bigger than α in this row is at least m . An $(m, 0)$ -rich row is also called m -rich. The matrix M is said to be (m, α) -rich (resp. m -rich) if every row of M is (m, α) -rich (resp. m -rich).

In case M is a neighborhood matrix of a graph G , then the i -th row of M is said to be (m, α) -rich if the number of entries that are bigger than α in this row is either at least m or the size of the connected component containing vertex i in G .

Remark: For the simplicity of our presentation, in the case that the i -th row of a neighborhood matrix of a graph G is m -rich, then by using m we mean the minimum of m and the size of the connected component containing vertex i in G . The corresponding arguments can be easily justified.

We have the following easy fact:

Proposition 2.1 *Let M be a nonnegative real matrix. If M is $(m, 2^{-t})$ -rich then $\lfloor M \rfloor_t$ is m -rich.*

2.2 Matrix Algorithms

In this section we review some definitions and standard facts about the computation of matrix algorithms.

A *matrix algorithm* A is an algorithm such that the inputs to A are ordered pairs (M, z) , where M is a square matrix and z is an auxiliary parameter, and in addition two indices $i, j \in [d]$ where $d = d(M, z)$; the output of A is interpreted as the (i, j) -th entry of a matrix denoted $A(M, z)$. Thus $\dim(A(M, z)) = d$. Since the entire matrix $A(M, z)$ can be obtained by running the algorithm over all the indices $i, j \in [d]$, in a sense, a matrix algorithm is a function that maps ordered pairs (M, z) to square matrices.

Understood that the computation operates in the way described, we will say that a matrix algorithm A on input (M, z) *computes* a matrix $A(M, z)$.

For typographical simplicity, we will denote a sequence $[x_1, \dots, x_p]$ by $[x_i]_p$.

Let $M_0 = M, M_1, \dots, M_p$ be a sequence of square matrices and $[z_i]_p$ be a sequence of auxiliary parameters. The following fact is well known.

Proposition 2.2 *Suppose there is a matrix algorithm A such that for any $0 < i \leq p$, A on input (M_{i-1}, z_i, i) computes M_i and runs in space $S_i \geq \log(\dim(M_{i-1}))$. Then there is a matrix algorithm F such that F on input $(M, [z_i]_p)$ computes M_p and runs in space $O(\sum_{i=1}^p S_i)$.*

We call such an algorithm F *recursive matrix algorithm* and we say that F on input $(M, [z_i]_p)$ *(recursively) computes* a sequence of matrices $[M_i]_p$.

2.3 Off-line Randomized Approximations

We consider a class of randomized algorithms called off-line randomized algorithms [SZ95]. An *off-line randomized* algorithm A is a randomized algorithm such that upon receipt of an input x , it first computes the total number $R(|x|)$ of random bits it will need for the computation and then requests a random string $y \in \{0, 1\}^{R(|x|)}$ from the random source and stores it in a designated section of memory for read-only; given x and y , the computation is then completely deterministic. We use the notation $(x; y)$ to separate the “true” input x from the “off-line random input” y .

An off-line randomized algorithm A is said to have *random bit complexity* $R(\cdot)$ and *processing space complexity* $S(\cdot)$ if on any input of length l , the algorithm requests $R(l)$ random bits and given these bits it runs in space $S(l)$.

For the following discussion, let A be an off-line randomized algorithm that has random bit complexity $R(\cdot)$ and processing space complexity $S(\cdot)$.

We say that A accepts a language with *one-sided error* if for a given input x in the language, the probability over a randomly chosen $y \in \{0, 1\}^{R(|x|)}$ that $A(x; y)$ outputs 1 is at least $1/2$; and for a given x not in the language, it always outputs 0.

There is a trivial way to derandomize an off-line randomized algorithm A with one-sided error: given an input x , we simply enumerate over all the choices of y and run $A(x, y)$, and output 1 if and only if any of the runs outputs 1. The enumeration over y clearly takes space $O(R(|x|))$ and for each y the computation of $A(x; y)$ takes processing space $O(S(|x|))$ by definition. Observing that the same processing space can be re-used for each y , the space needed for the trivial derandomization is $O(R(|x|) + S(|x|))$.

Suppose a is an integer and β is a real number. For an input pair (M, z) and a matrix N whose dimension is the same as that of $A(M, z; \cdot)$, we say that $A(M, z; y)$ approximates matrix N *with accuracy a and error probability β* if

$$\Pr[\|A(M, z; y) - N\| > 2^{-a}] \leq \beta,$$

where the probability is over a randomly chosen $y \in \{0, 1\}^{R(|M, z|)}$. That is, for all but β fraction of choices of $y \in \{0, 1\}^{R(|M, z|)}$, $A(M, z; y)$ approximates N to within 2^{-a} .

2.4 Perturbations and Truncations

We follow [SZ95] to review some basic properties of the perturbation and truncation operators. First we have the following easy facts.

Proposition 2.3 *Suppose M, N are nonnegative real matrices of the same dimension, t is a positive integer and δ is a positive real number. Then:*

1. $\|M - \Sigma_\delta(M)\| \leq \delta$
2. $\|M - \lfloor M \rfloor_t\| \leq 2^{-t}$
3. $\|\Sigma_\delta(M) - \Sigma_\delta(N)\| \leq \|M - N\|$.

A nonnegative real number r is said to be (a, t) -dangerous for positive integers $a > t$ if r can be written in the form $2^{-t}I + \rho$ where I is a positive integer and $\rho \in [-2^{-a}, 2^{-a})$, and is said to be (a, t) -safe otherwise. A matrix M is (a, t) -dangerous if any entry of M is (a, t) -dangerous and is (a, t) -safe if all of its entries are (a, t) -safe. For two positive integers $a \geq d$, we define $\Delta(a, d)$ to be the set of 2^d real numbers $\{q2^{-a} \mid \text{integer } q \in [0, 2^d - 1]\}$. (Sometime we may identify $\{0, 1\}^d$ with $[0, 2^d - 1]$.)

The proofs of the next two lemmas can be found in [SZ95] (Lemma 5.5 and Lemma 5.2, respectively).

Lemma 2.1 *Suppose M is an $n \times n$ nonnegative real matrix and $a \geq d$ are positive integers. Let $t = a - d$. Then for a randomly chosen $\delta \in \Delta(a, d)$*

$$\Pr[\Sigma_\delta(M) \text{ is } (a, t)\text{-dangerous}] \leq n^2 2^{-d+1}.$$

Lemma 2.2 *Suppose M is an (a, t) -safe nonnegative real matrix. Then for any nonnegative real matrix N such that $\|M - N\| \leq 2^{-a}$, $\lfloor M \rfloor_t = \lfloor N \rfloor_t$.*

For real x, δ and integer t , we define $\text{Per}T(x, \delta, t) = \lfloor \Sigma_\delta x \rfloor_t$. The next corollary can be easily derived from the above two lemmas.

Corollary 2.1 *Let A be an off-line randomized matrix algorithm such that given as input (M, z) , $A(M, z; y)$ uses R random bits and approximates a matrix N of dimension n with accuracy a and error probability β . Suppose $d \leq a$ is a positive integer. Then for randomly chosen $y \in \{0, 1\}^R$ and $q \in \{0, 1\}^d$,*

$$\Pr[\text{Per}T(A(M, z; y), q2^{-a}, a - d) \neq \text{Per}T(N, q2^{-a}, a - d)] \leq \beta + n^2 2^{-d+1}.$$

2.5 Oblivious Sampling

Oblivious sampling is a strong version of deterministic amplification. Given a function $f : \{0, 1\}^l \rightarrow [0, 1]$ we want to efficiently estimate the expectation over a randomly chosen $y \in \{0, 1\}^l$ of $f(y)$, i.e., to estimate

$$Ef = \frac{1}{2^l} \sum_{y \in \{0, 1\}^l} f(y).$$

Definition 2.1 [BR94, Zuc96] *An (ϵ, γ) oblivious sampler (or sampler) $A : \{0, 1\}^r \rightarrow \{0, 1\}^{p^l}$ is a deterministic algorithm that on an r -bit string input y , outputs a sequence of p sample points $A(y)[1], \dots, A(y)[p] \in \{0, 1\}^l$ subject to the property that for any function $f : \{0, 1\}^l \rightarrow [0, 1]$, for all but γ fraction of $y \in \{0, 1\}^r$, the corresponding output sequence $A(y)[\cdot]$ satisfies that*

$$\left| \frac{1}{p} \sum_{i=1}^p f(A(y)[i]) - Ef \right| \leq \epsilon.$$

It is often convenient to view such an oblivious sampler as an off-line randomized algorithm without true input: A takes an off-line random input $y \in \{0, 1\}^r$ and computes accordingly a sequence $A(y)[\cdot]$ of p sample points in $\{0, 1\}^l$ subject to the property that for any function $f : \{0, 1\}^l \rightarrow [0, 1]$,

$$\Pr\left[\left|\frac{1}{p}\sum_{i=1}^p f(A(y)[i]) - Ef\right| > \epsilon\right] \leq \gamma,$$

where the probability is over a randomly chosen $y \in \{0, 1\}^r$.

Clearly a natural way to obtain an (ϵ, γ) sampler is to randomly choose independent sample points. In fact, a simple counting argument shows that $p = O(\frac{1}{\epsilon^2} \log(\frac{1}{\gamma}))$ points are enough. However, this way we use too many random bits: $r = pl$. We would like to reduce r to about only $O(l)$ while keeping the number of sample points small with $p = \text{poly}(l, \frac{1}{\epsilon})$. Indeed, by using improved extractors, Zuckerman showed an explicit construction of such a randomness-efficient sampler ([Zuc96] Theorem 5.5). We will use a special case of this sampler:

Lemma 2.3 [Zuc96] *For any constant $\nu < 1$, any positive integer l and any ϵ such that $\log \frac{1}{\epsilon} \leq l^\nu$, there is an $(\epsilon, \frac{1}{2l})$ oblivious sampler $A : \{0, 1\}^{3l} \rightarrow \{0, 1\}^{p \cdot l}$ that runs in NC, where $p = \lfloor (2l/\epsilon)^{c_0} \rfloor$ for some absolute constant c_0 .*

This is obtained from Theorem 5.5 of [Zuc96] by setting $m = l, d = p, \alpha = \frac{1}{2}$ and $\gamma = \frac{1}{2l}$.

Remark: Here by *running in NC* we mean that given the index of any bit in the output sequence, the value of the bit is computable in time poly-logarithmic in the length of the input: $3l$ plus the length of the index which is $\log(pl) = O(\log l + \log \frac{1}{\epsilon})$. In particular, it implies that any bit in the output sequence is computable in space $\log^{O(1)}(l + \log \frac{1}{\epsilon})$.

3 Motivation and Overview

One common point of the algorithms presented in [Sav70] and [NSW92] for connectivity is that either of them can be viewed as a recursive matrix algorithm such that on an input instance (G, s, t) of undirected st -connectivity, it computes a sequence of graphs $[G_i]_r$ that satisfies the following properties which we call *recursive connectivity properties* (RCP):

1. For each $1 \leq i \leq r$, s, t are vertices in G_i and s, t are connected in G_i if and only if they are connected in G_{i-1} , where $G_0 = G$.
2. The sizes of G_i are non-increasing.
3. G_r is a transitive closure matrix.

The output of the algorithm is $G_r[s, t]$. Such an algorithm accepts the language undirected st -connectivity since if $[G_i]_r$ satisfies RCP (with respect to input (G, s, t)) then s, t are connected in G if and only if $G_r[s, t] = 1$. We say that a (recursive) matrix algorithm on input (G, s, t) *computes connectivity* if it computes a sequence of graphs that satisfies RCP.

In [Sav70], the algorithm computes the sequence of graphs $[G_i]_r$ such that $G_i = G_{i-1}^2 = G^{2^i}$. In words, vertices u, v are connected in G_i if and only if there is a path of length at most 2^i from u to v in G . We can see that the space used at every recursive level (to compute G_i given G_{i-1}) is $O(\log n)$ and r is $\log n$. So the total space required is $O(\log^2 n)$ (by Proposition 2.2).

The algorithm of [NSW92] computes a sequence of matrices $[N_i, G_i]_r$ depending on a parameter k which we call *shrinking parameter*. For any integer $k \leq n$, $[N_i, G_i]_r(k)$ satisfies the properties that for each i , N_i is a k -rich boolean neighborhood matrix of graph G_{i-1} and $\dim(G_i) \leq \dim(G_{i-1})/k$; moreover, $[G_i]_r$ satisfies RCP. That is, the algorithm “shrinks” the graph G by a factor of k at every recursive level and at the same time preserves RCP. The computation at each recursive level consists of computing two matrices N_i and G_i . N_i is obtained by taking a walk on graph G_{i-1} according to the universal traversal sequence constructed in [Nis90], which takes space $O(\log^2 k + \log n)$. The computation of G_i is based on a matrix algorithm which we denote by SRNK for *graph shrinking procedure*. This algorithm will be the basic building-block of the construction of our algorithm and we summarize its properties as follows.

Lemma 3.1 *Given as input (G, s, t, N) , where G is an undirected graph of size n , s, t are vertices in G and N is a k -rich neighborhood matrix of G , the algorithm SRNK runs in space $O(\log n)$ and computes a graph G' such that*

1. s, t are vertices in G' and s, t are connected in G' if and only if they are connected in G .
2. $\dim(G') \leq \dim(G)/k$.

G_i is computed as SRNK (G_{i-1}, s, t, N_i) .

Then we can see that the space required at every recursive level of the algorithm is $O(\log^2 k + \log n)$ and the number of recursive levels is $O(\log n / \log k)$. Thus the total space needed for the computation is $O(\log^2 n / \log k + \log n \log k)$. By choosing $\log k = \log^{1/2} n$ (as a result of minimizing the space over choices of k), the algorithm has space complexity $O(\log^{3/2} n)$.

To improve the result we adapt an idea of [SZ95]: we apply randomization in recursion and then reuse random bits at different recursive levels to achieve space savings. We replace the deterministic procedure of computing neighborhood matrices N_i in [NSW92] by an off-line randomized procedure which we call NBRs for *randomized neighborhood matrix approximations*. This procedure NBRs takes $O(\log^2 k)$ random bits but uses only $O(\log n)$ processing space, and with high probability outputs a k -rich neighborhood matrix N_i of any graph G_{i-1} . Moreover and importantly, the output matrix N_i is itself *fixed* in the sense that it (essentially) does not depend on the random bits used.

For the rest of the paper, we let $c = 960$ and $\nu = \frac{3}{4}$.

Lemma 3.2 *Given as input a graph G of size n , integers $m \leq \log n$, $a \leq (cm^2)^\nu$ and d such that $a - d > \log n$, the algorithm NBRs $(G, m, a, d; h, q)$ takes off-line random inputs $h \in \{0, 1\}^{3cm^2}$ and $q \in \{0, 1\}^d$, runs in space $O(a + \log n)$ and with probability at least $1 - n^2(2^{-3m^2} + 2^{-d+1})$ outputs a 2^{m-1} -rich neighborhood matrix N of G that does not depend on h .*

Now we apply algorithm NBRS to G_i with parameters a, d of $O(\log n)$ and shrinking parameter k , where $k = 2^{m-1}$ in the lemma, at all recursive levels but use only one single random string h of length $O(\log^2 k)$. (On the other hand, we use independent random strings q of length $d = O(\log n)$ at different recursive levels.) Since at any particular level, with high probability the output matrix is k -rich and does not depend on h , we can argue that with high probability, for a random h we obtain k -rich neighborhood matrices N_i of G_{i-1} at all levels.

Overall, the off-line randomized algorithm described, which we denote by RCon for *randomized connectivity*, takes a total of $O(\log^2 k + \log^2 n / \log k)$ random bits, uses $O(\log n)$ space at every recursive level, computes a sequence of matrices $[N_i, G_i]_r$ satisfying RCP with shrinking parameter k and has one-sided error. Applying the trivial derandomization, we can then obtain a deterministic algorithm for connectivity that runs in space $O(\log^2 k + \log^2 n / \log k)$. By minimizing the space over choices of k , we choose $\log k = \log^{2/3} n$ and the algorithm has space complexity $O(\log^{4/3} n)$.

Still it remains to construct the algorithm NBRS. To obtain such an algorithm we proceed in two stages. In the first stage of our construction, we develop a randomized approximation scheme that approximates the expectation of any off-line randomized algorithm B with sufficient accuracy and high probability at essentially the same cost as that of B ; moreover, the outcome of the approximation is (essentially) independent of the random bits used. More precisely we have the following lemma:

Lemma 3.3 *There is an off-line randomized procedure APRX (for approximations) that has the following properties: Let B be any off-line randomized matrix algorithm such that all the entries in any output matrix of B are in the range $[0, 1]$ and B has random bit complexity $R(\cdot)$ and processing space complexity $S(\cdot)$. Suppose x is an input to B such that $\dim(B(x; \cdot)) = K$ and let $a \geq d$ be positive integers. Then on input B (i.e. the program of B), x and a, d , APRX requests random strings $h \in \{0, 1\}^{3R(|x|)}$ and $q \in \{0, 1\}^d$ (thus uses $O(R(|x|) + d)$ random bits), runs in space $O(S(|x|) + a) + \log^{O(1)} R(|x|)$ and with probability at least $1 - K^2(2^{-R(|x|)} + 2^{-d+1})$ outputs exactly $\text{PerT}(E_y[B(x; y)], q2^{-a}, a - d)$, where the expectation is over randomly chosen $y \in \{0, 1\}^{R(|x|)}$.*

The construction of APRX uses the oblivious sampler of [Zuc96] (we may as well use the sampler constructed in [BGG93]) and the perturbation and truncation scheme of [SZ95]. We emphasize a few points of the procedure APRX that will be useful for the construction of our algorithm: (1) APRX has essentially the same random bit complexity and processing space complexity as that of algorithm B ; in addition, with high probability, (2) it approximates the expectation $E_y[B(x; y)]$ of algorithm B on input x , and (3) the outcome is independent of the random string h it uses.

Given the approximation scheme APRX, to obtain the desired algorithm NBRS, now it is sufficient for us to construct an off-line randomized procedure that satisfies the following somewhat relaxed properties: the random bit complexity and processing space complexity of the procedure can be of the same order as that of NBRS; its expectation is k -rich (but has “large” entries so that after applying PerT the result is still k -rich).

In the second stage of our construction, by applying Nisan’s pseudo-random generator for space-bounded computation [Nis90] to simulate short random walks on graphs, we construct

an off-line randomized algorithm, which we call WALK for *pseudo-random walks*, that gives a “weak” version of the procedure NBRS in the following sense: given any graph G_{i-1} , WALK takes $O(\log^2 k)$ random bits, runs in processing space $O(\log n)$, and computes a neighborhood matrix N_i of G_{i-1} such that for each vertex v in G_{i-1} , with high probability the v -th row of N_i is $(2k, \frac{1}{n})$ -rich.

Lemma 3.4 *Given as input a graph G of size n and an integer m , the algorithm WALK $(G, m; y)$ takes an off-line random input $y \in \{0, 1\}^{cm^2}$, runs in space $O(\log n)$ and computes a boolean neighborhood matrix N_y of G such that for each $1 \leq v \leq n$,*

$$\Pr[\text{the } v\text{-th row of } N_y \text{ is } 2^m\text{-rich}] \geq \frac{2}{3},$$

where the probability is over a randomly chosen $y \in \{0, 1\}^{cm^2}$.

The significance of this procedure WALK is that if we consider the expectation over y of $N_y = \text{WALK}(G, m; y)$:

$$\mathcal{N}(G, m) = E_y[\text{WALK}(G, m; y)] = 2^{-cm^2} \sum_{y \in \{0, 1\}^{cm^2}} \text{WALK}(G, m; y),$$

a straightforward averaging argument shows that $\mathcal{N}(G, m)$ is $(k, \frac{1}{n})$ -rich. Now applying the approximation scheme APRX to WALK we obtain the desired algorithm NBRS .

Putting all these together, we get our connectivity algorithm.

In the rest of the paper we present the details of our construction and give the correctness proofs. In Section 4, assuming that algorithm NBRS is given, we present the main algorithm RCon together with its correctness proof. In Section 5, we present the construction of the approximation scheme APRX and prove Lemma 3.3. In Section 6, assuming that algorithm WALK is given, we show the application of APRX to WALK to obtain algorithm NBRS and prove Lemma 3.2. Finally in Section 7 we give the description of algorithm WALK and prove Lemma 3.4.

4 Randomized Connectivity

In this section we present the description of algorithm RCon together with its correctness proof. As we have discussed in the previous section, this algorithm will satisfy the properties that on given input (G, s, t) where $\dim(G) = n$, it takes $O(\log^2 k + \log^2 n / \log k)$ off-line random bits, runs in processing space $O(\log n)$ per recursive level, has $O(\log n / \log k)$ levels (i.e., the shrinking parameter is k), and computes connectivity with one-sided error.

In addition to the input (G, u, v) , the algorithm RCon uses four parameters a, d, m and r : a is an accuracy parameter and is passed to algorithm NBRS to control the accuracy of the approximations. d is a perturbation parameter, $\Delta(a, d)$ gives the range of the perturbations that we apply (recall the definition of $\Delta(a, d)$ in Section 2.4). m determines the shrinking parameter which is equal to 2^{m-1} , and satisfies $a \leq (cm^2)^\nu$ which is the requirement in Lemma 3.2. r is the number of recursive levels and is thus $\log n / (m - 1)$.

Upon receipt of an input (G, u, v) , the algorithm first computes the four parameters that depend solely on the size of the input graph G . These parameters will then be used in the computation at all subsequent recursive levels. The algorithm requests two off-line random inputs $h \in \{0, 1\}^{3cm^2}$ and $\vec{q} = [q(1), \dots, q(r)] \in (\{0, 1\}^d)^r$, where h is the off-line random string that is used by algorithm NBRs at all recursive levels, and each $q(i) \in \{0, 1\}^d$ is identified with the integer in $[0, 2^d - 1]$ whose binary expansion is $q(i)$.

Algorithm RCon

Input: a graph G of size n and two vertices u, v in G ;

Initialization: compute (from the input) parameters $a, d; m$ and $r = \frac{\log n}{m-1}$;

Off-line Random Input: $h \in \{0, 1\}^{3cm^2}$ and $\vec{q} = [q(1), \dots, q(r)] \in (\{0, 1\}^d)^r$;

The algorithm computes the sequence of matrices $[G_i]_r$ defined as follows:

$$G_0 = G \text{ and for } 1 \leq i \leq r,$$

$$G_i = \text{SRNK} (G_{i-1}, u, v, N_i = \text{NBRs} (G_{i-1}, m, a, d; h, q(i)))$$

Output: $G_r[u, v]$

In words, the algorithm computes a sequence of matrices RCon $(G, u, v; h, \vec{q}) = [N_i, G_i]_r$, where N_i is obtained by applying algorithm NBRs to G_{i-1} with parameters m and a, d , and G_i is obtained by applying algorithm SRNK to (G_{i-1}, u, v, N_i) .

Let us first examine the space complexity of the algorithm.

Lemma 4.1 *The algorithm RCon has random bit complexity $O(m^2 + dr)$ and processing space complexity $O(r(a + \log n))$.*

Proof: The random bit complexity is clear from the description of the algorithm. For the processing space complexity, it suffices to show that the processing space of SRNK (G_{i-1}, u, v, N_i) , where N_i is computed as $\text{RNA}(G_{i-1}, m, a, d; h, q(i))$, is bounded by $O(a + \log n)$ for each i . Then we get that the total processing space is as desired.

First we notice that the sizes of G_i are non-increasing. Consider the computation of SRNK (G_{i-1}, u, v, N_i) , where the size of G_{i-1} is at most n . The call to NBRs takes space $O(a + \log n)$ by Lemma 3.2 (keep in mind that $h, q(i)$ are given and we count only the processing space), and the call to SRNK takes space $O(\log n)$ by Lemma 3.1. So the total processing space of SRNK (G_{i-1}, u, v, N_i) is $O(a + \log n)$ as required. \square

Recall that a recursive matrix algorithm is said to compute connectivity on input (G, s, t) if the algorithm computes a sequence of graphs satisfying RCP (with respect to the input (G, s, t)).

Lemma 4.2 *Given any input (G, u, v) where G is of size n , if $a - d$ is chosen to be at least $\log n + 1$ then*

$$\Pr[\text{RCon} (G, u, v; h, \vec{q}) \text{ computes connectivity}] \geq 1 - r n^2(2^{-3m^2} + 2^{-d+1}),$$

where the probability is over randomly chosen $h \in \{0, 1\}^{3cm^2}$ and $\vec{q} \in \{0, 1\}^{dr}$. Moreover, the algorithm has one-sided error.

Proof: We first observe the fact that if each neighborhood matrix N_i in the sequence $[N_i, G_i]_r$ computed by RCon $(G, u, v; h, \vec{q})$ is 2^{m-1} -rich, then by Lemma 3.1 $[G_i]_r$ satisfies the first two properties of RCP; moreover, since G_r is a graph with only two vertices u and v , the third property of RCP is satisfied as well. Thus to prove the lemma, it suffices to upper bound the probability over randomly chosen (h, \vec{q}) that $[N_i]_r$ is *not* 2^{m-1} -rich for all N_i by $r n^2(2^{-3m^2} + 2^{-d+1})$.

For this we need a definition. Consider an off-line random input (h, \vec{q}) , where $h \in \{0, 1\}^{3cm^2}$ and $\vec{q} = [q(1), \dots, q(r)] \in (\{0, 1\}^d)^r$. Suppose that $[N_i, G_i]_r$ is the corresponding sequence of matrices computed by RCon $(G, u, v; h, \vec{q})$. For $1 \leq l \leq r$, we denote by $\vec{q}[1, l]$ the sequence $[q(1), \dots, q(l)]$ (thus $\vec{q}[1, r] = \vec{q}$). We say that $(h, \vec{q}[1, l])$ is *good* if for each $1 \leq i \leq l$, N_i computed as NBRs $(G_{i-1}, m, a, d; h, q(i))$ is a 2^{m-1} -rich neighborhood matrix of G_{i-1} and moreover, N_i does not depend on h (in the sense as that in the statement of Lemma 3.2). Then we have:

$$\begin{aligned}
& \Pr[\text{RCon}(G, u, v; h, \vec{q}) \text{ does not compute connectivity}] \\
& \leq \Pr[[N_i]_r \text{ is not } 2^{m-1}\text{-rich for all } N_i] \\
& \leq \Pr[(h, \vec{q}) \text{ is not good}] \\
& \leq \sum_{l=1}^r \Pr[(h, \vec{q}[1, l-1]) \text{ is good but } (h, \vec{q}[1, l]) \text{ is not good}] \\
& \leq r n^2(2^{-3m^2} + 2^{-d+1}).
\end{aligned}$$

To see the last inequality, we first observe the fact that if $(h, \vec{q}[1, l-1])$ is good then G_{l-1} is independent of h . Then by using Lemma 3.2 it is not difficult to check that each term in the sum is bounded by $n^2(2^{-3m^2} + 2^{-d+1})$.

We can easily verify that algorithm WALK and subsequently algorithm NBRs never produce any non-neighborhood matrix. Thus algorithm RCon has one-sided error. \square

Now that we have Lemma 4.2, we can choose the parameters, e.g., $a = 7 \log n$, $d = 4 \log n$ (thus $a - d > \log n$); $m = \log^{2/3} n$ (note $a \leq (cm^2)^\nu$), thus $r = \Theta(\log^{1/3} n)$, to obtain an algorithm that takes $O(\log^{4/3} n)$ random bits, runs in $O(\log^{4/3} n)$ space, computes connectivity with probability $1 - \frac{1}{n}$ and has one-sided error. Finally by applying the trivial derandomization, we can obtain a deterministic algorithm for connectivity with space complexity $O(\log^{4/3} n)$. This completes the proof that undirected st -connectivity is in $DSPACE(\log^{4/3} n)$.

5 Approximation Scheme APRX

In this section we present the off-line randomized approximation scheme APRX and prove Lemma 3.3.

Fix any off-line randomized algorithm B that has random bit complexity $R(\cdot)$ and processing space complexity $S(\cdot)$. Let x be an input to B and $a \geq d$ be positive integers. Suppose that $\dim(B(x; \cdot)) = K$.

To obtain the approximation scheme APRX we first construct an off-line randomized algorithm, denoted SMPL for *sampling*, such that given as input (B, x, a) , SMPL takes a random string h of length $3R(|x|)$, runs in $O(S(|x|) + a) + \log^{O(1)} R(|x|)$ space and approximates $M = E_y[B(x; y)]$ with accuracy a and error probability $K^2 2^{-R(|x|)}$. The output of APRX is then set to be $PerT(\text{SMPL}(B, x, a; h), q2^{-a}, a - d)$, where $q \in \{0, 1\}^d$ is chosen at random.

It is clear that APRX constructed satisfies both the number of random bits and the processing space requirements. Moreover, by Corollary 2.1, APRX also satisfies the requirement for the error probability.

Thus all it remains is to construct the algorithm SMPL. In fact, for the last approximation requirement of SMPL it is sufficient to show that for any pair of indices $i, j \in [K]$,

$$Pr[|\text{SMPL}(B, x, a; h)[i, j] - M[i, j]| > 2^{-a}] \leq 2^{-R(|x|)}$$

where the probability is over a randomly chosen $h \in \{0, 1\}^{3R(|x|)}$. So let us fix two arbitrary indices $i, j \in [K]$.

For typographical simplicity, in the following discussion we let $l = R(|x|)$. Define function $f : \{0, 1\}^l \rightarrow [0, 1]$ to be such that for any $y \in \{0, 1\}^l$, $f(y) = B(x; y)[i, j]$. Thus by definition,

$$M[i, j] = \frac{1}{2^l} \sum_{y \in \{0, 1\}^l} B(x; y)[i, j] = Ef.$$

Now let $A : \{0, 1\}^{3l} \rightarrow \{0, 1\}^{p-l}$, where $p = \lfloor (2l \cdot 2^a)^{c_0} \rfloor$, be an $(\frac{1}{2^a}, \frac{1}{2^l})$ oblivious sampler given by Lemma 2.3. Define SMPL to be such that for $h \in \{0, 1\}^{3l}$,

$$\text{SMPL}(B, x, a; h) = \frac{1}{p} \sum_{k=1}^p B(x; A(h)[k]).$$

Then $\text{SMPL}(B, x, a; h)[i, j] = \frac{1}{p} \sum_{k=1}^p f(A(h)[k])$ and by Definition 2.1 of oblivious samplers, we have that for a randomly chosen $h \in \{0, 1\}^{3l}$,

$$Pr[|\text{SMPL}(B, x, a; h)[i, j] - M[i, j]| > 2^{-a}] \leq 2^{-l}.$$

Formally we have the following description of the approximation scheme.

Approximation Scheme APRX

Input: B, x, a and d ;

Off-line Random Input: $h \in \{0, 1\}^{3R(|x|)}$ and $q \in \{0, 1\}^d$;

Output: $PerT(\text{SMPL}(B, x, a; h), q2^{-a}, a - d)$

Algorithm SMPL

Input: B, x, a ;

Off-line Random Input: $h \in \{0, 1\}^{3l}$ where $l = R(|x|)$;

Let $A : \{0, 1\}^{3l} \rightarrow \{0, 1\}^{p \cdot l}$ be the $(\frac{1}{2^a}, \frac{1}{2^l})$ oblivious sampler given by Lemma 2.3, where $p = \lfloor (2l \cdot 2^a)^{c_0} \rfloor$.

Set Sum to be the $K \times K$ matrix with all 0 entries.

For each $1 \leq k \leq p$,

$Sum \leftarrow Sum + B(x; y = A(h)[k])$ where y is computed on-line.

Output: $\frac{1}{p}Sum$

Remark: One important point of the above algorithm is that y is computed in the “on-line” fashion. That is, whenever a bit of y is required by B , we run the sampler A on h to compute the bit. We do not (!) compute y once for all and store it in the memory since it takes too much space.

As we have analyzed above, the algorithm SMPL satisfies the requirements in terms of both the number of random bits used and the error probability achieved. Thus it only remains to examine the space complexity of the algorithm. Following the remark after Lemma 2.3, to compute any bit of y it takes $\log^{O(1)}(a+l)$ space. By definition, $B(x; y)$ runs in space $S(|x|)$ if y is given. Therefore the call to algorithm B in the algorithm, in which y is computed in the on-line fashion as described, takes space $O(S(|x|)) + \log^{O(1)}(R(|x|) + a)$. The enumeration, summation and computing the output take space no more than $O(a + \log p) = O(a + \log R(x))$. So the total space needed is $O(S(|x|) + a) + \log^{O(1)} R(|x|)$ as desired. \square

6 Algorithm NBRS

In this section we present the off-line randomized algorithm NBRS and prove Lemma 3.2. Algorithm NBRS is served as a subroutine in the main algorithm RCon . Recall the parameters of RCon defined in Section 4.

Algorithm NBRS

Input: a graph G of size n, m and a, d ;

Off-line Random Input: $h \in \{0, 1\}^{3cm^2}$ and $q \in \{0, 1\}^d$;

Output: APRX (WALK , $(G, m), a, d; h, q$)

That is, algorithm NBRS is obtained by applying the approximation scheme APRX to algorithm WALK .

We want to show that NBRS satisfies the requirements as stated in Lemma 3.2. From the description of algorithm NBRS , we can see that the requirement for the number of random bits used is clearly satisfied. Following Lemma 3.4 and Lemma 3.3, it is easy to check that the space requirement is satisfied as well and, for the approximation requirements, all we need to show is that $PerT(\mathcal{N}(G, m), q2^{-a}, a - d)$ is 2^{m-1} -rich (it is easy to check that the requirements for the error probability and for being independent of h are satisfied).

For this we first need the following lemma:

Lemma 6.1 *For any graph G of size n and any m , $\mathcal{N}(G, m)$ is $(2^{m-1}, \frac{1}{n})$ -rich.*

Assume the lemma is true. Then since $q(i)2^{-a} \leq 2^{-(a-d)} \leq \frac{1}{2n}$ (recall that we assumed $a - d \geq \log n + 1$), by Proposition 2.3 (1) we know that $\Sigma_{q(i)2^{-a}}\mathcal{N}(G, m)$ is $(2^{m-1}, \frac{1}{2n})$ -rich. Now Proposition 2.1 says that $PerT(\mathcal{N}(G, m), q2^{-a}, a - d) = \lfloor \Sigma_{q(i)2^{-a}}\mathcal{N}(G, m) \rfloor_t$ is 2^{m-1} -rich.

So it remains to show Lemma 6.1. Recall that

$$\mathcal{N}(G, m) = 2^{-R} \sum_{y \in \{0,1\}^R} \text{WALK}(G, m; y)$$

where R is the number of random bits requested by algorithm WALK on input (G, m) . Let $i \in [n]$ be arbitrary and let w be the number of entries of $\mathcal{N}(G, m)[i, \cdot]$ that are bigger than $\frac{1}{n}$. We want to show that $w \geq 2^{m-1}$.

By the definition of w , we know that

$$\sum_{j \in [n]} \mathcal{N}(G, m)[i, j] \leq w + (n - w)/n.$$

On the other hand,

$$\begin{aligned} \sum_{j \in [n]} \mathcal{N}(G, m)[i, j] &= 2^{-R} \sum_{y \in \{0,1\}^R} \sum_{j \in [n]} \text{WALK}(G, m; y)[i, j] \\ &\geq \frac{2}{3} 2^m, \end{aligned}$$

where the inequality follows from Lemma 3.4, because for at least $\frac{2}{3}$ of the y 's, $\text{WALK}(G, m; y)[i, \cdot]$ contains at least 2^m entries with value 1.

Thus $w \geq \frac{2}{3} 2^m - 1 \geq 2^{m-1}$ and the proof is complete. \square

7 Pseudo-random Walks

In this section we present algorithm WALK and prove Lemma 3.4. We start with the following Lemma whose proof is shown in [BF93].

Lemma 7.1 *Suppose G is a connected graph and i is an arbitrary vertex in G . Let p be an integer. Then with probability at least $3/4$, a random walk of length $l^* = l^*(p) = O(p^3)$ on G from i visits at least p distinct vertices.*

Since to proceed a random walk on a graph takes space $O(\log n)$, this lemma in effect gives a randomized algorithm that satisfies the requirements of Lemma 3.4 in terms of both the processing space and the probability that many distinct vertices are visited. However, the number of random bits it needs is $\Omega(l^*)$ which is too much for our purposes.

We will apply Nisan's pseudo-random generator [Nis90] to construct an off-line randomized algorithm that takes only $O(\log^2 p)$ random bits and "efficiently" simulates the random walk process in the sense that our algorithm runs in space $O(\log n)$ as well and with high probability, it visits as many distinct vertices as a random walk does.

First we need some preliminaries.

7.1 Preliminary Definitions and Facts

7.1.1 Finite State Machines and Sub-stochastic Matrices

In this section we review some definitions and basic facts about finite state machines and sub-stochastic matrices.

A real matrix M is said to be *stochastic* (resp. *sub-stochastic*) if all the entries of M are nonnegative and all the row sums are 1 (resp. at most 1).

A *finite state machine* Q of type (n, m) is a directed multi-graph on vertex set $\{0, 1, \dots, n\}$ such that each vertex in Q has 2^m outgoing edges that are labeled in one to one correspondence with the alphabet $\Sigma = \{0, 1\}^m$; and all 2^m edges leaving vertex 0 are self-loops. The vertex set of Q is called the set of *states* of the machine. We denote by $Q[i, j]$ the set of $\alpha \in \Sigma$ that label the edges directed from state i to state j .

It is clear that the finite state machine Q defines a Markov chain. We use Q^* to denote the *probability transition* matrix of Q deleting the 0-th row and the 0-th column. That is, Q^* is the $n \times n$ matrix such that the rows and columns of Q^* are indexed by $[n]$ and $Q^*[i, j] = 2^{-m}|Q[i, j]|$ for $i, j \in [n]$. In words, $Q^*[i, j]$ is equal to the fraction of edges leaving i that are directed to j . Clearly Q^* is sub-stochastic.

Suppose Q is a finite state machine of type (n, m) and i is a state in Q . Let $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_p)$ be a word in Σ^p . A *walk on Q from i according to α* is a process obtained by starting from state i and following in succession the edges labeled by $\alpha_1, \alpha_2, \dots, \alpha_p$; a *random walk of length p on Q from i* is a random process obtained by first choosing uniformly at random a word $\alpha \in \Sigma^p$ and then proceeding a walk on Q from i according to α . We say that a walk *visits* a state j if j belongs to the set of states in Q traversed by the walk, and we say that a walk *reaches* a state j if the last state visited by the walk is j .

We use $Q^p[i, j]$ to denote the set of all words $\alpha \in \Sigma^p$ such that the walk on Q from i according to α reaches j . We say that a word $\alpha \in \Sigma^p$ *maps* state i to state j in Q if $\alpha \in Q^p[i, j]$.

Note that Q^p can itself be viewed as a finite state machine of type (n, mp) . By definition, for any $i, j \in [n]$, $(Q^p)^*[i, j]$ is the probability that a random walk of length p on Q from i reaches j . The following fact is easily seen:

Proposition 7.1 *For any finite state machine Q of type (n, m) and any positive integer p , $(Q^p)^* = (Q^*)^p$.*

A square sub-stochastic matrix is of *type (n, m)* if it has dimension n and all of its entries are multiples of 2^{-m} . Given a sub-stochastic matrix M of type (n, m) , there is a canonical way to construct a finite state machine $Q(M)$ of type (n, m) such that $(Q(M))^* = M$, i.e., for each $i, j \in [n]$, the size of the set $Q(M)[i, j]$ is $2^m M[i, j]$:

Definition 7.1 [SZ95] *We identify each $\alpha \in \Sigma$ with the integer in $\{0, 1, \dots, 2^m - 1\}$ whose binary expansion is α . For each $i \in [n]$, define $Q(M)[i, j]$ to be the set of strings corresponding to the set of integers in the interval $[2^m \sum_{l=1}^{j-1} M[i, l], (2^m \sum_{l=1}^j M[i, l]) - 1]$ for $j \in [n]$, and define $Q(M)[i, 0]$ to be the set of strings corresponding to the set of integers in the interval $[2^m \sum_{l=1}^n M[i, l], 2^m - 1]$. Finally, define $Q(M)[0, 0] = \Sigma$ and $Q(M)[0, j] = \emptyset$ for all $j \neq 0$.*

It is easy to check that $(Q(M))^* = M$ as desired. The next proposition is shown in [SZ95] (Proposition 6.3) and is easy to verify.

Proposition 7.2 *There is an algorithm which, given as input a sub-stochastic matrix M of type (n, m) , $\alpha \in \{0, 1\}^m$, and $i \in [n]$, determines the index j such that $\alpha \in Q(M)[i, j]$ in space $O(m + \log n)$.*

7.1.2 Finite State Machines and Branching Programs

For a finite state machine Q of type (n, m) , the *branching program* of Q , denoted $BP(Q)$, is defined to be a finite state machine of type $((n + 1)^2, m)$ such that:

- $BP(Q)$ has $n + 1$ levels of vertices with $n + 1$ vertices at each level. The i -th vertex at the k -th level, where $0 \leq i, k \leq n$, is called *state* (k, i) .
- For any $0 \leq k < n$ and $0 \leq i, j \leq n$, there is an edge labeled by $\alpha \in \Sigma$ directed from state (k, i) to state $(k + 1, j)$ in $BP(Q)$ if and only if there is an edge labeled by α directed from state i to state j in Q .
- All 2^m edges leaving a state at the last level n are self-loops.

Formally we need a state 0 in $BP(Q)$: it is an isolated state with self-loops. For a state (k, i) in $BP(Q)$, we define $\rho(k, i) = i$ and we call i the *real state* (in Q) of (k, i) . If Y is a set of states in $BP(Q)$ then $\rho(Y)$ is defined to be the union over $s \in Y$ of $\{\rho(s)\}$.

The definition of $BP(Q)$ clearly gives the following:

Proposition 7.3 1. *For any integer $k \leq n$, a word $\alpha \in \Sigma^k$ maps state i to state j in Q if and only if α maps state $(0, i)$ to state (k, j) in $BP(Q)$.*

2. *For any integer $k < n$, $BP(Q)^*[(k, i), (k + 1, j)] = Q^*[i, j]$.*

In a sense, $BP(Q)$ is just a “layered” version of Q .

7.1.3 Generators and Finite State Machines

An (l, a) -generator T is a function that maps $\{0, 1\}^a$ to $(\{0, 1\}^a)^l$, i.e., a function that maps a string $x \in \{0, 1\}^a$ to a sequence of l strings $y_0, y_1, y_2, \dots, y_{l-1}$ where each $y_i \in \{0, 1\}^a$. The y_i 's are called the *output blocks* of the generator. For an integer $k \leq l$, we use $T]_k$ to denote the (k, a) -generator whose output blocks are the first k output blocks of T .

Suppose T is an (l, m) -generator. For a finite state machine Q of type (n, m) , we define a finite state machine Q_T of type (n, m) as follows: for $i, j \in [n]$, $Q_T[i, j]$ is the set of all $\alpha \in \{0, 1\}^m$ such that $T(\alpha)$ maps i to j in Q . (The fact that Q_T is a finite state machine of type (n, m) is not difficult to verify.)

Let Q be a finite state machine of type (n, m) and T be an (l, m) -generator. T is said to be ϵ -pseudo-random with respect to Q if $\|Q_T^* - (Q^l)^*\| \leq \epsilon$; and T is said to be (Q, ϵ) -good if for each $k \leq l$, $T]_k$ is ϵ -pseudo-random with respect to Q . Therefore, if T is (Q, ϵ) -good then for each pair of states $i, j \in [n]$ and for any $k \leq l$, the fraction of $\alpha \in \Sigma$ such that $T]_k(\alpha)$ maps i to j in Q is within ϵ to the probability that a random walk of length k on Q from i reaches j . The next proposition follows easily:

Proposition 7.4 *Let Q be a finite state machine of type (n, m) and T be an (l, m) -generator. If T is (Q, ϵ) -good then for each state i in Q , the walks on Q from i according to $T(\alpha)$ over $\alpha \in \Sigma$ visit every state j in Q satisfying the property that $(Q^*)^k[i, j] > \epsilon$ for some $k \leq l$.*

7.1.4 Nisan's Pseudo-random Generators

The following lemma is implicit in Nisan's work [Nis90] on constructing pseudo-random generators for space-bounded computation.

Lemma 7.2 *Let n, b, r be integers and $\epsilon > 0$. There is an explicit family $\{T^{\vec{h}} | \vec{h} \in \{0, 1\}^{2br}\}$ of $(2^r, b)$ -generators that satisfies the following properties:*

1. *For any finite state machine Q of type (n, b) , all but a fraction $\frac{n^{11}2^{3r}}{\epsilon^{2^b}}$ of the generators $T^{\vec{h}}$ in the family are (Q, ϵ) -good.*
2. *There is an algorithm which, given input \vec{h} , $\alpha \in \{0, 1\}^b$ and $i \in \{0, 1, \dots, 2^r - 1\}$, computes the i -th output block of $T^{\vec{h}}(\alpha)$ in space $O(b + r)$.*

7.1.5 More Graph Notation

Let G be a directed multi-graph of size n . (Note that undirected graphs are a subclass of directed multi-graphs and thus the following definitions apply to undirected graphs as well.) For a vertex u in G , we use $\Gamma_G(u)$ to denote the set of vertices v such that (u, v) is an edge in G . i.e., $\Gamma_G(u)$ is the set of *neighbors* of u . If X is a set of vertices in G , we use $|X|$ to denote the size of X and $\Gamma_G(X)$ to denote the union over $v \in X$ of $\Gamma_G(v)$.

The *transition* matrix of G , denoted $M(G)$, is defined to be the $n \times n$ stochastic matrix such that for any vertices i, j in G , if there are k edges directed from i to j in G then $M(G)[i, j] = k/\text{deg}(i)$, where $\text{deg}(i)$ denotes the out-degree of i . (Without loss of generality we attach self-loops to vertices without outgoing edges.) That is, $M(G)$ is the probability transition matrix of the Markov chain defined by G .

7.2 Algorithm WALK

7.2.1 The Description of the Algorithm

For a given graph G of size n , recall that $\lfloor M(G) \rfloor_b$ is a matrix of type (n, b) and thus by Definition 7.1, $Q(\lfloor M(G) \rfloor_b)$ is a finite state machine of type (n, b) .

Algorithm WALK

Input: a graph G of size n and a positive integer m ; in addition, two vertices $u, v \in [n]$;

Off-line Random Input: $\vec{h} \in \{0, 1\}^{2br}$ where $b = 120m$, $r = 4m$;

$N_{\vec{h}}[u, v] \leftarrow 0$;

For each $\alpha \in \{0, 1\}^b$

If the walk on $Q(\lfloor M(G) \rfloor_b)$ from u according to $T^{\vec{h}}(\alpha)$ visits v or some $w \in \Gamma_G(v)$

Then $N_{\vec{h}}[u, v] \leftarrow 1$;

Output: $N_{\vec{h}}[u, v]$.

In words, given as input a graph G of size n and an integer m , algorithm WALK takes an off-line random input $\vec{h} \in \{0, 1\}^{960m^2}$ and computes an $n \times n$ boolean matrix $N_{\vec{h}}$ such that for any $u, v \in [n]$, $N_{\vec{h}}[u, v] = 1$ if and only if v belongs to either the set of states $Y \subseteq [n]$ visited by the walks on $Q = Q(\lfloor M(G) \rfloor_b)$ from u according to $T^{\vec{h}}(\alpha)$ over $\alpha \in \{0, 1\}^b$ or the set of neighbors of Y in G (i.e., $v \in Y \cup \Gamma_G(Y)$).

To show the correctness of the algorithm, we first notice that $N_{\vec{h}}$ is indeed a boolean neighborhood matrix of G . This is because for any pair of states $i, j \in [n]$, if there is an edge directed from i to j in Q then, by definition, (i, j) must be an edge in G . Therefore the set of states in $[n]$ of Q visited by the walks are in the same connected component as u is in G .

Next let us see that the space requirements of Lemma 3.4 are satisfied. The length of the off-line random input is cm^2 as required. For the processing space, first we see that the enumeration of α takes space $O(m)$. To proceed the walk we need to generate the output blocks of $T^{\vec{h}}(\alpha)$ and for each block we need to determine to which state in Q the walk goes next. By Lemma 7.2 (2), each output block of $T^{\vec{h}}(\alpha)$ can be generated in space $O(m)$. Given graph G , to compute $\lfloor M(G) \rfloor_b$ clearly can be done in space $O(m + \log n)$. Thus by Proposition 7.2, to determine which state the walk goes to takes space $O(m + \log n)$. Finally to tell whether or not the current state is adjacent to v takes space no more than $O(\log n)$. So the total processing space needed is $O(m + \log n)$.

In the next subsection we prove the second part of Lemma 3.4.

7.2.2 The Correctness of Algorithm WALK

In the following discussion, we denote $p = 2^m$, $Q = Q(\lfloor M(G) \rfloor_b)$ and define $P = BP(Q)$. Notice that both Q and P are over alphabet $\{0, 1\}^b$.

For the proof, let us fix an arbitrary vertex $u \in [n]$. We want to show that for a randomly chosen $\vec{h} \in \{0, 1\}^{2br}$, with probability at least $2/3$, WALK $(G, m, u, v; \vec{h})$ outputs 1 for at least $p = 2^m$ distinct $v \in [n]$. In other words, the walks on Q from u according to $T^{\vec{h}}(\alpha)$ over $\alpha \in \{0, 1\}^b$ visit a set of states $Y \subseteq [n]$ such that Y together with its neighbors in G contain at least p distinct vertices, i.e., $|Y \cup \Gamma_G(Y)| \geq p$. Now Proposition 7.3(1) tells us that it would be sufficient for us to show that if these walks are proceeded on P from $(0, u)$, then the set of real states $Y \subseteq [n]$ visited by the walks satisfies the same property. We will work on P .

Overview of the Proof

We want to show that for any $p \leq n$, almost all strings \vec{h} of length $O(\log^2 p)$ are “good” for (G, p, u) in the sense that the walks on $P = P(G)$ from $(0, u)$ according to $T^{\vec{h}}(\cdot)$ visit at least p distinct real states (including neighbors) in $[n]$.

For the case that $p = n$, Nisan’s argument in [Nis90] on constructing universal traversal sequences essentially shows that the previous statement is indeed true. The difficulty arises in our case since we would like p to be independent of n , in particular, p could be much smaller than n .

By examining Lemma 7.2 (1) and Proposition 7.4, one can see that Nisan's generator in fact has the following property: Given a graph W of size $\text{poly}(p)$ such that, with respect to a random walk of length $\text{poly}(p)$ on W from state i , if the probability of reaching any state j in W is sufficiently large ($1/\text{poly}(p)$), then almost all strings \vec{h} of length $O(\log^2 p)$ are "good" for (W, p, i) .

In light of this observation, we proceed the proof as follows: We first construct a subgraph W of P such that, with respect to a random walk of length l^* on P starting from $(0, u)$, W consists of the state (k, i) in P with "heavy weights" in the sense that the probability of visiting (k, i) in the walk is large (at least $1/\text{poly}(p)$). (As a consequence, the size of W is small (at most $\text{poly}(p)$.) We then show that $\rho(W)$ together with its neighbors in G contain at least p distinct vertices. Finally we show that almost all the $T^{\vec{h}}$ in Nisan's generator family are "good" for W in the sense that the walks on P from $(0, u)$ according to $T^{\vec{h}}(\cdot)$ visit every state in W .

The Proof

Let us first construct the subgraph W of P as follows: W consists of $l + 1$ levels of vertices denoted W_0, W_1, \dots, W_l , where each W_i is a subset of states at the i -th level of P and $l = l(P) \leq l^*$ (l^* is as defined in Lemma 7.1). W is then defined to be the induced subgraph of P on the vertex set $\cup_{i=0}^l W_i$, which we also denote by W for simplicity.

The construction of W_0, W_1, \dots, W_l is inductive. W_0 consists of one vertex: state $(0, u)$ in P . Suppose that W_0, \dots, W_{k-1} have been defined, we show how to construct W_k and how to determine the value of l .

For this we need a definition. Given a vertex $j \in [n]$ of G , we define $R_k(j)$ to be the probability that a random walk of length k on P from $(0, u)$ reaches (k, j) and passes through *only* the states in $\cup_{i=1}^{k-1} W_i$. That is, $R_0(j)$ is 1 if $j = u$ and is 0 otherwise, and for $k \geq 1$,

$$R_k(j) = \sum_{(k-1, i) \in W_{k-1}} R_{k-1}(i) P^*[(k-1, i), (k, j)] = \sum_{(k-1, i) \in W_{k-1}} R_{k-1}(i) Q^*[i, j],$$

where the second equality follows from Proposition 7.3 (2).

Let

$$W_k = \{(k, j) \mid R_k(j) \geq \frac{1}{2l^*p}\}.$$

To determine the value of l , define

$$B_k = \{(k, j) \mid 0 < R_k(j) < \frac{1}{2l^*p}\}.$$

It is clear that W_k and B_k are disjoint sets and $W_k \cup B_k = \Gamma_P(W_{k-1})$. If either [Case 1]: $(k, 0) \in W_k$ or [Case 2]: $\sum_{(k, j) \in B_k} R_k(j) \geq \frac{1}{2l^*}$ then we let $l = k - 1$; otherwise, if $k = l^*$ then we let $l = l^*$. The construction continues until l is determined (obviously, it can proceed at most l^* steps).

Remark: We emphasize the fact that $0 \notin \rho(W) \subseteq [n]$.

The next two lemmas show that $\rho(W)$ together with its neighbors contain at least p distinct vertices in G .

Lemma 7.3 *If $l = l^*$, then $|\rho(W)| \geq p$.*

Proof: Following the construction of W , we know that for each $k \leq l$, $\sum_{(k,j) \in B_k} R_k(j) < \frac{1}{2l}$. Thus the probability that a random walk of length l on P from $(0, u)$ does *not* stay in W is at most

$$\sum_{k=1}^l \sum_{(k,j) \in B_k} R_k(j) < l \cdot \frac{1}{2l} = \frac{1}{2}.$$

Next we want to argue that, in fact, the probability that a random walk of length l on G from u that does not stay in $\rho(W)$ (i.e., does not stay in the subgraph of G induced on $\rho(W)$) is also less than $\frac{1}{2}$. Suppose this is true, then by Lemma 7.1 $\rho(W)$ must contain at least p vertices in G . This will complete the proof of the lemma.

For each $0 \leq k \leq l$ and any vertex $j \in [n]$, we define $R_k^G(j)$ to be the probability that a random walk of length k on G from u to j and passes through only the vertices in $\cup_{i=1}^{k-1} \rho(W_i)$. That is, $R_0^G(j)$ is 1 if $j = u$ and is 0 otherwise, and for $k \geq 1$,

$$R_k^G(j) = \sum_{(k-1,i) \in W_{k-1}} R_{k-1}^G(i) M(G)[i, j].$$

(Note that the only difference between $R_k^G(j)$ and $R_k(j)$ is that the former counts the probability in a random walk on G , while the latter counts the probability in a random walk on P .) We want to show that for any $j \in [n]$, $R_l^G(j) \geq R_l(j)$. This is clearly sufficient for completing the argument. We proceed by induction on k for $0 \leq k \leq l$.

The case where $k = 0$ is trivial since $R_0^G(j) = R_0(j)$ by definition. Suppose the statement holds for $k - 1$ and we show it for k .

$$\begin{aligned} R_k^G(j) &= \sum_{(k-1,i) \in W_{k-1}} R_{k-1}^G(i) M(G)[i, j] \\ &\geq \sum_{(k-1,i) \in W_{k-1}} R_{k-1}(i) \lfloor M(G) \rfloor_b [i, j] \\ &= \sum_{(k-1,i) \in W_{k-1}} R_{k-1}(i) Q^*[i, j] \\ &= R_k(j), \end{aligned}$$

where $R_{k-1}^G(i) \geq R_{k-1}(i)$ is by induction and $\lfloor M(G) \rfloor_b = Q^*$ follows from Definition 7.1. This concludes the argument. \square

Lemma 7.4 *If $l < l^*$, then $|\Gamma_G(\rho(W_l))| \geq p$.*

Proof: It suffices to show that $|\Gamma_P(W_l)| > p$. We examine the following two cases.

[Case 1]: The construction stops at W_l because $(l + 1, 0) \in W_{l+1}$, i.e., because $R_{l+1}(0) \geq \frac{1}{2l^*p}$.

Let $i \in [n]$ be an arbitrary state in Q . By Proposition 2.3 (2), we know that for any vertex $j \in \Gamma_G(i)$ (in fact, any $j \in [n]$)

$$|M(G)[i, j] - \lfloor M(G) \rfloor_b [i, j]| \leq 2^{-b}.$$

Following Definition 7.1, we can see that each state $j \in \Gamma_Q(i)$ can contribute at most 2^{-b} to the fraction of edges directed from state i to state 0 in Q , i.e., to $Q^*[i, 0]$.

Now suppose on the contrary that $|\Gamma_P(W_l)| \leq p$. This in particular implies that $|\Gamma_Q(i) \leq p|$ for any state $(l, i) \in W_l$. Thus $Q^*[i, 0] \leq p2^{-b}$ from the above analysis. Then

$$R_{l+1}(0) = \sum_{(l,i) \in W_l} R_l(i)Q^*[i, 0] \leq p2^{-b} \ll \frac{1}{2l^*p}$$

(recall that $b = 120m, p = 2^m$ and $l^* = O(p^3)$), which is a contradiction.

[Case 2]: The construction stops because $\sum_{(l+1,j) \in B_{l+1}} R_{l+1}(j) \geq \frac{1}{2l^*}$.

By the definition of B_{l+1} , we know that $\sum_{(l+1,j) \in B_{l+1}} R_{l+1}(j) < |B_{l+1}| \frac{1}{2l^*p}$. Therefore, $|B_{l+1}| > p$ and this concludes the proof of the lemma since B_{l+1} is a subset of $\Gamma_P(W_l)$. \square

Following the above two lemmas, to complete the proof it now suffices to show that for a randomly chosen $\vec{h} \in \{0, 1\}^{2br}$, with probability at least $2/3$, the walks on P from $(0, u)$ according to $T^{\vec{h}}(\alpha)$ over $\alpha \in \{0, 1\}^b$ visit every state in W .

To proceed we first notice that the size of W is at most $s = 2(l^*)^2p$: it has at most l^* levels and at each level i , W_i contains at most $2l^*p$ vertices by definition. Now we construct a finite state machine \tilde{W} of type (s, b) from W as follows: \tilde{W} is obtained by adding the state 0 to W and direct all the edges in P leaving a vertex in W but absent from W (recall W is a subgraph of P) to state 0. (If needed, we add isolated states with self-loops to make the total number of states in \tilde{W} equal to s .)

The next proposition follows immediately from the construction of W and the definition of \tilde{W} .

Proposition 7.5 *For any integer k and any $\alpha \in (\{0, 1\}^b)^k$, if a walk on \tilde{W} from $(0, u)$ according to α reaches (k, j) then the walk on W (therefore, on P) from $(0, u)$ according to α reaches (k, j) . As a consequence, for each $(k, j) \in W$, $(\tilde{W}^k)^*[(0, u), (k, j)] \geq \frac{1}{2l^*p}$.*

Now we have

Lemma 7.5 *If $T^{\vec{h}}$ is $(\tilde{W}, \frac{1}{3l^*p})$ -good, then the walks on P from $(0, u)$ according to $T^{\vec{h}}(\alpha)$ over $\alpha \in \{0, 1\}^b$ visit every state in W .*

Proof: $T^{\vec{h}}$ is clearly a $(2^r = p^4, b)$ -generator. Since $l^* = O(p^3) \leq p^4$ (for large enough p), the lemma follows easily from Propositions 7.4 and 7.5. \square

Finally, it remains to show that the fraction of $T^{\vec{h}}$ over $\vec{h} \in \{0, 1\}^{2br}$ that are $(\tilde{W}, \epsilon = \frac{1}{3l^*p})$ -good is at least $2/3$. By Lemma 7.2 (1), this fraction is at least $1 - \frac{s^{11}2^{3r}}{\epsilon^{22b}}$ which is at least $2/3$ (for large enough p). \square

Acknowledgement

The authors are grateful to Mike Saks for many helpful discussions on the subject.

References

- [AKLLR79] R. Aleliunas, R. Karp, R. Lipton, L. Lovasz and C. Rackoff, “Random walks, universal sequences and the complexity of maze problems”, *20th IEEE Symposium on Foundations of Computer Science*, 1979, pp. 218-223.
- [AKS87] M. Ajtai, J. Komlós, E. Szemerédi, “Deterministic Simulation of Logspace”, *Proc. 19th ACM Symposium on Theory of Computing*, 1987, pp. 132-140.
- [BNS89] L. Babai, N. Nisan, M. Szegedy, “Multiparty protocols and logspace-hard pseudorandom sequences”, *Proc. 21st ACM Symposium on Theory of Computing*, 1989.
- [BF93] G. Barnes and U. Feige, “Short random walks on graphs.” *Proc. 25th ACM Symposium on Theory of Computing*, pp. 728-737, 1993.
- [BGG93] M. Bellare, O. Goldreich and S. Goldwasser, “Randomness in interactive proofs.” *Computational Complexity*, 3(4):319-354, 1993.
- [BR94] M. Bellare and J. Rompel, “Randomness-Efficient Oblivious Sampling.” *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, 1994.
- [Gil77] J. Gill, “Computational complexity of probabilistic Turing machines”, *SIAM J. Computing* **6** (1977) 675-695.
- [LP82] H. Lewis and C. Papadimitiou, “Symmetric space-bounded computation”, *Theoretical Computer Science*, 19:161-187, 1982.
- [Nis90] N. Nisan, “Pseudorandom generators for space-bounded computation,” *Proc. 22nd ACM Symposium on Theory of Computing*, 1990, pp. 204-212.
- [Nis92] N. Nisan, “ $RL \subseteq SC$,” *Proc. ACM Symposium on Theory of Computing*, 1992, pp. 619-623.
- [NSW92] N. Nisan, E. Szemerédi, A. Wigderson, “Undirected Connectivity in $O(\log^{1.5}n)$ Space”, *Proc. 30th Symposium on Foundations of Computer Science*, 1992, pp. 248-253.
- [NZ93] N. Nisan and D. Zuckerman, “More Deterministic Simulation in Logspace”, *Proc. 25th ACM Symposium on Theory of Computing*, 1993, pp. 235-244.
- [Sak96] M. Saks, “Randomization and Derandomization in Space-Bounded Computation”, *11th Annual Conference on Computational Complexity*, 1996.
- [SZ95] M. Saks and S. Zhou. “ $RSPACE(S) \subseteq DSPACE(S^{3/2})$.” In *Proc. of 36th IEEE Symposium on Foundations of Computer Science*, pp. 344-353, 1995.
- [Sav70] W.J. Savitch, “Relationships between nondeterministic and deterministic space complexities.” *J. Comp. and Syst. Sci.*, 4(2):177-192, 1970.

- [Wig92] A. Wigderson. The complexity of graph connectivity. *Mathematical Foundations of Computer Science: Proceedings, 17th Symposium, Lecture Notes in Computer Science* 629:112-132, 1992.
- [Zuc96] D. Zuckerman. “Randomness-optimal sampling, extractors and constructive leader election.” In *Proc. ACM Symposium on Theory of Computing*, pp. 286-295, 1996.