REDUCING RANDOMNESS IN COMPUTATION VIA EXPLICIT CONSTRUCTIONS

BY SHIYU ZHOU

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Professor Michael Saks

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 1996

ABSTRACT OF THE DISSERTATION

Reducing Randomness in Computation via Explicit Constructions

by Shiyu Zhou

Dissertation Director: Professor Michael Saks

One of the important issues in both complexity theory and algorithm design is to understand whether randomness can be more powerful than determinism. In this dissertation, we explore some problems related to this issue and study how to reduce randomness in computation via explicit combinatorial constructions.

One approach to achieving determinism from randomness is by way of applying pseudorandom generators. We examine the relationship between constructing pseudorandom generators for space bounded computation and designing space-efficient deterministic algorithms for approximating matrix powering. As a consequence, we show that any randomized algorithm that runs in space $O(S)$ and terminates can be simulated deterministically in space $O(S^{3/2})$, where the previously best known simulation required space $O(S^2)$ following Savitch's Theorem.

Deterministic amplification is one of the major subjects in the study of reducing randomness. An important issue in randomized computation that is essentially equivalent to deterministic amplification is to design randomized algorithms that are robust with respect to deviations of the random source from true randomness. It has been observed that this issue is closely connected to the problem of efficiently constructing

graphs with certain expansion properties, called dispersers. We give an improved construction of a family of dispersers that can be used to design randomized algorithms achieving maximal robustness. Our construction settles a conjecture of Sipser that the complexity class *Strong Randomized Polynomial Time* (Strong-RP) is equivalent to *Randomized Polynomial Time* (RP).

Our effort at constructing small-sized sample spaces that keep certain randomness properties was motivated by designing efficient deterministic algorithms via derandomization. Here we look at the problem from a different perspective and investigate the application of sample spaces with small bias to designing error-correcting communication protocols. We study the issue of constructing sample spaces with small bias on certain sets of bit positions. By way of derandomization, we give a new construction whose size matches the bound given by the probabilistic argument, improving on the previously best known result. As an application of our improved construction, we present an essentially optimal communication protocol for performing a type of data consistency check in distributed databases.

Acknowledgements

First, I would like to acknowledge my indebtedness to my advisor Michael Saks; the work presented in this thesis could not have been done without his help, guidance and collaboration. His invaluable advice over these past three years has had a major impact on my development as a researcher as well as an individual. I am very grateful to him for the nurturing, encouragement and support he has provided me.

I wish to thank Endre Szemerédi for introducing me to the subject of randomized computation. I have benefited greatly from the numerous meetings that I have had with him, during the course of which I was influenced by his insights into many research problems.

It has been my pleasure and privilege to work with Avi Wigderson over the past one year. I am grateful to him for sharing with me his appreciation of mathematics and enthusiasm for research, as well as for advising me on both academic and non-academic matters.

I have enjoyed and learned much from the classes Eric Allender and Vašek Chvátal have taught, and from the discussions I have had with them. The first time I had the opportunity to study computational complexity in a systematic way was in Eric's class on the Foundations of Computer Science; the first research problem I have ever worked on was posed to me by Vašek when I took an independent study course with him.

One of the best aspects of the Computer Science program at Rutgers is that it provides students with the opportunity to take many wonderful courses in theoretical computer science and discrete mathematics. I thank József Beck, Mike Grigoriadis, Jeff Kahn, János Komlós, Gábor Tardos and Ann Yasuhara for offering these courses, which have been a tremendous learning experience for me.

I have had the opportunity to work with many great people and I am thankful

to all of them, especially Roy Armoni, Vivek Gore, Daniel Kudenko, Omer Reingold, Aravind Srinivasan (who also collaborated with me on the result described in Chapter 3), Martin Strauss and David Zuckerman.

During my graduate study I was supported for part of the time as a graduate research assistant under NSF grant CCR-9215293, and also by the Center for Discrete Mathematics and Computer Science (DIMACS) which is funded by NSF grant NSF-STC91-19999 and by the New Jersey Commission on Science and Technology.

Finally, I wish to thank my grandma and my parents for their love, understanding and inspiration through all these years away from home. This thesis is dedicated to them.

Dedication

To my grandma and my parents.

Table of Contents

Chapter 1

Introduction

Since its emergence in the late 1970's, randomized computation has become one of the major research fields in Computer Science. The use of randomization has played an important role in both complexity theory and algorithm design. Its apparent success in application has motivated a substantial effort to understand how much power randomness can provide over determinism.

For many problems, a randomized algorithm is by far faster (e.g. primality testing [SS77, Rab80, AH87]), more parallel (e.g. perfect matching construction [KUW86, MVV87]), more space-efficient (e.g. undirected graph connectivity [AKLLR79]), or simpler (e.g. the min-cut problem [Kar93]) than any known deterministic algorithm. In fact, there are many problems such as primality testing, perfect matching construction and undirected graph connectivity which (in certain respects) have very efficient randomized algorithms, but are not known to have corresponding deterministic counterparts.

In spite of its seemingly enhanced power, as far as we know, randomness might not provide any computational advantage over determinism. During the past decade, complementing the study of randomization, powerful techniques for reducing randomness have been developed and become effective tools in deterministic algorithm design and deterministic amplification. In this thesis, continuing the investigation of understanding the power of randomness, we study the issue of reducing randomness in computation via explicit combinatorial constructions.

1.1 Reducing Randomness in Computation

A randomized algorithm in one way can be viewed as containing two elements: a deterministic procedure and a random source. The random source of the algorithm is a family of distributions, each on a set of strings (*sample space*) corresponding to a different input length. The deterministic procedure has two inputs: the real input to the algorithm and a read-only auxiliary string coming from the random source to which the procedure has only one-way access. On an input, the randomized algorithm first requests the random source to provide an auxiliary string according to the distribution from the sample space that corresponds to the input length, and then executes the deterministic procedure on both the input and the auxiliary string. At any step of the computation, the deterministic procedure may read the next bit of the auxiliary string (*random bits*). It is generally assumed that the auxiliary random string consists of unbiased mutually independent bits. That is, the random source is perfect. The performance (*success probability*) of the execution of the algorithm then depends on the provable fact that there are a large proportion of points in the sample space that are "good" in the sense that the deterministic procedure would give the right answer on these points. We refer the reader to the textbook [MR95] for more background on randomized algorithms.

Generally speaking, reducing randomness has two broad meanings: deterministic amplification and derandomization. The objective of deterministic amplification is to achieve high success probability while minimizing the number of random bits, and the goal of derandomization is to eliminate randomness from computation.

There are two basic methods that are employed in deterministic amplification. The first method is to construct pseudorandom generators [Yao82, BM84, NW88, BNS89], which are functions that map short random seeds to much longer strings so that any randomized algorithm of a certain type cannot distinguish statistically between the output strings from the generators and the truly random strings. For polynomial time computation, unfortunately, the currently available constructions of pseudorandom generators

all rely on certain unproven assumptions. Nevertheless, for space-bounded computation, successful constructions of pseudorandom generators have been obtained. We refer the reader to [Sak96] for a survey on the latter subject. The most often used method in deterministic amplification, on the other hand, is to build a type of bipartite expander called dispersers [Sip88] (or its variant called extractors [NZ93]). The application of this approach gives extremely high success probability using only a moderate number of random bits. We refer the reader to [Nis96] for a survey on this method.

With respect to polynomial time randomized computation, there are two fundamental approaches to derandomization: the method of conditional probabilities [ES73, Spe87, Rag88] and constructing small-sized sample spaces [KW84, Lub85, ABI86, AKS87, NN90]. In the former approach, we search for a good point in a large sample space by improving certain conditional probabilities (or expectations) in an adaptive manner; in the latter method, we construct a sample space of polynomial size while guaranteeing the existence of a good point so that we could accomplish finding such a point by exhaustive search. The most commonly used tools for constructing small-sized sample spaces are: k-wise independent hashing [CW79], sample spaces with limited independence [KW84, Lub85, ABI86], sample spaces with small bias [NN90], and expander graphs [AKS87].

Remark: Here we have assumed that the polynomial time randomized computation has *one-side* error, in which case, finding a good sample point is sufficient for derandomization. In fact, for almost all the problems that are known to have randomized algorithms, their corresponding computations are of, or can be converted to, this type.

For space-bounded randomized computation, on the other hand, the main approach to derandomization which has been proven effective is the application of pseudorandom generators (see e.g. [Nis92, NSW92, NZ93, SZ95]).

1.2 Explicit Constructions

Explicit constructions are also called efficient constructions. Generally, the success of reducing randomness by way of constructing combinatorial objects does not merely depend on the existence of the objects, but mainly depends on the efficiency of the constructions.

A typical combinatorial construction is a function that maps binary strings of length l to binary strings of length m for some l and m depending on the construction itself. The combinatorial object built by the construction consists of all the domain-image pairs of the function, and therefore has output size $2^l(l+m)$. For this discussion, we call such a construction an (l,m)-construction.

Depending on its application, the construction of a combinatorial object can be defined in different ways. For example, consider the cases where we want to construct a graph of a certain type. If we need to know the neighborhood of an arbitrarily given vertex, the construction of the graph should be defined as a function that computes the neighboring node on input the index of a vertex and the index of one of its incident edges. On the other hand, if the edge layout of the graph is requested, we should define the construction as a Boolean function that takes the indices of two vertices and outputs 1 if and only if there is an edge between them. In a sense, in the former case we output the graph in the form of the its adjacency list; while in the latter one we output its adjacency matrix.

Next we discuss the efficiency issue in applying combinatorial constructions to reducing randomness in computation. Generally, there are two efficiency requirements for an (l,m)-construction in this context, depending on the type of the randomized algorithms to which we would apply the construction.

The first and most fundamental requirement is that the total computation time used by the construction to build the entire combinatorial object be polynomial in the size of the output, i.e., polynomial in $2^l(l+m)$. To see that this requirement is useful, let us consider an example in which we apply small-sized sample space constructions to derandomizing (sequential) polynomial time randomized algorithms.

Suppose that we are given a randomized polynomial time algorithm A (with one-sided error) and an input x of length n. Let $m = m(n)$ be the number of random bits requested by the computation of A on x, and let $T_D(n)$ be the running time of the deterministic procedure of A on x and an auxiliary random string. Clearly, both m and $T_D(n)$ are polynomial in n by assumption. Now suppose that we could obtain an (l, m)-construction whose image is a sample space that guarantees the existence of a good sample point; moreover, suppose that the total computation time needed by the construction is $T_C(n)$. Then by exhaustive search, i.e., by testing all the points in the constructed sample space using the deterministic procedure of A, we could find the good point and therefore derandomize the computation of A on x in total time of order $T_C(n) + 2^l T_D(n)$. In particular, if the sample space has polynomial size and the construction satisfies the first efficiency requirement (so $l = O(\log n)$ and $T_C(n) = n^{O(1)}$), then the derandomization can be accomplished in polynomial time.

One extreme case where we may employ this derandomization framework is that, as we know from a counting argument, there exists an $(O(\log n), n^{O(1)})$-construction of a sample space that can be used to derandomize any polynomial time randomized computation on inputs of length n. An explicit construction of such a sample space that satisfies the above efficiency requirement would then imply that randomness does not provide more than polynomial advantage in computation (at least).

It is worth mentioning that the application of the method of conditional probabilities can be thought of as an (l, m)-construction whose corresponding combinatorial object is one single string: a good point in the sample space $\{0,1\}^m$. Therefore $l = 0$. In this case, a construction that satisfies the requirement would give a procedure to find a good sample point in time polynomial in m, which consequently would imply that the corresponding randomized computation can be derandomized in polynomial time.

The second and stronger requirement for an (l, m)-construction for the purposes of reducing randomness in computation is that, for a given l-bit string α and the index of any bit position between 1 and m, the value of the corresponding bit in the output of the construction on input α should be computable in time polynomial in l (and in some occasions, in space linear to l as well). Usually, constructions that satisfy this

requirement are especially useful for designing or derandomizing parallel algorithms (see e.g. [KW84, Lub85, ABI86, NN90]), and for constructing pseudorandom generators for space bounded computation [BNS89, Nis90, NZ93, INW94, AW95, ASWZ96].

As a matter of fact, most of the known explicit constructions satisfy the second requirement (and thus satisfy the first as well). These include almost all the explicit constructions of pseudorandom generators (e.g. [NW88, Nis90]), expanders (e.g. [GG81, LPS86]), dispersers (e.g. [CW89, SSZ95]), extractors (e.g. [NZ93, TaS96]), k-wise independent sample spaces (e.g. [Lub85, ABI86]), small-bias sample spaces (e.g. [NN90, AGHP91]), etc. On the other hand, explicit constructions obtained via using the method of conditional probabilities (e.g. [Rag88, SZ96]) or solving linear constraints (e.g. [Sch92, KK94]) typically satisfy the first requirement but not necessarily the second.

1.3 The Organization of the Thesis

So far we have summarized the general approaches to reducing randomness in computation via explicit constructions. In this thesis, we examine these methods to a greater extent by formalizing the frameworks for these approaches and demonstrating the applications of these methods to a variety of problems related to deterministic amplification and derandomization.

The thesis consists of three major chapters. Each chapter deals with a different problem and the presentation in each chapter is self-contained.

In Chapter 2, we investigate the relationship between the construction of pseudorandom generators for space-bounded computation and the design of space-efficient deterministic algorithms for approximating matrix exponentiation. We present a new algorithm that approximates 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 [Nis90]. As a consequence, we show that any randomized algorithm (with possibly

two-sided error) that runs in space $O(S)$ and always halts can be simulated deterministically in space $O(S^{3/2})$, i.e., $BP_HSPACE(S) \subseteq DSPACE(S^{3/2})$. 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 [Sav70] and for two-sided error algorithms follows by reduction to recursive matrix exponentiation. Our result includes as a special case the result due to Nisan, Szemerédi and Wigderson [NSW92] that undirected graph connectivity can be computed in space $O(\log^{3/2} n)$.

In Chapter 3 we examine the connection between deterministic amplification and the disperser construction. One problem that is closely related to deterministic amplification is to design randomized algorithms that are robust under the presence of imperfect random sources. This problem itself is an important issue in randomized computation and has been studied extensively during the past decade (see e.g. [SV86, CG88, Zuc90]). Using the techniques developed in the study of computing with imperfect sources, we give an improved construction of a family of dispersers that can be applied to designing randomized polynomial time algorithms that achieve maximal robustness. As a result, we show that every randomized polynomial time algorithm has an extremely efficient amplification scheme. In particular, we resolve a conjecture posed by Sipser [Sip88] that the complexity class *Strong Randomized Polynomial Time* (Strong-RP) is equivalent to *Randomized Polynomial Time* (RP).

In our final chapter, Chapter 4, we study the problem of constructing sample spaces with small bias on designated sets of bit positions (neighborhoods). We give a construction that improves the previously best result which can be derived from the explicit construction of small-bias sample spaces in [NN90]. The size of our new construction matches the bound given by the probabilistic argument, and leads to the best efficient construction of k-wise small-bias sample spaces for fixed k. Our construction is obtained by way of derandomization and employs the explicit construction of sample spaces with small bias and the method of conditional probabilities. As an application of our construction, we study a variant of the equality problem in communication complexity where two communicating parties with limited computation power try to

recognize certain difference patterns on their private inputs. We present an essentially optimal error-correcting communication protocol for this problem.

The results in Chapter 2 and Chapter 4 appeared in [SZ95] and [SZ96], respectively, and are joint work with Michael Saks. The results in Chapter 3 appeared in [SSZ95] and are joint work with Michael Saks and Aravind Srinivasan.

Chapter 2

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

2.1 Matrix Repeated Squaring and Randomized Space Computation

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 chapter 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. $R_H SPACE(S(n))$ (resp. $BP_H SPACE(S(n))$) 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, $R_H SPACE(S(n)) \subseteq BP_H SPACE(S(n))$.

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 2.1.1 *Let $S(n)$ be a proper complexity function*[1] *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

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

performance of exact rather than approximate matrix computations, is not generalized by Theorem 2.1.1.) 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., $R_H SPACE(\log n)$) 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 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 chapter is organized as follows. In Section 2.2 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 2.3. Then assuming the algorithm PRS is given, we present our main approximation algorithm in Section 2.4 and give its correctness proof in Section 2.5. In Section 2.6, 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 2.7, 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 $O(\log d \log^{1/2} p)$.

2.2 Preliminary Definitions and Facts

2.2.1 Matrix Approximations

For integer d, we use $[d]$ to denote the set of integers $\{1, 2, \ldots, d\}$. 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 $x \in \mathrm{R}^d$, 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 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 $\|M\| = \sup\{\|xM\| \mid x \in \mathrm{R}^d \text{ such that } \|x\| = 1\}$.

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 2.2.1 *Let $M, N \in \mathrm{R}^{d \times d}$. 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 2.2.2 *Let M, N, M', N' be $d \times d$ substochastic matrices. Then $\|MN - M'N'\| \leq \|M - M'\| + \|N - N'\|$.*

Proof:

$$\|MN - M'N'\| \leq \|MN - M'N\| + \|M'N - M'N'\|$$

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

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. □

A corollary of the proposition is the following.

Proposition 2.2.3 *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* $\lfloor\ \rfloor_t$ 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 2.2.4 *Let $M, N \in \mathrm{R}^{d \times d}$, 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. $\|\Sigma_\delta(M) - \Sigma_\delta(N)\| \leq \|M - N\|$.

2.2.2 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 $F_1, F_2, \ldots, F_k$ is a sequence of matrix functions. For a square matrix M and a sequence of auxiliary parameters $z_1, z_2, \ldots, z_k$, we define a sequence of matrices $M_0, M_1, M_2, \ldots, M_k$ 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 $G(M, z_1, z_2, \ldots, z_k)$ evaluates to M_k. Thus G is essentially the composition of the matrix functions $F_1, F_2, \ldots, F_k$. Then the following fact is well known and easily verified:

Proposition 2.2.5 *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* $O(S_1 + S_2 + \ldots + S_k)$.

2.2.3 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 $A(x_1,\ldots,x_i;y_1,\ldots,y_j)$ if the true input to A is a list $x_1,\ldots,x_i$ of strings and the offline random input is a list $y_1,\ldots,y_j$ of strings. When accounting for space in an offline randomized algorithm, we explicitly divide the space requirements of the algorithm into two quanitities: 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 $B_1(x_1; y_1), B_2(x_2; y_2) \ldots$ 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 2.2.1 *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

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

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.

2.3 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 2.2.2, 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 2.3.1 *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 $O(\log^{3/2} d)$.*

2.4 The Algorithm

2.4.1 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 2.2.1 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 2.6. The relevant properties of this algorithm are summarized in the following:

Lemma 2.4.1 *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 $\Lambda^{(r_1)}$ 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 $M_0 = M, M_1, M_2, \ldots, M_{r_2}$ 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 $N_0 = M, N_1, N_2, \ldots, N_{r_2}$ where $N_i = N_{i-1}^{2^{r_1}}$, which is the exact sequence that $M_0, M_1, \ldots, M_{r_2}$ 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 $O(r_2 s)$ 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 $O(r^{1/2}s)$.

2.4.2 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 $[q(1), q(2), \ldots, q(r_2)]$, 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 $\vec{q} = [q(1), q(2), \ldots, q(r_2)]$ for each $q(i) \in \{0,1\}^D$;

Define the sequence of matrices

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

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

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

$$M_i \quad = \quad \lfloor M_i^{\Sigma} \rfloor_t$$

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 2.2.2), so we may apply Proposition 2.2.5 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 2.4.1, the space needed for the algorithm PRS other than the random bits, is $O(s)$. Thus the overall processing space can be bounded by $O(r_2 s)$ as required.

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

Theorem 2.4.1 *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 2.2.3 and

2.2.4, $\Lambda^{(r)}(M_0)$ 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 chapter we first prove Theorem 2.4.1. Then in Section 2.6, we examine in detail the connection between approximating substochastic matrix exponentiation and constructing pseudorandom generators, and describe the algorithm PRS and prove Lemma 2.4.1. This will complete the proof of Theorem 2.3.1 and Theorem 2.1.1.

2.5 The Correctness Proof

We will denote by $\Delta(M, K; \vec{h}, \vec{q})$, where $\vec{h} \in (\{0,1\}^{2m})^{r_1}$ and $\vec{q} = [q(1), q(2), \ldots, q(r_2)] \in (\{0,1\}^D)^{r_2}$, the sequence of matrices $[M_0; M_1^P, M_1^\Sigma, M_1; M_2^P, \ldots; M_{r_2}^P, M_{r_2}^\Sigma, M_{r_2}]$ 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 2.4.1. We will compare the sequence $\Delta(M, K; \vec{h}, \vec{q})$ of matrices to the sequence $\Gamma(M, K; \vec{q}) = [N_0; N_1^P, N_1^\Sigma, N_1; N_2^P, \ldots; N_{r_2}^P, N_{r_2}^\Sigma, N_{r_2}]$ 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 2.5.1 *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 2.5.1 *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 2.5.2 *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 2.5.2 *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 2.4.1 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 2.5.1 *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 2.5.2.

Lemma 2.5.3 *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 2.4.1.

Proof of Theorem 2.4.1: By Lemma 2.5.1, 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 2.5.2, and the second follows from Lemma 2.5.3 and Proposition 2.5.1. □

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

Proof of Lemma 2.5.1:

Definition 2.5.3 *A sequence of $d \times d$ matrices $[N_0, N_1, \ldots, N_p]$ is said to be (l,β)-*close *if for all $1 \leq i \leq p, \|N_i - N_{i-1}^l\| \leq \beta$.*

Lemma 2.5.4 *Let $[M = N_0, N_1, \ldots, N_p]$ be an (l,β)-close sequence of $d \times d$ substochastic matrices. Then*

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

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.

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

where the second inequality follows from the fact that the sequence is (l, β)-close and from Proposition 2.2.3, and the last inequality is by induction. □

We claim the sequence of matrices $[N_0, N_1, \ldots, N_{r_2}]$ 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 2.2.4 and the last inequality from the fact that $t = K - D$. Now Lemma 2.5.4 completes the proof. □

Proof of Lemma 2.5.2:

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 $\vec{h} \in \{0,1\}^{2mr_1}$ 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 2.2.4, which implies that for any $u, v \in [d]$,

$$|M_i^\Sigma[u,v] - N_i^\Sigma[u,v]| \leq 2^{-K}. \tag{2.1}$$

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 (2.1), $M_i^\Sigma[u,v] = \lfloor N_i^\Sigma[u,v] \rfloor_t + \rho'$ 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 (2.1) 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. □

Proof of Lemma 2.5.3:

We want to upper bound the probability μ, with respect to the random choice

of $\vec{q} = [q(1), q(2), \ldots, q(r_2)] \in (\{0,1\}^D)^{r_2}$, 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 $[N_0; N_1^P, N_1^\Sigma, N_1; N_2^P, \ldots; N_{r_2}^P, N_{r_2}^\Sigma, N_{r_2}]$ is completely determined by $\vec{q}$.

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

Now, by the definition of N_i^Σ, $\mu_i = Pr[\Sigma_{q(i)2^{-K}}(N_i^P)$ is (K,t)-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 $\Sigma_{q(i)2^{-K}}(W)$ 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 $q(1), q(2), \ldots, q(i-1)$, while the event $T_i(W)$ depends only on $q(i)$, so these events are independent. So finally it suffices to prove:

Lemma 2.5.5 *For any fixed substochastic matrix W of dimension d, the probability with respect to $q \in \{0,1\}^D$ that $\Sigma_{q2^{-K}}(W)$ is (K,t)-dangerous is at most $2d^2 2^{-D}$.*

Proof: We can bound the desired probability by d^2 times the probability that any particular entry of $\Sigma_{q2^{-K}}(W)$ 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, $\Sigma_{q2^{-K}}(W[u,v]) \leq W[u,v]$ is not (K,t)-dangerous by definition; if $I_2 = 2^D - 1$, then $\Sigma_{q2^{-K}}(W[u,v])$ 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 $\Sigma_{q2^{-K}}(W[u,v])$ is (K,t)-dangerous are the values I_2 and $I_2 + 1 \pmod{2^D}$.

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

2.6 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 2.4.1. 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 2.4.1.

2.6.1 Matrix Exponentiation and Pseudorandom Generators

Finite State Machines and Substochastic Matrices

Definition 2.6.1 *A* finite state machine Q *of type* (d,m) *is a directed graph on vertex set* $\{0,1,2,\ldots,d\}$ *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 $\{0,1,2,\ldots,d\}$ *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 $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_p) \in \Sigma^p$ defines a unique path of length p starting from i, obtained by following in succession the arcs labeled by $\alpha_1, \alpha_2, \ldots, \alpha_p$. 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 2.6.1 *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 2.6.2 *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 $\{0, 1, \ldots, 2^m - 1\}$ 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 2.6.3 *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

$$M[i,1], M[i,2], M[i,3], \cdots.$$

$\alpha \in Q(M)[i,j]$ 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 $O(m + \log d)$. □

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 $(\{0,1\}^a)^b$, i.e., a function that maps a string $x \in \{0,1\}^a$ to a sequence of b strings $y_0, y_1, y_2, \ldots, y_{b-1}$ where each $y_i \in \{0,1\}^a$. 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 $i,j \in \{0,1,2,\ldots,d\}$, $Q_G[i,j]$ 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 2.6.2 *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 2.6.1 *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 2.6.2 and the inequality follows from the assumption that G is ϵ-pseudorandom with respect to M. □

2.6.2 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.

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 $x \in \{0,1\}^a$, $G \circ G'(x) = G(G'_0(x))G(G'_1(x)) \ldots G(G'_{b'-1}(x))$.

Lemma 2.6.2 *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 $k_1, k_2, \ldots, k_{p'-1}$ 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 $G \circ G'(x) = G(G'_0(x))G(G'_1(x)) \ldots G(G'_{p'-1}(x))$ maps state i to state j in Q. The latter condition is equivalent to saying that there exist states $k_1, k_2, \ldots, k_{p'-1}$ 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 $G'_\ell(x) \in Q_G[k_\ell, k_{\ell+1}]$. □

Lemma 2.6.3 *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 2.6.2 and Proposition 2.6.1, the second inequality is by Proposition 2.6.2, and the last inequality follows from Proposition 2.2.3. □

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 2.6.2) and the composition of generators in the corresponding way so that Lemma 2.6.1, 2.6.2 and 2.6.3 hold analogously.

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

Universal Hash Function Families

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

Definition 2.6.3 *[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 2.6.4 *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 $h : \{0,1\}^p \to \{0,1\}^q$ 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 2.6.4 *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 2.6.4 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. □

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 2.6.5 *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 2.6.4, 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, $\sum_{j\in[d]} \rho(Q[i,j]) \leq 1$. □

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 $\vec{h} = (h_1, h_2, \ldots, h_r)$ where $h_k = (a_k, b_k) \in \mathrm{F}^2$, 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 $h = (a, b) \in \mathrm{F}^2$ we associate a function $f_h : \mathrm{F} \to \mathrm{F}$ given by $f_h(x) = ax + b$. In addition, we associate to each $h \in \mathrm{F}^2$ 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 $\vec{h} = (h_1, h_2, \ldots, h_r) \in (F^2)^r$, we define $G^{\vec{h}} = G^{h_1} \circ G^{h_2} \circ \ldots \circ G^{h_r}$.

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

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

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

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

Let $\vec{h} = (h_1, h_2, \ldots, h_r) \in (\mathrm{F}^2)^r$ and ℓ be an integer in the range $\{0, 1, \ldots, 2^r - 1\}$. Suppose $1 \le \ell_{i_1} < \ell_{i_2} < \ldots < \ell_{i_k} \le r$ are the positions of the 1's in the binary expansion of ℓ (so that $\ell = \sum_{l=1}^{k} 2^{\ell_{i_l}-1}$), we denote by $\vec{h}_\ell$ the subsequence $h_{\ell_1}, h_{\ell_2}, \ldots, h_{\ell_k}$ 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 $y_0, \ldots, y_{2^r-1}$ of 2^r blocks of strings in $\{0,1\}^m$, where y_ℓ is defined to be $f_{\vec{h}_\ell}(x)$ for $\ell = 0, 1, \ldots, 2^r - 1$.

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 2.6.6 *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 $\|(Q^2)^* - Q^*_{G^h}\| \le \epsilon d^2$. Let $i, k \in [d]$ be two arbitrary indices. It suffices to upper bound $|(Q^*)^2[i,k] - Q^*_{G^h}[i,k]|$ by ϵd.

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

$$|(Q^*)^2[i,k] - Q^*_{G^h}[i,k]|$$

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

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

For a finite state machine Q of type (d,m) and $\epsilon > 0$, a sequence $\vec{h} = (h_1, h_2, \ldots, h_r) \in (F^2)^r$ is said to be (Q,ϵ)-*well-behaved* if for each $1 \leq i \leq r$, f_{h_i} is $(Q_{G^{h_1}\circ G^{h_2}\circ\ldots\circ G^{h_{i-1}}}, \epsilon)$-good. Applying Lemma 2.6.5, a straightforward induction shows that:

Proposition 2.6.5 *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 2.6.7 *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 2.6.6. Assume it is true for $r-1$ and we show that the statement holds for r $(r \geq 2)$.

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

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 2.6.6, G^{h_r} is ϵd^2-pseudorandom with respect to $Q_{G^{\vec{h}'}}$.

Then, Lemma 2.6.3 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. □

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

Lemma 2.6.8 *Let m, d, r be integers and $\epsilon > 0$. There is an explicit family $\{G^{\vec{h}} | \vec{h} \in \{0,1\}^{2mr}\}$ 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 $\ell \in \{0, 1, \ldots, 2^r - 1\}$, computes the ℓ-th block of $G^{\vec{h}}(\alpha)$ in space $O(m + r)$.*

Consider the family of generators defined in Section 2.6.2. Then the first part of the lemma clearly follows from Proposition 2.6.5 and Lemma 2.6.7. 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 $O(m + r)$.

2.6.3 The PRS Algorithm

Finally, we give the description of the PRS algorithm and give the correctness proof of Lemma 2.4.1.

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 2.4.1 follows.

Proof of Lemma 2.4.1: 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 2.6.8, 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 2.6.1. 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 2.6.8, we can compute each successive block $G^{\vec{h}}_\ell(\alpha)$ for $\ell = 0, 1, \ldots, 2^r - 1$ in space $O(m + r)$ and by Proposition 2.6.3, 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. □

2.7 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 2.7.1 *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 $M^p = \Pi_{k=0}^{\lfloor \log p \rfloor} M^{p(k)2^k}$. 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 $\Pi_{k=i}^{j} M^{p(k)2^k}$. The definition of F is recursive:

$F(M, t, p, [i, i]) = \lfloor AMRS(M, p(i)2^i, 2^t)\rfloor_t$ and
$F(M, t, p, [i, j]) = \lfloor F(M, t, p, [i, \lfloor (i + j)/2\rfloor])\ F(M, t, p, [\lfloor (i + j)/2\rfloor + 1, j])\rfloor_t$ for $i < j$.

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 2.7.2 *(1) For $i \leq j$,*

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

(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 2.2.4 (1). Let us show the inductive step. We have that

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

where the first inequality follows from Proposition 2.2.4 (1), the second from Proposition 2.2.2, and the third is by induction.

Fact (2) is not difficult to verify using Proposition 2.7.1. □

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 $M^p = \Pi_{k=0}^{\lfloor \log p \rfloor} M^{p(k)2^k}$ 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 2.7.1 *There is a deterministic algorithm for AME with space complexity* $O(s \log^{1/2} p)$.

Chapter 3

Explicit Dispersers with Polylogarithmic Degree

3.1 Main Result and Applications

A disperser is a type of expander which was first introduced by Sipser in [Sip88]. Cohen and Wigderson [CW89] classified dispersers into two types: OR-dispersers and MAJORITY-dispersers. A bipartite multigraph $G = (V, W, E)$ with $|V| = N$ and $|W| = M$ is called an (N, M, T)-OR-disperser if any subset of V having at least T vertices has a neighbor set in W of size at least $M/2$; G is called an (N, M, T)-MAJORITY-disperser if it has the (essentially stronger) property that for any subset Y of W of size $M/3$, there are at most T vertices $v \in V$ such that a majority of v's neighbors are in Y. The *degree* of a disperser is defined to be the maximum degree of any vertex in V. In this chapter we investigate the problem of efficiently constructing OR-dispersers with small degree. For convenience, we will use the term "disperser" instead of "OR-disperser" unless otherwise specified.

It was proved in [San87] by a probabilistic argument that there exist (N, M, T)-dispersers for $N > T = M$ with degree at most $\log_2 N + 2$. The problem of giving an efficient algorithmic construction for dispersers with these or similar parameters, has remained open. (By *efficient*, we mean that for each vertex $v \in V$, its neighbor set can be generated in time polynomial in the output size of the set.) It has been shown that such an efficient construction would have a variety of applications in the theory of computation. The requirement for such a construction in many of the applications is that it works for constants $1 \geq \xi > \lambda \geq 0$ and for sufficiently large N, with $T \geq 2^{(\log N)^{\xi}}$, $M \leq 2^{(\log N)^{\lambda}}$, and with the degree being at most polylogarithmic in N.

As we will see, the problem of finding an efficient disperser construction is closely

related to the problem of making randomized algorithms robust with respect to imperfections in the random source. In fact, most of the previous disperser constructions are implicit in the work of simulating randomized algorithms using weak sources. The best previously known constructions are due to Zuckerman [Zuc91, Zuc96], who achieved degree polylogarithmic in N if $N = T^{O(1)}$, and Srinivasan and Zuckerman [SZ94], whose construction works for $N = 2^{polylog(T)}$ but requires degree $(\log N)^{O(\log\log N)}$. In recent work, Ta-Shma [TaS96] improved the degree of the construction in [SZ94] to $(\log N)^{O(\log^{(k)} \log N)}$ for any fixed integer k, where $\log^{(k)}$ denotes the logarithm to the base 2 iterated k times.

In this chapter, we give an improved construction of an (N, M, T)-disperser.

Main Theorem: $\forall \xi, \lambda, 1 \geq \xi > \lambda \geq 0$, $\exists N_0(\xi, \lambda)$ *such that if* $N \geq N_0(\xi, \lambda)$, *then for any* $2^{(\log N)^{\xi}} \leq T \leq N$ *and* $M \leq 2^{(\log N)^{\lambda}}$, *there is an efficient construction of an explicit* (N, M, T)*-disperser with degree polylogarithmic in* N.

It is worth noticing that although the constructions of [Zuc91], [SZ94] and [TaS96] do not give polylogarithmic degree, they do give MAJORITY-dispersers. Unfortunately, we could not strengthen our construction to give a MAJORITY-disperser. Nevertheless, the construction of a good OR-disperser is itself an interesting combinatorial problem and, as we will see, has significant applications to both complexity theory and randomized algorithm design.

In the remainder of this section, we describe two main applications of our new disperser. First we show that the complexity class Strong-RP is equal to RP. In effect, what this says is that there is an extremely efficient amplification scheme for randomized polynomial time algorithms. Then we show how our improved disperser family can be applied to designing randomized algorithms that achieve maximal robustness with respect to imperfections in the random source. There are other consequences of our disperser construction. For example, it gives improvements on the expander construction and the consequent applications given in [WZ93], on the the hardness results of approximating the clique function [Zuc93, SZ94], and on the results for a problem in

data structures: implicit $O(1)$ probe search [FN93, Zuc91]. These consequences were each observed by previous researchers, and provided much of the motivation for the search for good dispersers. The details of these improvements can be derived from the corresponding original papers by plugging in our construction, and will not be given here. We refer the reader to a comprehensive survey paper by Nisan [Nis96].

3.1.1 The Equivalence of RP and Strong-RP

Definition 3.1.1 *Random polynomial time (RP) is the set of languages* $L \subseteq \{0,1\}^*$ *such that there is a deterministic polynomial-time Turing machine* $M_L(\cdot,\cdot)$ *for which*

$$\begin{aligned} x \in L &\rightarrow Pr[M_L(x,y) \textit{ accepts}] > 1/2, \textit{ and} \\ x \notin L &\rightarrow Pr[M_L(x,y) \textit{ accepts}] = 0, \end{aligned}$$

where the probabilities are for a y *picked uniformly at random from* $\{0,1\}^m$ *where* $m = p(|x|)$*for some polynomial function* p*.*

We call $W_L^x = \{y \in \{0,1\}^m | M_L(x,y) \textit{ accepts}\}$ the *witness* set of M_L on input x, where m is the number of random bits used by M_L on inputs of length $|x|$.

Sipser [Sip88] defined the complexity class *strong random polynomial time* (Strong-RP) to be the class of languages L for which there is an RP machine $M_L(\cdot,\cdot)$ and a real number $0 < \eta < 1$ such that on input a string $x \in L$ of (any) length n, M_L uses $q(n)$ random bits for some polynomial $q(\cdot)$ and recognizes x with error probability at most $2^{-q(n)+q(n)^\eta}$. (If $x \notin L$, then the error probability is zero, as in Definition 3.1.1.) He asked whether RP = Strong-RP and showed that the existence of explicit constructions for sufficiently good dispersers would imply this equivalence. The construction we give here is sufficient for Sipser's purposes and so we obtain

Theorem 3.1.1 *RP = Strong-RP.*

Proof: The argument in this proof is essentially the same as in Sipser's original paper [Sip88]. We show the proof here for later reference. By definition, Strong-RP $\subseteq$ RP. We show that RP $\subseteq$ Strong-RP.

Suppose we have an RP machine M_L that needs m random bits on an input $x \in L$ of length n, for some m polynomial in n. We wish to simulate this by a machine that satisfies the conditions of Strong-RP.

Let $0 < \eta < 1$ be given. Let $\ell = m^{2/\eta} = n^{O(1)}$. By the Main Theorem with $\xi = \eta$ and $\lambda = \eta/2$, there is an explicit $(2^\ell, 2^m, 2^{\ell^\eta})$-disperser $G(\{0,1\}^\ell, \{0,1\}^m, E)$, and moreover, for any given $z \in \{0,1\}^\ell$, we can compute the neighborhood of z in $\{0,1\}^m$ in time $\ell^{O(1)} = n^{O(1)}$.

Our simulator on input x first samples a random $z \in \{0,1\}^\ell$, then computes the set of neighbors $Y \subset \{0,1\}^m$ of z. Finally, it runs $M_L(x,y)$ for each $y \in Y$ and accepts if and only if any of the runs accept. To see that it accepts with the required probability, let B be the set of strings in $\{0,1\}^\ell$ that do not have a neighbor in the witness set W_L^x (which are the strings on which the simulation fails to find a witness). Since the witness set W_L^x has more than half of the nodes in $\{0,1\}^m$, the definition of the disperser implies that B has size at most 2^{ℓ^η}. So the probability of failure is at most $2^{\ell^\eta - \ell}$. □

Sipser's introduction of Strong-RP was motivated by:

Theorem 3.1.2 ([Sip88]) *If $P \neq$ Strong-RP then there is a positive ϵ such that for any time bound $t(n)$, and for any language $L \in TIME(t(n))$, there is a machine that accepts L and, for infinitely many inputs, requires space at most $t(n)^{1-\epsilon}$.*

Sipser wanted to replace the hypothesis "P $\neq$ Strong-RP" by the hypothesis "P $\neq$ RP"; Theorem 3.1.1 says that this can indeed be done.

3.1.2 Computing With Weak Random Sources

In practice, randomized algorithms get their "random" bits by using pseudo-random number generators. Empirically, this often seems to be sufficient. However, there are reports of algorithms giving quite different results under different pseudo-random generators (see e.g., [FLW92] for such reports on Monte-Carlo simulations, and [Hsu93, HRD94] for the deviant performance of some RNC algorithms for graph problems). An alternative approach is to use the output of some physical source of randomness, such

as a Zener diode, or the last digits of a real-time clock. For such a source, it is plausible to assume that the string of bits output by the source are selected from some unknown distribution which, while not necessarily uniform, is "somewhat random". This leads to the question: to what extent is it possible to design randomized algorithms that are robust with respect to deviations of the random source from true randomness?

In general, one can define the following notion of an abstract source. An *n-bit source* is a probability distribution on $\{0,1\}^n$. A *source* is a sequence $S = S_1, S_2, \ldots$ where S_n is an n-bit source. We allow an algorithm to make a *single request* for R bits from the source for some R, and the source produces a string that is distributed according to S_R.

A natural idea to compute with imperfect random sources is to "convert" any source from some family of faulty sources into a distribution that is (nearly) random, which can then be used by randomized algorithms. That is, for a class of faulty sources, we would like to construct an easily computable function (family) f such that for any faulty source in the class, if x is a string selected according to the source then the distribution induced on $f(x)$ is (close to) uniform. For example, von Neumann [Neu63] presented a technique to convert independent but biased coin-flips into independent and unbiased ones; Blum [Blu86] showed how to convert the bits output by an unknown Markov chain into a sequence of perfectly random bits. However, for more general faulty sources, it can be shown (see e.g. [SV86]) that to construct such a direct conversion f is impossible.

On the other hand, the following broader idea of conversion (introduced in [VV85, Vaz86]) works for simulating randomized algorithms: Suppose the randomized computation C we wish to simulate needs m random bits. The simulation first requests $R = R(m)$ bits from the faulty source. Then it maps this sequence of $R(m)$ bits to a *set* of $t(m)$ strings each of m bits, called *test strings*, where $t(\cdot)$ is some function that depends on the simulation. The construction of the set of test strings does not depend on the computation being simulated or its input, but only on the number m and the sequence of bits output from the semi-random source. It then performs the computation C $t(m)$ separate times, one for each test string s, using s as the random string in the computation. If the algorithm being simulated is an RP algorithm then

the simulation accepts if and only if any of the runs of C accept, while if the algorithm being simulated is a BPP algorithm, the simulation accepts if and only if the majority of the runs of C accept. Simulating RP is of course no harder than simulating BPP.

It is easy to see that the running time of the simulation is $g(m) + t(m)TIME(C)$, where $g(m)$ is the time needed to generate the set of test strings, $t(m)$ is the size of the set of test strings and $TIME(C)$ is the time to perform the computation C. We say that the simulation is *efficient* if both $g(m)$ and $t(m)$ are bounded by polynomials of m. Note that in the case that we are simulating an RP or BPP algorithm, this is equivalent to saying that the simulation runs in polynomial time.

The high-level structure of this simulation is common to all subsequent work in this area. We refer to such a simulation as a "black-box" simulation. Applying this framework, efficient simulations of RP and BPP under various models of weak sources have been extensively studied. For example, Santha and Vazirani [SV86] studied the class of weak sources called "slightly random sources", which was further examined in [VV85, Vaz86, Vaz87a, Vaz87b]. A more general model "PRB-sources" is considered later by Chor and Goldreich [CG88]. In [Zuc90, Zuc91], Zuckerman introduced the model of δ-sources which generalizes all the previous models.

Let D be a probability distribution on a set X. The *min-entropy* of D is defined to be the maximum real number d such that $D(x) \leq 2^{-d}$ for all $x \in X$, where $D(x)$ is the probability that D assigns to x. For a function $\delta(\cdot)$ mapping the positive integers to $[0,1]$, a source $S = S_1, S_2, \ldots$ is said to be a *δ-source* if each S_R is a distribution of min-entropy at least $\delta(R) \cdot R$. The function δ is called the *entropy rate* of the source. In other words, for any number R of random bits requested, a δ-source outputs an R-bit string such that no string has probability more than $2^{-\delta(R)R}$ of being output. One example of an n-bit δ-source is the uniform distribution on a subset of $\{0,1\}^n$ of size at least $2^{\delta(n)n}$. Any source is a 0-source, and a 1-source is the pure random source. In general, the smaller that δ gets, the weaker (stochastically) the source can be.

For the case where δ is a fixed positive constant, Zuckerman [Zuc91, Zuc96] showed how to simulate any BPP algorithm efficiently with δ-sources. What about sources whose entropy rate decreases with R: how weak can the source get and still be usable

for efficient simulations? In [CW89], it was observed that, for information-theoretic reasons, if $\mathcal{S}$ is any class of sources for which there is an efficient black-box simulation of RP or BPP (that works correctly with high probability) using any source $S \in \mathcal{S}$, then every source S must be close to a δ-source with $\delta(R) \geq R^{\eta-1}$ for some fixed $\eta \in (0,1]$. Intuitively, if a source S is good enough to be used in an efficient simulation, then it must be sufficiently random so that to get m bits of entropy we only have to look at a polynomial in m number of bits from the source. This establishes a lower bound on the "amount of randomness" of the class of sources for which an efficient simulation is possible. For a given $\eta \in (0,1]$, we refer to the class of δ-sources with $\delta(R) = R^{\eta-1}$ as η-*minimally random* sources. The question is: for each $\eta > 0$, is there a polynomial-time simulation of BPP, or even RP, that works for all η-minimally random sources?

The previously best known simulation for RP with η-minimally random sources is due to [SZ94], which takes time $m^{O(\log m)}$, where m is the number of random bits needed by the original RP algorithm. (This result has been improved recently by Ta-Shma in [TaS96], where he presented a BPP simulation that uses time $m^{O(\log^{(k)} m)}$ for any fixed k.)

In [San87, Sip88, CW89] it was observed that the black-box simulation described above can be defined by a sequence $G_m = (V_m, W_m, E_m)$ of bipartite multigraphs, one for each m, where $V_m = \{0,1\}^{R(m)}$ and $W_m = \{0,1\}^m$. If the computation to be simulated needs m bits, then we use $R(m)$ bits from the faulty source to specify a vertex in V_m, and take as our test set the neighbors of the selected vertex. They further observed that to get an efficient simulation for RP (resp. BPP) that works for all δ-sources for a given function $\delta(\cdot)$, it suffices that the graphs G_m are good enough dispersers (resp. MAJORITY-dispersers).

The disperser construction we shall present is good enough to be used to do a polynomial-time black-box simulation of RP algorithms with η-minimally random sources for any fixed positive η:

Theorem 3.1.3 *For any fixed $0 < \eta \leq 1$, there is a polynomial time black-box simulation of RP using a δ-source with $\delta(R) = R^{\eta-1}$ and with error probability $2^{-n^{\Omega(1)}}$, where*

n is the length of the input to the RP algorithm.

Proof: Suppose an RP machine M_L needs m random bits on an input x of length n, for some m polynomial in n. Let $R = m^{3/\eta} = n^{O(1)}$, and let distribution D be any δ-source on $\{0,1\}^R$. Setting $\xi = \eta/2$ and $\lambda = \eta/3$ in the Main Theorem, we get a $(2^R, 2^m, 2^{R^{\eta/2}})$-disperser and use the same simulation as in the proof of Theorem 3.1.1. Again the probability of failure is the probability that $z \in B$. Since D has min-entropy R^η, we conclude that the probability that $z \in B$ is at most $2^{-R^\eta + R^{\eta/2}}$. □

The rest of this chapter contains six sections. In Section 3.2 we present various preliminary definitions and facts. The motivation and overview of the disperser construction is given in Section 3.3. We present the construction in Section 3.4 and its correctness proof is shown in Section 3.5, 3.6 and 3.7. Finally in Section 3.8 we give a summary of the parameters that appear in the construction and the proof.

3.2 Preliminaries

This section contains a large number of preliminary definitions concerning bipartite graphs and random sources. The key result of this section is Lemma 3.2.4, which provides a sufficient condition for a bipartite graph to be a disperser.

3.2.1 Bipartite Graphs

We consider bipartite multigraphs $G = (V, W, E)$. For simplicity, we will say "graphs" instead of "multigraphs". We use $deg(G)$ to denote the degree of G, which is defined to be the maximum degree of any vertex in V.

Suppose that $V_1, V_2, \ldots, V_k$ are disjoint sets and for each i between 1 and $k-1$, $G_i = (V_i, V_{i+1}, E_i)$ is a bipartite graph. The *composition* of $G_1, G_2, \ldots, G_{k-1}$, denoted $G_1 \circ G_2 \circ \cdots \circ G_{k-1}$, is defined to be the bipartite graph $G = (V_1, V_k, E)$ where $(v_1, v_k) \in E$ if and only if there exists a sequence $v_2, \ldots, v_{k-1}$ of vertices with $v_i \in V_i$ for $1 < i < k$ and $(v_i, v_{i+1}) \in E_i$ for $1 \leq i < k$. Observe:

Proposition 3.2.1 *If $G = G_1 \circ \cdots \circ G_{k-1}$, then $deg(G) \leq deg(G_1) \times \cdots \times deg(G_{k-1})$.*

3.2.2 Blocks, Segmentations and Bit-string Trees

For integers $i \leq j$, the *block* $[i, j]$ is defined to be the sequence of integers $(i, i+1, \ldots, j)$.

We will often be dealing with sequences $A = A_1, A_2, \ldots, A_k$ where the A_i are positive integers, sets or bit strings. If $[i, j]$ is a block contained in $[1, k]$ then we use the notation $A_{[i,j]}$ to denote the sum $A_i + A_{i+1} + \ldots + A_j$ in the case that the A_i are integers, the union $A_i \cup A_{i+1} \cup \ldots \cup A_j$ in the case that the A_i are sets and the concatenation $A_i A_{i+1} \ldots A_j$ in the case that the A_i are bit strings.

An (n, s)-*composition* is a sequence $l = (l_1, l_2, \ldots, l_s)$ of positive integers summing to n, an (n, s)-*tower* is a sequence $q = (q_0, q_1, \ldots, q_{s-1}, q_s)$ of integers with $0 = q_0 < q_1 < q_2 < \ldots < q_{s-1} < q_s = n$, and an (n, s)-*segmentation* is a sequence $\pi = (B_1, B_2, \ldots, B_s)$ of disjoint blocks whose union is $[1, n]$ such that for each $i < j$ every element of B_i is less than every element of B_j.

There is an obvious set of one-to-one correspondences between (n, s)-compositions, (n, s)-towers and (n, s)-segmentations as follows: the composition l corresponds to the tower q with $q_i = l_{[1,i]}$ and to the segmentation π such that $B_i = [q_{i-1} + 1, q_i]$ (thus $|B_i| = l_i$).

For example, for $n = 6$ and $s = 3$, the (n, s)-segmentation $(\{1, 2, 3\}, \{4, 5\}, \{6\})$ corresponds to the (n, s)-composition $l = (3, 2, 1)$ and the (n, s)-tower $q = (0, 3, 5, 6)$.

Let π be an (n, s)-segmentation, l the corresponding (n, s)-composition and q the corresponding (n, s)-tower. The *bit-string tree* T^π associated to π is the rooted tree with $s + 1$ layers of nodes labelled by bit-strings defined as follows:

1. the root of the tree is at depth 0 and is labelled by the empty string;
2. there are 2^{q_i} nodes at depth i labelled by the bit-strings of length q_i;
3. if a node at depth $i < s$ is labelled by y, then it has $2^{l_{i+1}}$ children at depth $i + 1$ such that for each bit-string z of length l_{i+1}, the node has exactly one child labelled by yz. We denote the edge from y to yz by $(z|y)$.

3.2.3 Probability Distributions, Converters and Carriers

We will be considering probability distributions D defined on a finite set X. We denote by $\mathrm{supp}(D)$ the support of D, i.e., the set of elements of X to which D assigns non-zero probability. It is often convenient to think of D as a (row) vector indexed by the set X. Thus, a family of distributions $\mathcal{D}$ on X can be viewed as a collection of vectors in R^X. We say that $\mathcal{D}$ is *convex* if the associated set of vectors is convex. For $x \in X$, we use $D(x)$ to denote the probability that D assigns to x, and for $S \subseteq X$, we define $D(S) = \sum_{x \in S} D(x)$.

We denote by U_X the uniform distribution on the set X, and use U_l instead of $U_{\{0,1\}^l}$ for simplicity. If D is a distribution on X and E is a distribution on Y, then $D \times E$ denotes the product distribution on the set $X \times Y$ given by $D \times E(x, y) = D(x)E(y)$.

Definition 3.2.1 *Let V and W be finite sets. A* converter Λ for V and W *is a matrix with rows indexed by V and columns indexed by W, such that all entries are nonnegative and all row sums are 1. For $v \in V$ and $w \in W$, $\Lambda(v, w)$ denotes the entry in row v and column w.*

The usual term for a converter is "stochastic matrix"; we choose the term converter because it provides a mnemonic for what we want to do with it. Each row $\Lambda(v, \cdot)$ of a converter can be viewed as a probability distribution over W. Given any probability distribution D on V, and a converter Λ for V and W, we define the distribution $D\Lambda$ on W as follows: select $v \in V$ according to D, and then select $w \in W$ according to $\Lambda(v, \cdot)$. Thus Λ "converts" any distribution on V to a distribution on W. If we view distributions as (row) vectors, then the transformed distribution of D through Λ is exactly the vector-matrix product $D\Lambda$.

Any function $f : V \to W$ induces a converter Λ^f for V and W in a natural way: for $v \in V$ and $w \in W$, $\Lambda^f(v, w)$ is 1 if $f(v) = w$ and is 0 otherwise. If D is any distribution on V, then $f(D)$ is defined to be the distribution on W such that for $w \in W$, $f(D)(w) = \sum_{v \in V : f(v) = w} D(v)$. That is, $f(D) = D\Lambda^f$. An easy induction shows the following:

Proposition 3.2.2 *Let $V_1, V_2, \ldots, V_k$ be sets and for each i between 1 and $k-1$, let f_i be a function mapping V_i to V_{i+1}. Suppose $f : V_1 \to V_k$ is defined as $f = f_{k-1} \circ \ldots \circ f_2 \circ f_1$, where $\circ$ denotes the composition of functions. Then for any distribution D on V_1, $f(D)$ is the distribution on V_k such that $f(D) = D\Lambda^{f_1}\Lambda^{f_2}\ldots\Lambda^{f_{k-1}}$.*

The *support* of a converter Λ is defined to be the bipartite graph (V, W, E) such that $(v, w) \in E$ if and only if $\Lambda(v, w) \neq 0$. If G is a bipartite graph on V and W that contains the support of Λ, then G is said to be a *carrier* for Λ. Given a carrier G of the converter Λ, it is often convenient to visualize Λ as an assignment of nonnegative weights to the edges of G, where the weight on edge (v, w) is the probability assigned to vertex w by the distribution $\Lambda(v, \cdot)$. Thus the sum of weights on edges leaving v is exactly 1.

We next recall a standard measure of distance between distributions.

Definition 3.2.2

1. *The* variational distance *between two distributions D_1 and D_2 on the same set X is defined to be $\|D_1 - D_2\| = \max_{Y \subseteq X} |D_1(Y) - D_2(Y)| = \frac{1}{2}\sum_{x \in X} |D_1(x) - D_2(x)|$.*

2. *D_1 is said to be* ϵ-near *to D_2 if $\|D_1 - D_2\| \leq \epsilon$.*

3. *The distribution D on X is said to be* quasi-random to within ϵ *if D is ϵ-near to the uniform distribution on X.*

Some useful, well-known, and easily proved facts about variational distance are summarized below.

Proposition 3.2.3

1. *If D_1 and D_2 are distributions on X, then $\|D_1 - D_2\| \leq \min\{D_1(X_1), D_2(X_2)\}$ where X_1 (resp. X_2) is the set of elements $x \in X$ such that $D_2(x) < D_1(x)$ (resp. $D_1(x) < D_2(x)$).*

2. *If D_1, D_2, D_3 are distributions on the same set X, then $\|D_1 - D_3\| \leq \|D_1 - D_2\| + \|D_2 - D_3\|$.*

3. *If D_1 and D_2 are distributions on V and Λ is a converter for V and W, then $\|D_1\Lambda - D_2\Lambda\| \leq \|D_1 - D_2\|$. In particular, for any function $f : V \to W$, $\|f(D_1) - f(D_2)\| \leq \|D_1 - D_2\|$.*

The *distance* between a distribution D and a family $\mathcal{D}$ of distributions on the same set is defined to be the minimum of $\|D - D'\|$ over all $D' \in \mathcal{D}$. D is said to be *ϵ-near to the family $\mathcal{D}$* if the distance from D to $\mathcal{D}$ is at most ϵ. We will need the following technical fact:

Lemma 3.2.1 *Let $\mathcal{D}$ be a convex family of distributions on X and suppose that $D_1, D_2, \ldots, D_k$ are distributions on X such that D_i is ϵ_i-near to the family $\mathcal{D}$. Suppose that $\lambda_1, \lambda_2, \ldots, \lambda_k$ are nonnegative reals summing to 1, and let $D = \sum_{i=1}^{k} \lambda_i D_i$. Then D is ϵ-near to the family $\mathcal{D}$, where $\epsilon = \sum_{i=1}^{k} \epsilon_i \lambda_i$.*

Proof: Let i be any integer between 1 and k. By assumption, D_i is ϵ_i-near to $\mathcal{D}$ and therefore there exists a $D_i' \in \mathcal{D}$ such that $\|D_i - D_i'\| \leq \epsilon_i$. Let $D' = \sum_{i=1}^{k} \lambda_i D_i'$. Since $\mathcal{D}$ is convex, $D' \in \mathcal{D}$. Furthermore,

$$\|D - D'\| \leq \sum_{i=1}^{k} \lambda_i \|D_i - D_i'\| \leq \sum_{i=1}^{k} \lambda_i \epsilon_i,$$

which concludes the proof. □

Finally, we adopt the convention that the conditional probability given some impossible event is 0.

3.2.4 Bit Sources

An *n-bit source* is a distribution on $\{0,1\}^n$. We typically use the symbol $\tilde{X}$ to denote a random n-bit string selected according to some n-bit source D. If y is a bit string of length $j \leq n$, we write $D(y)$ as the probability that $\tilde{X}_{[1,j]} = y$. If z is another bit string of length k with $j + k \leq n$, we write $D(z|y)$ as the conditional probability that $\tilde{X}_{[1,j+k]} = yz$ given $\tilde{X}_{[1,j]} = y$.

For $q \leq n$, we define the *q-bit truncation* of the n-bit source D to be the q-bit source $D^{(q)}$ such that for $y \in \{0,1\}^q, D^{(q)}(y) = D(y)$.

Lemma 3.2.2 *If D is an n-bit source quasi-random to within ϵ and $q \leq n$, then $D^{(q)}$ is quasi-random to within ϵ.*

Proof: Define $f : \{0,1\}^n \to \{0,1\}^q$ to be the q-bit truncation function, i.e., for $x \in \{0,1\}^n$, $f(x)$ is equal to the first q bits of x. Then $D^{(q)} = f(D)$ and $f(U_n) = U_q$. The lemma follows from Proposition 3.2.3(3). □

Let $x \in \{0,1\}^n$ and $1 \leq i \leq n$. The *information in bit i of x relative to D* is defined to be $I_i(x) = I_i^D(x) = -\log_2 D(x_i|x_{[1,i-1]})$. (If $D(x_i|x_{[1,i-1]}) = 0$ then $I_i(x) = \infty$.) Intuitively, $I_i(x)$ represents the amount of information (relative to the distribution) that the i-th bit of x provides if the bits are revealed one by one. We say that position i indexes a *good bit* in x ([NZ93]) if either $I_i(x) > 1$ or $I_i(x) = 1$ and $x_i = 0$.

The *total information* of x relative to D is defined to be the sum of the $I_i(x)$ over $1 \leq i \leq n$, which is equal to $-\log_2 D(x)$. The *min-entropy* of D is the minimum total information of any string relative to D. For a function $\delta = \delta(\cdot)$ that maps positive integers to $[0,1]$, an n-bit source D is said to be a *δ-source* if its min-entropy is at least δn for $\delta = \delta(n)$, i.e., $D(x) \leq 2^{-\delta n}$ for all $x \in \{0,1\}^n$.

More generally, if $B = [i,j]$ is any block of $[1,n]$ then the *information in block B of x relative to D* is given by $I_B(x) = I_B^D(x) = -\log_2 D(x_{[i,j]}|x_{[1,i-1]})$. Clearly, $I_B(x) = I_i(x) + I_{i+1}(x) + \ldots + I_j(x)$. In particular, $I_{[1,n]}(x)$ is just the total information of x as defined above.

The *good bit indicator* of x with respect to D, denoted $\chi^D(x)$, is the n-bit sequence whose i-th bit is 1 if i indexes a good bit and is 0 otherwise. This important notion gives us a simple way to measure the distribution of the information of a string in a source and, as we will see, has useful properties that provide much motivation for our construction. We examine a few basic properties of this notion.

Proposition 3.2.4 *Let D be an n-bit source. Then for $x, y \in supp(D)$, $\chi^D(x) = \chi^D(y)$ if and only if $x = y$.*

This is because $\chi^D(x)$ and $\chi^D(y)$ must be different at the first coordinate where x and y differ.

For a bit sequence β, we use $w(\beta)$ to denote the *weight* of β, i.e., the number of 1's in the sequence. An n-bit source D is said to be *δ-smooth* if $w(\chi^D(x)) \geq \delta n$ for every $x \in \mathrm{supp}(D)$, that is, every string in the support of D has at least δn good bits. Clearly, any δ-smooth source is a δ-source. Intuitively, if a source is smooth, then the information of every string in the source is well dispersed.

Let α be an n-bit sequence. A segmentation π of $[1, n]$ is said to be *t-equitable with respect to α* if for each block B in π we have $w(\alpha_B) \geq t$. We observe that :

Proposition 3.2.5 *For an n-bit source D and an n-bit string x, if π is a segmentation of $[1, n]$ that is t-equitable with respect to $\chi^D(x)$, then for each block B in π, $I_B(x) \geq t$.*

The *block-wise min-entropy of D with respect to segmentation π* is the minimum of $I_{B_j}(x)$ over all strings x and blocks B_j of π. We say that D is a *block-wise (π, b)-source* if the block-wise min-entropy of D with respect to π is at least b.

Lemma 3.2.3 *Let n, s be integers with $n \geq s$. Then for any (n, s)-segmentation π and real number $b > 0$, the class of block-wise (π, b)-sources on $\{0,1\}^n$ is convex.*

Proof: Let $\pi = (B_1, B_2, \ldots, B_s)$. Suppose D_1, D_2 are arbitrary block-wise (π, b)-sources on $\{0,1\}^n$ and λ_1, λ_2 are arbitrary nonnegative reals summing to 1. Let $D = \sum_{i=1}^{2} \lambda_i D_i$. We want to show that for any $x \in \{0,1\}^n$ and $1 \leq k \leq s$, $D(x_{B_k} | x_{B_{[1,k-1]}}) \leq 2^{-b}$.

$$\begin{aligned} D(x_{B_k} | x_{B_{[1,k-1]}}) &= D(x_{B_{[1,k]}}) / D(x_{B_{[1,k-1]}}) \\ &= (\sum_{i=1}^{2} \lambda_i D_i(x_{B_{[1,k]}})) / (\sum_{i=1}^{2} \lambda_i D_i(x_{B_{[1,k-1]}})) \\ &\leq \max_i \{D_i(x_{B_{[1,k]}}) / D_i(x_{B_{[1,k-1]}})\} \\ &\leq 2^{-b}, \end{aligned}$$

where the last inequality follows from the assumption that the D_i are block-wise (π, b)-sources. □

When trying to determine whether D is a block-wise (π, b)-source it is useful to represent D by an edge-labeling of the bit-string tree T^π in which edge $(z|y)$ is labeled by $D(z|y)$. We call this representation the *π-tree representation* of D. We note the following obvious facts:

Proposition 3.2.6 *Let D be an n-bit source. Suppose π is an (n,s)-segmentation and $l = (l_1, \ldots, l_s)$ the corresponding (n,s)-composition. Then in the π-tree representation of D:*

1. *for every internal node y at depth i, the sum of $D(z|y)$ over all $z \in \{0,1\}^{l_{i+1}}$ is equal to 1;*
2. *for any leaf $x \in \{0,1\}^n$, the probability $D(x)$ is just the product of the edge labels on the path to x;*
3. *D is a block-wise (π, b)-source if and only if every edge label is bounded above by 2^{-b}.*

3.2.5 Carriers for Families of Distributions

In this section we present the sufficient condition for a bipartite graph to be a disperser that we will use throughout the chapter.

Let V and W be sets, $\mathcal{D}_1$ be a family of distributions on V, and $\mathcal{D}_2$ be a family of distributions on W. A bipartite graph $G = (V, W, E)$ is a $(\mathcal{D}_1, \mathcal{D}_2, \epsilon)$-*carrier* if for each $D_1 \in \mathcal{D}_1$ there is a converter Λ, carried by G, such that $D_1\Lambda$ is ϵ-near to some distribution in $\mathcal{D}_2$. A $(\mathcal{D}_1, \mathcal{D}_2, \epsilon)$-carrier is said to be *strong* if there is a converter Λ carried by G such that, for every distribution $D_1 \in \mathcal{D}_1$, the transformed distribution $D_1\Lambda$ is ϵ-near to some distribution $D_2 \in \mathcal{D}_2$.

In the case that the class $\mathcal{D}_2$ is a singleton set of distribution D_2, we may use the notation $(\mathcal{D}_1, D_2, \epsilon)$-carrier.

Remark: The condition that G is a strong $(\mathcal{D}_1, \mathcal{D}_2, \epsilon)$-carrier is stronger than the condition that it is a $(\mathcal{D}_1, \mathcal{D}_2, \epsilon)$-carrier, since the latter condition does not require that there is a single converter Λ carried by G that "works" for all $D_1 \in \mathcal{D}_1$.

We denote by $\mathcal{T}(X, d)$ the set of distributions on X of min-entropy at least d. The next lemma shows that to construct a good disperser, it suffices to construct an appropriate carrier.

Lemma 3.2.4 *Let N, M, T be positive integers and let V, W be sets with $|V| = N$ and $|W| = M$. If $G = (V, W, E)$ is a $(\mathcal{T}(V, \log T), U_W, 1/2)$-carrier then G is an (N, M, T)-disperser.*

Proof: Assume that G is a $(\mathcal{T}(V, \log T), U_W, 1/2)$-carrier. Let X be an arbitrary subset of V of size T, and let Y be the neighbor set of X. We need to show that $|Y| \geq M/2$. Let D be the distribution that is uniform on X and 0 on $V - X$. Then $D \in \mathcal{T}(V, \log T)$, and so by the assumption on G, there is a converter Λ supported by G that converts D into a distribution $Q = D\Lambda$ that is $1/2$-near to U_W. Thus, $|Q(Y) - U_W(Y)| \leq 1/2$. Since Λ is supported by G, Q assigns positive probability only to vertices that are in the neighborhood of X, so $Q(Y) = 1$. The uniform distribution assigns probability $|Y|/M$ to Y. Thus $|Q(Y) - U_W(Y)| = 1 - |Y|/M \leq 1/2$, i.e., $|Y| \geq M/2$. □

Next we discuss the composition of converters and carriers. A straightforward induction shows the following fact.

Proposition 3.2.7 *Let $V_1, V_2, \ldots, V_k$ be sets and for each i between 1 and $k-1$, let Λ_i be a converter for V_i and V_{i+1}. Then the matrix product $\Lambda_1 \Lambda_2 \cdots \Lambda_{k-1}$ is a converter for V_1 and V_k. Furthermore, if G_i is a bipartite graph that carries Λ_i, then the composition $G_1 \circ G_2 \circ \cdots \circ G_{k-1}$ carries $\Lambda_1 \Lambda_2 \cdots \Lambda_{k-1}$.*

Lemma 3.2.5 *Let $V_1, \ldots, V_k$ be finite sets and for each i, $1 \leq i \leq k$, let $\mathcal{D}_i$ be a family of distributions on V_i. If for each i, $1 \leq i \leq k-1$, G_i on V_i, V_{i+1} is a $(\mathcal{D}_i, \mathcal{D}_{i+1}, \epsilon_i)$-carrier, then $G = G_1 \circ \cdots \circ G_{k-1}$ is a $(\mathcal{D}_1, \mathcal{D}_k, \epsilon)$-carrier where $\epsilon = \epsilon_1 + \cdots + \epsilon_{k-1}$.*

Proof: The proof is by induction on k. The case where $k = 2$ is trivial. Assume it holds for $k-1$ and we show for k.

Let $G' = G_1 \circ \cdots \circ G_{k-2}$. Then $G = G' \circ G_{k-1}$ and we know G' on V_1, V_{k-1} is a $(\mathcal{D}_1, \mathcal{D}_{k-1}, \epsilon')$-carrier for $\epsilon' = \epsilon_1 + \cdots + \epsilon_{k-2}$ by induction hypothesis.

Fix any distribution $D_1 \in \mathcal{D}_1$. We know there is a converter Λ_1 carried by G' and a distribution $D_{k-1} \in \mathcal{D}_{k-1}$ such that $\|D_1\Lambda_1 - D_{k-1}\| \leq \epsilon'$ by the carrier property of G' and also, there is a converter Λ_2 carried by G_{k-1} and a distribution $D_k \in \mathcal{D}_k$ such

that $\|D_{k-1}\Lambda_2 - D_k\| \leq \epsilon_{k-1}$ by the carrier property of G_{k-1}. Define $\Lambda = \Lambda_1\Lambda_2$. Then

$$\begin{aligned}\|D_1\Lambda - D_k\| &\leq \|(D_1\Lambda_1)\Lambda_2 - D_{k-1}\Lambda_2\| + \|D_{k-1}\Lambda_2 - D_k\| \\ &\leq \|D_1\Lambda_1 - D_{k-1}\| + \|D_{k-1}\Lambda_2 - D_k\| \\ &\leq \epsilon' + \epsilon_{k-1} = \epsilon,\end{aligned}$$

where the second inequality follows from Proposition 3.2.3(3). Now Proposition 3.2.7 completes the proof. □

3.3 Motivating the Disperser Construction

We present our disperser construction in Section 3.4. While the construction is itself elementary, it is not at all clear from the description why it is a disperser. A detailed proof of this is given following the description of the construction, but the technical details of the proof hide the key intuitions. In this section, we discuss the idea behind our construction. The discussion here is not rigorous, and is intended only to aid the reader in understanding what follows.

As Lemma 3.2.4 stated, to obtain an explicit construction of a sufficiently good disperser, it suffices to construct a carrier that converts distributions of small min-entropy to the ones that are nearly uniform. For this discussion, let us assume $V = \{0,1\}^n, n = \log N$ and $W = \{0,1\}^m, m = \log M$, and N, M, T are related as they are in the Main Theorem. Building on the ideas developed in [NZ93] and [SZ94], our construction $G_D = (V, W, E)$ of an (N, M, T)-disperser with degree polylogarithmic in N is obtained by composing two bipartite graphs. Let $Z = \{0,1\}^{sn}$ where $s = a \log n$ for some constant a. The first bipartite graph is between V and Z and gives a carrier G_A that converts any distribution on V of min-entropy $\log T$ to a distribution on Z that is close to a block-wise source with s equal-sized blocks and with block-wise min-entropy nearly $\log T$. The second is between Z and W and gives a carrier G_C that converts any block-wise source on Z as above to a nearly uniform distribution on W. Both of these constructions have degree polylogarithmic in N. G_D is defined to be $G_A \circ G_C$, and so it has degree polylogarithmic in N as well and it gives a desired carrier by Lemma 3.2.5.

If X and Y are arbitrary sets, a function f from X to Y can be viewed as a bipartite graph from X to Y with edge set $\{(x, f(x))|x \in X\}$. In our construction, the edge set of the carrier G_A is described by a family of functions $\mathcal{F}_A$, each mapping V to Z, and consists of the union of the graphs obtained from each function. Carrier G_C is similarly specified by a family $\mathcal{F}_C$ of functions.

The family $\mathcal{F}_C$ of functions mapping Z to W that specifies carrier G_C is obtained by adjusting parameters in the "block-wise extractor" as presented in [SZ94], which is in turn an improvement on a similar construction in [NZ93]. Generally speaking, a block-wise extractor is a function taking two input strings z and y such that if z comes from a block-wise source and y from a pure random source, then the distribution induced on the output of the function is nearly uniform. By modifying the construction in [SZ94], we obtain a block-wise extractor C such that on input a string z coming from any block-wise source on Z with s equal-sized blocks and with block-wise min-entropy nearly $\log T$, and a purely random string y of length $O(\log n)$, the distribution induced on $C(z, y)$ is close to uniform. We use each such string y to index a function f_y mapping Z to W defined by $f_y(z) = C(z, y)$. The family $\mathcal{F}_C$ is defined to be the set of all f_y's, and therefore the size of $\mathcal{F}_C$ is polynomial in n (polylogarithmic in N). The above fact about C now can be restated in terms of $\mathcal{F}_C$ as follows: for a string z selected according to a block-wise source on Z as above, if we uniformly choose a function $f_y \in \mathcal{F}_C$ at random, then the distribution of $f_y(z)$ on W is nearly uniform. This is exactly what is needed for G_C.

The main novelty of our construction is the construction of carrier G_A. Intuitively, to convert an arbitrary distribution D into a block-wise source we want to "chop up" the sequence of bits from D into a sequence of blocks so that each block has a sufficient amount of information relative to D. As in [NZ93], the good bit indicator is an appropriate way to measure the dispersal of information within any string from the source. As suggested by Proposition 3.2.5, ideally what we would like is to find an (n, s)-segmentation π such that for any string x from D, π is t-equitable with respect to $\chi^D(x)$ for some sufficiently large t.

There are a few obvious difficulties in finding such a π. First, since we don't know

what the source D is, we do not know how the information of any string is distributed. Second, it may be the case that for a particular string x from a particular source D, all the information of x is concentrated in a very few bits so that the weight of $\chi^D(x)$ is too small compared to the total information of x (or to the min-entropy of D). The second problem is easily handled: it turns out that any source D is close to a smooth source D' whose min-entropy is close to that of D. So we work with D'. The solution to the first problem is this: we look for a small family of segmentations (not just one single segmentation) of the bit positions such that if a string x has enough weight in $\chi^{D'}(x)$, then at least one of the segmentations in the family is t-equitable with respect to $\chi^{D'}(x)$ for some sufficiently large t.

Abstractly, here is the combinatorial problem we want to solve: for fixed $\eta' < \eta$ and sufficiently large n, we need an explicit polynomial-sized family of segmentations Π of $[1, n]$ into $s = a \log n$ blocks for some fixed a, such that for any n-bit sequence α of weight at least n^η, there is a segmentation in Π that is $n^{\eta'}$-equitable with respect to α.

Each segmentation $\pi \in \Pi$ into s blocks defines a function f_π mapping $V = \{0,1\}^n$ to $Z = \{0,1\}^{sn}$ as follows: $f_\pi(x)$ is obtained by splitting x into the s segments $x^1, x^2, \ldots, x^s$ corresponding to the segmentation, and then padding each x^i by $n - |x^i|$ 0's, so that each block is of length n. The family $\mathcal{F}_A$ of functions that specifies carrier G_A is the set of f_π over $\pi \in \Pi$.

3.4 The Disperser Construction

We are now ready to give a full description of our disperser construction. The proof of its correctness will be presented in the following sections. Note that for reference, a summary of parameters that appear in the construction and the proof appears at the end of the chapter. All logarithms are to the base 2, unless otherwise specified.

Fix ξ and λ with $1 \geq \xi > \lambda \geq 0$. We wish to construct an (N, M, T)-disperser for sufficiently large N, any $T \geq 2^{(\log N)^\xi}$, and any $M \leq 2^{(\log N)^\lambda}$. We assume that $N = 2^n$ and $M = 2^m$ for integers n, m, and take $V = \{0,1\}^n$ and $W = \{0,1\}^m$. This assumption is without loss of generality since in the case that N or M is not an integer

power of 2, we can take $n = \lceil \log N \rceil$, and take $m = \lceil \log M \rceil$ if $M > \frac{3}{2}2^{\lfloor \log M \rfloor}$ and $m = \lfloor \log M \rfloor$ otherwise, and construct a $(\mathcal{T}(V, \log T), U_W, 3/4)$-carrier. Our disperser will be denoted $G_D = (V, W, E)$.

We define G_D to be the composition of two bipartite graphs G_A and G_C, defined respectively in Sections 3.4.1 and 3.4.2. Let $a = \frac{\lambda}{\log \frac{9}{8}}$ and let $s = \lceil a \log n \rceil$, $r = sn$. Define $Z = \{0,1\}^r$. The first bipartite graph G_A is between V and Z. The second, denoted G_C, is between Z and W.

3.4.1 Constructing $G_A(V, Z, E_A)$

Given an (n, s)-segmentation $\pi = (B_1, \ldots, B_s)$ and $x \in \{0,1\}^n$, define $f_\pi(x)$ to be the sn-bit string obtained as follows: $f_\pi(x)$ consists of the concatenation of s n-bit strings, where the i-th string consists of the concatenation of x_{B_i} and $0^{n-|B_i|}$.

For integers n, k, d with $n \geq k, d \geq 4$, we will define a family $\Pi(n, k, d)$ of (n, k)-segmentations. The edge set E_A of G_A is then defined by $\{(x, f_\pi(x)) | \pi \in \Pi(n, s, d)\}$, and thus $deg(G_A)$ is $|\Pi(n, s, d)|$.

To describe $\Pi(n, k, d)$, we first define the *balanced d-ary tree T on $[1, n]$* to be the labelled rooted tree with n leaves constructed in the following way. Let $H = \lceil \log_d n \rceil$ and let T' be the rooted d-ary tree of depth H. Let T be the smallest subtree of T' containing the root and the leftmost n leaves of T'. Thus T has $\lceil \log_d n \rceil + 1$ layers of nodes and for each node v at depth $i \geq 0$, the subtree rooted at v has at most n/d^{i-1} leaves. The nodes of T are labelled as follows: the leaves are labelled left-to-right by $1, 2, \ldots, n$ and each internal node v is labelled by the interval formed by the labels of the leaves in the subtree of v.

For a node v in the tree, we denote by $L(v)$ the largest leaf label in the subtree of v. If $u_1, u_2, \ldots, u_{k-1}$ is any sequence of vertices in T such that $L(u_i)$ is strictly increasing and $L(u_{k-1}) < n$, we associate it with the (n, k)-segmentation whose corresponding (n, k)-tower is $(0, L(u_1), L(u_2), \ldots, L(u_{k-1}), n)$. We specify below a collection $\Gamma(n, k, d)$ of such sequences of vertices; $\Pi(n, k, d)$ is then defined to be the set of all (n, k)-segmentations arising from these sequences.

For each leaf w of T, let $A(w)$ be the set of vertices v such that (1) $L(v) < w$ and

(2) the parent of v belongs to the unique path joining w to the root. That is, $A(w)$ is the set of vertices v of T such that v is a left sibling of some vertex on the path joining w to the root. We notice that the vertices in $A(w)$ all have distinct $L(\cdot)$ values and they are all less than n. Order the vertices in $A(w)$ according to their $L(\cdot)$ values in increasing order. We take $\Gamma(n,k,d)$ to be the union over all leaves w of the set of $(k-1)$-element subsequences of the ordered $A(w)$. Notice that the size of any $A(w)$ is at most $(d-1)H$ and the number of subsequences of $A(w)$ is at most $2^{H(d-1)} \leq n^{d-1}$ (since $d \geq 4$). So the size of $\Gamma(n,k,d)$ (and hence the size of $\Pi(n,k,d)$) is at most n^d.

3.4.2 Constructing $G_C(Z,W,E_C)$

For some q that is $O(\log n)$, we will define a function C from $Z \times \{0,1\}^q = \{0,1\}^r \times \{0,1\}^q$ to $W = \{0,1\}^m$. We then define the edge set E_C of G_C as $\{(z,w) | \exists y \in \{0,1\}^q$ such that $C(z,y) = w\}$, and thus $deg(G_C)$ will be 2^q which is polynomial in n. The function C is a straightforward modification of the block-wise extractor of [SZ94]. To describe the construction, we need the following lemma which is derived from the improved Leftover Hash Lemma in [SZ94]:

Lemma 3.4.1 *Let $t \leq n$ be nonnegative integers. There is an explicit family H of functions such that:*

- *each function in H maps n bits to $\lfloor 3t/4 \rfloor$ bits;*
- *the functions of H are indexed by sequences of $4t + \lceil 6 \log n \rceil$ bits;*
- *given the index of a function $h \in H$ and $x \in \{0,1\}^n$, the value of $h(x)$ is computable in time polynomial in n; moreover,*
- *if D is an n-bit source of min-entropy t, $\tilde{X}$ is sampled from D and $\tilde{h}$ is chosen uniformly from H, then $(\tilde{h}, \tilde{h}(\tilde{X}))$ is a random variable whose distribution is quasi-random to within $2^{1-t/8}$ (on $H \times \{0,1\}^{\lfloor 3t/4 \rfloor}$).*

This is obtained from the improved Leftover Hash Lemma of [SZ94], by setting $k = t/8$ and $\epsilon = 2^{1-k}$. □

The following reformulation is more convenient for our purposes.

Corollary 3.4.1 *There is an explicit function*

$$\tau : \{0,1\}^n \times \{0,1\}^{4t+\lceil 6\log n\rceil} \to \{0,1\}^{4\lceil \frac{9}{8}t\rceil+\lceil 6\log n\rceil}$$

such that for any n-bit source D of min-entropy t ($t \geq 11$), the induced distribution $\tau(D \times U_{4t+\lceil 6\log n\rceil})$ on $\{0,1\}^{4\lceil \frac{9}{8}t\rceil+\lceil 6\log n\rceil}$ is quasi-random to within $2^{1-t/8}$. Furthermore, τ is computable in time polynomial in n.

To see the corollary, we view each $y \in \{0,1\}^{4t+\lceil 6\log n\rceil}$ as indexing a function $h_y \in H$, and define $\tau(x,y)$ to be the first $4\lceil \frac{9}{8}t\rceil + \lceil 6\log n\rceil$ bits of the concatenation of y and $h_y(x)$. Since the length of this concatenation is $4t + \lceil 6\log n\rceil + \lfloor 3t/4\rfloor$ which is at least $4\lceil \frac{9}{8}t\rceil + \lceil 6\log n\rceil$ for $t \geq 11$, we can see that τ is well-defined. □

Let s be as above. Let $t_s = 32$ and $t_{k-1} = \lceil \frac{9}{8}t_k\rceil$ for $0 < k \leq s$. Set $p_k = 4t_k + \lceil 6\log n\rceil$ for $0 \leq k \leq s$. Now, using the function τ, we define a family of functions

$$C_k : (\{0,1\}^n)^k \times \{0,1\}^{p_k} \to (\{0,1\}^n)^{k-1} \times \{0,1\}^{p_{k-1}}$$

for $1 \leq k \leq s$ as follows: for $z_1, \ldots, z_k \in \{0,1\}^n, y_k \in \{0,1\}^{p_k}$,

$$C_k(z_1 \ldots z_{k-1} z_k y_k) = z_1 \ldots z_{k-1} y_{k-1}, \text{ where } y_{k-1} = \tau(z_k, y_k).$$

Let $C^* = C_1 \circ C_2 \circ \ldots \circ C_s$ and let $q = p_s$ (thus $q = O(\log n)$). Finally, for each $z \in Z$ and $y \in \{0,1\}^q$, $C(z,y)$ is defined to be the first m bits of $C^*(z,y)$, where in C^* we view z as a series of s blocks of n bits each. To see that function C is well-defined, we notice that the length of $C^*(z,y)$ is $p_0 = 4t_0 + \lceil 6\log n\rceil$. This is bigger than m since by definition $t_{k-1} = \lceil \frac{9}{8}t_k\rceil$ for $0 < k \leq s$ and thus $t_0 \geq (\frac{9}{8})^s t_s \geq n^\lambda t_s \geq m$.

Thus far we have seen the constructions of G_A and G_C and hence that of our final disperser $G_D = G_A \circ G_C$, and the efficiency of G_D is easily seen. In the following sections, we will prove the following lemma, which clearly implies the Main Theorem:

Main Lemma: $\forall \xi, \lambda, 1 \geq \xi > \lambda \geq 0$, $\exists n_0(\xi,\lambda)$ *such that if* $n \geq n_0(\xi,\lambda)$, *then for any* T, m *such that* $\log T \geq n^\xi$ *and* $m \leq n^\lambda$, $G_D(V,W,E)$ *is a* $(2^n, 2^m, T)$*-disperser.*

3.5 Proof of the Main Lemma

Let $1 \geq \xi > \lambda \geq 0$ be arbitrary but fixed, and $n_0(\xi, \lambda)$ be a sufficiently large constant. (n_0 will be specified as the proof proceeds.) Let $n \geq n_0$, and T and m be any integers such that $\log T \geq n^{\xi}$ and $m \leq n^{\lambda}$.

As defined in Section 3.4, let $a = \frac{\lambda}{\log \frac{9}{8}}, s = \lceil a \log n \rceil$ and $r = sn$. We take $V = \{0,1\}^n, W = \{0,1\}^m$ and $Z = \{0,1\}^r$. We want to show that $G_D(V, W, E) = G_A(V, Z, E_A) \circ G_C(Z, W, E_C)$ as constructed in Section 3.4 is a $(2^n, 2^m, T)$-disperser.

For the proof, we define the parameters η_i for $0 \leq i \leq 3$ by $\eta_i = \xi(1 - i/3) + \lambda(i/3)$. Note that $\xi = \eta_0 > \eta_1 > \eta_2 > \eta_3 = \lambda$. Define $\delta_i = \delta_i(n) = n^{\eta_i - 1}$.

Let π^* denote the (sn, s)-segmentation of $[1, sn]$ into s equal-sized blocks each of length n. We denote by $\mathcal{B}(Z, \pi^*, b)$ the set of all block-wise (π^*, b)-sources on Z. In the next two sections, we will prove the following two lemmas:

Lemma 3.5.1 *$G_A = (V, Z, E_A)$ is a $(\mathcal{T}(V, \delta_0 n), \mathcal{B}(Z, \pi^*, \delta_2 n), \epsilon = 1/4)$-carrier.*

Lemma 3.5.2 *$G_C = (Z, W, E_C)$ is a strong $(\mathcal{B}(Z, \pi^*, \delta_2 n), U_W, \epsilon = 1/4)$-carrier.*

Remark: Both lemmas can easily be strengthened to $\epsilon = n^{-c}$ for any constant c, provided that n_0 is chosen large enough depending on c.

These two lemmas together with Lemma 3.2.5 imply that $G_D = (V, W, E)$ is a $(\mathcal{T}(V, \log T), U_W, 1/2)$-carrier. The Main Lemma follows from Lemma 3.2.4.

In what follows, we shall first examine the properties of G_A and prove Lemma 3.5.1 in Section 3.6. Then in Section 3.7 we show the proof of Lemma 3.5.2 for G_C.

3.6 Converting Weak Sources to Block-wise Sources

Our goal is to show that $G_A = (V, Z, E_A)$ is a $(\mathcal{T}(V, \delta_0 n), \mathcal{B}(Z, \pi^*, \delta_2 n), \epsilon)$-carrier. To do this, we fix an arbitrary δ_0-source D on $\{0,1\}^n$, and show that there is a converter Λ carried by G_A, such that $D\Lambda$ is ϵ-near to a block-wise $(\pi^*, \delta_2 n)$-source on $\{0,1\}^r$. It is important to emphasize the order of quantifiers here: we do not need one converter that works for all δ_0-sources, but can choose a different converter for each δ_0-source D.

Recall that for each vertex $x \in V$, its incident edges in G_A correspond to the (n,s)-segmentations in $\Pi(n,s,d)$. The converter we construct for D will be of a particularly simple form. For each $x \in V$ we will choose a segmentation $\pi_x \in \Pi(n,s,d)$. We then consider the edge set

$$g = \{(x, f_{\pi_x}(x)) | x \in \{0,1\}^n\} \subseteq E_A.$$

Observe that g is a function mapping V to Z, it thus induces a converter Λ^g for V and Z. We take as our converter the matrix Λ^g.

We will choose the segmentations π_x, and then show that for the resulting converter Λ^g, $D\Lambda^g$ is ϵ-near to a block-wise $(\pi^*, \delta_2 n)$-source.

3.6.1 Smoothing the Distribution

Let D be the fixed δ_0-source. It turns out that the analysis in the later sections will be simplified if we know that D is a smooth source. The next lemma says that any source is close to a smooth source with almost the same min-entropy.

Lemma 3.6.1 *There is a constant $c_0 \in (0,1)$ such that if D is a δ-source on $\{0,1\}^n$ with $1/n \leq \delta \leq 1/2$, then there is an n-bit source D' that is $2^{-c_0 \delta n}$-near to D and is $\psi(\delta)$-smooth, where the function $\psi : (0,1) \to (0,1)$ is defined as $\psi(\delta) = c_0 \delta / \log \delta^{-1}$.*

Proof: Let κ denote the probability with respect to D that x has fewer than $\psi(\delta)n$ good bits. We first give an upper bound on κ.

By Proposition 3.2.4, for any $\rho \in (0, \frac{1}{2}]$, the total number of n-bit strings x from $\mathrm{supp}(D)$ with $w(\chi^D(x)) \leq \rho n$ is at most

$$\sum_{i=0}^{\lfloor \rho n \rfloor} \binom{n}{i} \leq \rho n \binom{n}{\rho n} \leq \rho n (\frac{en}{\rho n})^{\rho n} \leq \rho n (\frac{e}{\rho})^{\rho n}.$$

To get an upper bound on κ we let $\rho = \psi(\delta) \leq \frac{1}{2}$. Since every string from D has probability at most $2^{-\delta n}$, we have $\kappa \leq 2^{-\delta n} \rho n (\frac{e}{\rho})^{\rho n}$. Elementary estimates show that if $\rho = c_0 \delta / \log \delta^{-1}$ where c_0 is sufficiently small, then $\kappa \leq 2^{-c_0 \delta n}$.

Next we define the distribution D' as follows: for $x \in \{0,1\}^n$,

$$D'(x) = \begin{cases} \frac{D(x)}{1-\kappa} & \text{if } w(\chi^D(x)) \geq \psi(\delta)n + 1 \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to check that D' is a well-defined distribution and $\|D' - D\| \leq \kappa$. To see that D' is $\psi(\delta)$-smooth, we notice that for each $x \in \{0,1\}^n$ and $1 < i \leq n$, $I_i^{D'}(x) = I_i^D(x)$ and therefore $w(\chi^{D'}(x)) \geq w(\chi^D(x)) - 1$. So by the definition of D', $w(\chi^{D'}(x)) \geq \psi(\delta)n$ for every $x \in \mathrm{supp}(D')$. □

Remark: In fact, it is not difficult to see that D' actually has min-entropy $\delta n - 2^{-c_0 \delta n + 1}$.

By this lemma, there is a $\psi(\delta_0)$-smooth source D' that is $\epsilon/2$-near to D for sufficiently large n. In what follows we will show that for an appropriate g, $D'\Lambda^g$ is $\epsilon/2$-near to a block-wise $(\pi^*, \delta_2 n)$-source. By Proposition 3.2.3 (2) and (3), $D\Lambda^g$ is ϵ-near to a block-wise $(\pi^*, \delta_2 n)$-source.

3.6.2 Extracting Blocks Using Segmentations

In this section we specify the converter Λ^g by choosing for each $x \in \{0,1\}^n$ a segmentation π_x from $\Pi(n, s, d)$.

In Section 3.4.1 we constructed a family $\Pi(n, s, d)$ of (n, s)-segmentations. This family was chosen to satisfy two properties. The first, which was noted earlier, is that its size is at most n^d. The second is as follows (recall the definition of t-equitable from Section 3.2.4):

Lemma 3.6.2 *Let n, d, s, t be positive integers with $n \geq d, s, t$ and let $H = \lceil \log_d n \rceil$, i.e., H is the least integer such that $d^H \geq n$. Let $\varphi \in (0,1)$. If α is an n-bit sequence with $w(\alpha) \geq \frac{2t}{\varphi^H (1-\varphi)^s}$, then there is a segmentation $\pi \in \Pi(n, s, d)$ that is t-equitable with respect to α.*

Proof: We define an algorithm **Segment** which takes as input an n-bit string α and d, s, t, φ as in the lemma, and computes an (n, k)-tower $Q = (Q_0 = 0, Q_1, \ldots, Q_{k-1}, Q_k = n)$ with $k \leq s$ whose corresponding (n, k)-segmentation is contained in $\Pi(n, k, d)$. The lemma follows immediately from:

Claim: If the hypotheses of the lemma are satisfied, then the segmentation π corresponding to the output Q of **Segment** is an (n, s)-segmentation that is t-equitable with respect to α.

The algorithm **Segment** is given formally below. For notational simplicity, we denote $w(\alpha_{[i,j]})$ by $w([i,j])$ and call it the *weight in the block* $[i,j]$.

The algorithm first constructs the balanced d-ary tree T on $[1,n]$ and then traverses a subset of the nodes of T in a top-down and left-to-right fashion. The integers $Q_1, Q_2, \ldots$ are generated sequentially during the traversal; the parameter q represents the length of the sequence generated so far, i.e., $Q_1, \ldots, Q_q$ have been selected. We initialize $q = 0$ and $Q_0 = 0$.

At any point of time, the algorithm looks at a particular node v_h in T at depth h, where h is an integer parameter initialized to be 0. Recall that $L(v)$ denotes the largest leaf label in the subtree of v. The interval $[Q_q + 1, L(v_h)]$ is called the *active range* from which all further Q_i will be selected. The algorithm examines left-to-right the children of v_h, denoted $C_1(v_h), C_2(v_h), \ldots, C_{d(v_h)}(v_h)$, where $d(v)$ represents the number of children of node v. (By definition, $d(v) \leq d$ for all v of T.) Each iteration of the loop in the algorithm corresponds to the examination of such a node.

If the i-th child of v_h is being examined, the algorithm takes $L(C_i(v_h))$ as a possible candidate for Q_{q+1}. One of three actions is taken depending on the weight in the block $[Q_q + 1, L(C_i(v_h))]$. If the weight in the block is "too small", i.e., less than t, then $L(C_i(v_h))$ is rejected. If the weight in the block is at least t, but not "too large", i.e., less than $\varphi w([Q_q + 1, L(v_h)])$, then Q_{q+1} is chosen to be $L(C_i(v_h))$. Finally if the weight in the block is too large, i.e., at least $\varphi w([Q_q + 1, L(v_h)])$, then the active range is reduced by setting $v_{h+1} = C_i(v_h)$. Consequently the algorithm moves one level down in the tree and h is increased by 1. The loop repeats until either $i > d(v_h)$ or $q \geq s$ or $h \geq h_{max}$ (where $h_{max} = \lceil \log_d \frac{n}{t} \rceil + 1$).

Algorithm **Segment**

Input: $\alpha \in \{0,1\}^n$, integers d, s, t, and $\varphi \in (0,1)$

1. Construct the balanced d-ary tree T on $[1,n]$;

2. $h_{max} \leftarrow \lceil \log_d \frac{n}{t} \rceil + 1$.

3. $q \leftarrow 0; Q_0 \leftarrow 0; h \leftarrow 0; v_0 \leftarrow \mathrm{root}(T); i \leftarrow 1;$

4. **Loop**

5. **if** $w([Q_q + 1, L(C_i(v_h))]) < t$

6. **then** $\{i \leftarrow i + 1;\}$

7. **else if** $t \leq w([Q_q + 1, L(C_i(v_h))]) < \varphi w([Q_q + 1, L(v_h)])$

8. **then** $\{Q_{q+1} \leftarrow L(C_i(v_h)); q \leftarrow q + 1; i \leftarrow i + 1;\}$

9. **else if** $w([Q_q + 1, L(C_i(v_h))]) \geq \varphi w([Q_q + 1, L(v_h)])$

10. **then** $\{v_{h+1} \leftarrow C_i(v_h); h \leftarrow h + 1; i \leftarrow 1;\}$

11. **Until** $(i > d(v_h))$ **or** $(q \geq s)$ **or** $(h \geq h_{max})$;

12. $k = q;$

Output: $Q = (0, Q_1, Q_2, \ldots, Q_{k-1}, n)$

The algorithm always terminates since at each iteration at least one of the i, q, h increases and therefore the loop condition will eventually be violated. It is easy to see from the description of the algorithm that at the termination, the sequence of vertices $v_0, v_1, \ldots, v_h$ form a path from the root and all the Q_j have been chosen as $L(u)$ for some u that is a left sibling of some node v_i on the path. Comparing this fact with the construction of $\Pi(n, k, d)$, we can see that the (n, k)-segmentation π corresponding to the output (n, k)-tower Q is contained in $\Pi(n, k, d)$.

We show that at the end of the algorithm, $w([Q_{i-1} + 1, Q_i]) \geq t$ for $1 \leq i \leq k$ and $k = s$. These two conditions would imply that π is an (n, s)-segmentation and is t-equitable with respect to α, which is sufficient for the proof of the claim and the lemma. By generating Q_s (rather than stopping at Q_{s-1}), we do not have to treat the last block of π separately; since the last block of π, $[Q_{k-1} + 1, n]$ contains $[Q_{k-1} + 1, Q_k]$, the first condition for $i = k$ guarantees that the last block has enough weight.

The first condition is obvious by Step 7 and 8 in the algorithm. To prove $k = s$, we first claim that the following invariants hold at the end of each iteration:

$$w([Q_q + 1, L(C_{i-1}(v_h))]) < t, \tag{3.1}$$

$$w([Q_q + 1, L(v_h)]) \geq \varphi^h(1-\varphi)^q w(\alpha), \tag{3.2}$$

$$w([Q_q + 1, L(v_h)]) < t + n/d^{h-1}. \tag{3.3}$$

(When $i = 1$, we define $L(C_0(v))$ to be the largest leaf label that precedes the subtree of v.) We emphasize that in the above invariants, we refer to the values of the parameters i, q and h at the end of the iteration.

We show the claim by induction on the number of iterations. These invariants hold initially before entering the loop. During an iteration, if Step 6 is executed then there is clearly no effect on the invariants (3.2) and (3.3). Invariant (3.1) also holds because of the precondition in Step 5. If Step 8 is executed then $w([Q_q + 1, L(C_{i-1}(v_h))])$ is 0 at the end and so invariant (3.1) holds trivially. After this step q increases by 1, $w([Q_q+1, L(v_h)])$ decreases and h is unchanged. Thus invariant (3.3) holds by induction. The precondition in Step 7 guarantees that $w([Q_q + 1, L(v_h)])$ is at least $1 - \varphi$ times what it was and so invariant (3.2) also holds. Let us now consider the case that Step 10 is executed. Then the left hand side of invariant (3.1) is $w([Q_q + 1, L(C_0(v_h))])$, which is exactly the $w([Q_q + 1, L(C_{i-1}(v_h))])$ at the end of the previous iteration, and is thus less than t by induction. Since $w([Q_q + 1, L(v_h)])$ is at least φ times what it was and h increases by 1, invariant (3.2) is also maintained. After this step, $w([Q_q + 1, L(v_h)])$ is equal to the sum of $w([Q_q + 1, L(C_0(v_h))])$ (which is less than t by invariant (3.1)) and the size of the interval labelling v_h (which is at most n/d^{h-1}). So invariant (3.3) holds as well.

Now suppose on the contrary that the algorithm terminates with $k < s$. Then at the termination we must have either $i > d(v_h)$ or $h \geq h_{max}$.

If $i > d(v_h)$, then either Step 6 or Step 8 is executed at the last iteration with $i = d(v_h)$. We note that $L(C_{d(v)}(v)) = L(v)$ for any v. If Step 6 is executed, then by the precondition of this step, $w[Q_q+1, L(v_h)] < t$. But this together with invariant (3.2) violate the hypothesis about $w(\alpha)$. The precondition of Step 8 can never be satisfied

in this case since we assumed $\varphi < 1$.

If $h \geq h_{max}$, then invariants (3.2) and (3.3) together imply that $\varphi^h(1-\varphi)^q w(\alpha) < 2t$, which again violates the hypothesis. The proof is complete. □

Corollary 3.6.1 *Let ξ, λ, s and η_i, δ_i for $i = 0, 1$ be as defined at the beginning of Section 3.5. There exists an integer $d(\xi, \lambda)$ such that for all sufficiently large n, if D' is a $\psi(\delta_0)$-smooth n-bit source, then for each string $x \in supp(D')$ there is a segmentation $\pi \in \Pi(n, s, d)$ that is $\delta_1 n$-equitable with respect to $\chi^{D'}(x)$.*

Proof: Fix any $x \in \mathrm{supp}(D')$ and let $\alpha = \chi^{D'}(x)$. Since D' is $\psi(\delta_0)$-smooth, $w(\alpha) \geq \psi(\delta_0)n$. Let $t = \delta_1 n$. We will show that constants d and φ can be chosen to satisfy $\psi(\delta_0)n \geq \frac{2t}{\varphi^H(1-\varphi)^s}$ for sufficiently large n so that the hypotheses in Lemma 3.6.2 hold. This will be sufficient for the proof of the corollary.

Since $s < 1 + a\log n$ and $H < 1 + \log n/\log d$, it suffices to have

$$\frac{c_0 n^{\eta_0 - 1}}{(1-\eta_0)\log n} n \geq \frac{2n^{\eta_1 - 1} n}{\varphi^{1+\log n/\log d}(1-\varphi)^{1+a\log n}},$$

which is equivalent to

$$\begin{aligned}(\eta_0 - \eta_1)\log n \geq \quad & 1 + \log c_0^{-1} + \log(1-\eta_0) + \log\varphi^{-1} + \log(1-\varphi)^{-1} \\ & + \log\log n + (\log\varphi^{-1}/\log d + a\log(1-\varphi)^{-1})\log n.\end{aligned}$$

Choosing φ sufficiently close to 0 (depending only on a, η_0, η_1 which in turn depend only on ξ and λ) and then choosing d sufficiently large (depending on φ) makes the coefficient of $\log n$ on the right less than $(\eta_0 - \eta_1)$. Then the inequality holds for all sufficiently large n. For example, we can set $\varphi = 1 - 2^{(\eta_1 - \eta_0)/3a}$ and $d = 2^{3\log\varphi^{-1}/(\eta_0 - \eta_1)}$ so that the coefficient of $\log n$ on the right is $2(\eta_0 - \eta_1)/3$. □

We now define π_x for $x \in \mathrm{supp}(D')$ to be the segmentation given by the corollary. For $x \notin \mathrm{supp}(D')$, define π_x to be an arbitrary segmentation of $\Pi(n, s, d)$. Then, $g : V \to Z$ defined by $g(x) = f_{\pi_x}(x)$ specifies the converter Λ^g. To complete the proof of Lemma 3.5.1, it now suffices to show

Lemma 3.6.3 *$D'\Lambda^g$ is $\epsilon/2$-near to a block-wise $(\pi^*, \delta_2 n)$-source on $\{0,1\}^r$.*

3.6.3 The Correctness Proof of the Conversion

We need to show that the distribution $D'\Lambda^g$ is within $\epsilon/2$ of a block-wise $(\pi^*, \delta_2 n)$-source. To do this, we will define a collection of distributions $(E^*_\pi : \pi \in \Pi(n,s,d))$, and express $D'\Lambda^g$ as a convex combination of such distributions. We will then use Lemma 3.2.1 to upper bound the distance of $D'\Lambda^g$ from a block-wise $(\pi^*, \delta_2 n)$-source, in terms of the distances of each $E^*\pi$ from such a source.

To define the distributions E^*_π, we first define for each $\pi \in \Pi(n,s,d)$ the *segmentation class* of π to be $S_\pi = \{x \in \mathrm{supp}(D') | \pi_x = \pi\}$. The next proposition follows easily from Corollary 3.6.1:

Proposition 3.6.1 *Suppose $\pi = (B_1, \ldots, B_s) \in \Pi(n,s,d)$. Then S_π is not empty implies that $|B_i| \geq \delta_1 n$ for all $1 \leq i \leq s$. Moreover, for any $x \in S_\pi$ and $1 \leq k \leq s$, we have $D'(x_{B_k} | x_{B_{[1,k-1]}}) \leq 2^{-\delta_1 n}$, i.e., $I^{D'}_{B_k}(x) \geq \delta_1 n$.*

Let $\kappa_\pi = D'(S_\pi)$.

By definition, the map g acts on all $x \in S_\pi$ as follows: segment the bits of x according to π, and pad enough 0's to each segment so that its length is n. That is, $g(x) = f_\pi(x)$ for $x \in S_\pi$. Note that while the map g is not necessarily one-to-one on $\{0,1\}^n$, it is one-to-one when restricted to any S_π. Since $\{S_\pi : \pi \in \Pi(n,s,d)\}$ clearly forms a partition of $\mathrm{supp}(D')$, we have $\sum_\pi \kappa_\pi = 1$.

If $\kappa_\pi > 0$, we define E_π to be the probability distribution over $\{0,1\}^n$ conditioned on S_π, i.e., for each $x \in \{0,1\}^n$,

$$E_\pi(x) = \begin{cases} \frac{D'(x)}{\kappa_\pi} & \text{if } x \in S_\pi \\ 0 & \text{otherwise.} \end{cases}$$

For $\kappa_\pi = 0$, we take E_π to be an arbitrary distribution. It is easy to see that $D' = \sum_\pi \kappa_\pi E_\pi$.

We define E^*_π to be the probability distribution on $\{0,1\}^r$ such that for $y \in \{0,1\}^r$:

$$E^*_\pi(y) = \begin{cases} E_\pi(x) & \text{if } \exists\ x \in S_\pi \text{ and } g(x) = y \\ 0 & \text{otherwise.} \end{cases}$$

From the fact that g is one-to-one on S_π through f_π, it is easy to check that E^*_π is well-defined and $E^*_\pi = E_\pi \Lambda^g = E_\pi \Lambda^{f_\pi}$.

We can now express $D'\Lambda^g$ as a convex combination of the E^*_π:

$$D'\Lambda^g = \sum_\pi \kappa_\pi E_\pi \Lambda^g = \sum_\pi \kappa_\pi E^*_\pi.$$

We want to use this to upper bound the distance of $D'\Lambda^g$ from a block-wise $(\pi^*, \delta_2 n)$ source. Let ϵ_π denote the distance from E_π to the family of block-wise $(\pi, \delta_2 n)$-sources on $\{0,1\}^n$. Thus for each $\pi \in \Pi(n,s,d)$ there is a block-wise $(\pi, \delta_2 n)$-source F_π on $\{0,1\}^n$ that is ϵ_π-near to E_π. Let F^*_π be the distribution on $\{0,1\}^r$ defined as follows: for $y \in \{0,1\}^r$,

$$F^*_\pi(y) = \begin{cases} F_\pi(x) & \text{if } f_\pi(x) = y \\ 0 & \text{otherwise.} \end{cases}$$

Since f_π clearly defines a one-to-one function on $\{0,1\}^n$, we can see that $F^*_\pi = F_\pi \Lambda^{f_\pi}$. Following the definition of f_π, we can also see that for any n-bit string x and $1 \le i \le s$, the information in the i-th block (w.r.t. π) of x relative to F_π is exactly the same as the information in the i-th block (w.r.t. π^*) of $f_\pi(x) \in \{0,1\}^r$ relative to F^*_π. Therefore, since F_π is a block-wise $(\pi, \delta_2 n)$-source on $\{0,1\}^n$, F^*_π is a block-wise $(\pi^*, \delta_2 n)$-source on $\{0,1\}^r$.

Observe:

$$\|E^*_\pi - F^*_\pi\| = \|E_\pi \Lambda^{f_\pi} - F_\pi \Lambda^{f_\pi}\| \le \|E_\pi - F_\pi\| = \epsilon_\pi.$$

Now by Lemma 3.2.1 and Lemma 3.2.3,

Lemma 3.6.4 *$D'\Lambda^g$ is γ-near to a block-wise $(\pi^*, \delta_2 n)$-source on $\{0,1\}^r$, where $\gamma = \sum_{\pi \in \Pi(n,s,d)} \kappa_\pi \epsilon_\pi$.*

So it suffices to upper bound γ in this lemma by $\epsilon/2$. We will prove:

Lemma 3.6.5 *For any $\pi \in \Pi(n,s,d)$ with $\kappa_\pi > 0$,*

$$\epsilon_\pi \le \frac{s 2^{(\delta_2 - \delta_1)n}}{\kappa_\pi}.$$

Combined with Lemma 3.6.4, we obtain $\gamma \leq |\Pi(n,s,d)| s 2^{(\delta_2-\delta_1)n}$. Thus to show $\gamma \leq \epsilon/2$ it suffices to show: $(\delta_1 - \delta_2)n \geq 1 + \log \epsilon^{-1} + \log s + \log |\Pi(n,s,d)|$. The left hand side is $n^{\Omega(1)}$ and the right hand side is bounded by a constant times $\log n$, so the desired bound on γ holds for sufficiently large n.

Thus it remains to prove Lemma 3.6.5.

Proof of Lemma 3.6.5:

Fix any $\pi \in \Pi(n,s,d)$ with $\kappa_\pi > 0$. We need to upper bound the distance of E_π from the family of block-wise $(\pi, \delta_2 n)$-sources on $\{0,1\}^n$. We first prove a general upper bound on the distance of an arbitrary n-bit source F from the family of block-wise (π, b)-sources on $\{0,1\}^n$.

The reader will find it useful to recall the definition of the π-tree representation of a distribution F on $\{0,1\}^n$. Recall that F is a block-wise (π, b)-source if and only if every edge label in this tree is at most 2^{-b}. With this in mind, we make the following definitions:

Definition 3.6.1 *Suppose F is an n-bit source. Let $\pi = [B_1, \ldots, B_s]$ be an (n,s)-segmentation and k be an integer satisfying $1 \leq k \leq s$. $x \in \{0,1\}^n$ is said to be* (π, b)-bad at block k relative to distribution F *if $F(x_{B_k} | x_{B_{[1,k-1]}}) > 2^{-b}$, i.e., the label of the k-th edge on the path from the root to x in T^π is greater than 2^{-b}. $x \in \{0,1\}^n$ is* (π, b)-bad relative to F *if there is a k such that x is (π, b)-bad at block k. We denote by $\beta_F(\pi, b; k)$ the probability that an n-bit string randomly chosen according to F is (π, b)-bad at block k and by $\beta_F(\pi, b)$ the probability that an n-bit string randomly chosen according to F is (π, b)-bad.*

So F is a block-wise (π, b)-source on $\{0,1\}^n$ if and only if none of the strings is (π, b)-bad. Generally, we have the following bound on the distance of F from a block-wise (π, b)-source:

Lemma 3.6.6 *Let F be an n-bit source, let b be a positive real number and let $\pi = [B_1, \ldots, B_s]$ be an (n,s)-segmentation in which each block has at least b elements. Then F is $\beta_F(\pi, b)$-near to a block-wise (π, b)-source.*

Proof: We will construct a block-wise (π, b)-source F' with the property that $F'(x) \leq F(x)$ implies that x is (π, b)-bad relative to F. By Proposition 3.2.3(1), this implies that F' is $\beta_F(\pi, b)$-near to F.

Consider the π-tree representation for F. We modify the probability labels of this tree to obtain the distribution F'. The tree for F' must satisfy that the conditional probability assigned to each edge is at most 2^{-b}. Consider an internal node z at depth $k < s$. The edges from z correspond to binary strings of length $|B_{k+1}| \geq b$. Thus z has at least 2^b edges coming from it. Therefore it is possible to choose $F'(y|z)$ for $y \in \{0,1\}^{|B_{k+1}|}$ so that $F'(y|z) = 2^{-b}$ if $F(y|z) > 2^{-b}$, $F(y|z) \leq F'(y|z) \leq 2^{-b}$ if $F(y|z) \leq 2^{-b}$, and the sum of $F'(y|z)$ over strings y of length $|B_{k+1}|$ is 1. Doing this for all internal nodes z yields the desired F'. It is easy to see that if x is not (π, b)-bad relative to F then the conditional probability labels on the path to x are at least as big for F' as for F, and so $F'(x) \geq F(x)$. □

By Proposition 3.6.1, for any $\pi \in \Pi(n, s, d)$ with $\kappa_\pi > 0$, each block in π has at least $\delta_1 n$ elements. Then the above lemma tells us that $\beta_{E_\pi}(\pi, \delta_2 n) \geq \epsilon_\pi$. Since $\beta_{E_\pi}(\pi, b) \leq \sum_{k=1}^{s} \beta_{E_\pi}(\pi, b; k)$, we will complete the proof of Lemma 3.6.5 by showing:

$$\beta_{E_\pi}(\pi, b; k) \leq \frac{2^{b-\delta_1 n}}{\kappa_\pi}. \tag{3.4}$$

By definition, $\beta_{E_\pi}(\pi, b; k)$ is equal to the sum of $E_\pi(x)$ over all the $x \in \mathrm{supp}(E_\pi)$ that are (π, b)-bad at block k relative to E_π. We know that for any $x \in \{0,1\}^n$, x is (π, b)-bad at block k relative to E_π if $E_\pi(x_{B_k}|x_{B_{[1,k-1]}}) > 2^{-b}$. Moreover, for $x \in S_\pi$, we have:

$$\begin{aligned}
E_\pi(x_{B_k}|x_{B_{[1,k-1]}}) &= \frac{E_\pi(x_{B_{[1,k]}})}{E_\pi(x_{B_{[1,k-1]}})} \\
&= \frac{D'(x_{B_{[1,k]}})}{\kappa_\pi E_\pi(x_{B_{[1,k-1]}})} \frac{D'(x_{B_{[1,k-1]}})}{D'(x_{B_{[1,k-1]}})} \\
&= D'(x_{B_k}|x_{B_{[1,k-1]}}) \frac{D'(x_{B_{[1,k-1]}})}{\kappa_\pi E_\pi(x_{B_{[1,k-1]}})} \\
&\leq \frac{2^{-\delta_1 n}}{\kappa_\pi} \frac{D'(x_{B_{[1,k-1]}})}{E_\pi(x_{B_{[1,k-1]}})},
\end{aligned}$$

where the inequality follows from Proposition 3.6.1. Putting these together, we have

that a necessary condition for $x \in S_\pi$ to be (π, b)-bad at block k relative to E_π is:

$$E_\pi(x_{B_{[1,k-1]}}) \leq \frac{2^{b-\delta_1 n}}{\kappa_\pi} D'(x_{B_{[1,k-1]}}). \tag{3.5}$$

Let X be the set of all $x \in S_\pi = \mathrm{supp}(E_\pi)$ that satisfy (3.5). The sum of $E_\pi(x)$ over $x \in X$ is then an upper bound on $\beta_{E_\pi}(\pi, b; k)$. Define $q = |B_{[1,k-1]}|$, and note that condition (3.5) depends only on the first q bits of x. Recall that $E_\pi^{(q)}$ denotes the q-bit truncation of E_π and $D'^{(q)}$ is the q-bit truncation of D'. Let Y be the set of all $y \in \{0,1\}^q$ such that $E_\pi^{(q)}(y) \leq 2^{b-\delta_1 n} D'^{(q)}(y)/\kappa_\pi$. By the definition of q-bit truncation, we know that for any $y \in \{0,1\}^q$, $E_\pi^{(q)}(y) = E_\pi(y)$ and $D'^{(q)}(y) = D'(y)$. It is then clear that the sum of $E_\pi^{(q)}(y)$ over $y \in Y$ is equal to the sum of $E_\pi(x)$ over $x \in X$. So finally we have:

$$\begin{aligned}
\beta_{E_\pi}(\pi, d; k) &\leq \sum_{y \in Y} E_\pi^{(q)}(y) \\
&\leq \sum_{y \in Y} \frac{2^{b-\delta_1 n} D'^{(q)}(y)}{\kappa_\pi} \\
&= \frac{2^{b-\delta_1 n}}{\kappa_\pi} \sum_{y \in Y} D'^{(q)}(y) \\
&\leq \frac{2^{b-\delta_1 n}}{\kappa_\pi},
\end{aligned}$$

which establishes inequality (3.4), and completes the proof. □

3.7 Converting Block-wise Sources to a Uniform Source

In this section we show Lemma 3.5.2 for G_C. The proof we present is a variant of the correctness proof of the extractor (Function C) in [NZ93].

In Section 3.4.2 we defined the parameters $t_s = 32$ and $q = p_s = 4t_s + \lceil 6 \log n \rceil$ (thus $q = O(\log n)$). Define Λ to be the matrix on $Z \times W$ such that for $z \in Z, w \in W$,

$$\Lambda(z, w) = |\{y \in \{0,1\}^q |\ C(z, y) = w\}|/2^q.$$

It is clear that Λ is a converter and that G_C is the support of Λ. To prove Lemma 3.5.2, now it suffices to show that Λ transforms every distribution in $\mathcal{B}(Z, \pi^*, \delta_2 n)$ to a

distribution that is ϵ-near to U_W. Fix any $D \in \mathcal{B}(Z, \pi^*, \delta_2 n)$ and we want to show that $\|D\Lambda - U_W\| \leq \epsilon$.

First we notice that, in fact,

$$\Lambda(z, w) = \sum_{y \in \{0,1\}^q} \Lambda^C((z, y), w) U_q(y),$$

where Λ^C is the converter for $Z \times \{0,1\}^q$ and W induced by the function C. Therefore $D\Lambda = (D \times U_q)\Lambda^C$. As we know, C is defined to be the m-bit truncation of C^*. Now Lemma 3.2.2 implies that to prove $\|(D \times U_q)\Lambda^C - U_W\| \leq \epsilon$, it suffices to show $\|(D \times U_q)\Lambda^{C^*} - U_{p_0}\| \leq \epsilon$. Furthermore, since $C^* = C_1 \circ C_2 \circ \ldots \circ C_s$, Proposition 3.2.2 says that this is equivalent to showing $\|(D \times U_q)\Lambda^{C_s} \ldots \Lambda^{C_2}\Lambda^{C_1} - U_{p_0}\| \leq \epsilon$.

Let k be any integer between 1 and s. For each $y \in \{0,1\}^{(k-1)n}$, let $D_{k|y}$ denote the distribution on $\{0,1\}^n$ such that for $z \in \{0,1\}^n$, $D_{k|y}(z) = D(z|y)$. Recall that for any integer $j \leq sn$, $D^{(j)}$ denotes the j-bit truncation of D. By definition,

$$D^{((k-1)n)}(y) D_{k|y}(z) = D^{(kn)}(yz) = D(yz).$$

As we know $D = D^{(sn)}$, we want to show

$$\|(D^{(sn)} \times U_{p_s})\Lambda^{C_s} \ldots \Lambda^{C_2}\Lambda^{C_1} - U_{p_0}\| \leq \epsilon. \tag{3.6}$$

We will prove that for each integer k between 1 and s,

$$\|(D^{(kn)} \times U_{p_k})\Lambda^{C_k} - D^{((k-1)n)} \times U_{p_{k-1}}\| \leq 2^{1-t_k/8}. \tag{3.7}$$

Assume this is true. Then by applying Proposition 3.2.3(2) and (3), a straightforward induction shows that the left hand side of (3.6) is at most $\sum_{k=1}^{s} 2^{1-t_k/8} \leq 2 \cdot 2^{1-t_s/8} \leq \epsilon$, where the last inequality follows from $t_s \geq 32$ and $\epsilon = 1/4$. This will complete the proof of the lemma.

So it remains to prove (3.7). Since D is a block-wise $(\pi^*, \delta_2 n)$-source on Z, by definition, for any $1 \leq k \leq s$ and any $y \in \{0,1\}^{(k-1)n}$, $D_{k|y}$ is an n-bit source with min-entropy $\delta_2 n$. This is at least t_k since $t_0 \leq (\frac{9}{8})^s t_s + \sum_{i=0}^{s-1} (\frac{9}{8})^i \leq \delta_2 n$ for n large enough and $t_0 > t_i$ for $1 \leq i \leq s$. Now Corollary 3.4.1 says that the distribution $\tau(D_{k|y} \times U_{p_k})$ on $\{0,1\}^{p_k - 1}$ is quasi-random to within $2^{1-t_k/8}$. Finally,

$$
\begin{aligned}
& \|(D^{(kn)} \times U_{p_k})\Lambda^{C_k} - D^{((k-1)n)} \times U_{p_{k-1}}\| \\
= \; & \frac{1}{2} \sum_{y \in \{0,1\}^{(k-1)n}, u \in \{0,1\}^{p_{k-1}}} D^{((k-1)n)}(y) |\tau(D_{k|y} \times U_{p_k})(u) - U_{p_{k-1}}(u)| \\
= \; & \sum_{y \in \{0,1\}^{(k-1)n}} D(y) \|\tau(D_{k|y} \times U_{p_k}) - U_{p_{k-1}}\| \\
\leq \; & 2^{1-t_k/8} \sum_{y \in \{0,1\}^{(k-1)n}} D(y) \\
= \; & 2^{1-t_k/8}.
\end{aligned}
$$

This completes the proof. □

Remark: In the case that $\epsilon = n^{-O(1)}$, we can take $t_s = 8 \log \frac{1}{\epsilon} + 16$ so that q is still $O(\log n)$ and $2 \cdot 2^{1-t_s/8} \leq \epsilon$.

3.8 Summary of Parameters

Here we give a summary of the parameters that appeared in the construction and the proof:

$1 \geq \xi > \lambda \geq 0$;

$n_0(\xi, \lambda)$ is sufficiently large and $n \geq n_0$;

$\log T \geq n^{\xi}$ and $m \leq n^{\lambda}$;

for $0 \leq i \leq 3$, $\eta_i = \xi(1 - i/3) + \lambda(i/3)$ and $\delta_i = \delta_i(n) = n^{\eta_i - 1}$;

$a = \frac{\lambda}{\log \frac{9}{8}}$, $s = \lceil a \log n \rceil$ and $r = sn$;

$\varphi = 1 - 2^{\frac{\eta_1 - \eta_0}{3a}}$ and $d = 2^{\frac{3 \log \varphi^{-1}}{\eta_0 - \eta_1}}$;

$\epsilon = 1/4$;

$t_s = 8 \log \epsilon^{-1} + 16 = 32$ and $t_{k-1} = \lceil \frac{9}{8} t_k \rceil$ for $0 < k \leq s$;

$p_k = 4t_k + \lceil 6 \log n \rceil$ for $0 \leq k \leq s$;

$q = p_s = O(\log n)$.

Chapter 4

Constructing Sample Spaces with Small Bias on Neighborhoods

4.1 Previous Work and Motivation

One of the approaches to derandomization that has been extensively studied is to use explicit constructions of small-sized sample spaces that guarantee limited independence or almost independence among some random variables so that certain randomness properties of the space are maintained. This approach has been widely used and proven effective in many applications (see e.g. [KW84, Lub85, ABI86, BR89, MNN89, NN90, Alo91, Sch92, KM93, KK94]).

Along the line of this approach, the construction of k-wise independent sample spaces is one of the earliest studied problems. Let Ω be a sample space (multiset) in $\{0,1\}^n$ and suppose $(X_1, X_2, \ldots, X_n)$ is a randomly selected point from Ω according to the uniform distribution. Then Ω is said to be k-wise independent if the random variables within every subset of $X_1, X_2, \ldots, X_n$ of size at most k are mutually independent. An efficient construction of k-wise independent sample spaces of size $n^{\lfloor k/2 \rfloor}$ was presented in [ABI86] (see also [Lub85]), whose size basically matches the lower bound given in [CGHFRS85] (see also [ABI86, KM94]). In a more general setting, Schulman [Sch92] considered the problem of constructing sample spaces uniform over a family of subsets of $\{1, 2, \ldots, n\}$ (neighborhoods), i.e., the distribution induced on the random variables in any particular neighborhood in the family is uniformly independent. The size of the sample space in the construction is $O(d2^l)$, where l is the maximum size of any neighborhood in the family and d is the maximum number of neighborhoods containing any element.

For a subset $S \subseteq \{1, 2, \ldots, n\}$, the quantity $|Pr[\sum_{i \in S} X_i = 1] - Pr[\sum_{i \in S} X_i = 0]|$,

where the sum is modulo 2, is called the *bias* of S ([Vaz86]). Naor and Naor [NN90] introduced the notion of ϵ-biased (resp. k-wise ϵ-biased) sample space, in which the bias of every subset (resp. every subset of size at most k) of $\{1, 2, \ldots, n\}$ is at most ϵ. They presented an efficient construction of ϵ-biased sample spaces of size $O(n/\epsilon^4)$. Alon et al. in [AGHP91] gave three more simple constructions of ϵ-biased sample spaces of size $O(n^2/\epsilon^2)$. Following the work in [NN90], these constructions of ϵ-biased sample spaces gave constructions of k-wise ϵ-biased sample spaces of size $O(k \log n/\epsilon^4)$ and $O(k^2 \log^2 n/\epsilon^2)$, respectively. In [EGLNV92], constructions of sample spaces with small bias for general independent distributions were studied.

Koller and Megiddo in [KM93] introduced the framework of constructing sample spaces that satisfy a set of probability constraints, which generalized the construction approach used in [Sch92]. This method was further extended by Karger and Koller in [KK94] where they showed how to deal with the more general expectation constraints. In the latter work, an NC algorithm for constructing sample spaces that approximate general independent distributions with relative error was shown.

In this chapter, we consider a variant of the problem studied by Schulman, in which we weaken the requirement "uniform on neighborhoods" to "ϵ-biased on neighborhoods". For a given family $\mathcal{F}$ of subsets of $\{1, 2, \ldots, n\}$ (neighborhoods), we construct a small sample space associated with $X_1, X_2, \ldots, X_n$ such that the bias of any particular neighborhood in the family is at most ϵ.

Let us denote the maximum cardinality of any element of $\mathcal{F}$ by $\Upsilon(\mathcal{F})$. Then two previously best known constructions for this problem are of sizes $O(\Upsilon(\mathcal{F}) \log n/\epsilon^4)$ [NN90] and $O(\Upsilon(\mathcal{F})^2 \log^2 n/\epsilon^2)$ [AGHP91], respectively, which are obtained by constructing $\Upsilon(\mathcal{F})$-wise ϵ-biased sample spaces. We shall present a new construction whose size is $O(\log |\mathcal{F}|/\epsilon^2)$. The running time needed in our construction is polynomial in n, ϵ^{-1} and $|\mathcal{F}|$. Since $|\mathcal{F}| \leq n^{\Upsilon(\mathcal{F})}$ always holds, the size of our construction can never exceed the order of the sizes of the previous constructions. In fact, in the case that $\mathcal{F}$ is of size polynomial in n, our construction runs in polynomial time and has size $O(\log n/\epsilon^2)$ which is independent of $\Upsilon(\mathcal{F})$.

It is not difficult to show that if we select $O(\log(|\mathcal{F}|)/\epsilon^2)$ points uniformly at random

from $\{0,1\}^n$, then the resulting set is almost surely a desired sample space. Although an existence proof of this type provides an efficient randomized procedure for constructing a needed sample space, typically it does not help explicit constructions. Nevertheless, one key observation to our construction is that such an existence proof holds even if we select the points from an $\frac{\epsilon}{2}$-biased sample space. Then we apply the method of conditional probabilities to searching for such a desired small subset of points from an $\frac{\epsilon}{2}$-biased sample space. Since the previous work in [NN90] and [AGHP91] give us explicit constructions of such sample spaces of polynomial size, the size of our search space becomes sufficiently small so that we are able accomplish the construction efficiently. The size of the sample space of our construction matches the bound given by the probabilistic argument, which leads to the best known efficient construction of k-wise ϵ-biased sample spaces of size $O(\log n/\epsilon^2)$ for the case that k is fixed.

One motivation for our sample space construction is a problem of designing error-correcting communcation protocols which arises in the context of data replication in distributed database systems.

Usually in a distributed database system, the system maintains several (read-only) physical copies of a single logical database at distinct locations to increase data availability and decrease data access time. Efficient communication protocols for checking data consistency between distinct databases are important for replicated data management.

Consider the situation where two (read-only) copies of a logical database are located far from each other and the communication between them is expensive compared to the local computation. These two databases regularly need to check the data consistency between each other to find and correct possible data errors that may occur unpredictably, e.g. because of undetected hardware failures. Their objective is to minimize the communication needed.

It is a well known fact in communication complexity that in the worst case they can do nothing better than the trivial solution: send a copy of the *whole* database from one side to the other. However, this worst case assumption is overly pessimistic. In our case that the databases are maintained for read-only, typically only a small portion

of data is likely to be inconsistent. In fact, depending on the data-storage layout, the inconsistencies between the two databases most probably would occur in some restricted set of patterns. More precisely, what we would expect is that there is a family of sets of bit positions (difference patterns) such that, in (almost) all the cases, the set of positions where the inconsistencies occur is a member in the family (although neither of the databases should necessarily know which member this is without communication); moreover, the cases where the inconsistencies fail to follow the family of difference patterns are so rare that it would not affect the checking performance if we disregard these cases. For example, suppose that the data stored in each database are divided into blocks and at any point of time, (most likely) there can be at most one of the data blocks in which at most 13 bits differ from the other copy. Then the family of difference patterns we would expect is the family of sets of bit positions of size at most 13 from each block. The questions we would like to ask then are whether the databases can do anything better in this situation and how well they can do.

Abstractly, we have the following communication problem. Let $\mathcal{F}$ be a family of subsets of $\{1, 2, \ldots, n\}$. Two parties A and B with limited computation power (polynomial time machines) are given an n-bit string x and an n-bit string y respectively such that if ever x and y are not the same, the set of positions they differ is (almost certainly) a member in $\mathcal{F}$. Knowing $\mathcal{F}$, A and B want to determine whether or not x and y are equal (error-detecting), and in the case they are not, A and B want to tell the *exact* positions in which x and y differ (error-correcting). The goal is to mininize the communication.

The error-detecting case of this problem is a natrual variant of the *equality* problem in the study of communication complexity (see e.g. [Yao79]), in which A and B have unlimited computation power and try to tell whether or not x and y are the same (using the above language, $\mathcal{F}$ contains all the non-empty subsets of $\{1, 2, \ldots, n\}$).

We will be interested in the case where the size of $\mathcal{F}$ is polynomial in n. Typically, a deterministic protocol for the problem needs $\Omega(\log n)$ bits of communication in this case. On the other hand, if we do not restrict the computation power of both parties, then a simple counting argument shows that there is a deterministic protocol

that uses $O(\log n)$ bits to solve the problem. In our case that both parties have limited computation power, the previously best known protocol one could achieve, which uses $O(\Upsilon(\mathcal{F})\log n)$ bits of communication, is by way of applying the explicit construction of $\Upsilon(\mathcal{F})$-wise ϵ-biased sample spaces [NN90]. In this work, using our construction of sample spaces with small bias on neighborhoods, we close the gap between these two bounds by presenting a deterministic protocol for the problem using $O(\log n)$ bits.

In the rest of this chapter, we first give some basic notation and definitions in Section 4.2. Then in Section 4.3 we present our construction of sample spaces with small bias on neighborhoods. Finally in Section 4.4 we show applications of our construction to designing error-correcting communication protocols and to constructing universal sets.

4.2 Notation, Definitions and Facts

4.2.1 Basic Notation

For a positive integer n, we let $[n]$ denote the set of integers $\{1, 2, \ldots, n\}$ and let $\binom{[n]}{k}$ denote the family of subsets of $[n]$ of size k. If $\mathcal{F}$ is a family of sets, we use $\Upsilon(\mathcal{F})$ to denote the maximum size of any set in the family. We define $\Lambda(\mathcal{F})$ to be the ideal generated by $\mathcal{F}$ excluding the empty set, i.e., $\Lambda(\mathcal{F}) = \{S \mid S \neq \phi \text{ and } S \subseteq T \text{ for some } T \in \mathcal{F}\}$. For example, $\Lambda(\binom{[n]}{k})$ is the family of non-empty subsets of $[n]$ of size at most k. In the case that $\mathcal{F}$ is a singleton set $\{S\}$, we may use $\Lambda(S)$ instead of $\Lambda(\{S\})$ for simplicity. For example, $\Lambda([n])$ is the family of all non-empty subsets of $[n]$. We use the notation $\sqsubseteq$ for multiset inclusion.

In many cases, we will identify a multiset $\Omega \sqsubseteq \{-1, 1\}^n$ with a matrix of dimension $|\Omega| \times n$ whose rows consists of all the points in Ω. (The order of the rows is inconsequential and we may fix it in an arbitrary way.) We denote this matrix by $M(\Omega)$. For a subset $S \subseteq [n]$, we use $\psi_S(M(\Omega))$ to denote the column vector obtained by the coordinate-wise product of the columns of $M(\Omega)$ with indices in S.

In what follows, we will identify the term "sample space" with "multiset" without ambiguity. All the probability distributions are assumed to be uniform.

4.2.2 Sample Spaces with Small Bias

For a multiset $\Omega \sqsubseteq \{-1,1\}^n$ and a subset $S \subseteq [n]$, the *bias of S with respect to Ω* is defined to be

$$|Pr[\Pi_{i\in S} X_i = 1] - Pr[\Pi_{i\in S} X_i = -1]| = |E[\Pi_{i\in S} X_i]|,$$

where the probability is with respect to a randomly selected $(X_1, X_2, \ldots, X_n)$ from Ω.

Definition 4.2.1 *Let $\mathcal{F}$ be a family of subsets of $[n]$. A multiset $\Omega \sqsubseteq \{-1,1\}^n$ is said to be* ϵ-biased with respect to *$\mathcal{F}$ if for each $J \in \mathcal{F}$, the bias of J with respect to Ω is at most ϵ. It is said to be* ϵ-biased *if it is ϵ-biased with respect to $\Lambda([n])$, and it is said to be* k-wise ϵ-biased *if it is ϵ-biased with respect to $\Lambda(\binom{[n]}{k})$.*

We will need the next result due to [AGHP91]:

Theorem 4.2.1 *Let n be a positive integer and $0 < \epsilon \leq 1$. There is an efficient construction of an ϵ-biased sample space in $\{-1,1\}^n$ of size $O(n^2/\epsilon^2)$.*

Here by *efficient construction* we mean that the construction time is polynomial in the size of the output sample space.

The next proposition reformulates the definition of ϵ-bias in terms of matrices:

Proposition 4.2.1 *Let $\mathcal{F}$ be a family of subsets of $[n]$. Suppose $\Omega \sqsubseteq \{-1,1\}^n$ is ϵ-biased with respect to $\mathcal{F}$. Then for any $J \in \mathcal{F}$, the difference between the proportion of 1's and the proportion of -1's in $\psi_J(M(\Omega))$ is within ϵ.*

Remark: In the case that Ω is a multiset of $\{0,1\}^n$ instead of $\{-1,1\}^n$, we may apply the following modifications to make the previous definitions and statements work: we map 1 to 0, -1 to 1 and change the product correspondingly to the sum modulo 2.

4.2.3 The Method of Conditional Probabilities

The method of conditional probabilities, which was introduced by Erdös and Selfridge[ES73], Spencer[Spe87], and Raghavan [Rag88], is a powerful tool in derandomization. In this

section, we review a basic framework for this method in terms of the conditional expectations.

For a set S and a positive integer n, let $\Omega \sqsubseteq S^n$. We call S the *sample range* of Ω. Suppose $V_1, V_2, \ldots, V_n$ are mutually independent random variables on Ω that take values over range S and let $Z = Z(V_1, V_2, \ldots, V_n)$. For $1 \leq i \leq n$ and $v_1, \ldots, v_i \in S$, we use $E[Z|v_1, \ldots, v_i]$ to denote the conditional expectation of Z given $V_1 = v_1, \ldots, V_i = v_i$.

Suppose $E[Z] \leq c$ for some constant c. Knowing S and $\{Pr[V_i = v] \mid 1 \leq i \leq n, v \in S\}$, we want to construct a sample point $\omega \in \Omega$ such that $Z(\omega) \leq c$ (such an ω clearly exists).

We first observe that for any given $v_1, \ldots, v_{i-1} \in S$ where $1 \leq i \leq n$,

$$\begin{aligned} E[Z|v_1, \ldots, v_{i-1}] &= \sum_{v \in S} Pr[V_i = v]\ E[Z|v_1, \ldots, v_{i-1}, v] \\ &\geq \min_{v \in S}\{E[Z|v_1, \ldots, v_{i-1}, v] \mid Pr[V_i = v] \neq 0\}, \end{aligned}$$

where the equality follows from the assumption that the V_i's are mutually independent. Now the construction goes as follows: Sequentially for $i = 1, 2, \ldots, n$, search over all $v \in S$ where $Pr[V_i = v] \neq 0$, to find the one with minimum $E[Z|v_1, \ldots, v_{i-1}, v]$; set $v_i = v$. Finally, let $\omega = (v_1, v_2, \ldots, v_n)$. By the above inequality, it is easy to check that $Z(\omega) \leq c$ as desired.

From the algorithmic point of view, if for each $1 \leq i \leq n$ and for any given $v_1, \ldots, v_i \in S$, the conditional expectation $E[Z|v_1, \ldots, v_i]$ is computable in time $t(n)$, then the total time needed for the construction is clearly of order $nt(n)|S|$. We say that the expectations are *easy to compute* if the function $t(\cdot)$ is a polynomial. In this case, the total time needed for the construction is $n^{O(1)}|S|$.

Remark: Here by *computing* we mean that the expectations can be either calculated exactly, or approximated with sufficient accuracy so as to guarantee that after the i-th iteration for each i, $E[Z|v_1, \ldots, v_i] \leq c$.

Nevertheless, in many circumstances, direct computations of conditional expectations (or conditional probabilities) are not easy. One effective technique to overcome

this difficulty that has been proven successful in applications is by way of employing *pessimistic estimators*, which was first introduced by Raghavan [Rag88]. We describe below a modified version of pessimistic estimators due to Fundia [Fun94] (see also [Sak94]).

Continuing the above discussion, we suppose that for some given $v_1, \ldots, v_i \in S$, the conditional expectation $E[Z|v_1, \ldots, v_i]$ is difficult to compute. We would like then to introduce a new random variable $W = W(V_1, V_2, \ldots, V_n)$ (pessimistic estimator) defined on Ω that satisfies the following conditions:

(A) $Z(\omega) \leq W(\omega)$ for all $\omega \in \Omega$;

(B) $E[W] \leq c$; and

(C) for each $1 \leq i \leq n$ and for all $v_1, \ldots, v_i \in S$, $E[W|v_1, \ldots, v_i]$ is easy to compute.

Now, we can apply the approach described above to W and construct a sample point ω satisfying $W(\omega) \leq c$ in time $n^{O(1)}|S|$. By condition (A), ω is also good for Z.

4.3 Sample Spaces with Small Bias on Neighborhoods

In this section, we present a construction of sample spaces with small bias on designated neighborhoods.

The idea behind our construction is rather straightforward. Suppose $\mathcal{F}$ is a family of subsets of $[n]$ and $0 < \epsilon \leq 1$. We want to construct efficiently a small-sized sample space in $\{-1, 1\}^n$ that is ϵ-biased with respect to $\mathcal{F}$. An easy counting argument shows that if we randomly select $m = O(\log|\mathcal{F}|/\epsilon^2)$ points from $\{-1, 1\}^n$, then with high probability the resulting set is ϵ-biased with respect to $\mathcal{F}$. The starting point of our construction is the observation that even if these points are randomly selected from an $\frac{\epsilon}{2}$-biased sample space $\Omega_{\frac{\epsilon}{2}}$, the resulting set is still almost surely as desired. Notice that in either of these two cases, the random point-selecting process generates a sample space $\Delta = S^m$, where the sample range S of Δ is $\{-1, 1\}^n$ in the former case, and is $\Omega_{\frac{\epsilon}{2}}$ in the latter one; moreover, the counting argument shows that almost all the points in Δ (each of which is a sequence of m points in S) are ϵ-biased with respect

to $\mathcal{F}$. What we would like then is to apply the method of conditional probabilities to searching for such a point from Δ. We certainly would fail in the former case since the construction time needed in this case is more than $|S| = 2^n$. On the other hand, we should succeed in the latter case since the work in [NN90] and [AGHP91] provide us explicit constructions of $\Omega_{\frac{\epsilon}{2}}$ of size polynomial in n and ϵ^{-1}, and therefore, provided that the conditional expectations are easy to compute, the construction time we need is $n^{O(1)}|S|$ which is polynomial in n and ϵ^{-1}. At the same time, the size of the sample space of our construction matches the bound given by the probabilistic argument.

4.3.1 The Existence Proof

We show that in every sample space with small bias, there exists a small (multi)subset of it that still keeps certain desired small-bias property.

We will need a technical lemma whose proof is given in [AS91].

Lemma 4.3.1 *For all $\alpha \in [0,1]$ and all β, $\alpha e^{\beta(1-\alpha)} + (1-\alpha)e^{-\beta\alpha} \leq e^{\beta^2/8}$.*

Lemma 4.3.2 *Let $\Omega_\delta \sqsubseteq \{-1,1\}^n$ be a δ-biased sample space and let $\mathcal{F}$ be a family of subsets of $[n]$. Then for any $\epsilon \geq \delta$, there is a sample space $\Omega_{(\epsilon,\mathcal{F})} \sqsubseteq \Omega_\delta$ of size $\min(|\Omega_\delta|, \frac{2\ln(3|\mathcal{F}|)}{(\epsilon-\delta)^2})$ that is ϵ-biased with respect to $\mathcal{F}$.*

Proof: Let $m = \frac{2\ln(3|\mathcal{F}|)}{(\epsilon-\delta)^2}$. Consider the process in which we select vectors $v_1, \ldots, v_m \in \Omega_\delta$ uniformly and independently at random. Clearly, the sample space Δ generated by this random process is the set of all sequences of m vectors in Ω_δ. We want to show that there exists an $\Omega_{(\epsilon,\mathcal{F})} \in \Delta$ that is ϵ-biased with respect to $\mathcal{F}$.

Let $(V_1, V_2, \ldots, V_m)$ be a randomly selected point from Δ, where each

$$V_i = V_{i1}V_{i2}\ldots V_{in} \in \{-1,1\}^n.$$

By the construction of Δ, it is clear that the random variables V_i are mutually independent. For a subset $S \subseteq [n]$ and each $1 \leq i \leq m$, we define a random variable $Y_i^S = Y_i^S(V_1, V_2, \ldots, V_m)$ mapping Δ to $\{-1,1\}$ as follows:

$$Y_i^S(V_1, V_2, \ldots, V_m) = \prod_{j \in S} V_{ij}.$$

It follows immediately from the definition that for any fixed $S \subseteq [n]$, each Y_i^S depends only on the i-th point V_i and therefore, the Y_i^S's are mutually independent as well. Moreover, we observe that:

Proposition 4.3.1 *For all $S \subseteq [n]$ and for each $1 \leq i \leq m$,*

1. *$Pr[Y_i^S = 1] = p_S$, where p_S is the fraction of 1's in the vector $\psi_S(M(\Omega_\delta))$, which clearly depends only on Ω_δ;*
2. *$E[Y_i^S] = 2p_S - 1$;*
3. *$|E[Y_i^S]|$ is equal to the bias of S with respect to Ω_δ, which is at most δ by assumption.*

Let $Y^S = \sum_{i=1}^m Y_i^S$. Now, for any $\omega \in \Delta$, $|Y^S(\omega)|/m$ is by definition the bias of S with respect to ω. Therefore, to show the existence of an $\Omega_{(\epsilon,\mathcal{F})}$ in Δ, it suffices to prove that for all $J \in \mathcal{F}$, $Pr[|Y^J| > \epsilon m] < 1/|\mathcal{F}|$.

Let us fix an arbitrary $J \in \mathcal{F}$. In order to estimate $Pr[|Y^J| > \epsilon m]$, we introduce another random variable $X^J = Y^J - E[Y^J]$. We notice that

$$|Y^J| > \epsilon m \text{ implies that } |X^J| > (\epsilon - \delta)m. \tag{4.1}$$

This is because

$$|X^J| = |Y^J - E[Y^J]| \geq |Y^J| - |E[Y^J]| \geq |Y^J| - \sum_{i=1}^m |E[Y_i^J]| > \epsilon m - \delta m,$$

where the last inequality uses the fact that $|E[Y_i^J]| \leq \delta$ for $1 \leq i \leq m$ which follows from Proposition 4.3.1 (3). Therefore,

$$Pr[|Y^J| > \epsilon m] \leq Pr[|X^J| > (\epsilon - \delta)m].$$

Now to complete the proof, it suffices to show that the right hand side of the inequality is strictly less than $1/|\mathcal{F}|$.

To accomplish this, we apply the standard Chernoff bound for large deviations. The argument we shall present uses the proof of a theorem in [AS91] (Corollary A.7). We show the details here for later reference.

For each $1 \leq i \leq m$, let us define a random variable

$$X_i^J = Y_i^J - E[Y_i^J].$$

Clearly, $X^J = \sum_{i=1}^m X_i^J$. As we have noticed, the Y_i^J's are mutually independent, so are the X_i^J's. Moreover, by Proposition 4.3.1 (1),

$$\begin{aligned} Pr[X_i^J = 1 - E[Y_i^J]] &= Pr[Y_i^J = 1] = p_J, \text{ and} \\ Pr[X_i^J = -1 - E[Y_i^J]] &= Pr[Y_i^J = -1] = 1 - p_J. \end{aligned}$$

Now, for any λ,

$$\begin{aligned} E[e^{\lambda X_i^J}] &= p_J e^{\lambda(1-E[Y_i^J])} + (1-p_J)e^{\lambda(-1-E[Y_i^J])} \\ &= p_J e^{2\lambda(1-p_J)} + (1-p_J)e^{-2\lambda p_J} \qquad (4.2) \\ &\leq e^{\lambda^2/2}, \end{aligned}$$

where the second equality follows from Proposition 4.3.1 (2) and the inequality follows Lemma 4.3.1 by setting $\alpha = p_J, \beta = 2\lambda$. Therefore,

$$E[e^{\lambda X^J}] = \prod_{i=1}^m E[e^{\lambda X_i^J}] \leq e^{m\lambda^2/2}. \qquad (4.3)$$

By applying Markov's inequality, we have

$$Pr[X^J > a] = Pr[e^{\lambda X^J} > e^{\lambda a}] < \frac{E[e^{\lambda X^J}]}{e^{\lambda a}} \leq e^{m\lambda^2/2 - \lambda a}.$$

The right hand side is minimized at $\lambda = a/m : Pr[X^J > a] < e^{-a^2/2m}$. By symmetry, $Pr[|X^J| > a] < 2e^{-a^2/2m}$. Substituting $(\epsilon - \delta)m$ for a and using our choice of m, we have

$$Pr[|X^J| > (\epsilon - \delta)m] < 1/|\mathcal{F}|.$$

This completes the proof. □

In particular, we have $|\Omega_{(\epsilon,\mathcal{F})}| = O(\frac{\log n}{\epsilon^2})$ if $\mathcal{F}$ has size polynomial in n and if $\epsilon - \delta = \Theta(\epsilon)$.

4.3.2 The Explicit Construction

Let $\mathcal{F}$ be a family of subsets of $[n]$ of size polynomial in n and let $0 < \epsilon \leq 1$ be polynomially small, i.e., $\epsilon = n^{-O(1)}$. As we have shown in Lemma 4.3.2, in every $\frac{\epsilon}{2}$-biased sample space there exists a sample space of size $O(\log|\mathcal{F}|/\epsilon^2)$ that is ϵ-biased with respect to $\mathcal{F}$. This existence proof in effect gives us an efficient randomized algorithm for constructing a desired sample space. In this subsection, we present an efficient deterministic algorithm to build such a sample space by derandomizing the above randomized procedure via the method of conditional probabilities.

Let $\delta = \epsilon/2$. By Theorem 4.2.1, there is an efficient construction of a δ-biased sample space $\Omega_\delta \sqsubseteq \{-1,1\}^n$ of size $O(n^2/\epsilon^2)$.

Let $m = \frac{2\ln(3|\mathcal{F}|)}{(\epsilon-\delta)^2}$. We define Δ to be the set of all sequences of m vectors in Ω_δ, i.e., $\Delta = (\Omega_\delta)^n$. By Lemma 4.3.2, we know that there exists a "good " point in Δ in the sense that its corresponding sequence of m vectors in Ω_δ is ϵ-biased with respect to $\mathcal{F}$. What we want is to deterministically find such a good point in Δ.

First of all, we remind the reader that for a random point $(V_1, V_2, \ldots, V_m) \in \Delta$, the random variables V_i are mutually independent. This fact will help us to apply the method of conditional probabilities to our algorithmic construction. To describe the algorithm we need some preliminaries. For notational convenience, we will use some of the notation from the proof of Lemma 4.3.2.

For each $J \in \mathcal{F}$, we define random variables X^J, Y^J on Δ in the same way as that in the proof of Lemma 4.3.2, and we define

$$Z^J = \begin{cases} 1 & \text{if } |Y^J| > \epsilon m \\ 0 & \text{otherwise.} \end{cases}$$

Let $Z = \sum_{J\in\mathcal{F}} Z^J$. Since a sample point $\omega \in \Delta$ is ϵ-biased with respect to $\mathcal{F}$ if and only if for each $J \in \mathcal{F}$ the corresponding $|Y^J(\omega)| \leq \epsilon m$, it suffices for us to construct an ω that satisfies $Z(\omega) < 1$. In fact, the existence proof of Lemma 4.3.2 exactly says that such a point exists since for all $J \in \mathcal{F}$, $E[Z^J] = Pr[|Y^J| > \epsilon m] < 1/|\mathcal{F}|$, and thus $E[Z] < 1$. We want to apply the method of conditional probabilities to Z and construct such a desired point in Δ.

Unfortunately, there seems to be no easy way to directly compute the conditional expectation of Z. To overcome this difficulty, we adopt the idea of pessimistic estimators [Rag88, Fun94] and introduce another random variable W^J to estimate each Z^J, where W^J is defined as

$$W^J = e^{-(\epsilon-\delta)^2 m}(e^{(\epsilon-\delta)X^J} + e^{-(\epsilon-\delta)X^J}).$$

We let $W = \sum_{J\in\mathcal{F}} W^J$.

What we would like to see is that the random variable W satisfies the conditions (A), (B) and (C) for pessimistic estimators in Section 4.2.3 with respect to the random variable Z, so that we can apply the method of conditional probabilities to W and construct a desired sample point. Let us first show the following:

Proposition 4.3.2 *For each $J \in \mathcal{F}$,*

(a) *$Z^J(\omega) \leq W^J(\omega)$ for all $\omega \in \Delta$;*

(b) *$E[W^J] < 1/|\mathcal{F}|$; and*

(c) *for each $1 \leq i \leq n$ and for all $v_1, \ldots, v_i \in \Omega_\delta$, $E[W^J|v_1, \ldots, v_i]$ is easy to compute.*

Proof: Since W^J is always nonnegative, to prove (a) it suffices to show that for any $\omega \in \Delta$, $Z^J(\omega) = 1$ would imply $W^J(\omega) \geq 1$. We know that $Z^J(\omega) = 1$ if and only if $|Y^J(\omega)| > \epsilon m$. By (4.1) in the proof of Lemma 4.3.2, now it is enough to show that if $|X^J(\omega)| > (\epsilon - \delta)m$ then $W^J(\omega) \geq 1$. This is easy to verify.

By substituting $(\epsilon - \delta)$ for λ in (4.3), (b) follows from our choice of m.

For (c) it suffices to show that $E[e^{(\epsilon-\delta)X^J}|v_1, \ldots, v_i]$ is easy to compute. Since by definition $X^J = \sum_{k=1}^m X_k^J$ and the X_k^J's are mutually independent, we know that

$$E[e^{(\epsilon-\delta)X^J}|v_1, \ldots, v_i] = \Pi_{k=1}^m E[e^{(\epsilon-\delta)X_k^J}|v_1, \ldots, v_i].$$

It is therefore enough to show that each term in the product is easy to compute. We first notice that p_J is easy to compute given Ω_δ: p_J is just the proportion of 1's in the vector resulting from the coordinate-wise product of the columns of $M(\Omega_\delta)$ with indices in J. Since each $X_k^J = Y_k^J - E[Y_k^J]$ depends only on the k-th point v_k by definition, we

know then for each $1 \leq k \leq i$,

$$E[e^{(\epsilon-\delta)X_k^J}|v_1,\ldots,v_i] = e^{(\epsilon-\delta)(Y_k^J(v_k)-(2p_J-1))}$$

which is easy to compute; and for each $i+1 \leq k \leq m$,

$$E[e^{(\epsilon-\delta)X_k^J}|v_1,\ldots,v_i] = E[e^{(\epsilon-\delta)X_k^J}]$$

which by (4.2) is easy to compute as well. □

Now that we have the proposition, conditions (a) and (b) clearly give conditions (A) and (B), respectively. That condition (c) implies condition (C) is because $|\mathcal{F}| = n^{O(1)}$ and therefore

$$E[W|v_1,\ldots,v_i] = \sum_{J\in\mathcal{F}} E[W^J|v_1,\ldots,v_i]$$

is easy to compute as well.

Now we are ready to describe our algorithm: Knowing Ω_δ, we first compute p_J for each $J \in \mathcal{F}$. The next part of the algorithm runs in m steps. At the beginning, $\Omega_{(\epsilon,\mathcal{F})}$ is empty. We add one vector of Ω_δ to $\Omega_{(\epsilon,\mathcal{F})}$ in each step. In the i-th step, $1 \leq i \leq m$, for each $v \in \Omega_\delta$ we compute $E[W|v_1,\ldots,v_{i-1},v]$. Choose v_i to be the v such that the conditional expectation is less than 1.

Proposition 4.3.2 guarantees that the construction can proceed at every step and the final sequence $v_1, v_2, \ldots, v_m$ is ϵ-biased with respect to $\mathcal{F}$.

In fact, it is not difficult to see that the running time of the algorithm is polynomial in n, ϵ^{-1} and $|\mathcal{F}|$ for general n, ϵ and $\mathcal{F}$. So we have shown:

Theorem 4.3.1 *Let $\mathcal{F}$ be a family of subsets of $[n]$ and let $0 < \epsilon \leq 1$, There exists an algorithm that constructs a sample space $\Omega_{(\epsilon,\mathcal{F})} \sqsubseteq \{-1,1\}^n$ (thus $\{0,1\}^n$) of size $O(\frac{\log|\mathcal{F}|}{\epsilon^2})$ that is ϵ-biased with respect to $\mathcal{F}$. The running time of the algorithm is polynomial in n, ϵ^{-1} and $|\mathcal{F}|$.*

In particular, in the case that $\mathcal{F}$ has size polynomial in n and ϵ is polynomially small, the algorithm constructs $\Omega_{(\epsilon,\mathcal{F})}$ of size $O(\frac{\log n}{\epsilon^2})$ in polynomial time.

The theorem clearly leads to the best efficient construction of k-wise ϵ-biased sample spaces of size $O(\log n/\epsilon^2)$ for fixed k and polynomially small ϵ.

4.4 Applications

4.4.1 Error-Correcting Communication Protocols

In the standard model of the communication complexity study [Yao79], there are two parties A and B with unlimited computation power. Given a private input to each party, A and B communicate with each other following a specified protocol to compute a function of both inputs. The goal is find a protocol that performs the computation with minimum amount of communication. (Without loss of generality, here we assume that it suffices for either party to know the answer.)

One of the classical problems in this study is the EQUALITY problem, in which each of A, B gets an n-bit string x and y, respectively, and the function that A and B compute is $EQ : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$ such that $EQ(x,y) = 1$ if and only if $x = y$. It has been shown [Yao79] that for this problem, the two parties in general cannot do anything better than trivially sending the *whole* input from one side to the other.

In this section, we consider a communication problem analogous to EQUALITY in which the two parties want to recognize certain difference patterns between the input strings.

Notation and Definitions

Throughout the rest of this section, the "+" operation is assumed to be modulo 2.

Let x and y be two n-bit strings. The *difference* between x and y is defined to be the set of positions where x and y differ; and the *difference pattern* of x and y is defined to be the characteristic vector of the difference between them, which is equal to the n-bit vector $x + y$. If $\mathcal{F}$ is a family of subsets of $[n]$, then we use $x \neq_{\mathcal{F}} y$ to denote that x is not equal to y and moreover, the difference between x and y is a member of $\mathcal{F}$.

The Communication Problem and the Approach

In our problem, the two communicating parties A and B have limited computation power: they are polynomial time machines. We suppose that a family $\mathcal{F}$ of subsets of

$[n]$, which we call the set of *tolerable differences*, is known to both parties beforehand. On input an n-bit string x to A and an n-bit string y to B, A and B communicate with each other and try to recognize the difference pattern between x and y with respect to $\mathcal{F}$ in the following sense: A and B compute the function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ such that

$$f(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq_{\mathcal{F}} y \\ 0 \text{ or } 1 & \text{otherwise.} \end{cases}$$

In words, A and B definitely output 1 if x is equal to y, definitely output 0 if x, y are not equal but the difference between them is a member of $\mathcal{F}$, and may output either 0 or 1 (we don't care) in other cases. We call the problem at this stage *error-detecting*. Moreover, we require that in the case that $x \neq_{\mathcal{F}} y$, at least one of A, B should know the *exact* difference between x and y. We call the problem at this stage *error-correcting*. Of course, error-detecting is no harder than error-correcting.

We can see that if $\mathcal{F}$ consists of $\Lambda([n])$, then the error-detecting case of our problem is reduced to EQUALITY. Therefore, if we allow $\mathcal{F}$ to be arbitrary then generally A and B can do nothing better than communicating all the input bits from one side to the other. We will be interested in the case where the size of $\mathcal{F}$ is polynomial in n. For such a problem, we shall present a deterministic protocol using $O(\log|\mathcal{F}|)$ bits of communication.

It is worth noticing that, as a matter of fact, this bound given by our protocol is essentially tight in many cases. Let us look at the following two examples just for the case of error-detecting. (For this discussion, we remind the reader a well known fact in communication complexity that for a communication problem, the number of bits of communication needed in any deterministic protocol is at least the logarithm of the rank of the communication matrix associated with the problem. We will not provide all the proof details here since they are out of the scope of this thesis. We refer the reader to the book by Kuselevitz and Nisan [KN95] on communication complexity.)

Example 1: We consider the case where $\mathcal{F}$ includes $\Lambda(\binom{[n]}{2})$. Then in the communication matrix associated with the problem, the submatrix indexed by the set of n-bit strings with a single 1-bit has full rank n. Therefore, the number of bits of communication needed by any deterministic protocol in this case is at least $\log n$.

Example 2: For a subset $S \subseteq [n]$ of size $\Theta(\log n)$, we suppose that $\mathcal{F}$ includes $\Lambda(S)$. Then in the communication matrix associated with the problem, the submatrix indexed by the set of all n-bit strings that correspond to the characteristic vectors of the sets in the ideal generated by S, i.e., in $\Lambda(S) \cup \{\phi\}$, has full rank $2^{|S|}$. Therefore, any deterministic protocol in this case needs at least $|S|$ which is $\Theta(\log n)$ bits of communication as well.

On the other hand, since in both of these examples $|\mathcal{F}| = n^{O(1)}$ by assumption, our protocol thus uses $O(\log(|\mathcal{F}|) = O(\log n)$ bits which matches the communication lower bound to within a constant factor.

The design of our protocol exploits the explicit construction of sample spaces with small bias on neighborhoods.

Applying explicit constructions of sample spaces with small bias to the design of communication protocols was first introduced by Naor and Naor in [NN90]. In their paper, they presented a randomized protocol for the EQUALITY problem that achieves polynomially small error using $O(\log n)$ bits. A fundamental idea in their approach is based on the next observation which follows easily from Proposition 4.2.1:

Proposition 4.4.1 *Let $\mathcal{F}$ be a family of subsets of $[n]$. Suppose $\Omega \sqsubseteq \{0,1\}^n$ is ϵ-biased with respect to $\mathcal{F}$, where $0 \leq \epsilon < 1$. If x and y are two n-bit strings whose difference is a member in $\mathcal{F}$, then $M(\Omega)(x+y) \neq 0$ and moreover, the fraction of 1's in $M(\Omega)(x+y)$ is at least $\frac{1-\epsilon}{2}$.*

Let x, y and Ω be as in the proposition. We suppose that both A and B know Ω beforehand and they want to compute $EQ(x,y)$. Then, in a randomized protocol, A could first pick a row ω from $M(\Omega)$ at random and then send the index of ω together with the bit ωx to B; in turn, B could compute ωy and output 1 if and only if $\omega y = \omega x$.

By Proposition 4.4.1, we know that with probability at least $\frac{1-\epsilon}{2}$, B will give the right answer. Applying the standard amplification techniques, they can then achieve high success probability. For a deterministic protocol, on the other hand, A could send $M(\Omega)x$ to B and B would output 1 if and only if $M(\Omega)x = M(\Omega)y$. (Here we emphasize the fact that in this approach, the amount of communication in the protocol depends completely on the size of the sample space Ω: the logarithm of the size is the length of the index sent in the randomized protocol, while the size itself is the number of bits sent in the deterministic protocol.)

Using this idea, let us first consider the error-detecting case of our problem. The work in [NN90] provides an explicit construction of $(\Upsilon(\mathcal{F}))$-wise $\frac{1}{2}$-biased sample spaces of size $\Theta(\Upsilon(\mathcal{F})\log n)$, which in our case is polylogarithmic in n. Such a sample space is by definition $\frac{1}{2}$-biased with respect to $\mathcal{F}$. Therefore, the randomized protocol described above works for our problem and uses $O(\log\log n)$ bits of communication. The deterministic protocol however uses $\Theta(\Upsilon(\mathcal{F})\log n)$ bits. In particular, in the case that $\Upsilon(\mathcal{F})$ is $\omega(1)$, this deterministic protocol would use $\omega(\log n)$ bits. On the other hand, if in this case we allow both parties to have unlimited computation power, then the counting argument of Lemma 4.3.2 in effect shows that there exists a deterministic protocol using $O(\log|\mathcal{F}|) = O(\log n)$ bits of communication.

In what follows, using our new sample space construction, we present a deterministic error-correcting protocol using $O(\log|\mathcal{F}|)$ bits for the case that both parties have limited computation power.

The Protocol

For a family $\mathcal{F}$ of sets, we define $\mathcal{F}^* = \mathcal{F} \cup \{X \triangle Y \mid X, Y \in \mathcal{F}\}$, where $X \triangle Y$ is the symmetric difference of X and Y. Clearly, $|\mathcal{F}^*| = O(|\mathcal{F}|^2)$. If S is a subset of $[n]$, we use χ_S to denote the (column) characteristic vector of S.

Lemma 4.4.1 *Let $\mathcal{F}$ be family of subsets of $[n]$ and let $0 \leq \epsilon < 1$. Suppose $\Omega_{(\epsilon,\mathcal{F}^*)} \sqsubseteq \{0,1\}^n$ is ϵ-biased with respect to $\mathcal{F}^*$. Then for all distinct $S, T \in \mathcal{F}$, $M(\Omega_{(\epsilon,\mathcal{F}^*)})\chi_S \neq M(\Omega_{(\epsilon,\mathcal{F}^*)})\chi_T$.*

Proof: We want to show that for all distinct $S, T \in \mathcal{F}$, $M(\Omega_{(\epsilon,\mathcal{F}^*)})(\chi_S + \chi_T) = M(\Omega_{(\epsilon,\mathcal{F}^*)})\chi_{S\triangle T} \neq 0$. Since $S \neq T$, we know $\chi_{S\triangle T} \neq 0$; and since $S, T \in \mathcal{F}$, $S\triangle T \in \mathcal{F}^*$. Now the lemma follows from Proposition 4.4.1. □

Let $\mathcal{F}$ be a family of subsets of $[n]$ of size polynomial in n. Then it is clear that $\mathcal{F}^*$ is also a family of subsets of $[n]$ of size polynomial in n. Therefore, Theorem 4.3.1 guarantees the efficient construction of $\Omega_{(1/2,\mathcal{F}^*)}$ of size $O(\log|\mathcal{F}^*|) = O(\log|\mathcal{F}|)$, which is $O(\log n)$ in our case. So we may assume without loss of generality that both party A and B keep the same matrix $M(\Omega_{(1/2,\mathcal{F}^*)})$ of dimension $O(\log n) \times n$. At the same time, we assume that both A and B keep a table $\mathcal{T}$ containing all the pairs $< M(\Omega_{(1/2,\mathcal{F}^*)})\chi_S, S >$ for $S \in \mathcal{F}$. (Note that $\mathcal{T}$ is constructable in time polynomial in n as well.) The mutual distinctness of these pairs is guaranteed by Lemma 4.4.1.

The protocol is as follows:

1. A computes $M(\Omega_{(1/2,\mathcal{F}^*)})x$ and sends the resulting vector to B;

2. B computes $M(\Omega_{(1/2,\mathcal{F}^*)})y$.

   If $M(\Omega_{(1/2,\mathcal{F}^*)})y = M(\Omega_{(1/2,\mathcal{F}^*)})x$, then he outputs 1. Otherwise, he outputs 0 and looks up the table $\mathcal{T}$ for the entry $< M(\Omega_{(1/2,\mathcal{F}^*)})(x + y), S >$ and takes S to be the difference between x and y.

The protocol clearly uses $O(\log n)$ bits of communication. It is obvious that the parties definitely output 1 in the case x is equal to y. In the case that $x \neq_{\mathcal{F}} y$, we know that $x + y$ is exactly the difference pattern of x and y, and thus $x + y = \chi_S$ for some $S \in \mathcal{F}$ by the definition of $\neq_{\mathcal{F}}$. Now the correctness of the protocol for this case follows easily from Lemma 4.4.1.

4.4.2 Constructing $(n, \mathcal{F})$-universal Sets

For a given family $\mathcal{F}$ of subsets of $[n]$, a set $T \sqsubseteq \{0,1\}^n$ is said to be $(n, \mathcal{F})$-*universal* if for each $S \in \mathcal{F}$, the projection of T on the indices in S contains $\{0,1\}^{|S|}$. In this subsection, using our construction of sample spaces with small bias on neighborhoods, we show how to construct $(n, \mathcal{F})$-universal sets and prove the following:

Theorem 4.4.1 *Let $\mathcal{F}$ be a family of subsets of $[n]$. Then there is an explicit construction of an $(n, \mathcal{F})$-universal set of size $O(2^{\Upsilon(\mathcal{F})} \log |\Lambda(\mathcal{F})|)$. The construction uses time polynomial in n and $|\Lambda(\mathcal{F})|$.*

To prove the theorem we need the following fact which was shown in [NN90] (whose proof in turn is based on a lemma in [Vaz86]):

Proposition 4.4.2 *If $\Omega \sqsubseteq \{0,1\}^k$ is 2^{-k-1}-biased, then $\{0,1\}^k \subseteq \Omega$.*

Corollary 4.4.1 *If $\Omega \sqsubseteq \{0,1\}^n$ is $2^{-\Upsilon(\mathcal{F})-1}$-biased with respect to $\Lambda(\mathcal{F})$, then Ω is $(n, \mathcal{F})$-universal.*

Now the corollary together with Theorem 4.3.1 clearly give us Theorem 4.4.1.

Constructing $(n, \mathcal{F})$-universal sets is useful for exhaustively testing combinatorial logic circuits in which each functional component depends only on a subset of inputs (see [SB88]). Schulman in [Sch92] gave an explicit construction of $(n, \mathcal{F})$-universal sets of size $O(d2^{\Upsilon(\mathcal{F})})$, where d denotes the maximum number of neighborhoods containing any particular element. The size of our construction is smaller than that of [Sch92] when $d > c \log |\Lambda(\mathcal{F})|$ for some constant c.

Theorem 4.4.1 in particular gives a polynomial time construction of an $(n, \binom{[n]}{k})$-universal set of size $O(2^k k \log n)$ for fixed k, which is optimal following the lower bound argument in [SB88]. Alon [Alo86] showed an explicit construction of $(n, \binom{[n]}{k})$-universal sets of size $\log n \cdot 2^{O(k^2)}$ for the case k is fixed. In [NN90] and [ABNNR92], nearly optimal constructions of size $\log n \cdot 2^{O(k)}$ were given. The previously best known construction in this case is due to [NSS95] and has size $2^k k^{O(\log k)} \log n$. Our construction has the least size and at the same time matches the bound given by the probabilistic argument.

References

[AH87] L. Adelman and M. Huang. Recognizing primes in random polynomial time. In *Proc. of 19th ACM Symposium on Theory of Computing*, 1987.

[AKS87] M. Ajtai, J. Komlós and E. Szemerédi. Deterministic Simulation of Logspace. In *Proc. of 19th ACM Symposium on Theory of Computing*, pp. 132-140, 1987.

[AKLLR79] R. Aleliunas, R. Karp, R. Lipton, L. Lovasz and C. Rackoff. Random walks, universal sequences and the complexity of maze problems. In *Proc. of 20th IEEE Symposium on Foundations of Computer Science*, pp. 218-223, 1979.

[AO94] E. Allender and M. Ogihara. Relationships among **PL**, #**L**, and the determininant. *Proc. of 9th Conference on Structure in Complexity Theory*, pp. 267-279, 1994.

[Alo86] N. Alon. Explicit constructions of exponential sized families of k-independent sets. *Discrete Math*, 58:191-193, 1986.

[Alo91] N. Alon. A parallel algorithm version of the Local Lemma. *Random Structures and Algorithms*, 2(4):367-378, 1991.

[ABI86] N. Alon, L. Babai and A. Itai. A fast and simple randomized parallel algorithm for the Maximal Independent Set Problem. *J. Algorithms* 7:567-583, 1986.

[ABNNR92] N. Alon, J. Bruck, J. Naor, M. Naor and R. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Trans. on Info. Theory*, 38(2):509-515, 1992.

[AGHP91] N. Alon, O, Goldreich, J. Hastad and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms* 3(3):289-303, 1992.

[AS91] N. Alon and J. Spencer. *The Probabilistic Method.* Wiley, 1991.

[ASWZ96] R. Armoni, M. Saks, A. Wigderson and S. Zhou. Discrepancy sets and pseudorandom generators for combinatorial rectangles. *Manuscript*, 1996.

[AW95] R. Armoni and A. Wigderson. Pseudorandomness for space bounded computations. *Manuscript*, 1995.

[BNS89] L. Babai, N. Nisan and M. Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences. In *Proc. of 21st ACM Symposium on Theory of Computing*, 1989.

[BR89] B. Berger and J. Rompel. Simulating ($\log^c n$)-wise independence in NC. *J. Assoc. Comput. Mach.*, 38(4):1026-1046, 1991.

[Blu86] M. Blum. Independent Unbiased Coin Flips from a Correlated Biased Source: a Finite Markov Chain. *Combinatorica*, 6(2):97-108, 1986.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *J. SIAM*, 13(4):850-864, 1984.

[BCP83] A. Borodin, S. Cook and N. Pippenger. Parallel computation and well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58:113-136, 1983.

[CW79] L. Carter and M. Wegman. Universal hash functions. *J. Comp. and Syst. Sci.*, 18(2):143-154, 1979.

[CFL83] A. Chandra, M. Furst and R. Lipton. Multi-party protocols. In *Proc. of 15th ACM Symposium on Theory of Computing*, pp. 94-99, 1983.

[CG88] B. Chor and O. Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM J. Comput.*, 17(2):230-261, 1988.

[CG86] B. Chor and O. Goldreich. On the power of two point based sampling. *Journal of Complexity*, 5:96-106, 1989.

[CGHFRS85] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich and R. Smolensky. The bit extraction problem or t-resilient functions. In *Proc. of 26th IEEE Symposium on Foundations of Computer Science*, pp. 396-407, 1985.

[CRS94] S. Chari, P. Rohatgi and A. Srinivasan. Improved algorithms via approximations of probability distributions. In *Proc. of 26th ACM Symposium on Theory of Computing*, pp. 584-592, 1994.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2-22, 1985.

[CW89] A. Cohen and A. Wigderson. Dispersers, Deterministic Amplification, and Weak Random Sources. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pp. 14-19, 1989.

[EGLNV92] G. Even, O. Goldreich, M. Luby, N. Nisan and B. Velickovic. Approximations of general independent distributions. In *Proc. of 24th ACM Symposium on Theory of Computing*, pp. 10-16, 1992.

[ES73] P. Erdös and J. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory*, Series A 14:298-301, 1973.

[FLW92] A. M. Ferrenberg, D. P. Landau and Y. J. Wong. Monte Carlo simulations: Hidden errors from "good" random number generators. *Physical Review Letters*, 69(23):3382-3384, 1992.

[FN93] A. Fiat and M. Naor. Implicit $O(1)$ probe search. *SIAM J. Comput.*, 22:1-10, 1993.

[Fun94] A. Fundia. Derandomizing Chebyshev's inequality to find independent sets in uncrowded hypergraghs. *Manuscript*, 1994.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM J. Computing*, 6:675-695, 1977.

[GG81] O. Gabber and Z. Galil. Explicit construction of linear-sized superconcentrators. *J. Comp. and Syst. Sci.*, 22:407-420, 1981.

[HRD94] T.-s. Hsu, V. Ramachandran and N. Dean. Parallel implementation of algorithms for finding connected components. In *Proc. DIMACS International Algorithm Implementation Challenge*, pp. 1-14, 1994.

[Hsu93] T.-s. Hsu. *Graph augmentation and related problems: theory and practice.* PhD thesis, Department of Computer Sciences, University of Texas at Austin, October 1993.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal of Computing*, 17:935-938, 1988.

[IZ89] R. Impagliazzo and D. Zuckerman. How to Recycle Random Bits. In *Proc. of 30th Symposium on Foundations of Computer Science*, pp. 248-253, 1989.

[ILL89] R. Impagliazzo, L. Levin and M. Luby. Pseudo-Random Generation from One-Way Functions. In *Proc. of ACM Symposium on Theory of Computing*, pp. 12-24, 1989.

[INW94] R. Impagliazzo, N. Nisan and A. Wigderson. Pseudorandomness for network algorithms. In *Proc. of ACM Symposium on Theory of Computing*, 1994.

[Jun81] H. Jung. Relationships between probabilistic and deterministic tape complexity. In *10th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, 118:339-346, Springer-Verlag.

[Kar93] D. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proc. of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 21-30, 1993.

[KK94] D. Karger and D. Koller. (De)randomized construction of small sample spaces in NC. In *Proc. of 35th IEEE Symposium on Foundations of Computer Science*, pp. 252-263, 1994.

[KM94] H. Karloff and Y. Mansour. On construction of k-wise independent random variables. In *Proc. of the 26th Annual ACM Symposium on Theory of Computing*, 1994.

[KUW86] R. Karp, E. Upfal and A. Wigderson. Constructing a maximum matching is in random NC. *Combinatorica*, 6(1):35-48, 1986.

[KW84] R. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. In *Proc. of the 16th Annual ACM Symposium on Theory of Computing*, 1984.

[KM93] D. Koller and N. Megiddo. Finding small sample spaces satisfying given constraints. In *Proc. of the 25th Annual ACM Symposium on Theory of Computing*, pp. 268-277, 1993.

[KN95] E. Kushilevitz and N. Nisan. *Communication Complexity.*

[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.

[LLSZ95] N. Linial, M. Luby, M. Saks and D. Zuckerman. Efficient construction of a small hitting set for combinatorial rectangles in high dimension. In *Proc. of 25th ACM Symposium on Theory of Computing*, pp. 258-267, 1993.

[LPS86] A. Lubotzky, R. Phillips and P. Sarnak. Explicit expanders and the Ramanujan conjectures. In *Proc. of 18th ACM Symposium on Theory of Computing*, pp. 240-246, 1986.

[Lub85] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036-1053, 1986.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[MNN89] R. Motwani, J. Naor and M. Naor. The probabilistic method yields deterministic parallel algorithms. In *Proc. of 30th IEEE Symposium on Foundations of Computer Science*, pp. 8-13, 1989.

[MVV87] K. Mulmuley, U. Vazirani and V. Vazirani. Matching is as easy as matrix inversion. In *Proc. of 19th ACM Symposium on Theory of Computing*, 1987.

[NN90] J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. *SIAM J. Comput.*, 22(4):838-856, 1993.

[NSS95] M. Naor, L. Schulman and A. Srinivasan. Splitters and near-optimal derandomization. In *Proc. of 36th IEEE Symposium on Foundations of Computer Science*, pp 182-191, 1995.

[Neu63] J. von Neumann. Various techniques for use in connection with random digits. In *von Neumann's Collected works*, pp 768-770, Pergaman, New York, 1963.

[Nis90] N. Nisan. Pseudorandom generators for space-bounded computation. In *Proc. of 22nd ACM Symposium on Theory of Computing*, pp. 204-212, 1990.

[Nis91] N. Nisan. *Using hard problems to create pseudorandom generators.* The MIT Press, 1991. (An ACM Distinguished Dissertation)

[Nis92] N. Nisan. $RL \subseteq SC$. In *Proc. of ACM Symposium on Theory of Computing*, pp. 619-623, 1992.

[Nis93] N. Nisan. On read-once vs. multiple access to randomness in Logspace. *Theoretical Computer Science*, 107:135-144, 1993.

[Nis96] N. Nisan. Extracting Randomness: How and Why. In *Proc. of 11th Annual Conference on Computational Complexity*, pp. 44-58, 1996.

[NSW92] N. Nisan, E. Szemerédi and A. Wigderson. Undirected Connectivity in $O(\log^{1.5} n)$ Space. In *Proc. of 30th Symposium on Foundations of Computer Science*, pp. 248-253, 1992.

[NT95] N. Nisan and A. Ta-Shma. Symmetric logspace is closed under complement. In *Proc. of 27th ACM Symposium on Theory of Computing*, pp. 140-146, 1995.

[NW88] N. Nisan and A. Wigderson. Hardness vs. randomness. In *Proc. of 29th IEEE Symposium on Foundations of Computer Science*, pp. 2-12, 1988.

[NZ93] N. Nisan and D. Zuckerman. More Deterministic Simulation in Logspace. In *Proc. of ACM Symposium on Theory of Computing*, pp. 235–244, 1993.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Rab80] M. Rabin. Probabilistic algorithm for testing primality. *J. of Number Theory*, 12:128-138, 1980.

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *J. Comp. and Syst. Sci.*, 37:130-143, 1988.

[Sak94] M. Saks. *Lecture notes on Random Structures and Algorithms* (manuscript). Applied Math 587, Rutgers University, 1994.

[Sak96] M. Saks. Randomization and Derandomization in Space-Bounded Computation. In *Proc. of 11th Annual Conference on Computational Complexity*, 1996.

[SSZ95] M. Saks, A. Srinivasan and S. Zhou. Explicit dispersers with polylog degree. In *Proc. of 27th ACM Symposium on Theory of Computing*, pp. 479-488, 1995.

[SZ95] M. Saks and S. Zhou. $RSPACE(S) \subseteq DSPACE(S^{3/2})$. In *Proc. of 36th IEEE Symposium on Foundations of Computer Science*, pp. 344-353, 1995.

[SZ96] M. Saks and S. Zhou. Sample spaces with small bias on neighborhoods and error-correcting communication protocols. *Manuscript*, 1996.

[San87] M. Santha. On Using Deterministic Functions in Probabilistic Algorithms. *Information and Computation*, 74(3):241-249, 1987.

[SV86] M. Santha and U. Vazirani. Generating Quasi-Random Sequences from Slightly Random Sources. *Journal of Computer and System Sciences*, 33:75–87, 1986.

[Sav70] W.J. Savitch. Relationships between nondeterministic and deterministic space complexities. *J. Comp. and Syst. Sci.*, 4(2):177-192, 1970.

[Sch92] L. Schulman. Sample spaces uniform on neighborhoods. In *Proc. of 24th ACM Symposium on Theory of Computing*, pp. 17-25, 1992.

[SB88] G. Seroussi and N. Bshouty. Vector sets for exhaustive testing of logic circuits. *IEEE Trans. on Info. Theory*, 34(3):513-522, 1988.

[Sip88] M. Sipser. Expanders, Randomness, or Time versus Space. *Journal of Computer and System Sciences*, 36:379-383, 1988.

[SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM J. on Computing*, 6:84-85, 1977.

[Spe87] J. Spencer. *Ten Lectures on the Probabilistic Method*, SIAM(Philadelphia), 1987.

[SZ94] A. Srinivasan and D. Zuckerman. Computing with Very Weak Random Sources. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pp. 264-275, 1994. Full version available as Technical Report TRA4/96, Department of Information Systems and Computer Science, National University of Singapore, April 1996.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*. 26:279-284, 1988.

[TaS96] A. Ta-Shma. On extracting randomness from weak random sources. In *Proc. of ACM Symposium on Theory of Computing*, 1996.

[Vaz86] U. Vazirani. *Randomness, Adversaries and Computation*. Ph.D. Thesis, University of California, Berkeley, 1986.

[Vaz87a] U. Vazirani. Efficiency Considerations in Using Semi-Random Sources. In *Proc. of ACM Symposium on Theory of Computing*, pp. 160-168, 1987.

[Vaz87b] U. Vazirani. Strong Communication Complexity or Generating Quasi-Random Sequences from Two Communicating Semi-Random Sources. *Combinatorica*, 7(4):375-392, 1987.

[VV85] U. Vazirani and V. Vazirani. Random Polynomial Time is Equal to Slightly-Random Polynomial Time. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pp. 417-428, 1985.

[Wig92] A. Wigderson. The complexity of graph connectivity. *Mathematical Foundations of Computer Science: Proceedings, 17th Symposium, Lecture Notes in Computer Science* 629:112-132, 1992.

[WZ93] A. Wigderson and D. Zuckerman. Expanders that Beat the Eigenvalue Bound: Explicit Construction and Applications. In *Proc. of ACM Symposium on Theory of Computing*, pp. 245-251, 1993.

[Yao79] A. C-C Yao. Some complexity questions related to distributive computing. In *Proc. of 11th ACM Symposium on Theory of Computing*, pp 209-213, 1979.

[Yao82] A. C-C Yao. Theory and applications of trapdoor functions. In *Proc. of 23rd IEEE Symposium on Foundations of Computer Science*, pp. 80-91, 1982.

[Zuc90] D. Zuckerman. General Weak Random Sources. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pp. 534-543, 1990.

[Zuc91] D. Zuckerman. Simulating BPP Using a General Weak Random Source. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pp. 79-89, 1991. Final version to appear in *Algorithmica*.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *Proc. of IEEE Conference on Structure in Complexity Theory*, pp. 305-312, 1993.

[Zuc96] D. Zuckerman. Randomness-optimal sampling, extractors and constructive leader election. In *Proc. ACM Symposium on Theory of Computing*, 1996.

Vita

Shiyu Zhou

1985	Graduated from the First Associate High School of Beijing Normal University, Beijing, China.
1985-90	Attended Tsinghua University, Beijing, China.
1990	B.E. in Computer Science and Technology, Tsinghua University.
1990-91	Assistant Software Engineer, Beifang Computer Corporation, Beijing, China.
1991-96	Graduate study in Computer Science, Rutgers, The State University of New Jersey, New Brunswick, New Jersey.
1994	M.S. in Computer Science, Rutgers, The State University of New Jersey.
1996	Ph.D. in Computer Science, Rutgers, The State University of New Jersey.