

$$BP_HSPACE(S) \subseteq DSPACE(S^{3/2})^*$$

Michael Saks[†] Shiyu Zhou[‡]

Abstract

We prove that any language that can be recognized by a randomized algorithm (with possibly two-sided error) that runs in space $O(S)$ and always terminates can be recognized by a deterministic algorithm running in space $O(S^{3/2})$. This improves the best previously known result that such algorithms have deterministic space $O(S^2)$ simulations which, for one-sided error algorithms, follows from Savitch's Theorem and for two-sided error algorithms follows by reduction to recursive matrix powering. Our result includes as a special case the result due to Nisan, Szemerédi and Wigderson that undirected connectivity can be computed in space $O(\log^{3/2} n)$. It is obtained via a new algorithm for repeated squaring of a matrix: we show how to approximate the 2^r -th power of a $d \times d$ matrix in space $O(r^{1/2} \log d)$, improving on the bound of $O(r \log d)$ that comes from the natural recursive algorithm. The algorithm employs Nisan's pseudorandom generator for space bounded computation, together with some new techniques for reducing the number of random bits needed by an algorithm.

*This work was supported in part by NSF grant CCR-9215293 and by DIMACS (Center for Discrete Mathematics & Theoretical Computer Science), through NSF grant NSF-STC91-19999 and by the New Jersey Commission on Science and Technology. A preliminary version of this paper appeared in the proceedings of the 36th IEEE Symposium on Foundations of Computer Science.

[†]Department of Mathematics, Rutgers University, New Brunswick, NJ 08854, USA. E-mail: saks@math.rutgers.edu.

[‡]Department of Computer Science, Rutgers University, New Brunswick, NJ 08854, USA. E-mail: szhou@cs.rutgers.edu.

1 Introduction

For a nonnegative integer r , the *repeated squaring* function $\Lambda^{(r)}$ is defined on square matrices M by $\Lambda^{(0)}(M) = M$ and $\Lambda^{(i)}(M) = (\Lambda^{(i-1)}(M))^2$, i.e., $\Lambda^{(r)}(M) = M^{2^r}$. A *stochastic* (resp. *substochastic*) matrix is a matrix with all entries nonnegative and all row sums equal to (resp. at most) 1. In this paper we investigate the problem of approximating $\Lambda^{(r)}(M)$ for substochastic matrices by space-bounded deterministic algorithms. Given as input a $d \times d$ substochastic matrix M , integers 2^r , 2^a in unary and indices i, j , such an algorithm estimates the (i, j) -th entry of $\Lambda^{(r)}(M)$ with accuracy 2^{-a} and runs in space polylogarithmic in the length of the input. For the simplicity of our presentation, we will measure the space complexity of the algorithm in terms of r and a parameter $s = s(M, r, a) = \max\{r, a, \log d\}$.

The most space efficient algorithm for this problem that was previously known is the straightforward recursive algorithm, which uses space $O(s)$ for each of r recursive levels for a total space of $O(rs)$. In this work we present a deterministic algorithm that uses only $O(r^{1/2}s)$ space. Our algorithm has $r^{1/2}$ recursive levels of $O(s)$ space each, and additional overhead of $O(r^{1/2}s)$ space.

It is well known (see e.g. [Gil77], [BCP83]) that approximating substochastic matrix repeated squaring is closely related to the problem of derandomizing space-bounded randomized algorithms. In this section we give a review of the connection between these two problems and summarize the consequences of our new result. For a general retrospective on the problems and developments in the related subjects of randomized space computation, we refer the reader to the survey by Saks [Sak96].

A *randomized space* $S(n)$ machine is a nondeterministic Turing machine that runs in space $S(n)$ on any input of length n , and has two nondeterministic choices at any stage of the computation depending on an unbiased coin-flip. By standardizing the model, we may assume that the machine has a read-only input tape, one work tape and a one-way output tape. Such a machine is said to be *halting* if for any input and any sequence of coin-flips, the computation terminates. $R_HSPACE(S(n))$ (resp. $BP_HSPACE(S(n))$) is the class of languages L for which there is a halting randomized space $S(n)$ machine such that for each input $x \in L$, the machine accepts x with probability at least $1/2$ (resp. at least $2/3$), and for any input $x \notin L$, the machine rejects with probability 1 (resp. at least $2/3$). Clearly, $R_HSPACE(S(n)) \subseteq BP_HSPACE(S(n))$.

Given a halting randomized space $S(n)$ machine T and input x , we may visualize the computation of T on x as a *state transition graph* $Q = Q(T, x)$ defined as follows: Q is a directed graph whose vertex set is the set of configurations in the computation, where each configuration consists of the instantaneous content of the work-tape, the heads' positions, and the value of the finite state control. Each vertex has two outgoing edges labelled by either 0 or 1. There is an edge directed from vertex i to vertex j with label b if and only if starting at configuration i , the computation goes to configuration j if the coin-flip of this step turns out to be b . We assume without loss of generality that there are exactly two halting configurations in the computation, one "ACCEPT" and one "REJECT", and the outgoing edges from these vertices are self-loops.

It is a standard fact that the total number of the configurations in such a computation is $2^{\Theta(S(|x|))}$ (provided $S(n) \geq \log n$), which we denote by d . Since the computation is guaranteed

to terminate, the configuration transitions in Q are acyclic except for the halting configurations. Therefore any execution of the computation must reach a halting state within d steps.

Clearly, the state transition graph Q defines a Markov chain, and the accepting probability of the computation is the probability that it reaches ACCEPT in a d step random walk on the chain starting from the initial configuration, denoted INITIAL. We now associate to the computation the *state transition matrix* $M = M(T, x)$ of the Markov chain. Such a matrix has rows and columns indexed by the set of configurations in the computation, whose (i, j) -th entry is the transition probability that the computation goes from configuration i to configuration j in one step. We can see then the accepting probability of the computation is just the (INITIAL, ACCEPT)-entry of M^d . In fact, since the halting configurations are assumed to be the absorbing states in the chain (with self-loops), this entry is the same for M^k where k is any integer greater than or equal to d .

Now recall that in the definition of $BP_HSPACE(S(n))$, there is a non-trivial gap between the accepting probability and the rejecting probability in the computation. Thus to tell whether or not the computation accepts, it would suffice to approximate the (INITIAL, ACCEPT)-entry of M^k for any $k \geq d$ with sufficient accuracy. We will approximate M^k for k an integer power of 2. In particular, in the case that we take $r = a = \lceil \log d \rceil$, our algorithm runs in space $O(\log^{3/2} d)$ and approximates $M^{2^{\lceil \log d \rceil}}$ with accuracy $2^{-\lceil \log d \rceil}$. As an immediate consequence, we have shown:

Theorem 1.1 *Let $S(n)$ be a proper complexity function¹ such that $S(n) \geq \log n$. Then*

$$BP_HSPACE(S(n)) \subseteq DSPACE(S(n)^{3/2}).$$

Previously, the best general result of this form was

$$BP_HSPACE(S(n)) \subseteq DSPACE(S(n)^2),$$

which can be deduced from the recursive matrix repeated squaring approximation algorithm. In fact, the weaker containment for $R_HSPACE(S(n))$ also follows from Savitch's theorem [Sav70] $NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$. (Independently, Borodin, Cook and Pippenger [BCP83] and Jung [Jun81] (see also [AO94]) showed that the same $DSPACE(S(n)^2)$ bound can be achieved in Gill's [Gil77] stronger model of randomized space computation in which cyclic configuration transitions are allowed. This model, in particular, contains $NSPACE(S(n))$. Their result, which requires the performance of exact rather than approximate matrix computations, is not generalized by Theorem 1.1.) In recent years, despite considerable progress on deterministic simulations of small space randomized algorithms [AKS87, BNS89, Nis90, Nis92, NSW92, NZ93] there has been no general improvement on this space bound. The central result in the area is Nisan's marvelous pseudorandom generator for space-bounded computation [Nis90], which he used to show that RL (i.e., $R_HSPACE(\log n)$) can be simulated by a deterministic algorithm that is simultaneously in polynomial time and $O(\log^2 n)$ space. Using Nisan's generator, Nisan, Szemerédi, Wigderson [NSW92] showed

¹The notion of proper complexity function is formally defined in [Pap94] and includes most "reasonable" functions, including polynomials, polylogarithmic functions, exponential functions, etc.

that undirected (s, t) connectivity, which is in RL [AKLLR79] and is complete for the complexity class SL [LP82], can be computed in $DSPACE(\log^{3/2} n)$. It was this result that motivated our research that we present in this work.

As with these other results, Nisan’s pseudorandom generator is a major component of our simulation. One key observation that enables us to apply Nisan’s generator to approximating matrix repeated squaring is the exposition of a canonical connection between substochastic matrices and finite state machines by which the randomized space-bounded computation can be modeled. We then develop from Nisan’s generator construction a randomized approximation algorithm for matrix repeated squaring, which we call PRS for pseudorandom repeated squaring, and apply it in a recursive fashion. A major technical contribution of our work is then to apply the idea of random perturbation to overcoming the stochastic dependence among different recursive levels.

The rest of this paper is organized as follows. In Section 2 we review some elementary concepts in matrix approximation and space-bounded matrix computation, and examine some of the basic properties. We formalize the approximation problem and state the main result in Section 3. Then assuming the algorithm PRS is given, we present our main approximation algorithm in Section 4 and give its correctness proof in Section 5. In Section 6, we examine in detail the connection between approximate matrix repeated squaring and Nisan’s pseudorandom generator construction and describe the algorithm PRS . Finally in Section 7, using the algorithm we develop for approximate repeated squaring, we present a deterministic algorithm for approximating an arbitrary integer power of a substochastic matrix in small space: the algorithm approximates the p -th power of a $d \times d$ substochastic matrix and runs in space $O(\log d \log^{1/2} p)$.

2 Preliminary Definitions and Facts

2.1 Matrix Approximations

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

We will be dealing almost exclusively with substochastic matrices whose entries are represented in binary.

For a vector $x \in \mathbb{R}^d$, we define $\|x\| = \sum_i |x_i|$, i.e., $\|x\|$ is the L_1 -norm of x . For a $d \times d$ matrix M over \mathbb{R} , we define the *norm* of M , $\|M\|$, to be the maximum over all rows $M[i, \cdot]$ of $\|M[i, \cdot]\|$. An equivalent definition is $\|M\| = \sup\{\|xM\| \mid x \in \mathbb{R}^d \text{ such that } \|x\| = 1\}$.

If M and N are square matrices of the same dimension and a is a positive real number, we say that M *approximates* N with accuracy a if $\|M - N\| \leq 2^{-a}$.

We collect some basic facts about the matrix norm. These facts are standard, and are given here without proof.

Proposition 2.1 *Let $M, N \in \mathbb{R}^{d \times d}$. Then:*

1. $\|M + N\| \leq \|M\| + \|N\|$ (the matrix norm is subadditive),

2. $\|MN\| \leq \|M\|\|N\|$ (the matrix norm is submultiplicative).

Now we have the following proposition.

Proposition 2.2 *Let M, N, M', N' be $d \times d$ substochastic matrices. Then $\|MN - M'N'\| \leq \|M - M'\| + \|N - N'\|$.*

Proof:

$$\begin{aligned} \|MN - M'N'\| &\leq \|MN - M'N\| + \|M'N - M'N'\| \\ &\leq \|M - M'\|\|N\| + \|M'\|\|N - N'\| \\ &\leq \|M - M'\| + \|N - N'\|, \end{aligned}$$

where the first inequality uses the subadditivity of the matrix norm, the second uses the submultiplicativity, and the last follows from the fact that N and M' are substochastic. \square

A corollary of the proposition is the following.

Proposition 2.3 *Let M, N be $d \times d$ substochastic matrices. Then for any nonnegative integer p , $\|M^p - N^p\| \leq p\|M - N\|$.*

Next we define two types of operators mapping $[0, 1]$ to $[0, 1]$. Let δ be a nonnegative real number. We define the *perturbation operator* Σ_δ as a function mapping any nonnegative real number $z \in [0, 1]$ to $\Sigma_\delta(z) = \max\{z - \delta, 0\}$. For a positive integer t , we define the *truncation operator* $\lfloor \cdot \rfloor_t$ as 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. It is obvious that these operators map substochastic matrices to substochastic matrices. We make the following simple observations about how these operators interact with the matrix norm.

Proposition 2.4 *Let $M, N \in \mathbb{R}^{d \times d}$, t be a positive integer and δ a positive real number. Then:*

1. $\|M - \lfloor M \rfloor_t\| \leq d2^{-t}$,
2. $\|M - \Sigma_\delta(M)\| \leq \delta d$,
3. $\|\Sigma_\delta(M) - \Sigma_\delta(N)\| \leq \|M - N\|$.

2.2 Space Bounded Computation of Matrices

We review some of the standard facts about space-bounded computation of matrices. We restrict attention to certain special classes of algorithms that will be useful later for our purposes.

We will be considering the computation of a class of functions we call *matrix functions*. The domain of a matrix function F consists of ordered pairs (M, z) where M is a square

matrix and z is an auxiliary parameter. $F(M, z)$ is a matrix whose dimension is the same as that of M . (Note that we do not require the presence of the auxiliary parameter in the input of a matrix function.)

We will consider algorithms A that compute such functions in the following sense. A will take as input M and z and in addition will take two indices i, j between 1 and $\dim(M)$. The output of A is interpreted as the (i, j) -th entry of a matrix denoted $A(M, z)$. Thus the entire matrix $A(M, z)$ could be determined by running the algorithm over all i, j . For (M, z) in the domain of F , we say that A *computes* F on input (M, z) if $A(M, z) = F(M, z)$ and that A *approximates* F on input (M, z) with accuracy a if $\|A(M, z) - F(M, z)\| \leq 2^{-a}$. Such an algorithm will be called a *matrix algorithm*. We will often identify a matrix algorithm with the matrix function it computes.

A matrix function F (resp., a matrix algorithm A) is said to be *substochastic* if $F(M, z)$ (resp., $A(M, z)$) is substochastic whenever M is, i.e., for any choice of z , F (resp., A) maps substochastic matrices to substochastic matrices.

A matrix function F is *computable in space* $S(\cdot)$ if there is a matrix algorithm that computes F on arbitrary input (M, z) from the domain of F and runs in space $S(M, z)$.

We next review recursive matrix computation in this context. Suppose F_1, F_2, \dots, F_k is a sequence of matrix functions. For a square matrix M and a sequence of auxiliary parameters z_1, z_2, \dots, z_k , we define a sequence of matrices $M_0, M_1, M_2, \dots, M_k$ recursively as follows:

$$M_0 = M \text{ and } M_i = F_i(M_{i-1}, z_i) \text{ for } 1 \leq i \leq k.$$

Let G be the matrix function such that $G(M, z_1, z_2, \dots, z_k)$ evaluates to M_k . Thus G is essentially the composition of the matrix functions F_1, F_2, \dots, F_k . Then the following fact is well known and easily verified:

Proposition 2.5 *Suppose for each $1 \leq i \leq k$, F_i is computable in space S_i , where $S_i \geq \log(\dim(M))$. Then G is computable in space $O(S_1 + S_2 + \dots + S_k)$.*

2.3 Offline Randomization

The deterministic algorithm we present for approximating substochastic matrix repeated squaring will be obtained by developing a randomized algorithm, and then derandomizing it. In the general view of randomized algorithms, random bits are used in an “on-line” fashion, that is, the algorithm requests random bits as it needs it, and need not store the bits unless necessary. The randomized algorithm we develop will have the following restricted structure: the algorithm, upon receipt of the input x , computes the total number $R(x)$ of random bits that will be required for the algorithm. It then obtains a string $y \in \{0, 1\}^{R(x)}$ of random bits from the random source and stores y in a designated section of memory, which from then on is available for reading only and can be accessed globally at any stage of the subsequent computation. Given x and y , the operation of the algorithm is completely deterministic. We will refer to this method of using random bits as “offline randomization”. This paradigm has been used implicitly in previous work on derandomizing randomized algorithms (see e.g. [Nis92, NZ93]), here we make it explicit in order to clarify certain subtleties.

We think of an offline randomized algorithm A as a function $A(x; y)$ of two input strings: x the “true” input and y , the “offline random input”. More generally we write $A(x_1, \dots, x_i; y_1, \dots, y_j)$ if the true input to A is a list x_1, \dots, x_i of strings and the off-line random input is a list y_1, \dots, y_j of strings. When accounting for space in an offline randomized algorithm, we explicitly divide the space requirements of the algorithm into two quantities: the space $R(x)$ needed to store the random bits, called the *random bit complexity* and the space required to run $A(x; y)$ once y is written down, called the *processing space complexity*. The overall space is the sum of these. (Technically, we should also account for the space needed to compute the number $R(x)$ of random bits needed but, in all of our algorithms and in virtually any conceivable situation, this is trivial compared to the other requirements, and we ignore it.)

Remark: One point in the above description should be emphasized. Suppose that $A(x; y)$ is an offline randomized algorithm and suppose that during the execution of A , various randomized subroutines $B_1(x_1; y_1), B_2(x_2; y_2) \dots$ are called (including recursive calls). Then each B_i does not generate its own random bits, since we require that the operation of A be completely deterministic given x and y . The call to B_i must specify y_i as well as x_i . Typically (and always in what we do here), y_i will be some designated subset of the *globally* accessible string y . Thus when we account for random bit complexity, we do not need to count the random bit requirements of every call of a subroutine, since all random bits are accounted for at the top level of the algorithm.

Now suppose that A is an offline randomized matrix algorithm. This means that it takes as true input (M, z) , and also an offline random input $y \in \{0, 1\}^{R(M, z)}$. For an integer a and a real number β , we say that A approximates the matrix function F on input (M, z) with accuracy a and error probability β if the following holds:

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

where the probability is over y chosen uniformly at random from $\{0, 1\}^{R(M, z)}$. In other words, for each fixed (M, z) all but a fraction β of the possible strings y will cause the algorithm to output an approximation to $F(M, z)$ with accuracy a .

There is a trivial way to convert an offline randomized algorithm A into a deterministic algorithm: for fixed input x , we enumerate over all choices of y and compute $A(x; y)$, then take the arithmetic average over all y of $A(x; y)$. We call this procedure the *naive derandomization* of A .

Lemma 2.1 *Let F be a substochastic matrix function and A be an offline randomized substochastic matrix algorithm. Suppose that on input (M, z) , A requires $R = R(M, z)$ random bits, and has processing space complexity $S = S(M, z)$, and approximates F on (M, z) with accuracy a and error probability β , where $\beta \leq 2^{-(a+1)}$. Then the naive derandomization B of A runs in space $O(S + R)$ and $B(M, z)$ approximates $F(M, z)$ with accuracy $a - 1$.*

Proof: From the construction of the naive derandomization algorithm, we have

$$B(M, z) = \frac{1}{2^R} \sum_{y \in \{0, 1\}^R} A(M, z; y).$$

The computation of $B(M, z)$ is done by first running through all the choices of y , computing $A(M, z; y)$ for each y and summing them up, and then taking the average of the sum. The space needed for the computation consists of three parts: the space to enumerate y , the space to compute $A(M, z; y)$, and the space to compute and store the value of the sum and to calculate the average. Enumerating over y takes space R . Once the value of y is stored, the deterministic procedure $A(M, z; y)$ takes y as part of the input and thus uses only S space by definition. The space can be reused for each successive y . Since each entry of $A(M, z; y)$ takes space at most S , the space needed to compute and store the value of the sum is thus bounded by $O(S + R)$, as is the space needed to compute the average. Then we conclude that B runs in space $O(S + R)$.

We now estimate the accuracy. Denote by $G_a(M, z)$ the set of all $y \in \{0, 1\}^R$ such that $\|A(M, z; y) - F(M, z)\| \leq 2^{-a}$. By assumption, $|G_a(M, z)| \geq (1 - \beta)2^R$. Then

$$\begin{aligned}
\|B(M, z) - F(M, z)\| &\leq \frac{1}{2^R} \sum_{y \in \{0, 1\}^R} \|A(M, z; y) - F(M, z)\| \\
&= \frac{1}{2^R} \left(\sum_{y \in G_a(M, z)} \|A(M, z; y) - F(M, z)\| \right. \\
&\quad \left. + \sum_{y \in \{0, 1\}^R - G_a(M, z)} \|A(M, z; y) - F(M, z)\| \right) \\
&\leq \frac{1}{2^R} (2^{-a} |G_a(M, z)| + 2 \cdot (2^R - |G_a(M, z)|)) \\
&\leq \frac{1}{2^R} (2^{-a} 2^R + 2 \cdot \beta 2^R) \\
&= 2^{-a} + 2\beta \\
&\leq 2^{-a+1}. \quad \square
\end{aligned}$$

What this lemma says, in effect, is that to construct a space efficient deterministic matrix algorithm that approximates a matrix function F with sufficient accuracy, it suffices to construct a space efficient randomized matrix algorithm that approximates F with sufficient accuracy and does not use “too many” random bits.

3 Formal Statement of Main Result

We formalize the repeated squaring problem as follows:

Approximate Substochastic Matrix Repeated Squaring (AMRS)

Input: a $d \times d$ substochastic matrix M , integers 2^r and 2^a in unary.

Output: a $d \times d$ substochastic matrix M' such that $\|M' - M^{2^r}\| \leq 2^{-a}$.

We want a matrix algorithm for AMRS in the sense of Section 2.2, i.e., one that takes as input $M, 2^r, 2^a$ and indices $u, v \in [d]$ and outputs $M'[u, v]$.

For the convenience of our presentation, we define the parameter $s = \max\{r, a, \log d\}$. We measure the space complexity of the algorithm for AMRS in terms of r and s .

In what follows, we present an explicit algorithm for AMRS to prove:

Theorem 3.1 *There is a deterministic algorithm for AMRS with space complexity $O(r^{1/2}s)$. In particular, in the case that the parameters a, r are of $O(\log d)$, the space complexity is $O(\log^{3/2} d)$.*

4 The Algorithm

4.1 Motivation and Overview

As mentioned in the introduction, the straightforward recursive algorithm for AMRS requires space $O(rs)$. The algorithm we present will reduce this space to $O(r^{1/2}s)$. Our approach will be to find a randomized approximation algorithm that runs in this space, and uses only $O(r^{1/2}s)$ random bits. The algorithm will approximate $\Lambda^{(r)}(M) = M^{2^r}$ with accuracy $a + 1$ and error probability $2^{-(a+2)}$. Then we use the naive derandomization to construct a deterministic algorithm, which by Lemma 2.1 will run in space $O(r^{1/2}s)$ and achieve approximation accuracy a .

The starting point for constructing such a randomized algorithm is Nisan’s pseudorandom generator for space bounded computation. By examining Nisan’s derivation of the generator, one can see that it implicitly gives an offline randomized algorithm for AMRS, which we call *PRS* for *pseudorandom repeated squaring*. The details of the algorithm are given in Section 6. The relevant properties of this algorithm are summarized in the following:

Lemma 4.1 *Let a be an integer. Given as input a $d \times d$ substochastic matrix M and integers r, m , the algorithm $PRS(M, r, m; \vec{h})$ takes a random string $\vec{h} \in (\{0, 1\}^{2m})^r$, runs in space $O(m + r + \log d)$ and computes a substochastic matrix of dimension d subject to the property that the algorithm approximates $\Lambda^{(r)}(M)$ (a matrix function on input (M, r)) with accuracy a and error probability $\frac{2^{2a+3r+5\log d}}{2^m}$.*

The lemma says that for any substochastic matrix M , almost all choices of \vec{h} are “good” for M in the sense that $PRS(M, r, m; \vec{h})$ computes a “close” approximation to M^{2^r} . We call the parameter m in the lemma the *randomization parameter*. From the lemma, we can choose $m = O(s)$ and get error probability that is exponentially small in s . However, to achieve this we require $\Theta(rs)$ random bits, and so the naive derandomization will need $\Theta(rs)$ space, which is the same as the standard recursive algorithm. The one thing we have gained however, is that the processing space complexity (the space over and above the random bits) is only $O(s)$, which is much smaller than we can afford. So our aim will be to reduce the number of random bits, and we can afford to increase the processing space complexity, if necessary. We will succeed in making both $O(r^{1/2}s)$.

The next idea is to try to apply *PRS* recursively. Suppose that r is factored as $r_1 \times r_2$ (eventually we will assume without loss of generality that r is a perfect square and choose both r_1, r_2 to be $r^{1/2}$.) Then $\Lambda^{(r)}(M) = (\Lambda^{(r_1)})^{r_2}(M)$, i.e., $\Lambda^{(r)}(M)$ can be computed by r_2 applications (compositions) of the function $\Lambda^{(r_1)}$ to M . Now instead of computing $\Lambda^{(r)}$ we can use *PRS* to approximate it. The i -th application of *PRS*, $1 \leq i \leq r_2$, requires

a random string $\vec{h}^{(i)}$ of length $2r_1m$. Each level of recursion thus needs $2r_1m$ bits for $\vec{h}^{(i)}$ and an additional $O(s)$ space. The total amount of processing space is thus $r_2 \times O(s)$ while the number of random bits we need is $r_2 \times 2r_1m = 2rm$. This hardly looks like progress: we've increased the processing space complexity from $O(s)$ to $O(r_2s)$ without getting any reduction in the number of random bits!

But now there is an obvious way to try to reduce the number of random bits: choose a single random vector \vec{h} of length $2r_1m$ and use it for $\vec{h}^{(i)}$ at every level of the recursion. Intuitively, it seems reasonable that this might work, since we know from the properties of the *PRS* algorithm that at each level of recursion, almost all choices of \vec{h} are “good” for the matrix being powered at that level. The problem is that there seems to be no easy way to prove that almost every \vec{h} is good simultaneously for every level of recursion (in fact we don't even know how to prove that there is even one such \vec{h}). There is a natural approach to doing this that “almost” works, which we sketch because it will motivate what comes later. Consider the sequence $M_0 = M, M_1, M_2, \dots, M_{r_2}$ where for $i \geq 1$, M_i is the result of *PRS*($M_{i-1}, r_1, m; \vec{h}^{(i)}$) computed at recursive level i (where we number recursive levels by assigning small numbers to deeper levels). We want to show that for most choices of \vec{h} , if we take $\vec{h}^{(i)} = \vec{h}$ at every level then all of these approximations are “close enough”. Now, as mentioned, we cannot just apply the fact that almost all choices of \vec{h} are good at each level, because the event that $\vec{h}^{(i)}$ is good at level i depends on the matrix M_{i-1} , which in turn depends on $\vec{h}^{(i-1)}$. Instead, it is natural to look at the sequence $N_0 = M, N_1, N_2, \dots, N_{r_2}$ where $N_i = N_{i-1}^{2r_1}$, which is the exact sequence that M_0, M_1, \dots, M_{r_2} is attempting to estimate. Now since all of the N_i are fixed by M , and do not themselves depend on \vec{h} , we can use the fact that almost all \vec{h} are good for each N_i to conclude that almost all \vec{h} are simultaneously good for all of the N_i . It is then reasonable to conjecture that any \vec{h} that is good for all of the N_i is good for all of the M_i as well. The idea for proving this is an induction. Since $M_0 = N_0$, \vec{h} is good for M_0 . Now since \vec{h} is good for M_0 , this means that the matrix M_1 computed by applying *PRS* to $(M_0, r_1, m; \vec{h})$ is “close” to N_1 . What we then want to prove is that since M_1 is “close” to N_1 and \vec{h} is good for N_1 , then \vec{h} is also good for M_1 . If this is true we can continue by induction. This can be reduced to showing that if N_i and M_i are “close” and \vec{h} is good for N_i , then the matrices N_i^* and M_i^* obtained by applying *PRS* to N_i and M_i are also “close”. In the qualitative way we have stated things, this is indeed true; the problem comes in making it quantitative. The best upper bound we have obtained on the ratio $\|M^* - N^*\|/\|M - N\|$ (which can be viewed as the rate of growth of error as we proceed through recursive levels) is too large to be able to prove that the final approximation to M^{2^r} is a good one.

Thus, while it is still possible that the simple recursive algorithm that reuses \vec{h} does indeed estimate the repeated square accurately, we could not prove it. The algorithm we present is derived from the approach just outlined, but requires an additional idea to circumvent the above obstacle. The idea, very roughly, is this: What we would like is to make the sequence M_i equal to N_i so as to avoid the dependencies on \vec{h} . In order to accomplish this, at every recursive level instead of applying *PRS* directly to the matrix M_{i-1} computed at the previous level of recursion, we apply it to a modified version of M_{i-1} which is obtained by “perturbing” the entries at random and then truncating them. The intuition is that in

typical cases, we can make M_i and N_i equal by truncating the low order bits from each entry in M_i and N_i since they are “close”; nevertheless, to prevent the cases where bad roundings might occur, we first apply random perturbations. We can then show that almost all choices of \vec{h} are “good” simultaneously for every level of recursion. So we will only need $2r_1m$ random bits to generate this single \vec{h} . However, the random perturbations will themselves require additional random bits, but the number will only be $O(s)$ per level of recursion for a total of at most $O(r_2s)$ random bits. Thus the total number of random bits needed will be $O((r_1 + r_2)s)$ which for $r_1 = r_2 = r^{1/2}$ is $O(r^{1/2}s)$.

4.2 The Description of the Algorithm

In this subsection, we describe our randomized algorithm for AMRS in detail. The key properties of this randomized algorithm, which we call MAIN for reference, will be that it uses $O(r^{1/2}s)$ random bits, has processing space complexity $O(r^{1/2}s)$ and approximates $\Lambda^{(r)}(M)$ with accuracy $a + 1$ and error probability $2^{-(a+2)}$. Using the naive derandomization, we obtain the desired deterministic algorithm.

The algorithm MAIN is a simple variant of the recursive *PRS* algorithm described above. As in the previous subsection, we suppose that r is factored as $r_1 \times r_2$, and eventually we will take $r_1 = r_2 = r^{1/2}$. (Thus we will assume that r is a perfect square. This assumption is unimportant for two reasons: (1) for purposes of derandomizing random space algorithms it does not hurt to replace r by the least perfect square greater than it, and (2) if one really cares to estimate $\Lambda^{(r)}(M)$ where r is not a perfect square then we may write $r = u + v$ where u is a perfect square and $v = O(\sqrt{r})$. Then we use the algorithm we present to compute $N = \Lambda^{(u)}(M)$ and use the simple recursive algorithm to estimate $\Lambda^{(v)}(N)$.)

Besides input M, r and a , the algorithm MAIN has four additional parameters m, t, D and K that are computed from the input and do not change thereafter. We will see below that the values for these parameters are all $\Theta(s)$. The parameter m is the randomization parameter to be used in each call of the *PRS* algorithm. The parameter t is a precision parameter, which represents the number of bits of precision that are passed from one recursive level to another. D and K are parameters that describe the range of possible perturbations that can be applied: when we perturb a matrix, we decrease all entries by the same amount, which is a number of the form $2^{-K}q$ where $0 \leq q < 2^D$. The parameters t, D, K satisfy $t + D = K$.

The number of random bits required by the algorithm is $2mr_1 + Dr_2$ which is $O(s(r_1 + r_2)) = O(r^{1/2}s)$. These random bits are represented as a pair (\vec{h}, \vec{q}) where \vec{h} consists of $2mr_1$ bits corresponding to the offline random input to the algorithm *PRS* for approximating $\Lambda^{(r_1)}(M)$, and \vec{q} is a vector $[q(1), q(2), \dots, q(r_2)]$, where each $q(i)$ is interpreted as a D -bit integer.

Now we are ready to describe our algorithm.

Algorithm MAIN

Input: a $d \times d$ substochastic matrix M , integers r ($r = r_1 \times r_2$ as assumed) and a , and indices $u, v \in [d]$;

Initialize (compute from the input) parameters: m, D, K and $t = K - D$;

Offline Random Input: $\vec{h} \in \{0, 1\}^{2mr_1}$ and $\vec{q} \in \{0, 1\}^{Dr_2}$, where $\vec{q} = [q(1), q(2), \dots, q(r_2)]$

for each $q(i) \in \{0, 1\}^D$;

Define the sequence of matrices

$$M_0; M_1^P, M_1^\Sigma, M_1; M_2^P, M_2^\Sigma, M_2; M_3^P \dots M_{r_2-1}; M_{r_2}^P, M_{r_2}^\Sigma, M_{r_2}$$

by $M_0 = M$ and for $1 \leq i \leq r_2$,

$$\begin{aligned} M_i^P &= PRS(M_{i-1}, r_1, m; \vec{h}) \\ M_i^\Sigma &= \Sigma_{\delta_i}(M_i^P) \text{ where } \delta_i = q(i)2^{-K} \\ M_i &= \lfloor M_i^\Sigma \rfloor_t \end{aligned}$$

The algorithm MAIN recursively computes $M_{r_2}[u, v]$.

Output: $M_{r_2}[u, v]$

In words, M_i^P is a pseudorandom approximation of $\Lambda^{(r_1)}(M_{i-1})$, M_i^Σ is obtained by ‘‘perturbing’’ M_i^P by decreasing its entries by δ_i (subject to staying nonnegative), M_i is obtained from M_i^Σ by truncating each entry of M_i^Σ after t bits.

As noted above, the number of random bits used in the algorithm is $O(r^{1/2}s)$. Let us analyze the space needed for this algorithm. The algorithm computes the composition of a sequence of $3r_2$ matrix functions (as defined in Section 2.2), so we may apply Proposition 2.5 to say that the processing space of the computation is the sum of the processing space needed for computing each function. The perturbation and truncation functions are both trivially computable in space $O(s)$. By Lemma 4.1, the space needed for the algorithm PRS other than the random bits, is $O(s)$. Thus the overall processing space can be bounded by $O(r_2s)$ as required.

What is not clear is that this algorithm produces a good approximation for $\Lambda^{(r)}$. We will prove:

Theorem 4.1 *The algorithm MAIN approximates $\Lambda^{(r)}(M)$ with accuracy $K - D - 2r - \log d$ and error probability $\frac{2^{r+2\log d}}{2^D} + \frac{2^{2K+4r+5\log d}}{2^m}$.*

In light of this Theorem, we can now choose the parameters, e.g. $m = 39s$, $t = 6s$, $D = 7s$ and $K = 13s$ to obtain an algorithm whose accuracy is at least $a + 1$ with error probability bounded above by $2^{-(a+2)}$. Applying the naive derandomization yields the desired deterministic algorithm for AMRS.

Remark: For simplicity of presentation we assumed that the maximum number of bits per entry of the input matrix M is at most m . We can always replace M by the matrix $M_0 = \lfloor M \rfloor_m$ at the lowest level of the recursion. By Propositions 2.3 and 2.4, $\Lambda^{(r)}(M_0)$ is very close to $\Lambda^{(r)}(M)$. Thus since the algorithm produces a good approximation to $\Lambda^{(r)}(M_0)$, it is also a good approximation to $\Lambda^{(r)}(M)$.

In the remainder of this paper we first prove Theorem 4.1. Then in Section 6, we examine in detail the connection between approximating substochastic matrix exponentiation and

constructing pseudorandom generators, and describe the algorithm *PRS* and prove Lemma 4.1. This will complete the proof of Theorem 3.1 and Theorem 1.1.

5 The Correctness Proof

We will denote by $\Delta(M, K; \vec{h}, \vec{q})$, where $\vec{h} \in (\{0, 1\}^{2m})^{r_1}$ and $\vec{q} = [q(1), q(2), \dots, q(r_2)] \in (\{0, 1\}^D)^{r_2}$, the sequence of matrices $[M_0; M_1^P, M_1^\Sigma, M_1; M_2^P, \dots; M_{r_2}^P, M_{r_2}^\Sigma, M_{r_2}]$ computed by the algorithm MAIN. Note that $\Delta(M, K; \vec{h}, \vec{q})$ depends implicitly on m and r_1 (through its dependence on \vec{h}) and on D and r_2 (through its dependence on \vec{q}). Recall that the structure of the algorithm is such that M_i is computed from M_{i-1} by first applying *PRS* to approximate its 2^{r_1} -th power, perturbing the matrix, and then truncating it. We want to show that for most choices of the random bits to the algorithm, after r_2 iterations, the resulting matrix is a good approximation to the 2^r -th power of the matrix.

The analysis proceeds in a way analogous to the approach sketched in Section 4.1. We will compare the sequence $\Delta(M, K; \vec{h}, \vec{q})$ of matrices to the sequence $\Gamma(M, K; \vec{q}) = [N_0; N_1^P, N_1^\Sigma, N_1; N_2^P, \dots; N_{r_2}^P, N_{r_2}^\Sigma, N_{r_2}]$ of matrices which is obtained by a nearly identical process to MAIN: $N_0 = M$ and for $1 \leq i \leq r_2$,

$$\begin{aligned} N_i^P &= N_{i-1}^{2^{r_1}} \\ N_i^\Sigma &= \Sigma_{\delta_i}(N_i^P) \text{ where } \delta_i = q(i)2^{-K} \\ N_i &= \lfloor N_i^\Sigma \rfloor_t. \end{aligned}$$

The only difference between this process and the one in MAIN is that N_i^P is defined to be the *exact* 2^{r_1} -th power of N_{i-1} , rather than the pseudorandom approximation.

Note that we are not interested in computing this sequence, but only consider it for the purpose of analysis. Note also that $\Gamma(M, K; \vec{q})$ does not depend on \vec{h} .

We will first prove:

Lemma 5.1 *For any choice of $\vec{q} \in \{0, 1\}^{Dr_2}$, the sequence $\Gamma(M, K; \vec{q})$ has the property that N_{r_2} approximates $\Lambda^{(r)}(M)$ with accuracy $K - D - 2r - \log d$.*

Once we have this lemma, to prove the theorem, it would be enough to show that for almost all choices of \vec{q} and \vec{h} , the final matrix M_{r_2} in $\Delta(M, K; \vec{h}, \vec{q})$ is suitably close to the final matrix N_{r_2} in $\Gamma(M, K; \vec{q})$. In fact, we will show that for almost all \vec{q} and \vec{h} , M_{r_2} is actually equal to N_{r_2} .

For this we need two definitions.

Definition 5.1 *Let W be a substochastic matrix and r, a, m be integers. We say a string $\vec{h} \in \{0, 1\}^{2mr}$ is a -pseudorandom with respect to W if $PRS(W, r, m; \vec{h})$ approximates $\Lambda^{(r)}(W)$ with accuracy a .*

Definition 5.2 *A nonnegative real number r is (b, t) -dangerous for positive integers $b > t$, if r can be written in the form $2^{-t}I + \rho$ where I is a positive integer and $\rho \in [-2^{-b}, 2^{-b})$, and is (b, t) -safe otherwise. A matrix W is (b, t) -dangerous if any entry of it is (b, t) -dangerous and is (b, t) -safe if all of its entries are (b, t) -safe.*

Lemma 5.2 Fix $\vec{q} \in \{0, 1\}^{Dr_2}$. In $\Gamma(M, K; \vec{q})$, suppose that all of the matrices N_i^Σ are (K, t) -safe, and suppose that $\vec{h} \in \{0, 1\}^{2mr_1}$ is a vector that is K -pseudorandom with respect to each N_i for $0 \leq i < r_2$. Then $M_{r_2} = N_{r_2}$. (In fact, $M_i = N_i$ for $0 \leq i \leq r_2$.)

Now, for any fixed $\vec{q} \in \{0, 1\}^{Dr_2}$, the sequence $\Gamma(M, K; \vec{q})$ does not depend on \vec{h} . Since for each N_i , we know by Lemma 4.1 that all but a fraction $2^{-m+2K+3r+5\log d}$ of the $\vec{h} \in \{0, 1\}^{2mr_1}$ are K -pseudorandom with respect to N_i , we have:

Proposition 5.1 For any $\vec{q} \in \{0, 1\}^{Dr_2}$, all but a fraction $2^{-m+2K+4r+5\log d}$ of the $\vec{h} \in \{0, 1\}^{2mr_1}$ are K -pseudorandom with respect to each of the N_i for $0 \leq i < r_2$.

Next we see where the perturbations help: we show that almost all vectors $\vec{q} \in \{0, 1\}^{Dr_2}$ satisfy the hypothesis of Lemma 5.2.

Lemma 5.3 For a randomly chosen $\vec{q} \in \{0, 1\}^{Dr_2}$, the probability that all of the matrices N_i^Σ in $\Gamma(M, K; \vec{q})$ are (K, t) -safe is at least $1 - 2^{-D+r+2\log d}$.

Assuming these three lemmas, we now prove Theorem 4.1.

Proof of Theorem 4.1: By Lemma 5.1, it suffices to upper bound the probability that, for a randomly chosen pair (\vec{h}, \vec{q}) , the matrix M_{r_2} in the sequence $\Delta(M, K; \vec{h}, \vec{q})$ does not equal to the matrix N_{r_2} in $\Gamma(M, K; \vec{q})$.

$$\begin{aligned} Pr[M_{r_2} \neq N_{r_2}] &\leq Pr[\text{not all } N_i \in \Gamma(M, K; \vec{q}) \text{ are } (K, t)\text{-safe}] + \\ &\quad Pr[\vec{h} \text{ is not } K\text{-pseudorandom w.r.t. each } N_i \in \Gamma(M, K; \vec{q})] \\ &\leq 2^{-D+r+2\log d} + 2^{-m+2K+4r+5\log d}, \end{aligned}$$

where the first inequality follows from Lemma 5.2, and the second follows from Lemma 5.3 and Proposition 5.1. \square

In the remainder of the section we prove the three lemmas.

Proof of Lemma 5.1:

Definition 5.3 A sequence of $d \times d$ matrices $[N_0, N_1, \dots, N_p]$ is said to be (l, β) -close if for all $1 \leq i \leq p$, $\|N_i - N_{i-1}^l\| \leq \beta$.

Lemma 5.4 Let $[M = N_0, N_1, \dots, N_p]$ be an (l, β) -close sequence of $d \times d$ substochastic matrices. Then

$$\|N_p - M^{lp}\| \leq \sum_{i=1}^p (l^{i-1} \beta).$$

Proof: We proceed by induction on p . The case where $p = 0$ is trivial. Assume it is true for $p - 1$ and we show that the lemma holds for p .

$$\begin{aligned}
\|N_p - M^{lp}\| &\leq \|N_p - N_{p-1}^l\| + \|N_{p-1}^l - (M^{lp-1})^l\| \\
&\leq \beta + l\|N_{p-1} - M^{lp-1}\| \\
&\leq \beta + l \sum_{i=1}^{p-1} (l^{i-1}\beta) \\
&= \sum_{i=1}^p (l^{i-1}\beta),
\end{aligned}$$

where the second inequality follows from the fact that the sequence is (l, β) -close and from Proposition 2.3, and the last inequality is by induction. \square

We claim the sequence of matrices $[N_0, N_1, \dots, N_{r_2}]$ in $\Gamma(M, K; \vec{q})$ is $(2^{r_1}, 2^{-t+\log d+1})$ -close. Recall that $N_i^P = N_{i-1}^{2^{r_1}}$, $N_i^\Sigma = \Sigma_{\delta_i}(N_i^P)$ where $\delta_i = q(i)2^{-K}$, and $N_i = \lfloor N_i^\Sigma \rfloor_t$. Therefore,

$$\begin{aligned}
\|N_i - N_{i-1}^{2^{r_1}}\| &\leq \|N_i - N_i^\Sigma\| + \|N_i^\Sigma - N_i^P\| \\
&\leq d2^{-t} + \delta_i d \\
&\leq 2^{-t+\log d+1},
\end{aligned}$$

where the second inequality follows from Proposition 2.4 and the last inequality from the fact that $t = K - D$. Now Lemma 5.4 completes the proof. \square

Proof of Lemma 5.2:

We prove by induction that $M_i = N_i$ for $0 \leq i \leq r_2$. The basis $i = 0$ is trivial. For $i > 0$, assume $M_{i-1} = N_{i-1}$. Since $\vec{h} \in \{0, 1\}^{2^{mr_1}}$ is K -pseudorandom with respect to N_{i-1} , by definition, $\|M_i^P - N_i^P\| \leq 2^{-K}$. Then we have $\|M_i^\Sigma - N_i^\Sigma\| \leq 2^{-K}$ from the third fact in Proposition 2.4, which implies that for any $u, v \in [d]$,

$$|M_i^\Sigma[u, v] - N_i^\Sigma[u, v]| \leq 2^{-K}. \quad (1)$$

Now, the fact that N_i^Σ is (K, t) -safe implies that either $N_i^\Sigma[u, v] \in [0, 2^{-t} - 2^{-K})$ or $N_i^\Sigma[u, v] = \lfloor N_i^\Sigma[u, v] \rfloor_t + \rho$ where $\rho \in [2^{-K}, 2^{-t} - 2^{-K})$. Thus by (1), $M_i^\Sigma[u, v] = \lfloor N_i^\Sigma[u, v] \rfloor_t + \rho'$ where $\rho' \in [0, 2^{-t})$. (This is because otherwise $M_i^\Sigma[u, v] < \lfloor N_i^\Sigma[u, v] \rfloor_t$, which is impossible in the former case and violates (1) in the latter one.) Therefore $\lfloor M_i^\Sigma[u, v] \rfloor_t = \lfloor N_i^\Sigma[u, v] \rfloor_t$. Since this holds for arbitrary u, v , we have $M_i = \lfloor M_i^\Sigma \rfloor_t = \lfloor N_i^\Sigma \rfloor_t = N_i$ as required. \square

Proof of Lemma 5.3:

We want to upper bound the probability μ , with respect to the random choice of $\vec{q} = [q(1), q(2), \dots, q(r_2)] \in (\{0, 1\}^D)^{r_2}$, that at least one of the matrices N_i^Σ is (K, t) -dangerous. Recall that $D+t = K$. Note that for any fixed M , the sequence $[N_0; N_1^P, N_1^\Sigma, N_1; N_2^P, \dots; N_{r_2}^P, N_{r_2}^\Sigma, N_{r_2}]$ is completely determined by \vec{q} .

We can upper bound μ by $\sum_{i=1}^{r_2} \mu_i$ where $\mu_i = \Pr[N_i^\Sigma \text{ is } (K, t)\text{-dangerous}]$. We will bound each of the probabilities in this sum by $2d^22^{-D}$, which will be sufficient for the proof of the lemma.

Now, by the definition of N_i^Σ , $\mu_i = \Pr[\Sigma_{q(i)2^{-\kappa}}(N_i^P) \text{ is } (K, t)\text{-dangerous}]$. Let \mathcal{N}_i^P denote the set of all possible values for the matrix N_i^P (which is finite since the number of choices for \vec{q} is finite) and for any matrix W , let $T_i(W)$ denote the event that $\Sigma_{q(i)2^{-\kappa}}(W)$ is (K, t) -dangerous. Then

$$\begin{aligned} \mu_i &= \sum_{W \in \mathcal{N}_i^P} \Pr[N_i^P = W] \Pr[T_i(W) | N_i^P = W] \\ &= \sum_{W \in \mathcal{N}_i^P} \Pr[N_i^P = W] \Pr[T_i(W)] \\ &\leq \max_{W \in \mathcal{N}_i^P} \Pr[T_i(W)], \end{aligned}$$

where the second equality comes from the fact that for each fixed W the event $N_i^P = W$ depends only on $q(1), q(2), \dots, q(i-1)$, while the event $T_i(W)$ depends only on $q(i)$, so these events are independent. So finally it suffices to prove:

Lemma 5.5 *For any fixed substochastic matrix W of dimension d , the probability with respect to $q \in \{0, 1\}^D$ that $\Sigma_{q2^{-\kappa}}(W)$ is (K, t) -dangerous is at most $2d^22^{-D}$.*

Proof: We can bound the desired probability by d^2 times the probability that any particular entry of $\Sigma_{q2^{-\kappa}}(W)$ is (K, t) -dangerous.

Fix any $(u, v) \in [d] \times [d]$. Then $W[u, v] = 2^{-t}I_1 + 2^{-K}I_2 + \rho$ for a unique (I_1, I_2, ρ) such that I_1 is an integer, I_2 is an integer in $[0, 2^D - 1]$ and $\rho \in [0, 2^{-K}]$. In the case where $I_1 = 0$, if $I_2 < 2^D - 1$, i.e., $W[u, v] < 2^{-t} - 2^{-K}$, then for any value of q , $\Sigma_{q2^{-\kappa}}(W[u, v]) \leq W[u, v]$ is not (K, t) -dangerous by definition; if $I_2 = 2^D - 1$, then $\Sigma_{q2^{-\kappa}}(W[u, v])$ is (K, t) -dangerous only when $q = 0$. In all other cases, it is easy to check that the only values of q for which $\Sigma_{q2^{-\kappa}}(W[u, v])$ is (K, t) -dangerous are the values I_2 and $I_2 + 1 \pmod{2^D}$.

Thus the probability that $\Sigma_{q2^{-\kappa}}(W[u, v])$ is (K, t) -dangerous is at most $2/2^D$ and so the probability that $\Sigma_{q2^{-\kappa}}(W)$ is (K, t) -dangerous is at most $2d^22^{-D}$ as required. \square

6 Matrix Pseudorandom Repeated Squaring

In this section, we complete both the description of the algorithm and the proof of correctness by presenting the *PRS* algorithm satisfying Lemma 4.1. As stated earlier, this algorithm is derived from Nisan's pseudorandom generator construction [Nis90]. In what follows, we first examine the general relationship between approximating substochastic matrix exponentiation and constructing pseudorandom generators. Then we present in detail the construction of Nisan's pseudorandom generator and analyze its properties. Finally we describe the algorithm *PRS* and prove Lemma 4.1.

6.1 Matrix Exponentiation and Pseudorandom Generators

6.1.1 Finite State Machines and Substochastic Matrices

Definition 6.1 A finite state machine Q of type (d, m) is a directed graph on vertex set $\{0, 1, 2, \dots, d\}$ such that

- each vertex has 2^m outgoing arcs, which are labeled in one to one correspondence with the alphabet $\Sigma = \{0, 1\}^m$, and
- all 2^m arcs leaving node 0 are self-loops.

The vertex set $\{0, 1, 2, \dots, d\}$ is called the set of states of the machine. We use $Q[i, j]$ to denote the set of $\alpha \in \Sigma$ that appear as labels on arcs directed from state i to state j .

Given any state i , each word $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_p) \in \Sigma^p$ defines a unique path of length p starting from i , obtained by following in succession the arcs labeled by $\alpha_1, \alpha_2, \dots, \alpha_p$. We denote by $Q^p[i, j]$ the set of all words in Σ^p that define a path from i to j . Note that Q^p can itself be viewed as a finite state machine of type (d, mp) . We say that a word $\alpha \in \Sigma^p$ maps state i to state j if $\alpha \in Q^p[i, j]$. It is easily seen that:

Proposition 6.1 For any finite state machine Q of type (d, m) and any positive integers p_1 and p_2 , $Q^{p_1 p_2} = (Q^{p_1})^{p_2}$.

For a finite state machine Q of type (d, m) , we define the $d \times d$ matrix Q^* to be such that the rows and columns of Q^* are indexed by $[d]$ and $Q^*[i, j] = 2^{-m}|Q[i, j]|$ for $i, j \in [d]$, i.e. $Q^*[i, j]$ is equal to the fraction of arcs leaving i that are directed to j . It is easy to see that Q^* is substochastic. In fact, for each i , the sum of entries in row i is equal to 1 minus the fraction of arcs leaving i that are directed to state 0.

The following fact is easy to verify:

Proposition 6.2 For any finite state machine Q of type (d, m) , and any positive integer p , $(Q^p)^* = (Q^*)^p$. In words, for each pair of states $i, j \in [d]$, the fraction of strings in Σ^p that map i to j is equal to the (i, j) -th entry of the p -th power of the substochastic matrix Q^* .

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

Identify each $\alpha \in \Sigma$ with the integer in $\{0, 1, \dots, 2^m - 1\}$ whose binary expansion is α . For each $i \in [d]$, 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 $j \in [d]$, and define $Q(M)[i, 0]$ to be the set of strings corresponding to the set of integers in the interval $[2^m \sum_{l=1}^d M[i, l], 2^m - 1]$. 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.

Proposition 6.3 *There is an algorithm which, given as input a substochastic matrix M of type (d, m) , $\alpha \in \{0, 1\}^m$, and $i \in [d]$, determines the index j such that $\alpha \in Q(M)[i, j]$ in space $O(m + \log d)$.*

Proof: The algorithm examines the i -th row of M entry by entry in the order of

$$M[i, 1], M[i, 2], M[i, 3], \dots$$

$\alpha \in Q(M)[i, j]$ if j is the first state such that $\alpha < 2^m \sum_{k=1}^j M[i, k]$. In the case that $\alpha \geq 2^m \sum_{k=1}^d M[i, k]$, it returns $j = 0$. The space used by the algorithm is obviously $O(m + \log d)$. \square

6.1.2 Approximate Matrix Exponentiation and Pseudorandom Generators

Suppose we have a substochastic matrix M of type (d, m) , and we want to estimate the entries of M^p for some positive integer p . By the discussion of the previous subsection, $M^p[i, j]$ is equal to the fraction of words in $(\{0, 1\}^m)^p$ that map state i to state j in the finite state machine $Q(M)$. Thus, one way to compute $M^p[i, j]$ is simply to enumerate over all words in $(\{0, 1\}^m)^p$ and count the number of words that map i to j in the finite state machine. The space complexity of this algorithm is $\Theta(pm + \log d)$. In order to obtain more space efficiency, one desirable modification of the above naive enumeration is to find a space-efficient way to sample a small set of words in $(\{0, 1\}^m)^p$ and estimate $M^p[i, j]$ to be the fraction of words in this small set that map i to j . The application of pseudorandom generators provides a way to do this efficient sampling while guaranteeing the accuracy of the estimation.

An (a, b) -generator is defined to be a function mapping $\{0, 1\}^a$ to $(\{0, 1\}^b)^b$, i.e., a function that maps a string $x \in \{0, 1\}^a$ to a sequence of b strings $y_0, y_1, y_2, \dots, y_{b-1}$ where each $y_i \in \{0, 1\}^b$. The y_i 's are called the *output blocks* of the generator, and we use the convention that they are indexed starting from 0. If G is an (a, b) -generator and $x \in \{0, 1\}^a$, we write $G_\ell(x)$, $0 \leq \ell \leq b - 1$, for the ℓ -th output block of $G(x)$.

Suppose G is an (m, p) -generator. Given a finite state machine Q of type (d, m) , we define a finite state machine Q_G of type (d, m) as follows: for $i, j \in \{0, 1, 2, \dots, d\}$, $Q_G[i, j]$ is the set of all $\alpha \in \{0, 1\}^m$ such that $G(\alpha)$ maps i to j in Q . (The fact that Q_G is a finite state machine of type (d, m) is not difficult to verify.)

Definition 6.2 *Let G be an (m, p) -generator.*

1. *If Q is a finite state machine of type (d, m) , G is called ϵ -pseudorandom with respect to Q if $\|Q_G^* - (Q^p)^*\| \leq \epsilon$.*
2. *If M is a substochastic matrix of type (d, m) , G is called ϵ -pseudorandom with respect to M if it is ϵ -pseudorandom with respect to the machine $Q(M)$.*

The above definition captures the property that G provides a good subset $\{G(\alpha) | \alpha \in \{0, 1\}^m\}$ of $(\{0, 1\}^m)^p$ for purposes of estimating the fraction of the total number of words in $(\{0, 1\}^m)^p$ that map one state to another in the finite state machine, and thus for purposes of estimating the corresponding entry of the matrix M^p .

Lemma 6.1 *Let M be a substochastic matrix of type (d, m) and G an (m, p) -generator. If G is ϵ -pseudorandom with respect to M , then $\|(Q(M))_G^* - M^p\| \leq \epsilon$.*

Proof:

$$\begin{aligned} \|(Q(M))_G^* - M^p\| &= \|(Q(M))_G^* - ((Q(M))^*)^p\| \\ &= \|(Q(M))_G^* - ((Q(M))^p)^*\| \\ &\leq \epsilon, \end{aligned}$$

where the first equality follows from the definition of $Q(M)$, the second from Proposition 6.2 and the inequality follows from the assumption that G is ϵ -pseudorandom with respect to M . \square

6.2 Nisan's Pseudorandom Generator Family

For completeness, we explicitly describe Nisan's construction of a family of $(m, 2^r)$ -generators and examine its properties. The arguments here mainly follow Nisan's work in [Nis90]. The presentation is self-contained and is a variant of the one given in [Nis90].

In the following discussion, if α and β are two strings then $\alpha\beta$ denotes their concatenation, and all the probability distributions are assumed to be uniform. First we need some preliminaries.

6.2.1 The Composition of Generators

Generally speaking, pseudorandom generators are functions that expand short random seeds to much longer strings and guarantee certain randomness properties of the outputs. However, it is typically much easier to construct pseudorandom generators that expand random seeds to strings that are relatively short. The idea behind composing generators is to apply these generators with short outputs in a recursive fashion so that the composition expands random seeds at an exponential rate without losing too much randomness of the outcomes. In this section we discuss a general framework for generator composition.

For an (a, b) -generator G and an (a, b') -generator G' , the *composition* of G and G' , denoted $G \circ G'$, is defined to be the (a, bb') -generator such that for any $x \in \{0, 1\}^a$, $G \circ G'(x) = G(G'_0(x))G(G'_1(x)) \dots G(G'_{b'-1}(x))$.

Lemma 6.2 *Let Q be a finite state machine of type (d, m) . Suppose G is an (m, p) -generator and G' is an (m, p') -generator. Then $Q_{G \circ G'} = (Q_G)_{G'}$.*

Proof: We want to show that for all $i, j \in [d]$, $x \in Q_{G \circ G'}[i, j]$ if and only if $x \in (Q_G)_{G'}[i, j]$, i.e., if and only if there exist states $k_1, k_2, \dots, k_{p'-1}$ such that for $0 \leq \ell \leq p' - 1$, $G'_\ell(x) \in Q_G[k_\ell, k_{\ell+1}]$ where $k_0 = i$ and $k_{p'} = j$.

Fix arbitrary $i, j \in [d]$. By definition, $x \in Q_{G \circ G'}[i, j]$ if and only if $G \circ G'(x) = G(G'_0(x))G(G'_1(x)) \dots G(G'_{p'-1}(x))$ maps state i to state j in Q . The latter condition is equivalent to saying that there exist states $k_1, k_2, \dots, k_{p'-1}$ such that for $0 \leq \ell \leq p' - 1$, $G(G'_\ell(x))$ maps state k_ℓ to state $k_{\ell+1}$ in Q where $k_0 = i$ and $k_{p'} = j$, which means $G'_\ell(x) \in Q_G[k_\ell, k_{\ell+1}]$. \square

Lemma 6.3 *Let Q be a finite state machine of type (d, m) . Suppose G is an (m, p) -generator and G' is an (m, p') -generator. If G is ϵ -pseudorandom with respect to Q and G' is ϵ' -pseudorandom with respect to Q_G , then $G \circ G'$ is an (m, pp') -generator that is $(\epsilon' + p'\epsilon)$ -pseudorandom with respect to Q .*

Proof: By assumption, we have $\|Q_G^* - (Q^p)^*\| \leq \epsilon$ and $\|(Q_G)_{G'}^* - ((Q_G)^{p'})^*\| \leq \epsilon'$. Then,

$$\begin{aligned} \|Q_{G \circ G'}^* - (Q^{pp'})^*\| &= \|(Q_G)_{G'}^* - ((Q^p)^{p'})^*\| \\ &\leq \|(Q_G)_{G'}^* - ((Q_G)^{p'})^*\| + \|((Q_G)^{p'})^* - ((Q^p)^{p'})^*\| \\ &\leq \epsilon' + \|(Q_G^*)^{p'} - ((Q^p)^*)^{p'}\| \\ &\leq \epsilon' + p'\epsilon, \end{aligned}$$

where the equality follows from Lemma 6.2 and Proposition 6.1, the second inequality is by Proposition 6.2, and the last inequality follows from Proposition 2.3. \square

Remark: More generally, one can define a class of generators called (a_1, a_2, b) -generators. An (a_1, a_2, b) -generator is a function that maps $\{0, 1\}^{a_1}$ to $(\{0, 1\}^{a_2})^b$. As a special case, the (a, b) -generator we defined is an (a, a, b) -generator.

In this general setting, one can modify the definitions of Q_G , ϵ -pseudorandomness (Definition 6.2) and the composition of generators in the corresponding way so that Lemma 6.1, 6.2 and 6.3 hold analogously.

For the purpose of constructing Nisan's generator, we will be interested in the composition of $(m, 2)$ -generators.

6.2.2 Universal Hash Function Families

We review some of the properties of universal hash function families.

Definition 6.3 [CW79] *A family H of functions that map $\{0, 1\}^p$ to $\{0, 1\}^q$ is called a universal hash function family if for all $x_1 \neq x_2 \in \{0, 1\}^p$ and all $y_1, y_2 \in \{0, 1\}^q$,*

$$Pr_{h \in H}[h(x_1) = y_1 \text{ and } h(x_2) = y_2] = 2^{-2q}.$$

It follows immediately from the definition that:

Proposition 6.4 *Let H be a universal hash function family that maps $\{0, 1\}^p$ to $\{0, 1\}^q$. Suppose h is selected from H uniformly at random. Then for each $x \in \{0, 1\}^p$, the random variable $h(x)$ is uniformly distributed over $\{0, 1\}^q$ and for any $x \neq y \in \{0, 1\}^p$, $h(x)$ and $h(y)$ are pairwise independent.*

For any positive integer n and a subset $X \subseteq \{0, 1\}^n$, we define

$$\rho(X) = Pr_{x \in \{0, 1\}^n}[x \in X] = |X|2^{-n}.$$

Let $A \subseteq \{0, 1\}^p$ and $B \subseteq \{0, 1\}^q$. A function $h : \{0, 1\}^p \rightarrow \{0, 1\}^q$ is said to be (A, B, ϵ) -good if

$$|Pr_{x \in \{0, 1\}^p}[x \in A \text{ and } h(x) \in B] - \rho(A)\rho(B)| \leq \epsilon.$$

Lemma 6.4 *Let H be a universal hash function family that maps $\{0, 1\}^p$ to $\{0, 1\}^q$. Then for any $A \subseteq \{0, 1\}^p$, $B \subseteq \{0, 1\}^q$ and any $\epsilon > 0$,*

$$Pr_{h \in H}[h \text{ is not } (A, B, \epsilon)\text{-good}] \leq \frac{\rho(A)\rho(B)(1 - \rho(B))}{\epsilon^2 2^p}.$$

Proof: For each $x \in A$, we define a random variable X_x with respect to the uniform distribution on H such that for an $h \in H$,

$$X_x(h) = \begin{cases} 1 & \text{if } h(x) \in B \\ 0 & \text{otherwise.} \end{cases}$$

It follows easily from Proposition 6.4 that for all $x \in A$, $E[X_x] = \rho(B)$ and the X_x 's are pairwise independent.

Let $X = \sum_{x \in A} X_x$. Then $E[X] = |A|\rho(B)$ and by definition, for each $h \in H$,

$$\begin{aligned} X(h) &= |\{x \in A | h(x) \in B\}| \\ &= |A|Pr_{x \in A}[h(x) \in B]. \end{aligned}$$

Now we have:

$$\begin{aligned} Pr_{h \in H}[h \text{ is not } (A, B, \epsilon)\text{-good}] &= Pr_{h \in H}[|Pr_{x \in \{0, 1\}^p}[x \in A \text{ and } h(x) \in B] - \rho(A)\rho(B)| > \epsilon] \\ &= Pr_{h \in H}[||A|Pr_{x \in A}[h(x) \in B] - |A|\rho(B)| > \epsilon 2^p] \\ &= Pr_{h \in H}[|X - E[X]| > \epsilon 2^p] \\ &\leq \frac{Var(X)}{\epsilon^2 2^{2p}} \\ &= \frac{\sum_{x \in A} Var(X_x)}{\epsilon^2 2^{2p}} \\ &= \frac{\rho(A)\rho(B)(1 - \rho(B))}{\epsilon^2 2^p}, \end{aligned}$$

where the inequality follows from Chebyshev's inequality, and the second to last equality follows from the fact that the X_x 's are pairwise independent. \square

Now suppose H is a family of functions that map $\{0, 1\}^m$ to $\{0, 1\}^m$ and Q is a finite state machine of type (d, m) . We say that an $h \in H$ is (Q, ϵ) -good if for all $i, j, k \in [d]$, h is $(Q[i, j], Q[j, k], \epsilon)$ -good.

Lemma 6.5 *Let H be a universal hash function family that maps $\{0, 1\}^m$ to $\{0, 1\}^m$ and let Q be a finite state machine of type (d, m) . Then for any $\epsilon > 0$,*

$$Pr_{h \in H}[h \text{ is not } (Q, \epsilon)\text{-good}] \leq \frac{d}{\epsilon^2 2^m}.$$

Proof:

$$\begin{aligned} Pr_{h \in H}[h \text{ is not } (Q, \epsilon)\text{-good}] &\leq \sum_{i, j, k \in [d]} Pr_{h \in H}[h \text{ is not } (Q[i, j], Q[j, k], \epsilon)\text{-good}] \\ &\leq \sum_{i, j, k \in [d]} \frac{\rho(Q[i, j])\rho(Q[j, k])}{\epsilon^2 2^m} \\ &\leq \frac{d}{\epsilon^2 2^m}, \end{aligned}$$

where the second inequality follows from Lemma 6.4, and the last one follows from the fact that for any fixed j , $\sum_{k \in [d]} \rho(Q[j, k]) \leq 1$ and for any fixed i , $\sum_{j \in [d]} \rho(Q[i, j]) \leq 1$. \square

6.2.3 The Descriptions of the Generator Family

Consider the finite field F on 2^m elements. It is well known that it is possible to give an efficient encoding of the elements of F by strings in $\{0, 1\}^m$ so that field addition and multiplication can be done in space $O(m)$. (The main thing that we need for this is an irreducible polynomial of degree m over $GF(2)$, which can be constructed in space $O(m)$. We refer the reader to [LN86] for more background on finite fields.) Fix such an encoding. Having done this, in what follows, we view the elements of F^k as binary strings of length mk and vice versa.

We are going to associate to each sequence $\vec{h} = (h_1, h_2, \dots, h_r)$ where $h_k = (a_k, b_k) \in F^2$, a function $G^{\vec{h}}$ which maps F to F^{2^r} . The final family of $(m, 2^r)$ -generators is defined to be $\{G^{\vec{h}} \mid \vec{h} \in (F^2)^r\}$.

First, to each $h = (a, b) \in F^2$ we associate a function $f_h : F \rightarrow F$ given by $f_h(x) = ax + b$. In addition, we associate to each $h \in F^2$ an $(m, 2)$ -generator G^h defined as follows: for $x \in \{0, 1\}^m$, $G^h(x) = x f_h(x)$. Finally, for each $\vec{h} = (h_1, h_2, \dots, h_r) \in (F^2)^r$, we define $G^{\vec{h}} = G^{h_1} \circ G^{h_2} \circ \dots \circ G^{h_r}$.

An alternative description of $G^{\vec{h}}$ is given below, whose equivalence to the above description is not difficult to verify.

If h_1, h_2, \dots, h_k is a sequence of elements in F^2 , we define the function f_{h_1, h_2, \dots, h_k} to be the composition of $f_{h_1}, f_{h_2}, \dots, f_{h_k}$, i.e., for $x \in F$,

$$f_{h_1, h_2, \dots, h_k}(x) = f_{h_1}(f_{h_2}(\dots(f_{h_k}(x))\dots)).$$

For the empty sequence λ , f_λ is defined to be the identity function.

Let $\vec{h} = (h_1, h_2, \dots, h_r) \in (F^2)^r$ and ℓ be an integer in the range $\{0, 1, \dots, 2^r - 1\}$. Suppose $1 \leq \ell_{i_1} < \ell_{i_2} < \dots < \ell_{i_k} \leq r$ are the positions of the 1's in the binary expansion of ℓ (so that $\ell = \sum_{i=1}^k 2^{\ell_{i_i}-1}$), we denote by \vec{h}_ℓ the subsequence $h_{\ell_1}, h_{\ell_2}, \dots, h_{\ell_k}$ of \vec{h} .

Finally, we complete this alternative description by defining $G^{\vec{h}}$ to be the $(m, 2^r)$ -generator that maps $x \in \{0, 1\}^m$ to a sequence y_0, \dots, y_{2^r-1} of 2^r blocks of strings in $\{0, 1\}^m$, where y_ℓ is defined to be $f_{\vec{h}_\ell}(x)$ for $\ell = 0, 1, \dots, 2^r - 1$.

6.2.4 The Properties of the Generator Family

Let $H = \{f_h \mid h \in F^2\}$. It is a well known and easily proved fact that H is a universal hash function family that maps F to F (or equivalently, maps $\{0, 1\}^m$ to $\{0, 1\}^m$).

Lemma 6.6 *Let Q be a finite state machine of type (d, m) . Suppose for $\epsilon > 0$, $h \in F^2$ is such that $f_h \in H$ is (Q, ϵ) -good. Then the $(m, 2)$ -generator G^h is ϵd^2 -pseudorandom with respect to Q .*

Proof: We want to prove $\|(Q^2)^* - Q_{G^h}^*\| \leq \epsilon d^2$. Let $i, k \in [d]$ be two arbitrary indices. It suffices to upper bound $|(Q^2)^2[i, k] - Q_{G^h}^2[i, k]|$ by ϵd .

By definition, $Q_{G^h}[i, k] = \{x \in \{0, 1\}^m \mid \exists j \in [d] \text{ such that } x \in Q[i, j] \text{ and } f_h(x) \in Q[j, k]\}$. So we have

$$\begin{aligned}
& |(Q^*)^2[i, k] - Q_{G^h}^*[i, k]| \\
&= \left| \sum_{j \in [d]} \rho(Q[i, j])\rho(Q[j, k]) - \sum_{j \in [d]} Pr_{x \in \{0, 1\}^m} [x \in Q[i, j] \text{ and } f_h(x) \in Q[j, k]] \right| \\
&\leq \sum_{j \in [d]} |\rho(Q[i, j])\rho(Q[j, k]) - Pr_{x \in \{0, 1\}^m} [x \in Q[i, j] \text{ and } f_h(x) \in Q[j, k]]| \\
&\leq \epsilon d,
\end{aligned}$$

where the last inequality is by the assumption that f_h is (Q, ϵ) -good. \square

For a finite state machine Q of type (d, m) and $\epsilon > 0$, a sequence $\vec{h} = (h_1, h_2, \dots, h_r) \in (F^2)^r$ is said to be (Q, ϵ) -well-behaved if for each $1 \leq i \leq r$, f_{h_i} is $(Q_{G^{h_1 \circ G^{h_2} \circ \dots \circ G^{h_{i-1}}}}, \epsilon)$ -good. Applying Lemma 6.5, a straightforward induction shows that:

Proposition 6.5 *For any finite state machine Q of type (d, m) and $\epsilon > 0$, all but a fraction $\frac{rd}{\epsilon^2 2^m}$ of $\vec{h} \in (F^2)^r$ are (Q, ϵ) -well-behaved.*

We will need:

Lemma 6.7 *Let Q be a finite state machine of type (d, m) and let $\epsilon > 0$. If $\vec{h} \in (F^2)^r$ is (Q, ϵ) -well-behaved, then $G^{\vec{h}}$ is $(2^r \epsilon d^2)$ -pseudorandom with respect to Q .*

Proof: We will prove by induction on r that $G^{\vec{h}}$ is $(\sum_{i=0}^{r-1} 2^i \epsilon d^2)$ -pseudorandom with respect to Q . This will clearly be sufficient for the proof of the lemma.

The case where $r = 1$ follows immediately from Lemma 6.6. Assume it is true for $r - 1$ and we show that the statement holds for r ($r \geq 2$).

Let $\vec{h} = (h_1, \dots, h_{r-1}, h_r) \in (F^2)^r$ and let $\vec{h}' = (h_1, \dots, h_{r-1}) \in (F^2)^{r-1}$. By definition, $G^{\vec{h}} = G^{h_1} \circ \dots \circ G^{h_{r-1}} \circ G^{h_r} = G^{\vec{h}'} \circ G^{h_r}$.

Since \vec{h} is (Q, ϵ) -well-behaved, by definition, we have that (i) \vec{h}' is also (Q, ϵ) -well-behaved; and (ii) f_{h_r} is $(Q_{G^{\vec{h}'}}, \epsilon)$ -good. Now by (i) and the induction hypothesis, $G^{\vec{h}'}$ is $(\sum_{i=0}^{r-2} 2^i \epsilon d^2)$ -pseudorandom with respect to Q ; by (ii) and Lemma 6.6, G^{h_r} is ϵd^2 -pseudorandom with respect to $Q_{G^{\vec{h}'}}$.

Then, Lemma 6.3 says that $G^{\vec{h}'} \circ G^{h_r}$ is γ -pseudorandom with respect to Q , where $\gamma = \epsilon d^2 + 2 \sum_{i=0}^{r-2} 2^i \epsilon d^2 = \sum_{i=0}^{r-1} 2^i \epsilon d^2$. This concludes the proof. \square

Finally we prove the main technical result of Nisan in [Nis90]:

Lemma 6.8 *Let m, d, r be integers and $\epsilon > 0$. There is an explicit family $\{G^{\vec{h}} \mid \vec{h} \in \{0, 1\}^{2mr}\}$ of $(m, 2^r)$ -generators that has the following property:*

1. *For any finite state machine Q of type (d, m) (and therefore for any substochastic matrix M of type (d, m)), all but a fraction $\beta = \frac{rd^5 2^{2r}}{\epsilon^2 2^m}$ of the generators $G^{\vec{h}}$ in the family are ϵ -pseudorandom with respect to Q (resp. M).*

2. There is an algorithm which, given input \vec{h} , $\alpha \in \{0, 1\}^m$ and $\ell \in \{0, 1, \dots, 2^r - 1\}$, computes the ℓ -th block of $G^{\vec{h}}(\alpha)$ in space $O(m + r)$.

Consider the family of generators defined in Section 6.2.3. Then the first part of the lemma clearly follows from Proposition 6.5 and Lemma 6.7. To see the second part of the lemma, let us recall the second description of $G^{\vec{h}}$: for any $x \in \{0, 1\}^m$ and any $0 \leq \ell \leq 2^r - 1$, $G^{\vec{h}}_\ell(x)$ is defined to be $f_{\vec{h}_\ell}(x)$, which obviously is computable in space $O(m + r)$.

6.3 The PRS Algorithm

Finally, we give the description of the PRS algorithm and give the correctness proof of Lemma 4.1.

Algorithm PRS

Input: a $d \times d$ substochastic matrix M of type (d, m) , integers r, m ; indices $i, j \in [d]$;

Offline Random Input: $\vec{h} \in \{0, 1\}^{2mr}$;

Set $count \leftarrow 0$;

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

If $G^{\vec{h}}(\alpha)$ maps i to j in $Q(M)$ then $count \leftarrow count + 1$;

Output: $count/2^m$

Let us see that Lemma 4.1 follows.

Proof of Lemma 4.1: It is easy to verify the fact that if we run the algorithm for all $i, j \in [d]$, PRS computes the matrix $(Q(M))_G^*$ where $G = G^{\vec{h}}$.

From the forgoing discussion, it is clear that the matrix $(Q(M))_G^*$ is substochastic and has dimension d . Furthermore, from the first part of Lemma 6.8, we can see that for a randomly chosen \vec{h} , with probability at least $1 - \frac{2^{2a+3r+5\log d}}{2^m}$, G is 2^{-a} -pseudorandom with respect to M . However, if G is 2^{-a} -pseudorandom with respect to M then $\|M^{2^r} - (Q(M))_G^*\| \leq 2^{-a}$ by Lemma 6.1. Therefore, we conclude that the algorithm approximates $\Lambda^{(r)}(M)$ with accuracy a and error probability $\frac{2^{2a+3r+5\log d}}{2^m}$.

Let us examine the space requirements. For each $\alpha \in \{0, 1\}^m$, we want to determine whether or not $G^{\vec{h}}(\alpha)$ maps i to j in $Q(M)$. By the second part of Lemma 6.8, we can compute each successive block $G^{\vec{h}}_\ell(\alpha)$ for $\ell = 0, 1, \dots, 2^r - 1$ in space $O(m + r)$ and by Proposition 6.3, for each block we can determine which state it goes to in space $O(m + \log d)$. Thus the overall space to determine, for a fixed α , whether $G^{\vec{h}}(\alpha)$ maps i to j is at most $O(m + r + \log d)$. Enumerating over all $\alpha \in \{0, 1\}^m$ and counting take no more than $O(m)$ space. So the overall space needed is $O(m + r + \log d)$. The proof of the lemma is complete. \square

7 Approximating Substochastic Matrix Exponentiation

The *Approximate Substochastic Matrix Exponentiation* problem is the generalization of the AMRS problem in which we want to approximate an arbitrary integer power of a substochastic matrix. The formal statement of the problem is as follows:

Approximate Substochastic Matrix Exponentiation (AME)

Input: a $d \times d$ substochastic matrix M , integers p and 2^a in unary.

Output: a $d \times d$ substochastic matrix M' such that $\|M' - M^p\| \leq 2^{-a}$.

Let $s = \max\{\log p, a, \log d\}$. Using the algorithm AMRS as a black-box, we sketch an algorithm for AME with space complexity $O(s \log^{1/2} p)$.

Let t be a positive integer. Suppose F_1, F_2 are two substochastic matrix functions whose ranges are square substochastic matrices with at most t bits per entry. Let G be the substochastic matrix function such that $G(M, z_1, z_2) = \lfloor F_1(M, z_1)F_2(M, z_2) \rfloor_t$. The following fact is easily seen:

Proposition 7.1 *If F_1, F_2 are computable in space S_1, S_2 respectively, then G is computable in space $\max\{S_1, S_2\} + O(t + \log d)$.*

Let $p(i)$, $0 \leq i \leq \lfloor \log p \rfloor$, be the i -th coordinate of the binary expansion of p , i.e., $p = \sum_{i=0}^{\lfloor \log p \rfloor} p(i)2^i$. Then we have $M^p = \prod_{k=0}^{\lfloor \log p \rfloor} M^{p(k)2^k}$. We define a substochastic matrix function F with parameters M, t, p and $[i, j]$ where $0 \leq i \leq j \leq \lfloor \log p \rfloor$, such that $F(M, t, p, [i, j])$ is a substochastic matrix with at most t bits per entry and is supposed to estimate the matrix $\prod_{k=i}^j M^{p(k)2^k}$. The definition of F is recursive:

$$\begin{aligned} F(M, t, p, [i, i]) &= \lfloor AMRS(M, p(i)2^i, 2^t) \rfloor_t \text{ and} \\ F(M, t, p, [i, j]) &= \lfloor F(M, t, p, [i, \lfloor (i+j)/2 \rfloor]) F(M, t, p, [\lfloor (i+j)/2 \rfloor + 1, j]) \rfloor_t \text{ for } i < j. \end{aligned}$$

For some $t = O(s)$, define $AME(M, p, 2^a) = F(M, t, p, [0, \lfloor \log p \rfloor])$. It is helpful to view the recursive computation of $AME(M, p, 2^a)$ as the depth-first traversal on a labelled binary tree defined in the following way: The root of the tree is labelled by $AME(M, p, 2^a) = F(M, t, p, [0, \lfloor \log p \rfloor])$. For a node with label $F(M, t, p, [i, j])$, which we call the $[i, j]$ -node, if $i < j$ then it is an internal node and has two children labelled by $F(M, t, p, [i, \lfloor (i+j)/2 \rfloor])$ and $F(M, t, p, [\lfloor (i+j)/2 \rfloor + 1, j])$ respectively; otherwise (i.e., $i = j$) it is a leaf. An internal node computes its labelling function as the t -bit truncation of the product of the outcomes of its two children. It is easily seen that the tree has $\lfloor \log p \rfloor + 1$ leaves and for any $[i, j]$ -node in the tree, the subtree rooted at the node has height $\lceil \log(j - i + 1) \rceil$.

We have the following facts:

Proposition 7.2 (1) *For $i \leq j$,*

$$\|F(M, t, p, [i, j]) - \prod_{k=i}^j M^{p(k)2^k}\| \leq (j - i + 1)(d + 1)2^{-t} + (j - i)d2^{-t}.$$

(2) *The space complexity to compute $F(M, t, p, [i, j])$ is upper bounded by the sum of $\lceil \log(j - i + 1) \rceil O(t + \log d)$ and the space needed to compute $AMRS(M, 2^j, 2^t)$.*

Proof: We prove these two facts by induction on $j - i$.

For fact (1), the basis $j - i = 0$ can be easily verified using the definition of $F(M, t, p, [i, i])$ and Proposition 2.4 (1). Let us show the inductive step. We have that

$$\begin{aligned}
& \|F(M, t, p, [i, j]) - \prod_{k=i}^j M^{p(k)2^k}\| \\
&= \| \lfloor F(M, t, p, [i, \lfloor (i+j)/2 \rfloor]) F(M, t, p, [\lfloor (i+j)/2 \rfloor + 1, j]) \rfloor_t - \prod_{k=i}^j M^{p(k)2^k} \| \\
&\leq \|F(M, t, p, [i, \lfloor (i+j)/2 \rfloor]) F(M, t, p, [\lfloor (i+j)/2 \rfloor + 1, j]) \\
&\quad - \prod_{k=i}^{\lfloor (i+j)/2 \rfloor} M^{p(k)2^k} \prod_{k=\lfloor (i+j)/2 \rfloor + 1}^j M^{p(k)2^k}\| + d2^{-t} \\
&\leq \|F(M, t, p, [i, \lfloor (i+j)/2 \rfloor]) - \prod_{k=i}^{\lfloor (i+j)/2 \rfloor} M^{p(k)2^k}\| \\
&\quad + \|F(M, t, p, [\lfloor (i+j)/2 \rfloor + 1, j]) - \prod_{k=\lfloor (i+j)/2 \rfloor + 1}^j M^{p(k)2^k}\| + d2^{-t} \\
&\leq (\lfloor (i+j)/2 \rfloor - i + 1)(d+1)2^{-t} + (\lfloor (i+j)/2 \rfloor - i)d2^{-t} \\
&\quad + (j - \lfloor (i+j)/2 \rfloor)(d+1)2^{-t} + (j - \lfloor (i+j)/2 \rfloor - 1)d2^{-t} + d2^{-t} \\
&= (j - i + 1)(d+1)2^{-t} + (j - i)d2^{-t},
\end{aligned}$$

where the first inequality follows from Proposition 2.4 (1), the second from Proposition 2.2, and the third is by induction.

Fact (2) is not difficult to verify using Proposition 7.1. □

Thus by choosing $t = O(s)$, fact (1) says that $AME(M, p, 2^a) = F(M, t, p, [0, \lfloor \log p \rfloor])$ approximates $M^p = \prod_{k=0}^{\lfloor \log p \rfloor} M^{p(k)2^k}$ with accuracy 2^{-a} , and fact (2) implies that the space needed to compute $AME(M, p, 2^a)$ is dominated by the space needed to compute $AMRS(M, 2^{\lfloor \log p \rfloor}, 2^t)$, which is $O(s \log^{1/2} p)$. So we have

Theorem 7.1 *There is a deterministic algorithm for AME with space complexity $O(s \log^{1/2} p)$.*

Acknowledgement

We thank Eric Allender and Avi Wigderson for helpful comments, and Dieter Van Melkebeek and an anonymous referee for their corrections of earlier drafts.

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.
- [AO94] E. Allender, M. Ogihara, “Relationships among \mathbf{PL} , $\#\mathbf{L}$, and the determinant”, *Proc. 9th Conference on Structure in Complexity Theory*, 1994, pp. 267-279.

- [BNS89] L. Babai, N. Nisan, M. Szegedy, “Multipart protocols and logspace-hard pseudorandom sequences”, *Proc. 21st ACM Symposium on Theory of Computing*, 1989.
- [BCP83] A. Borodin, S. Cook, and N. Pippenger, “Parallel computation and well-endowed rings and space-bounded probabilistic machines”, *Information and Control* (58), 1983, 113-136.
- [CG86] B. Chor and O. Goldreich, “On the power of two point based sampling”, *Journal of Complexity*, 5:96-106, 1989.
- [CW79] L. Carter and M. Wegman, “Universal hash functions”, *J. Comp. and Syst. Sci.*, 18(2):143-154, 1979.
- [Gil77] J. Gill, “Computational complexity of probabilistic Turing machines”, *SIAM J. Computing* **6** (1977) 675-695.
- [Jun81] H. Jung, “Relationships between probabilistic and deterministic tape complexity”, in *10th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, 118, Springer-Verlag, 339-346.
- [LP82] H. Lewis and C. Papadimitiou, “Symmetric space-bounded computation”, *Theoretical Computer Science*, 19:161-187, 1982.
- [LN86] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and their applications*, Cambridge University Press, 1986.
- [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.
- [Pap94] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.
- [Sak96] M. Saks, “Randomization and Derandomization in Space-Bounded Computation”, *11th Annual Conference on Computational Complexity*, 1996.
- [Sav70] W.J. Savitch, “Relationships between nondeterministic and deterministic space complexities,” *J. Comp. and Syst. Sci.*, 4(2):177-192, 1970.