\[ SL \subseteq L^{4/3} \]

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].

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 combinators 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 \( L^\alpha = \text{DSPACE}(\alpha(\log n)^{\beta}) \).

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 sequence 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 [AKL+79] 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 [Nis92] 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 \( O(\log n) \) 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 [Nis94] 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 (\( O(\log^2 k) \) 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 offline 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 gen-
operator 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 deterministic sampling of either [BGG93] or the recent extractor constructions 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 $\{1, 2, \ldots, n\}$. 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 $L_\infty$ norm of $M$.

We will need the following two approximation operators, perturbation and truncation, defined in [SZ95]. Let $\delta$ be a nonnegative real number. The perturbation operator is a function mapping any nonnegative real number $z \in [0,1]$ to $z - \delta = \max\{z - \delta, 0\}$. Let $t$ be a positive integer. The truncation operator $\lfloor \cdot \rceil_t$ is a function mapping any nonnegative real number $z \in [0,1]$ to $\lfloor z \rceil_t$ obtained by truncating the binary expansion of $z$ after $t$ binary digits. Thus $\lfloor z \rceil_t = 2^{-t} \lfloor 2^t z \rfloor$. 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 $x$.

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 $\alpha$ be a nonnegative real number. Suppose $M$ is a nonnegative real matrix. The $i$-th row of $M$ is said to be $(m, \alpha)$-rich if the number of entries that are bigger than $\alpha$ in this row is at least $m$. An $(m, 0)$-rich row is also called $m$-rich. The matrix $M$ is said to be $(m, \alpha)$-rich (resp. $m$-rich) if every row of $M$ is $(m, \alpha)$-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 $(m, \alpha)$-rich if the number of entries that are bigger than $\alpha$ 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 $\|M\|_t$ 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 $[x_1, \ldots, x_p]$ by $[x_i]_p$. 
Let $M_0 = M, M_1, \ldots, M_p$ be a sequence of square matrices and $[z_i]$ 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 $(M_{i-1}, z_i)$ 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 $(M, [z_i])$ computes $M_p$ and runs in space $O(\sum_{i=1}^p S_i)$.

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

### 2.3 Off-line Randomized Approximations

We consider a class of randomized algorithms called off-line randomized algorithms [SZ95]. They are sometimes called algorithms with 2-way random input. An off-line randomized algorithm $A$ is a randomized algorithm such that for any input $x$, $A$ computes the total number $R(|x|)$ of random bits it will need. Then upon receipt of an input $x$, it first samples a random string $y \in \{0,1\}^{R(|x|)}$ from the random source and stores it in a read-only section of memory; given $x$ and $y$, the algorithm 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 $\epsilon$ is a real number. For an input pair $(M, z)$ and a matrix $N$ whose dimension is the same as that of $A(M, z)$, we say that $A(M, z)$ approximates matrix $N$ with accuracy $a$ and error probability $\beta$ if

$$\Pr_{y \in \{0,1\}^{R(|M,z|)}}[\|A(M,z;y) - N\| > 2^{-a}] \leq \beta$$

That is, for all but $\beta$ 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 $\delta$ is a positive real number. Then:

1. $\|M - M_{-\delta}\| \leq \delta$
2. $\|M - [M]_t\| \leq 2^{-t}$
3. $\|M_{-\delta} - N_{-\delta}\| \leq \|M - N\|$. 

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 $2^{-\alpha}I + \rho$, 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 $\{2^{-a}q \mid q \in [0,2^{d-1}] \}$. 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,

$$\Pr_{\delta \in \Delta(a,d)}[M_{-\delta} \text{ is } (a,t)\text{-dangerous}] \leq n^2 2^{-d+1}$$

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

For real $x, \delta$, and integer $t$, we define $PerT(x, \delta, t) = \lfloor x - \delta \rfloor / t$. 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 $(M, z)$, $A(M, z)$ uses $R$ random bits and approximates a matrix $N$ of dimension $n$ with accuracy $a$ and error probability $\beta$. Suppose $d \leq a$ is a positive integer. Then for randomly chosen $y \in \{0,1\}^R$ and $q \in \{0,1\}^d$, 

$$\Pr[PerT(A(M,z),q2^{-\alpha},a-d) \neq PerT(N,q2^{-\alpha},a-d)] \leq \beta + n^2 2^{-d+1}$$
2.5 Deterministic Sampling

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

\[
E[f] = \mathbb{E}_{x \in \{0,1\}^l} |f(x)| = \frac{1}{2^l} \sum_{x \in \{0,1\}^l} f(x)
\]

to a given accuracy, minimizing the number of random bits used. Two optimal methods are known, [BG93] and [Zuc96].

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

\[
\left| \mathbb{E}_y \left[ \frac{1}{p} \sum_{i=1}^{p} f(A(y)[i]) - E[f] \right] \right| \leq \epsilon
\]

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 \( A(y)[i] \) of \( p \) sample points in \( \{0,1\}^l \) subject to the property that for any function \( f : \{0,1\}^l \rightarrow [0,1] \),

\[
\Pr_{y \in \{0,1\}^r} \left( \left| \frac{1}{p} \sum_{i=1}^{p} f(A(y)[i]) - E[f] \right| > \epsilon \right) \leq \gamma
\]

Clearly a natural way to obtain an \((\epsilon, \gamma)\) 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 \( \epsilon \) such that \( \epsilon \leq \nu \), there is an \((\epsilon, \frac{1}{2})\) oblivious sampler \( A : \{0,1\}^{l^3} \rightarrow \{0,1\}^{pl} \) that runs in \( NC \), where \( p = \lceil (2l/\epsilon)^{\nu} \rceil \) for some absolute constant \( c_0 \).

This is obtained from [Zuc96, Theorem 5.5] by setting \( m = l, d = p, \alpha = \frac{1}{2} \) and \( \gamma = \frac{3}{4} \).

**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 they both 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 \( G_{i-1} \), where \( G_0 = G \).
2. The sizes of \( G_i \) are non-increasing.
3. \( G_r \) is a transitive closure graph (union of cliques).

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 \( G_i = G_{i-1}^2 \)). 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 \( r = \log n \). 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 [Nis92], 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. $\text{dim}(G') \leq \text{dim}(G)/k$.

$G_i$ 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 $\log k = \log^{1/2} n$ 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 $\nu = \frac{3}{4}$.

**Lemma 3.2** Given as input a graph $G$ of size $n$, integers $m \leq \log n$, $a = O(m^2)$ and $d$ such that $a - d > \log n$, the algorithm NBRs ($G, m, a, d; h, q$) takes off-line random inputs $h \in \{0, 1\}^{O(m^2)}$ 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 $2^{-m} \geq$-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 order $O(\log n)$ and neighborhood size $k = 2^{m-1}$. We then use the $k$-rich neighborhood matrix in algorithm SRNK. The crucial point is that we use the same string $h$ (of long length $O(\log^2 k)$) as the random off-line input at all different recursive levels. 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 $[N_i, G_i]$, 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 $\text{dim}(B(x; \cdot)) = K$ 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 $O(S(|x| + a) + O(1)) R(|x|)$ and with probability at least $1 - K^2 (2^{-R(|x|) - 2^{-d+1}})$ outputs exactly $\Pr[T(x_{\{h\}})] B(x_{\{h\}})$ of algorithm $B$ on input $x$, and (3) the outcome is independent of the random string $h$ it uses.

In the second stage of our construction, by applying Nisan’s pseudo-random generator for space-bounded computation [Nis92] 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 $(2k, \frac{1}{4})$-rich.

**Lemma 3.4** Given as input a graph $G$ of size $n$ and an integer $m$, the algorithm WALK ($G, m; h$) takes an off-line random input $h \in \{0, 1\}^{O(m^2)}$, runs in space $O(\log n)$ and computes a boolean neighborhood matrix $N_h$ of $G$ such that for each $1 \leq v \leq n$,$$
\Pr[h \in \{0, 1\}^{O(m^2)} \mid \text{the } v\text{-th row of } N_h \text{ is } 2^m \text{-rich}] \geq \frac{2}{3}
$$

The significance of this procedure WALK is that if we consider the expectation over $h$ $N(G, m) = E_h[N_h] = E_{h \in \{0, 1\}^{O(m^2)}}[\text{WALK}(G, m; h)]$
then, a straightforward averaging argument shows that \( N(G, m) \) is \((k, \frac{1}{2})\)-rich. Now applying the approximation scheme APRX to WALK we obtain the desired algorithm NBRs.

Notice that the APRX algorithm allows us to relax the requirements from algorithm WALK; instead of requiring that most \( h \)'s are "good" for every vertex \( v \), it is sufficient to have that for every vertex \( v \), most \( h \)'s are "good" (for different \( v \)'s, different \( h \)'s are "good"). Moreover, the matrix \( N(G, m) \) does not depend on \( h \) since it is an average on \( h \) of \( N_h \) and algorithm APRX can compute \( N(G, m) \) with almost no additional cost. 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 a given input \((G, s, t)\) where \( \text{dim}(G) = n \), 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 \( k = 2^m - 1 \), and satisfies \( a = O((m^2)^r) \) 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 \( h \in [0,1]^{O(m^2)} \) and \( \vec{q} = [q(1), \ldots, q(r)] \in \{(0,1)^d\}^r \), 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 \( r = \log n / m - 1 \);

Off-line Random Input: \( h \in [0,1]^{O(m^2)} \) and \( \vec{q} = [q(1), \ldots, q(r)] \in \{(0,1)^d\}^r \);

The algorithm computes the sequence of matrices \([G_i]_r\) defined as follows: \( G_0 = G \), and for \( 1 \leq i \leq r \), \( G_i = \text{SRNK}(G_{i-1}, u, v, N_i) \) where \( N_i = \text{NBRs}(G_{i-1}, m, a, d; h, q(i)) \)

Output: \([G_r]_{u, v}\)

In words, the algorithm computes a sequence of matrices \( \text{RCON}(G, u, v; h, \vec{q}) = [N_i, G_i]_r \), 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 NBRs \((G_{i-1}, m, a, d; h, q(i))\), 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

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

where the probability is over randomly chosen \( h \in \{0,1\}^{O(m^2)} \) 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
$RCON (G, u, v; h, q)$ is $2^{m-1}$-rich, then by Lemma 3.1 \cite{G}, 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.

Recall that $q = [q(1), \ldots, q(r)] \in \{0, 1\}^{dr}$. Thus to prove the lemma, it suffices to upper bound for every $i$ the probability over randomly chosen $(h, q(i))$ that $N_i$ computed as NBRS ($G_{1-i, m, a, d; h, q(i)}$), is either not $2^{m-1}$-rich or it depends on $h$ (in the sense as that in the statement of Lemma 3.2). Thus,

$$\Pr \left[ RCON (G, u, v; h, q) \text{ does not compute connectivity} \right] \leq \sum_{i=1}^{r} \Pr \left[ N_i \text{ depends on } h \text{ or is not } 2^{m-1}\text{-rich :} \right. \left. N_{i-1} \text{ does not depend on } h \right] \leq r n^2 (2^{-3m^2} + 2^{-d+1})$$

To see the last inequality, we first observe the fact that $N_1, \ldots, N_{i-1}$ are all independent of the fact that if $N_1, \ldots, N_{i-1}$ are all independent of $h$, then so is $G_{1-i}$. 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.

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 $a = O((m^2)^{c})$), thus $r = O(\log^{1/3} n)$, to obtain an algorithm that takes $O(\log^{1/3} n)$ 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 $M = \mathbf{E}_a [B(x;y)]$ with accuracy $a$ and error probability $K^2 2^{-R(|x|)}$. The output of APRX is then set to be $Pr_{h \in \{0, 1\}^{3R(|x|)}} \left[ |SMPL (B, x, a; h)| - M \right] > 2^{-a} \leq 2^{-R(|x|)}$

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 $f : \{0, 1\}^l \to \{0, 1\}$ to be such that for any $y \in \{0, 1\}^l$, $f(y) = B(x; y)[i, j]$. Thus by definition,

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

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

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

$Pr_{h \in \{0, 1\}^{3l}} \left[ |SMPL (B, x, a; h)[i, j] - M[i, j]| > 2^{-a} \right] \leq 2^{-l}$

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:** $Pr_{h \in \{0, 1\}^{3l}} \left[ |SMPL (B, x, a; h)| - M \right] > 2^{-a}$

**Algorithm SMPL.**

**Input:** $B, x, a$.

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

Let $A : \{0, 1\}^{3l} \to \{0, 1\}^{p \cdot l}$ be the $(\frac{1}{p}, \frac{1}{p})$ oblivious sampler given by Lemma 2.3, where $p = [(2l \cdot 2^a)^{c}]$.

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 $O(S(|x|))$ 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. 

\[ \text{Lemma 6.1} \]

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

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

So it remains to show Lemma 6.1. Recall that

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

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,

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

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

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

\[ \text{Lemma 7.1} \]

Suppose $G$ is a connected graph and $s$ is an arbitrary vertex in $G$. Let $k$ be an integer. Then with probability at least $3/4$, a random walk of length $l = l(k) = O(k^3)$ on $G$ from $s$ visits at least $k$ distinct vertices or the whole connected component of $s$.

We could equally use a slightly weaker Lemma with $l = O(k^4)$ of [Lin92].

For the rest of this section, let us fix the graph $G$, the starting vertex (the row of the output matrix) $s$, the parameter $m$, the number of vertices that we should visit $k = 2^m$, and the length $l = O(k^3)$ given by Lemma 7.1. We will analyze random walks on $G$, that start from $s$. Let us start with the intuition behind the formal definitions. We wish to track random walks, starting at $s$, as long as they pass only through vertices with non-negligible reaching probability. $W_t$ will be the vertices with “high” reaching probability in the $t$th step, $B_t$ are the vertices with “low” reaching probability and $R_t(v)$
is the probability of reaching $v$ in the $t$th step when walking only through vertices in $W_j$. To achieve that, we will make sure that it would be possible to reach $W_i$ or $B_k$ only from $W_{l-1}$. $q$ will be the number of steps that we want to analyze. It is at most $l$, but if at time $t < l$ the probability of entering $B_{l+1}$ is too large, or, we discover a vertex with at least $k$ neighbors, then we stop. Let us give now the formal definitions. Let $P_{v,w}$ be the probability of a random step to move from $v$ to $w$, i.e., if $(v,w)$ is an edge of $G$ then $P_{v,w}$ is $\frac{1}{deg(v)}$ and 0 otherwise. Now, define $R_0(s) = 1$ and $\forall v \neq s$, $R_0(v) = 0$ and for every $t \geq 0$ define

$$W_t = \{v : R_t(v) \geq \frac{1}{2l^k}\}$$

$$B_t = \{v : 0 < R_t(v) < \frac{1}{2l^k}\}$$

$$R_{t+1}(v) = \sum_{w \in W_t} R_t(w) \cdot P_{w,v}$$

Denote by $\Gamma(v) = \{w : (u,w) \text{ is an edge of $G$}\}$ the set of neighbors of $v$ and for a subset $A$ of vertices, let $\Gamma(A)$ denote the set of neighbors of $A$. Define

$$q = \min\{|\{t \cup \{v \in W_t, |\Gamma(v)| \geq k\}| \cup \{t : \sum_{v \in B_{l+1}} R_{t+1}(v) > \frac{1}{2l}\}\}$$

Denote by $W = \bigcup_{t=0}^\infty W_t$ and get the following Lemma:

**Lemma 7.2** $|\Gamma(W)| \geq k$

**Proof:** If $q = l$, the probability that a random walk leaves $W$ is at most $l \cdot \frac{1}{2l} = \frac{1}{2}$ and the probability that it doesn’t visit $k$ distinct vertices is at most $\frac{1}{4}$. Thus, with probability at least $\frac{1}{4}$, the random walk both stays in $W$ and visits $k$ distinct vertices, thus $|W| \geq k$. In this case $W \subseteq \Gamma(W)$ we conclude that $|\Gamma(W)| \geq k$.

If $q < l$, then there are two possible cases: (1) for some $v \in W_q$, $|\Gamma(v)| \geq k$, or (2) $\sum_{v \in B_{q+1}} R_{q+1}(v) > \frac{1}{2l}$ but for every $v \in B_{q+1}$, $R_{q+1}(v) \leq \frac{1}{2l^k}$, thus the size of $B_{q+1}$ is at least $k$. Moreover, $B_{q+1} \subseteq \Gamma(W_q) \subseteq \Gamma(W)$. \square

At this point, we wish to show that instead of taking a random walk of length $l$ on $G$ from $s$, we can take a pseudo-random walk using pseudo-random bits from the generator of Nisan [Nis92], with log $l$ hash functions, each of size $O(\log k)$. Unfortunately, such a generator is only known to fool automata of size polynomial in $k$. To this end, we will construct an automaton $A$ of such a size.

The set of states of the automaton $A$ is

$$S = \{\{t,u\} : 0 \leq t \leq k, u \in W_t \cup \{0\}\}$$

It is easy to see that $|S| = O(l^2 k) = O(k^7) = O(2^{7m})$. To define the alphabet of this automaton, we need the following Lemma proved by Nisan [Nis92, Lemma 2]:

**Lemma 7.3** For every $|S| = O(2^{7m})$ and $l = O(2^m)$ there exists $p = O(m)$ and a pseudo-random generator $F : \{0,1\}^p \times (\{0,1\}2^p)^{\log l} \to (\{0,1\}^p)^l$ such that for every automaton $Q$ of size $|S|$, for all but $k^{-5}$ of the choices of $h \in (\{0,1\}^{2^p})^{\log l}$, the distance between the distribution over the states of $Q(c)$ where $c \in (\{0,1\}^p)^l$ is random and $Q(F(x,h))$ where $x \in \{0,1\}^p$ is random, is at most $k^{-5}$.

We use the parameter $p$ of the generator ($p = O(m) = O(\log k)$) to define the alphabet of our automaton as $\{0,1\}^p$. Now, it’s time to define the transition function of the automaton. For every $t$, $0 \leq t < q$, and for every $v \in W_t$, we look at input strings $b \in \{0,1\}^p$. Say that a random walk from $v$ that takes its decision according to $b$ takes us to a vertex $w$. If $w$ is in $W_{t+1}$, then the transition function of $(t,v)$ and $b$ is $(t+1,w)$; otherwise, it is $(t+1,0)$ (note that since the degree of every vertex in $W_t$ is at most $2^p$, we do not face a situation in which we need more than $p$ bits to take a step from $v$). For all other states in $S$, the transition function on every input is a self loop. The starting state of our automaton is of course $(0, s)$.

Now, one should note that by the construction of the automaton $A$, for every string $b \in (\{0,1\}^p)^l$ where $t < q$, if $b$ takes $A$ to some state $(t,v)$ with $v \neq 0$, then a walk on $G$ according to $b$ will end up at vertex $v$ as well. Moreover, for every $t \leq q$ and for every $v \in W_t$, the probability over $b \in (\{0,1\}^p)^l$ that a walk on $A$ according to $b$ reaches $(t,v)$ is at least $R_t(v) - k^{-5}$.

Consider now the pseudo-random generator $F$ of Lemma 7.3. For all but $k^{-5}$ of the choices of $b \in (\{0,1\}^{2^p})^{\log l}$, for every $t \leq q$ and for every $v \in W_t$,

$$\Pr_{x \in \{0,1\}^p} [A(F(x,h)) \text{ passes through } (t,v)] \geq R_t(v) - 2k^{-5}$$

Since $R_t(v) \geq \Omega(k^{-4})$, it follows that for at least one of the $x \in \{0,1\}^p$, the walk according to the output of $F$ in $G$ will pass through the vertex $v$.

Having all this knowledge at our disposal, we can define the algorithm WALK and finish the proof of Lemma 3.4.

**Algorithm WALK**

**Input:** A graph $G$ of $n$ vertices and a parameter $m$; two vertices $s, u$.

**Initialization:** Set $k = 2^m$. Compute the parameters $l = O(k^3)$ and $p = O(m)$.

**Off-line Random Input:** $h \in (\{0,1\}^{2p})^{\log l}$.

**Output:** Try every $x \in \{0,1\}^p$. If for one of the $x$’s, a walk on $G$ according to $F(x,h)$ from $s$ passes through $u$ or through a neighbor of $v$ of $u$, output 1. Otherwise, output 0.

Notice that the length of the off-line random input is $2p \cdot \log l = O(m^2)$. As for the space complexity, $p = O(m)$ and since $m \leq \log n$, the space needed to store an $x \in \{0,1\}^p$ or an output block of $F$ is certainly $O(\log n)$.
The space complexity of computing a block of $F(x,h)$ is also $O(\log n)$ and the space required to walk on a graph is of course $O(\log n)$ as well.

The last claim of Lemma 3.4 is that for every $s$, for at least $\frac{3}{4}$ of the choices of $h$, the algorithm will reply 1 for at least $k$ different $u$'s or for the whole connected component of $s$ if it’s smaller than $k$. If the connected component of $s$ is indeed smaller than $k$, then the walks according to the output of $F$ will indeed traverse it completely for almost every $h$ since $F$ is a pseudo-random generator for automata of such size. In the other case where the connected component of $s$ is at least of size $k$, then we know that for all but $k^{-5}$ of the possible $h$'s, the probability over the $x$'s that a walk on $G$ according to $F(x,h)$ passes through a vertex $v$ in $W$ is positive. Thus, for every vertex $v \in W$, the algorithm WALK will traverse $v$. But, by Lemma 7.2, $|\Gamma(W)| \geq k$, and WALK outputs 1 for every $u$ that it either traverse or a neighbor of a traversed vertex $v$, thus WALK will output 1 for every vertex of $\Gamma(W)$. This concludes the proof of Lemma 3.4.

Acknowledgment

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

References


