On the Random Resources Needed by Space Bounded Computational Models

Thesis for the degree of Doctor of Philosophy in Computer Science

by Roy Armoni

Submitted to the senate of the Hebrew University in 1998

This work was supervised by
Prof. Avi Wigderson
and
Prof. Michael Benor

Abstract

In this work we investigate the need for randomness in three different space bounded computational models:

For $BPSPACE(S)$ machines that use R random bits we construct a pseudo-random generator that expands a seed of length $O(\frac{(S+4^{\log^* R}\log R)\log R}{\max\{1,\log S-\log\log R\}})$. For most of the range of the parameters S and R, this seed is shorter than that of Nisan [Nis92] and of Impagliazzo, Nisan and Wigderson[INW94], and almost bridging the gap to [NZ93]. Then, we derandomize these machines in space $O(\frac{(S+e(R)\log R)\sqrt{\log R}}{\sqrt{\max\{1,\log S-\log\log R\}}})$ which again, for most of the range of the parameters is better than the previous derandomization of Saks and Zhou [SZ95].

Next, we investigate the deterministic space complexity of SL. Its randomized space complexity is known to be logarithmic. We show that SL is contained in $L^{4/3}$, improving on a previous result of Nisan, Szemerédi and Wigderson [NSW92].

Last, we look at the problem of constructing discrepancy sets for high dimension combinatorial rectangles. For the class $\mathcal{R}(m,d)$ of combinatorial rectangles of dimension d and side m we show how to construct an ϵ-discrepancy set of size polynomial in m, d and $(\frac{1}{\epsilon})^{\log\frac{1}{\epsilon}}$.

Acknowledgments

Special thanks to my advisors Avi Wigderson and Michael Benor for their guidance. Thanks to Avi for many discussions on so many aspects of complexity theory, and for his great support through out the whole period of my studies. Thanks to Michael for all our talks on cryptography, algorithms and quantum computations.

I would also like to thanks the members of my steering committee, Noam Nisan and Oded Goldreich for the fruitful discussions and for their comments on my research.

Thanks to Amnon Ta-Shma, Shiyu Zhou and Mike Saks for their collaboration with me in producing some of the results of this thesis.

And thanks to my wife Michal, my daughter Gal, my son Alon, my mother Ofra and my father Uri.

Contents

Chapter 1

Introduction

1.1 The Randomness Resource

Complexity theory is a branch of computer science that analyzes the efficiency of computational problems. The complexity of a given algorithm is a measure of its use of computational resources such as time and space. For randomized algorithms there is a further measure - that of randomized bit complexity - which refers to the number of random coins that the algorithm flips while trying to solve an instance of the problem. Unlike time and space complexities, the randomized bit complexity of any problem can be totally eliminated at the expense of time and space, inviting interest in the interrelationship between randomized bit complexity and the other two complexity measures.

Since the late '70s people have been designing random algorithms in which time and space are more efficient than their known deterministic counterparts. Examples of this include: primality testing, which has randomized polynomial time algorithms [SS77, Rab80, AH87] but no known deterministic polynomial time algorithm; s-t connectivity in undirected graphs, which is solvable in randomized-logspace [AKL$^+$79] but has no known deterministic logspace algorithm. This apparent power of randomization, together with the fact that we do not currently have real random bits, has turned randomization into an important research issue.

Randomized and probabilistic variants are defined for almost any natural complexity class. For example, if P is the class of all problems solvable by a deterministic polynomial time algorithm, then RP is the corresponding randomized class with one sided error and

BPP is the corresponding probabilistic class with bounded two sided error. Similarly (and closer to the context of this thesis), if L is the class of problems that are deterministically computable in logspace, then RL is the class of problems computable in randomized logspace with one sided error and BPL is the class of problems computable in probabilistic logspace with bounded two sided error.

This work investigates the power of randomness when space resources are limited. Three computational models are investigated here. For two of these, we build pseudo-random generators. The derandomization of two of these three models is also described below.

1.2 Derandomization and Pseudo-randomness

The area of *Derandomization* is concerned with eliminating the randomized bit complexity while at the same time minimizing the expense of time and space.

While investigating the power of randomness, our primary concern is whether or not the randomness is at all necessary. More precisely, let C be an arbitrary complexity class and RC be one of the randomized variants of C that we wish to investigate. If we can prove that $RC = C$ we can say that randomness is not really necessary for C. Otherwise, we say that randomness is helpful to C. *Derandomization* involves trying to prove that $RC = C$. Failing this, we try to prove that RC is contained in another deterministic class C', where C' is as small as possible.

A useful tool for derandomization is a pseudo-random generator: this concept was introduced in the early '80s by Shamir [Sha81], Blum and Micali [BM84], and by Yao [Yao82]. A *pseudo-random generator* for a complexity class C is a deterministic algorithm $G : \{0,1\}^l \to \{0,1\}^R$ with the following two properties: (1) $l \ll R$, and (2) if a *seed* $s \in \{0,1\}^l$ was chosen uniformly at random, then $G(s)$ fools C. We say that $G(s)$ *fools* a complexity class C, if, for any algorithm A in C, when s is uniformly distributed over $\{0,1\}^l$, there is a very small statistical difference between the distribution of the output of $A(G(s))$, and the distribution of $A(y)$ when y is truly random over $\{0,1\}^R$. The lower the complexity of G, and the smaller l in relation to R, the better G is as a pseudo-random generator.

While the notion of pseudo-random generators developed within the field of computer science, the parallel notion of discrepancy sets developed within the field of combinatorics. In combinatorics, much research has focussed on building discrepancy sets for the class of geometrical rectangles[BC87, Nie].

Let $A \subseteq \{0,1\}^R$. The volume of A is defined as $vol(A) = |A|2^{-R}$. The volume of A with respect to some subset $S \subseteq \{0,1\}^R$ is defined as $vol_S(A) = |A \cap S| \ / \ |S|$. We say that S is an *ϵ-discrepancy set* for A if the difference between $vol_S(A)$ and $vol(A)$ is no greater than ϵ. If $C \subseteq 2^{\{0,1\}^R}$ is a class of subsets of $\{0,1\}^R$, then S is an ϵ-discrepancy set for C if it is an ϵ discrepancy set for every $A \in C$. This means that for every A in C, if we try to find the volume of A by sampling, then sampling from S will give results equivalent to sampling from the entire universe $\{0,1\}^R$.

Using the Chernoff bound we can prove that for any class C, there exists an ϵ-discrepancy set S of size $O(\epsilon^{-2} \log |C|)$. The interesting question is how to efficiently and deterministically construct discrepancy sets for certain classes C, and to have these sets be as small as possible.

It should be noted at this point that if we limit ourselves to the construction of pseudo-random generators for decision problems, then the definitions of discrepancy sets and pseudo-random generators are equivalent: An algorithm A is equivalent to a subset of $\{0,1\}^R$ and a non uniform complexity class C is a class of subsets As; the generator G is equivalent to the efficient deterministic procedure that creates the elements of S; the seeds s are the indices of elements in S, and the discrepancy set $S = \{G(s) : s \in \{0,1\}^l\}$ is the image of G.

In this thesis we deal with three classes. The first is bounded probabilistic space, denoted by $BPSPACE(\cdot)$ which contains randomized space, denoted by $RSPACE(\cdot)$. The second is symmetric logspace, denoted by SL and contained in $RL = RSPACE(\log n)$. These two classes have their origins in computer science. The third class is the class of all combinatorial rectangles which is also contained in RL and which has its origins in combinatorics. We construct pseudo-random generators for $BPSPACE(\cdot)$ and for combinatorial rectangles, and enforce derandomization methods on $BPSPACE(\cdot)$ and SL. (For combinatorial rectangles, derandomization is irrelevant.)

1.3 $BPSPACE(S)$

Most of the pseudo-random generators that were constructed to derandomize $BPSPACE(S)$ output R pseudo-random bits, where $R = 2^{\Theta(S)}$ [BNS89, Nis92, INW94]. The two generators of Nisan [Nis92] and of Impagliazzo, Nisan and Wigderson[INW94] give the shorter seed, which is of length $O(S \log R)$.

Nisan and Zuckerman [NZ93] studied a subclass of $BPSPACE(S)$, namely, machines that use only $R = S^{O(1)}$ random bits. For these machines, they constructed pseudo-random generators that stretch a seed of length $O(S)$.

In chapter 2 - which is based on the paper [Arm98] - we construct a new pseudo-random generator whose seed length almost bridges the gap between the results of [Nis92] and [NZ93]. Denote $e(z) = 4^{\log^* z}$. We prove that

Theorem 1.1: For every S and $R = 2^{O(S)}$ there exists an $R^{-\Theta(1)}$-pseudo-random generator $G : \{0,1\}^l \to \{0,1\}^R$ that runs in space $O(l)$ and fools any space-S machine that uses R random bits, where $l = O(\frac{(S+e(R)\log R)\log R}{\max\{1, \log S - \log\log R\}})$.

This theorem is proven as corollary 2.7. Note that this generator uses a shorter seed than the generators of [Nis92] and [INW94] when $R < 2^{S^{1-\epsilon}}$ for some $\epsilon > 0$.

We use this generator to derandomize space-bounded computations. The best derandomization known so far is the one of Saks and Zhou [SZ95]. They proved that any $BPSPACE(S)$ algorithm that uses $R = 2^{O(S)}$ random bits can be turned into a deterministic algorithm that runs in space $O(S\sqrt{\log R})$. In chapter 2 we prove that

Theorem 1.2: For every S and $R = 2^{O(S)}$, every randomized algorithm that runs in space S and uses R random bits can be turned into a deterministic algorithm that runs in space $O(\frac{(S+e(R)\log R)\sqrt{\log R}}{\sqrt{\max\{1, \log S - \log\log R\}}})$.

We prove this theorem as corollary 2.9. Again, when $R < 2^{S^{1-\epsilon}}$ for some $\epsilon > 0$, this result is better than the derandomization of [SZ95].

1.4 SL

The class SL of symmetric logspace computations is contained in RL. The problem of s-t connectivity in undirected graphs ($USTCON$) is an interesting SL-complete problem. Since Savitch's derandomization of $USTCON$ [Sav70], showing that the more general class NL is in L^2 (where we use the notation that $L^{\alpha} = DSPACE((\log n)^{\alpha})$), a lot of research has been invested to find the exact space complexity of $USTCON$. The best previous result of Nisan, Szemerédi and Wigderson [NSW92] shows that $SL \subseteq L^{3/2}$. In chapter 3 which is based on the paper [ATSWZ97] we prove that

Theorem 1.3: $SL \subseteq L^{4/3}$

1.5 Combinatorial Rectangles

Combinatorial rectangles of dimension d and side m are elements of the form $R = R_1 \times \ldots \times R_d$ where for every $i \in [d]$,$R_i \subseteq [m]$ is a subset of $[m]$. For any positive integers m and d we denote by $\mathcal{R}(m, d)$ the class of all combinatorial rectangles of dimension d and side m.

Given m, d and $\epsilon > 0$, we wish to construct an ϵ-discrepancy set S for the class $\mathcal{R}(m, d)$, i.e., we want to have $|vol_S(R) - vol(R)| \le \epsilon$ for every $R \in \mathcal{R}(m, d)$. As mentioned earlier, it is easy to show that such a set of size $O(\epsilon^{-2} \log |\mathcal{R}(m, d)|) = O(md\epsilon^{-2})$ exists. However, the efficient and deterministic construction of a discrepancy set of this size remains an open problem.

Note that every rectangle in $\mathcal{R}(m, d)$ is accepted by a $(2, d, m)$-BP (see definition 2.3) thus an ϵ-pseudo-random generator (definition 2.4) for such branching programs will produce an ϵ-discrepancy set for $\mathcal{R}(m, d)$. Using this fact, [EGL$^+$92] constructed a discrepancy set of size $(md/\epsilon)^{O(\log d)}$ by employing Nisan's generator [Nis92], and constructed a set of size $(md/\epsilon)^{O(\log 1/\epsilon)}$ by using k-wise independence.

In chapter 4 which is based on the paper [ASWZ96] we construct an ϵ-discrepancy set for $\mathcal{R}(m, d)$, of size polynomial in $m \cdot d \cdot (1/\epsilon)^{\log 1/\epsilon}$.

Chapter 2

Space-Bounded Computations

This chapter is based on the paper [Arm98]. The computational model that we try to fool in this chapter is space-bounded computations, i.e., the complexity class $BPSPACE(S)$ which contains all the problems solvable by a Turing machine in space S using random bits with a bounded two sided error. The two best pseudo-random generators known today for space-bounded computations are [Nis92] and [NZ93]. In [Nis92] Nisan shows a generator that fools any space S machine that requires $R \leq 2^{O(S)}$ random bits using a seed of length $O(S \log R)$. However, when $R = S^{O(1)}$ Nisan and Zuckerman [NZ93] introduce a more powerful generator which requires only $O(S)$ random bits for its seed. Our generator combines ideas from the two generators of [NZ93] and [INW94], which almost interpolates the whole gap between the two results above. For the rest of this chapter denote $e(z) = 4^{\log^* z}$. We show that for $R \leq 2^{O(S)}$ there exists a pseudo-random generator for space-S computations with seed of length $O(\frac{(S+e(R)\log R)\log R}{\max\{1,\log S - \log\log R\}})$ and statistical error $R^{-\Omega(1)}$. This result is better than both earlier results for R such that $S^{\omega(1)} < R < 2^{S^{1-\epsilon}}$ for some $\epsilon > 0$. For $R = S^k$, the generator of Nisan and Zuckerman uses a seed of length $2^{\Omega(k)}S$ while our generator G uses a seed of length $O(kS)$, and for $R < 2^{S^{1-\epsilon}}$ for some $\epsilon > 0$, the length of the seed is only $O(\frac{S \log R}{\log S})$, while the generator of Nisan needs a seed of length $\Theta(S \log R)$. Note also, that for $S^{\omega(1)} < R < 2^{o(\log^2 S)}$, our generator gives an estimation to the accepting probability of a $BPSPACE(S)$ machine in less space than [SZ95], since they use $\Theta(S\sqrt{\log R})$ space which is larger than our seed.

We would like to remark here that ideally, the term $e(R)$ should simply be constant, but unfortunately this $e(R)$ term arises in the known constructions of extractors, and no better one is known yet.

The best derandomization known so far for $BPSPACE(S)$ is the one of Saks and Zhou [SZ95]. They proved that any problem that is randomly solvable in space S using $R \leq 2^{O(S)}$ random bits, can be solved deterministically in space $O(S\sqrt{\log R})$ as well. Here we show that this task can be done in space $O(\frac{(S+e(R)\log R)\sqrt{\log R}}{\sqrt{\max\{1,\log S-\log\log R\}}})$. For $R < 2^{S^{1-\epsilon}}$ for some $\epsilon > 0$, the space required according to our result is only $O(S\sqrt{\frac{\log R}{\log S}})$ which is less by a factor of $\sqrt{\log S}$ than the space used by Saks and Zhou [SZ95].

2.1 Preliminaries

2.1.1 Definitions and Notations

Let $x \in \mathbb{R}^l$ be a vector, then its ith entry is denoted by $x[i]$. The L_1-norm of x is $\|x\|_1 = \sum_{i=1}^{l} |x[i]|$. Let M be a $l \times l$ matrix over $\mathbb{R}$, then its L_1-norm is $\|M\|_1 = \sup\{xM : x \in \mathbb{R}^l, \|x\|_1 = 1\}$. The statistical distance between two distributions D and D' on the space $\{0,1\}^l$ is $\|D - D'\|_1$.

Let A_1 and A_2 be two events in a probability space. We denote by $\Pr[A_1 : A_2]$ the probability of A_1 conditioned upon A_2. Let $f : W \to R$ be a function, and let X be a random variable distributed on W, then $\mathbf{E}_{x\in X}[f(x)]$ denotes the expectation of $f(X)$. We denote by D_f the distribution over R of the random variable $f(Y)$ where Y is a random variable uniformly distributed over W. If $f : W \to R$ and $g : U \to W$, then $f \circ g$ is the composition of f and g, thus, $D_{f\circ g}$ is the distribution over R of the random variable $f(g(Y))$ where Y is a random variable uniformly distributed over U.

2.1.2 Extractors

Definition 2.1: [NZ93] A distribution D on $\{0,1\}^l$ is called a δ-source if for all $x \in \{0,1\}^l$, $D(x) \leq 2^{-\delta l}$.

Definition 2.2: [Zuc96, Definition 1.4] A function $E : \{0,1\}^k \times \{0,1\}^t \to \{0,1\}^m$ is called a

$(k, \delta, t, m, \epsilon)$-extractor if for every δ-source D on $\{0,1\}^k$, if X is a random variable distributed according to D and Y uniformly distributed on $\{0,1\}^t$, the distribution of $E(X,Y) \circ Y$ is within statistical distance of ϵ from the uniform distribution on $\{0,1\}^{m+t}$.

Lemma 2.1: There is a constant $c_1 > 0$ such that for every integer k and every $\epsilon > 2^{-\frac{k}{4^{\log^* k}}}$ there exists a $(k, \frac{1}{2}, c_1 \log \frac{k}{\epsilon}, \frac{k}{4}, \epsilon)$- extractor which runs in NC.

For the proof, refer to [Zuc96, Theorem 3.13,Remark 3.14] and substitute $n = k$, $\delta = \frac{1}{2}$ and $\alpha = \frac{1}{4}$. Note that although the original proof uses a different definition for statistical distance between distributions, which differs from ours by a factor of 2, the Lemma still holds.

2.1.3 Space Bounded Computations and Branching Programs

In this subsection we show how to simulate a space bounded computation by branching programs (BPs) in order to conclude that a pseudo-random generator for BPs fools space bounded machines as well. For that purpose we need the following definition of oblivious read-once BP.

Definition 2.3: A (w, n, r)-BP (branching program) is a graph of $n+1$ layers, indexed by $0, \ldots, n$, with w vertices at each layer (length n and width w). For each vertex v at layer $i < n$ there are 2^r outgoing multi-edges to vertices at layer $i+1$ where each of these edges is labeled by a distinct string from $\{0,1\}^r$. Upon receiving an input string from $\{0,1\}^{rn}$ which can be viewed as n blocks of length r, denoted $R_1, \ldots, R_n$, all edges are deleted from the BP, except for the edges between layer $i-1$ and i labeled by R_i, for every $i \leq n$. Let s be one of the vertices at layer 0 - we call s the initial vertex. The value computed by the BP is the index (in $[w]$) of the last vertex in the path that was started from s. Thus, a (w, n, r)-BP can be viewed as a function from $\{0,1\}^{rn}$ to $[w]$.

Now, we define a pseudo-random generator for BPs.

Definition 2.4: A function $G : \{0,1\}^l \to \{0,1\}^{rn}$ is called a γ-pseudo-random generator for (w, n, r)-BPs if for every (w, n, r)-BP P, $\|D_P - D_{P \circ G}\|_1 \leq \gamma$.

Definition 2.5: A function $G : \{0,1\}^l \to \{0,1\}^R$ is called a γ-pseudo-random generator for space-S machines that use R random bits, if for every machine M that runs in space S, uses R random bits and outputs $\{accept, reject\}$, then $\|D_M - D_{M \circ G}\|_1 \leq \gamma$.

Lemma 2.2: Let $S, R > 0$ be integers and let $\alpha > 0$. Then, if $G : \{0,1\}^l \to \{0,1\}^R$ is an α-pseudo-random generator for $(2^S, R, 1)$-BPs, then G is an α-pseudo-random generator for space-S machines that use R random bits.

Proof: Let M be a space-S machine that uses R random bits. Following [INW94], we simulate M by a $(2^S, R, 1)$-BP P. At each layer of P, the 2^S different vertices will correspond to the different configurations of M. An edge from vertex v at layer i, labeled by $x \in \{0,1\}$ will end up at vertex u at layer $i+1$ iff the input x on the random tape of M takes it from configuration v to u. Let s be the initial configuration of M. It follows that P computes the ending configuration of P. Now, since the output of M is a mapping from configurations to $\{accept, reject\}$, we get that for every $G : \{0,1\}^l \to \{0,1\}^R$, $\|D_M - D_{M \circ G}\|_1 \leq \|D_P - D_{P \circ G}\|_1$. We conclude that if G is α-pseudo-random for $(2^S, R, 1)$-BPs, then G is α-pseudo-random for space-S machines that use at most R random bits. $\Box$

Remark 2.1: If $G : \{0,1\}^l \to \{0,1\}^{rn}$ is γ-pseudo-random generator for (w, n, r)-BPs, and if for some $r' < r$, $G' : \{0,1\}^l \to \{0,1\}^{r'n}$ is defined to output the first r' bits of every output block of G, then G' is γ-pseudo-random generator for (w, n, r')-BPs.

2.2 Pseudo-randomness

In this section we show a pseudo-random generator that fools any (w, n, r)-BP P. Our building block will be the extractor from sub-section 2.1.2.

2.2.1 Pseudo-randomness for Branching Programs

Definition 2.6: Let c_1 be the constant of Lemma 2.1. Given n, w, r and γ, fix the following parameters: Let $\epsilon \stackrel{\text{def}}{=} \frac{\gamma}{2n}$, $k \stackrel{\text{def}}{=} \max\{4r, 2\log w + e(\frac{1}{\epsilon}) \log \frac{1}{\epsilon}\}$ (we remind the reader that

$e(z) = 4^{\log^* z}$) and $t \stackrel{\text{def}}{=} c_1 \log \frac{k}{\epsilon}$. Let E be the $(k, \frac{1}{2}, t, r, \epsilon)$-extractor of Lemma 2.1. Define a generator $\hat{G} : \{0,1\}^{k+nt} \to \{0,1\}^{nr}$ as follows:

$$\forall x \in \{0,1\}^k, y_1, \ldots, y_n \in \{0,1\}^t, \quad \hat{G}(x, y_1, \ldots, y_n) \stackrel{\text{def}}{=} E(x, y_1), \ldots, E(x, y_n)$$

For every $y_1, \ldots, y_n \in \{0,1\}^t$, we denote $\hat{G}^{(y_1,\ldots,y_n)} : \{0,1\}^k \to \{0,1\}^{nr}$ as $\hat{G}^{(y_1,\ldots,y_n)}(x) = \hat{G}(x, y_1, \ldots, y_n)$.

Lemma 2.3: For every n, w, r and γ, for every (w,n,r)-BP P and for every random variable Y uniformly distributed on $\{0,1\}^{tn}$, $\mathbf{E}_{y \in Y}[\|D_P - D_{P \circ \hat{G}^{(y)}}\|_1] \leq \gamma$.

A similar version of this Lemma has already been proven by [NZ93, Lemma 2] for space-bounded machine, and we state our Lemma for BPs. Another difference is that their Lemma is weaker, since they only proved that

$$\|\mathbf{E}_{y \in Y}[D_P - D_{P \circ \hat{G}^{(y)}}]\|_1 = \|D_P - D_{P \circ \hat{G}}\|_1 \leq \gamma$$

Proof: Let P be a (w,n,r)-BP. We denote by $S_i(u,v) \subseteq \{0,1\}^r$ the set of labels of edges going from vertex u at layer $i-1$ to vertex v at layer i of P. We define the following $w \times w$ stochastic matrices. M_i is the matrix with $\frac{|S_i(u,v)|}{2^r}$ at its (u,v) coordinate and for every $x \in \{0,1\}^k$ and $y \in \{0,1\}^t$ we define the matrix $\tilde{M}_i(x,y)$ with 1 at its (u,v) coordinate if $E(x,y) \in S_i(u,v)$ and 0 otherwise. Let u_0 be the initial state of P, then let e be a unit vector of length w with 1 at its u_0 coordinate and 0 elsewhere.

Let X be a random variable uniformly distributed on $\{0,1\}^k$. Notice that $D_P = e \prod_{i=1}^n M_i$ and that for every $y_1, \ldots, y_n \in \{0,1\}^t$,

$$D_{P \circ \hat{G}^{(y_1,\ldots,y_n)}} = \mathbf{E}_{x \in X}[e \prod_{i=1}^{n} \tilde{M}_i(x, y_i)]$$

Let $Y_1, \ldots, Y_n$ be random variables uniformly distributed on $\{0,1\}^t$. We will show by induction on j that

$$\mathbf{E}_{y_1 \in Y_1, \ldots, y_j \in Y_j}[\|e((\prod_{i=1}^{j} M_i) - \mathbf{E}_{x \in X}[\prod_{i=1}^{j} \tilde{M}_i(x, y_i)])\|_1] \leq \frac{j\gamma}{n}$$

For $j = 0$ we trivially get equality. For $j > 0$, assume correctness for $j-1$ and prove for j.

$$\begin{aligned}
&\mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|e((\prod_{i=1}^{j} M_i) - \mathbf{E}_{x\in X}[\prod_{i=1}^{j} \tilde{M}_i(x,y_i)])\|_1] \leq \\
\leq\ & \mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|e((\prod_{i=1}^{j-1} M_i) - \mathbf{E}_{x\in X}[\prod_{i=1}^{j-1} \tilde{M}_i(x,y_i)])M_j\|_1] + \\
& + \mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|e\mathbf{E}_{x\in X}[(\prod_{i=1}^{j-1} \tilde{M}_i(x,y_i))(M_j - \tilde{M}_j(x,y_j))]\|_1] \\
\leq\ & \frac{(j-1)\gamma}{n} + \mathbf{E}_{y_1\in Y_1,\ldots,y_j\in Y_j}[\|\mathbf{E}_{x\in X}[v(x,y_1,\ldots,y_{j-1})(M_j - \tilde{M}_j(x,y_j))]\|_1]
\end{aligned}$$

where in the last inequality, the first summand is bounded by the induction hypothesis and $\|M_j\|_1 = 1$ as it is stochastic, and in the second summand, $v(x,y_1,\ldots,y_{j-1}) = e\prod_{i=1}^{j-1} \tilde{M}_i(x,y_i)$. Note that $v(x,y_1,\ldots,y_{j-1})$ is a distribution vector over $[w]$. To conclude the proof, it is enough to show that for every $y_1,\ldots,y_{j-1} \in \{0,1\}^t$,

$$\mathbf{E}_{y_j\in Y_j}[\|\mathbf{E}_{x\in X}[v(x,y_1,\ldots,y_{j-1})(M_j - \tilde{M}_j(x,y_j))]\|_1] \leq \frac{\gamma}{n}$$

For the rest of the proof we fix the values of $y_1,\ldots,y_{j-1}$.

Let $\mathcal{V}$ be a random variable distributed on $[w]$ according to the distribution $\mathbf{E}_{x\in X}[v(x,y_1,\ldots,y_{j-1})]$. Let $L = \{v \in [w] : \Pr[\mathcal{V} = v] > \frac{\epsilon}{w}\}$.

For $v \in L$, $\Pr[X = x : \mathcal{V} = v] \leq \frac{\Pr[X=x]}{\Pr[\mathcal{V}=v]} \leq \frac{w}{\epsilon}\cdot \Pr[X = x] \leq 2^{\frac{k}{2}} \cdot \Pr[X = x]$. We get that the distribution of X conditioned upon $\mathcal{V} = v$ is a $\frac{1}{2}$-source. Since E is an extractor, we get that for a random variable Z which is uniformly distributed on $\{0,1\}^r$,

$$\sum_{y_j\in\{0,1\}^t} \sum_{z\in\{0,1\}^r} |\Pr[(Z = z) \wedge (Y_j = y_j)] - \Pr[(E(X,Y_j) = z) \wedge (Y_j = y_j) : \mathcal{V} = v]| \leq \epsilon$$

thus,

$$\begin{aligned}
&\mathbf{E}_{y_j\in Y_j}[\|\mathbf{E}_{x\in X}[v(x,y_1,\ldots,y_{j-1})(M_j - \tilde{M}_j(x,y_j))]\|_1] = \\
=\ & \mathbf{E}_{y_j\in Y_j}[\|\mathbf{E}_{x\in X}[\sum_{v\in[w]} \Pr[\mathcal{V} = v](M_j[v,\cdot] - \tilde{M}_j(x,y_j)[v,\cdot])]\|_1] \\
\leq\ & \sum_{v\in L} \Pr[\mathcal{V} = v]\mathbf{E}_{y_j\in Y_j}[\sum_{v'\in[w]} |\Pr[Z \in S_j(v,v')] - \Pr[E(X,y_j) \in S_j(v,v') : \mathcal{V} = v]|] \\
& + \sum_{v\notin L} \Pr[\mathcal{V} = v]\mathbf{E}_{y_j\in Y_j}[\|\mathbf{E}_{x\in X}[(M_j[v,\cdot] - \tilde{M}_j(x,y_j)[v,\cdot])]\|_1] \\
\leq\ & \sum_{v\in L} \Pr[\mathcal{V} = v]\cdot\epsilon + \sum_{v\notin L} \Pr[\mathcal{V} = v]\cdot 1 \leq \epsilon + \sum_{v\notin L}\frac{\epsilon}{w} \leq 2\epsilon \leq \frac{\gamma}{n}
\end{aligned}$$

□

Theorem 2.4: There exist constants c_2, c_3 such that for every n, w, r and γ, if $k = c_2(r + \log w + e(\frac{n}{\gamma}) \log \frac{n}{\gamma})$ and $t = c_3 \log \frac{nr \log w}{\gamma}$ then $\hat{G} : \{0,1\}^{k+nt} \to \{0,1\}^{nr}$ is a γ-pseudo-random generator for (w, n, r)-BPs that runs in NC.

Proof: Let $c_2 = 4$ and $c_3 = 2c_1$. Let Y be a random variable uniformly distributed on $\{0,1\}^{nt}$. By Lemma 2.3, for every (w, n, r)-BP P

$$\|D_P - D_{P \circ \hat{G}}\|_1 = \|D_P - \mathbf{E}_{y \in Y}[D_{P \circ \hat{G}(y)}]\|_1 \leq \mathbf{E}_{y \in Y}[\|D_P - D_{P \circ \hat{G}(y)}\|_1] \leq \gamma$$

Since each of the output bits of $\hat{G}$ is simply computed by applying an extractor to some predefined subset of the input bits, and since the extractor runs in NC, then each of the output bits of $\hat{G}$ is computed in NC as well. □

2.2.2 Composing Generators for Branching Programs

In this subsection, we will see that the generator $\hat{G}$ that was defined in subsection 2.2.1 can be composed on itself. To this end, we need the following proposition:

Proposition 2.5: Let P be a (w, n, r)-BP and let $\hat{G} : \{0,1\}^{k+nt} \to \{0,1\}^{nr}$ be the pseudo-random generator for (w, n, r)-BPs of Theorem 2.4. Then, for every $x \in \{0,1\}^k$,

$$P^{(x)}(y_1, \ldots, y_n) \stackrel{\text{def}}{=} P(\hat{G}(x, y_1, \ldots, y_n))$$

is a (w, n, t)-BP.

Proof: Note that $P^{(x)}$ has the same vertex set as P has, but its edge set is smaller. Every edge from layer $i-1$ to layer i of P with label from $E(x, y_i)$ for some value of $y_i \in \{0,1\}^t$ is labeled in $P^{(x)}$ by y_i while all other edges of P do not appear in $P^{(x)}$. Hence, $P^{(x)}$ is a (w, n, t)-BP. □

In order to be able to compose the generator, we would need to view the BP $P^{(x)}$ as a (w, n', r)-BP for some n'. In the following definition we formally show how we do so.

Definition 2.7: Let Q be a (w,n,t)-BP. We define the operation *collapse-c*, the result of which is a $(w,\lceil\frac{n}{c}\rceil,tc)$-BP, $\tilde{Q}$, as follows. Denote $n' \stackrel{\text{def}}{=} \lceil\frac{n}{c}\rceil$, then $\tilde{Q}$ has $n'+1$ layers indexed by $0,\ldots,n'$, each of which contains w vertices.
For every $i<n'$, for every $u,v\in[w]$, the edge between u at layer i and v at layer $i+1$ exists in $\tilde{Q}$ if and only if there exists a path in Q between vertex u at layer ic and vertex v at layer $(i+1)c$ (in the case that $(i+1)c>n$, we look for a path ending at vertex v at layer n, which is of course of length shorter than c).
The label of the edge (u,v) in $\tilde{Q}$ is the concatenation of the labels on the edges in the corresponding path in Q. In the case that the length of the paths to the last layer in Q are of length c' shorter than c, the edges in $\tilde{Q}$ corresponding to these paths have labels of length only $c't$. In this case we lengthen these labels by concatenating to each of them, all the possible suffixes from $\{0,1\}^{(c-c')t}$ which results in replacing each edge by $2^{(c-c')t}$ parallel edges.

Assuming that $t<r$, we would like to perform the operation collapse-$\lceil\frac{r}{t}\rceil$ on $P^{(x)}$ and get a new BP, $\tilde{P}^{(x)}$, with labels of length at least r. This fact allows us to feed the output of $\hat{G}$ instead of a real random string to $\tilde{P}^{(x)}$. For the rest of this subsection, we will properly define the composition of $\hat{G}$ and prove that this composition really gives us a pseudo-random generator for BPs.

Definition 2.8: Let n,w,r and α be given. Let $\gamma \stackrel{\text{def}}{=} \frac{\alpha}{\log n}$ and let $k=c_2(r+\log w+e(\frac{n}{\gamma})\log\frac{n}{\gamma})$. Let $t=c_3\log\frac{2nk\log w}{\gamma}$, $r'=t\max\{2,\lceil\frac{2\log w+e(\frac{n}{\gamma})\log\frac{n}{\gamma}}{t}\rceil\}$ and $k'=2c_2r'$.

Let $h \stackrel{\text{def}}{=} \lceil\frac{\log n}{\log\frac{r'}{t}}\rceil$, let n_0 be n and for $i>0$, let $n_i \stackrel{\text{def}}{=} \frac{n_{i-1}}{\frac{r'}{t}}$. For $i>0$, we define generators $G_i:\{0,1\}^k\times\{0,1\}^{(i-1)\cdot k'}\times\{0,1\}^{n_{i-1}\cdot t}\to\{0,1\}^{n\cdot r}$ recursively as follows:

$$G_1(x_1,y_1,\ldots,y_n) \stackrel{\text{def}}{=} \hat{G}(x_1,y_1,\ldots,y_n)$$

and for $i>1$

$$G_i(x_1,\ldots,x_i,y_1,\ldots,y_{n_{i-1}}) \stackrel{\text{def}}{=} G_{i-1}(x_1,\ldots,x_{i-1},\hat{G}(x_i,y_1,\ldots,y_{n_{i-1}}))$$

where we think of each of the n_{i-1} output blocks of $\hat{G}$ of length r' as partitioned into $\frac{r'}{t}$ blocks of length t, which gives a total of n_{i-2} blocks from $\{0,1\}^t$ each. Now, for every $x_1 \in \{0,1\}^k$, every $x_2, \ldots, x_h \in \{0,1\}^{k'}$ and for every $y \in \{0,1\}^t$ we define

$$G(x_1, \ldots, x_h, y) \stackrel{\text{def}}{=} G_h(x_1, \ldots, x_h, y)$$

Theorem 2.6: G is an α-pseudo-random generator for (w,n,r)-BPs that expands a seed of length $l = O(r + \log n \frac{\log rw + e(\frac{n}{\alpha}) \log \frac{n}{\alpha}}{\max\{1, \log\log w - \log\log \frac{nr}{\alpha}\}})$ and it runs in space $O(l)$.

Proof: The length of the seed that G uses is $k + k'(h-1) + t = O(r + k'h)$.

$$\begin{aligned} k'h &= O(k' \frac{\log n}{\log \frac{r'}{t}}) = O(\frac{r' \log n}{\max\{1, \log\log w - \log t\}}) \\ &= O(\log n \frac{\log rw + 4^{\log^* \frac{n}{\alpha}} \log \frac{n}{\alpha}}{\max\{1, \log\log w - \log\log \frac{nr}{\alpha}\}}) \end{aligned}$$

thus, the total length of the seed is $O(r + \log n \frac{\log rw + e(\frac{n}{\alpha}) \log \frac{n}{\alpha}}{\max\{1, \log\log w - \log\log \frac{nr}{\alpha}\}})$. Since each of the output bits of $\hat{G}$ is computed in NC, and G is a composition of $\hat{G}$ at most h times, then G does not use more than $k'h = O(l)$ space.

Let P be a (w,n,r)-BP with n and r that satisfy the conditions in Definition 2.8. We prove by induction on i that $\|D_P - D_{P \circ G_i}\|_1 \leq i\gamma$.

For $i = 1$ the claim follows immediately from the fact that by the choice of k and t and by Theorem 2.4, $\hat{G}$ is a γ-pseudo-random generator for (w,n,r)-BPs. Assume correctness for $i-1$ and prove for $i > 1$.

$$\|D_P - D_{P \circ G_i}\|_1 \leq \|D_P - D_{P \circ G_{i-1}}\|_1 + \|D_{P \circ G_{i-1}} - D_{P \circ G_i}\|_1$$

By the induction hypothesis, the first summand is bounded by $(i-1)\gamma$ so we are left to bound the second. Let $x_1 \in \{0,1\}^k$, $x_2, \ldots, x_{i-1} \in \{0,1\}^{k'}$ and $y_1, \ldots, y_{n_{i-1}} \in \{0,1\}^t$ we define

$$P^{(x_1, \ldots, x_{i-1})}(y_1, \ldots, y_{n_{i-1}}) \stackrel{\text{def}}{=} P(G_{i-1}(x_1, \ldots, x_{i-1}, y_1, \ldots, y_{n_{i-1}}))$$

Let X_1 be a random variable uniformly distributed over $\{0,1\}^k$ and let $X_2,\ldots,X_{i-1}$ be random variables uniformly distributed over $\{0,1\}^{k'}$. Then,

$$\begin{aligned} D_{P\circ G_{i-1}} &= \mathbf{E}_{x_1\in X_1}[\ldots\mathbf{E}_{x_{i-1}\in X_{i-1}}[D_{P^{(x_1,\ldots,x_{i-1})}}]\ldots] \\ D_{P\circ G_i} &= \mathbf{E}_{x_1\in X_1}[\ldots\mathbf{E}_{x_{i-1}\in X_{i-1}}[D_{P^{(x_1,\ldots,x_{i-1})}\circ\hat{G}}]\ldots] \end{aligned}$$

we get that

$$\|D_{P\circ G_{i-1}} - D_{P\circ G_i}\|_1 \leq \mathbf{E}_{x_1\in X_1}[\ldots\mathbf{E}_{x_{i-1}\in X_{i-1}}[\|D_{P^{(x_1,\ldots,x_{i-1})}} - D_{P^{(x_1,\ldots,x_{i-1})}\circ\hat{G}}\|]_1\ldots] \leq \gamma$$

where the last inequality is correct since $P^{(x_1,\ldots,x_{i-1})}$ is a (w,n_{i-1},t)-BP and by performing $\frac{r'}{t}$-collapse we get a (w,n_i,r')-BP and by the choice of k' and t and by Theorem 2.4, $\hat{G}$ is a γ-pseudo-random generator for (w,n_i,r')-BPs. We conclude that $\|D_P - D_{P\circ G_h}\|_1 \leq h\gamma \leq \alpha$ □

Corollary 2.7: There is an α-pseudo-random generator $G:\{0,1\}^l \to \{0,1\}^R$ for space-S machines that use R random bits, where G uses a seed of length $l = O(\frac{(S+e(\frac{R}{\alpha})\log\frac{R}{\alpha})\log R}{\max\{1,\log S-\log\log\frac{R}{\alpha}\}})$.

Proof: Use Theorem 2.6 and Lemma 2.2. □

2.3 Derandomization of Space Bounded Computations

In this section we show how to derandomize space bounded computations using our generator. The generator is used to approximate the probabilities of moving from one configuration to another in R steps in the space bounded machine. This is equivalent to approximating the R's power of the stochastic (or sub stochastic) matrix N that contains the probabilities of moving from one configuration to another in one step of the machine. We will fix two parameters, t_1 and t_2 such that their product is $\log R$. Then, we will approximate $N^{2^{t_1}}$ using the generator, and repeating this process t_2 times will give us an approximation to N^R. Approximating $N^{2^{t_1}}$ using the generator involves trying all possible seeds for the generator, which requires storing the seed which is too long. Instead, we may use samplers [BGG93, Zuc96] which allow us to use a random input of length linear in the length of the seed for the generator, and other than that, the approximation procedure needs only $O(S + \log R)$ space.

Another important point is that as in [SZ95], we wish to reuse the same random bits in all of the t_2 repetitions of the approximation procedure. However, a dependency problem arises since the outcome of one repetition depends on the random input and serves as an input for the next. We need to break this dependency.

The two tasks described above become straight forward when using the repeated averaging Lemma (Lemma 3.6). We show the following Lemma which is a consequence of our generator and the repeated averaging Lemma:

Lemma 2.8: Let $\beta < 1$. Let S and $R < 2^{O(S)}$ be given positive integers. Let $\alpha > 0$. Set $t_1 = \lceil\sqrt{(\log R)\max\{1, \log S - \log\log\frac{R}{\alpha}\}}\rceil$ and $t_2 = \lceil\frac{\log R}{t_1}\rceil$ (for simplicity assume that $R = 2^{t_1 t_2}$). Set $l = O(\frac{(S+e(\frac{2^{t_1}}{\alpha})\log\frac{2^{t_1}}{\alpha})t_1}{\max\{1, \log S - \log(t_1 \log\frac{1}{\alpha})\}})$ of corollary 2.7. Let $b < l^\beta$.
Then for every space-S machine M that uses R random bits there exists a machine M' that runs in space $O(t_2(S + b) + \text{poly}\log l)$, uses $O(l + t_2 b)$ random bits, and with probability $1 - t_2(2^{2S-l} + 2^{2S-b+1})$ computes an estimation to the accepting probability of M which is at distance at most $R^2(\alpha + 2^{S-b})$.

Proof: Let N be the $2^S \times 2^S$ stochastic matrix associated with the machine M, i.e., N contains in its i, j entry the probability of moving from configuration i to configuration j in M on a single random bit. Let $G : \{0,1\}^l \to \{0,1\}^{2^{t_1}}$ be the α-pseudo-random generator for space-S machines that use 2^{t_1} random bits. Define a randomized algorithm $A(N'; y)$ that uses the generator G to produce an estimation of distance α to the matrix $N'^{2^{t_1}}$ using a random string y (see [SZ95, algorithm PRS]).

By lemma 3.6 there exists an algorithm M' that outputs a sequence $N_1, \ldots, N_{t_2}$ (where $N_0 = N$) such that for all but $t_2(2^{2S-l} + 2^{2S-b+1})$ fraction of its random inputs

$$\|N_i - \mathcal{A}(N_{i-1})\|_\infty \leq 2^{-b}$$

where $\mathcal{A}(N')$ is the average on $y \in \{0,1\}^l$ of $A(N'; y)$. It follows that with probability

$1 - t_2(2^{2S-l} + 2^{2S-b+1})$

$$\begin{aligned} \|N_i - N^{2^{it_1}}\|_1 &\leq \|N_i - \mathcal{A}(N_{i-1})\|_1 + \|\mathcal{A}(N_{i-1}) - N_{i-1}^{2^{t_1}}\|_1 + \|N_{i-1}^{2^{t_1}} - N^{2^{it_1}}\|_1 \\ &\leq 2^{S-b} + \alpha + 2^{t_1} 2^{2(i-1)t_1}(\alpha + 2^{S-b}) \leq 2^{2it_1}(\alpha + 2^{S-b}) \end{aligned}$$

where the first summand is bounded by lemma 3.6 with a 2^S factor to compensate for using L_1 norm on matrices instead of L_∞, the second summand is bounded by the fact that G is an α-pseudo-random generator, and the last summand is bounded by an implicit induction hypothesis.

It is left to see that M' (again, by lemma 3.6) uses $O(l + t_2 b)$ random bits and runs in space $O(t_2(S+b) + \text{poly}\log l)$, and that by computing an approximation to $N^{2^{t_1 t_2}}$ we actually approximate the accepting probability of the machine M. □

Corollary 2.9: Every language which is decidable randomly in space S using $R \leq 2^S$ random bits with two sided error, can be decided deterministically in space $O(\frac{(S+e(R)\log R)\sqrt{\log R}}{\sqrt{\max\{1, \log S - \log\log R\}}})$.

Proof: Let M be the Turing machine that decides the language in space S using R random bits and set $\alpha = R^{-3}$ and $b = 4S$. We now use lemma 2.8 and exhaust all the possible random inputs $y \in \{0,1\}^{O(l+t_2 b)}$ to the algorithm M' and run it on every input to get an estimation to M^R by taking average of every entry.

By lemma 2.8 we approximates the matrix M^R with error smaller than $\frac{1}{10}$ (by the choice of parameters) and in space $O(\frac{(S+e(R)\log R)\sqrt{\log R}}{\sqrt{\max\{1, \log S - \log\log R\}}})$. Taking majority vote on the accepting configuration, we decide whether the given input is in the language. □

Chapter 3

Symmetric Log-Space

This chapter is based on the paper [ATSWZ97]. Undirected st-connectivity ($USTCON$) is a fundamental computational problem, and algorithms for it serve as basic subroutines for more complex graph problems. It is complete for the class SL of symmetric nondeterministic log-space computations [LP82], and is a subproblem of Directed st-connectivity, which captures the class NL of general nondeterministic computation. The combinatorics of $USTCON$, as well as its time complexity, are extremely well understood. However, its space complexity is still a mystery, which was a source of some beautiful discoveries in complexity theory. We remind the reader that we adopt NC-style notations and let $L^{\alpha} = DSPACE((\log n)^{\alpha})$.

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

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

Aleliunas et al [AKL+79] proved not only the *existence* of such a UTS of polynomial length, but did it via the probabilistic method, giving in particular a probabilistic log-space (RL) algorithm for $USTCON$. Thus they established $SL \subseteq RL$. Unfortunately, it did not provide deterministic space-efficient algorithms to generate such short *UTS*.

In a seminal paper, Nisan [Nis92] proved that a *UTS* can be constructed in L^2. This construction was based on a pseudo-random generator that fools *RL* machines. In particular, it can be used to derandomize the above probabilistic algorithm. This hierarchical generator requires $\log n$ universal hash functions of $O(\log n)$ bits each. While not directly improving the deterministic space bound for *USTCON*, Nisan's techniques were the basis of all subsequent progress, starting with his own paper [Nis94] proving $RL \subseteq SC$ (which in particular gives a $(\log n)^2$-space, polynomial time algorithm for *USTCON*).

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

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

In this chapter we prove theorem 1.3 that says that $SL \subseteq L^{4/3}$.

The bound of this chapter, $SL \subseteq L^{4/3}$, requires a careful combination of the ingredients of both papers above, with some new ideas. We will follow the shrinking scheme of [NSW92].

However, we replace the *UTS* of [NSW92] by a pseudo-random walk using Nisan's generator with *short* hash functions. We show that such a walk of length $k^{O(1)}$, just like a random one, will visit (the neighbors of) $O(k)$ vertices with constant probability on every graph of any size! Note that since only $(\log k)^2$ bits are used, Nisan's analysis does not apply, so we provide a different analysis.

Next we would like to repeatedly use the same hash functions of this pseudo-random walk in many iterations, in [SZ95] style. To remove dependencies we approximate the average behavior of this set of functions (an object independent of any particular one function) with sufficient accuracy and high probability. This is achieved without space and random bit penalty via deterministic sampling of either [BGG93] or the recent extractor constructions of [Zuc96]. This technique for removing dependencies, stated as Theorem 3.5, may become a useful derandomization tool.

The rest of the chapter is organized as follows. We start with some preliminaries (Section 3.1), continue with an informal overview (Section 3.2) followed by a high level description of our algorithm (Section 3.3). We then describe the two results upon which the construction is based: the sampler that (w.h.p) does not depend on its random coins (Section 3.4), and the derandomized short random walks (Section 3.5).

3.1 Preliminaries

3.1.1 Random Computations

The usual definition of a randomized space bounded computation allows the machine to flip a random coin at any point of the computation. It does not allow the machine to recall the outcome of previous coin tosses. This is equivalent to giving the machine a *read-once* access to a long-enough random tape.

A different variant to this model is the one where the machine is allowed *multiple access* to the random tape. We call such algorithms *off-line randomized algorithms* [SZ95]. They are sometimes also called algorithms with 2-way random input. An *off-line randomized* algorithm

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

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

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

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

3.1.2 Matrix Algorithms

A *matrix algorithm* A is an algorithm that gets a primary input x and in addition two indices $i, j \in [d]$; the output of A is interpreted as the (i, j)-th entry of a $d \times d$ matrix over the reals, denoted $A(x)$. Since the entire matrix $A(x)$ can be obtained by running the algorithm over all the indices $i, j \in [d]$, in a sense, a matrix algorithm is a function that maps inputs x to square matrices over the reals, given in some finite representation. Understood that the computation operates in the way described, we will say that a matrix algorithm A on input x *computes* a matrix $A(x)$.

Let $M_0 = M, M_1, \ldots, M_p$ be a sequence of square matrices and $z_1, \ldots, z_p$ be a sequence of additional parameters. For typographical simplicity, we will denote a sequence $[x_1, \ldots, x_p]$ by $[x_i]_p$, e.g., $[M_i]_p$ stands for $(M_1, \ldots, M_p)$ and $[z_i]_p$ for $(z_1, \ldots, z_p)$. We also denote the dimension of M by $dim(M)$, i.e., if M is a $d \times d$ matrix then $dim(M) = d$. The following fact is well known:

Proposition 3.1: Suppose there is a matrix algorithm A such that for any $1 \leq i \leq p$, A on input (M_{i-1}, z_i, i) computes M_i and runs in space $S_i \geq \log(dim(M_{i-1}))$. Then there is a matrix algorithm F such that F on input $(M, [z_i]_p)$ computes M_p and runs in space $O(\sum_{i=1}^{p} S_i)$.

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

3.2 Informal Overview

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

1. For each $1 \leq i \leq p$, s, t are vertices in G_i and s, t are connected in G_i if and only if they are connected in G_{i-1}, where $G_0 = G$.

2. The sizes of G_i are non-increasing.

3. G_p is a transitive closure graph (union of cliques).

The output of the algorithm is "yes" iff s is connected to t in G_p. Clearly such an algorithm recognizes the language undirected st-connectivity.

3.2.1 Savitch's Algorithm

We will identify a graph G with its adjacency matrix, denoted also by G, in which $G[i, j]$ is 1 if (i, j) is an edge in G and is 0 otherwise.

In [Sav70], the algorithm computes the sequence of graphs $[G_i]_p$ such that $G_i = G_{i-1}^2 = G^{2^i}$. In words, vertices u, v are connected in G_i if and only if there is a path of length at most 2^i from u to v in G. Clearly, $[G_i]_p$ satisfy RCP. We can see that the space used at every recursive level (to compute G_i given G_{i-1}) is $O(\log n)$ and p is $\log n$. So the total space required is $O(\log^2 n)$ (by Proposition 3.1).

3.2.2 The Algorithm of Nisan, Szemerédi and Wigderson

The algorithm of [NSW92] computes a sequence of matrices $[N_i, G_i]_p$ depending on a parameter k which we call the *shrinking parameter*. The sequence $[G_i]_p$ satisfies RCP, and has the property that $dim(G_i) \leq dim(G_{i-1})/k$, i.e., the algorithm "shrinks" the graph G by a factor of k at every recursive level and at the same time preserves RCP.

The computation at each recursive level consists of computing two matrices N_i and G_i. N_i is a k-rich *neighborhood matrix*:

Definition 3.1: [neighborhood matrix] Let G be a graph. A *neighborhood matrix* N of G is a real matrix of dimension $dim(G)$ such that all the entries are in the range $[0, 1]$ and if $N[i, j] \neq 0$ then vertex i is connected (by a path) to vertex j in G. A *Boolean* neighborhood matrix of G is a neighborhood matrix of G with *Boolean* entries.

Definition 3.2: [rich Boolean neighborhood matrix] A Boolean neighborhood matrix N of G is k-rich if any row of N is k-rich. A row i is k-rich if it contains at least $\min\{k, \#CC(i)\}$ "1" entries, where $\#CC(i)$ is the number of the vertices in the connected component of i in G.

N_i is obtained by taking a walk on graph G_{i-1} according to the universal traversal sequence constructed in [Nis92], which takes space $O(\log^2 k + \log n)$. Using the UTS property, it can be easily proved that N_i is indeed k-rich. The computation of G_i is based on a matrix algorithm which we denote by SRNK for *graph shrinking procedure*. This algorithm will be the basic building-block of the construction of our algorithm and we summarize its properties as follows.

Lemma 3.2: Given as input (G, s, t, N), where G is an undirected graph of size n, s, t are vertices in G and N is a k-rich neighborhood matrix of G, the algorithm SRNK runs in space $O(\log(n))$ and computes a graph G' such that

1. s, t are distinct vertices in G' and $G'[s, t] \iff G[s, t]$.

2. $dim(G') \leq dim(G)/k$.

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

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

3.2.3 Our Variant - Intuition

In [NSW92] the rich neighborhood matrix N was computed using a UTS. We replace this by a randomized algorithm that for most random strings y' computes a (possibly different) rich neighborhood matrix $N_{y'}$. This can be thought of as replacing the guarantee that at least one random string y' is good, by a guarantee that almost all random strings are good. Still, this by itself does not improve the space complexity of the algorithm.

We then notice that if we average over all the possible neighborhood matrices $N_{y'}$ that our randomized algorithm can give, i.e., taking $\mathcal{N} = \mathbf{E}_{y'}[N_{y'}]$, we also get a rich neighborhood matrix. Here we see the advantage of replacing the UTS by a randomized algorithm: since most y''s are good the average is also good. We also notice that the average does not depend on y'.

It is well known that even though the average can not be computed exactly, it can be extremely well approximated by a randomized algorithm using not too many random coins - only $O(|y'|)$. This can be achieved using efficient samplers [BGG93, Zuc96]. It is also easy to see that the approximated average also gives a rich neighborhood matrix. The only remaining problem is that each random string gives a different approximation and thus the result of the randomized algorithm does depend on the random coins it uses.

To solve this dependency problem we use a technique of [SZ95]. We randomly perturb the approximated matrix $\mathcal{N}_y$ and then we truncate it to the desired level of accuracy. We show that with high probability, the result depends only on the perturbation q and not on the random string y used in the approximation.

Thus, we can use the same string y (of long length $O(\log^2 k)$) as the random off-line input at all different recursive levels. On the other hand, we use independent random strings q of length $b = O(\log n)$ for the perturbations at different recursive levels. Since at any particular level, with high probability, the output matrix is k-rich and does not depend on y, we can argue that with high probability, for a random y we obtain k-rich neighborhood matrices N_i of G_{i-1} at all levels.

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

3.3 The New Algorithm

We now formally describe how to implement the ideas presented in Section 3.2.3.

3.3.1 An algorithm for finding rich neighborhoods

We apply Nisan's pseudo-random generator for space-bounded computation [Nis92] to simulate short random walks on graphs, and construct an off-line randomized algorithm, which we call WALK for *pseudo-random walks*, that gives the following: given any graph G_{i-1}, WALK takes $O(\log^2 k)$ random bits, runs in processing space $O(\log n)$, and computes a neighborhood matrix N_i of G_{i-1} such that for each vertex v in G_{i-1}, with high probability v has at least k neighbors in N_i.

Lemma 3.3: Given as input a graph G of size n and an integer m, the algorithm WALK $(G, m; y')$ takes an off-line random input $y' \in \{0,1\}^r$ of length $|y'| = r = O(m^2)$, runs in space $O(\log n)$ and computes a Boolean neighborhood matrix $N_{y'}$ of G such that for each $1 \leq v \leq n$,

$$\Pr_{y' \in \{0,1\}^r}[\text{the } v\text{-th row of } N_{y'} \text{ is } 2^m\text{-rich}] \geq \frac{2}{3}$$

We prove this lemma in Section 3.5.

3.3.2 The Average is Also Rich

The significance of procedure WALK can be seen if we consider the expectation over y'

$$\mathcal{N}(G, m) = \mathbf{E}_{y'}[N_{y'}] = \mathbf{E}_{y' \in \{0,1\}^{O(m^2)}}[\text{WALK } (G, m; y')]$$

A straightforward averaging argument shows that $\mathcal{N}(G, m)$ is $(k, \frac{1}{n})$-rich:

Definition 3.3: [rich neighborhood matrix] A neighborhood matrix N of G is (k, α)-rich if any row of N is (k, α)-rich. A row i is (k, α)-rich if it contains at least $max\{k, \#CC(i)\}$ entries j with $N[i, j] > \alpha$, where $\#CC(i)$ is the number of vertices in the connected component of i in G.

Lemma 3.4: For any graph G of size n and any m, $\mathcal{N}(G, m)$ is a $(2^{m-1}, \frac{1}{n})$-rich neighborhood matrix of G.

Proof: If $\mathcal{N}(G, m)[i, j] > 0$ then there is at least one y' s.t. WALK $(G, m; y')[i, j] > 0$. Thus, i must be connected to j in G, and $\mathcal{N}(G, m)$ is a neighborhood matrix of G. Now we prove it is rich. Recall that

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

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

By the definition of w, we know that

$$\sum_{j\in[n]} \mathcal{N}(G,m)[i,j] \leq w \cdot 1 + (n-w) \cdot \frac{1}{n}$$

On the other hand,

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

where the inequality follows from Lemma 3.3, because for at least $\frac{2}{3}$ of the y''s, WALK $(G, m; y')[i, \cdot]$ contains at least 2^m entries with value 1. Thus $w \geq \frac{2}{3}2^m - 1 \geq 2^{m-1}$ and the proof is complete. □

Notice that by averaging we can allow ourselves having relaxed requirements from algorithm WALK ; instead of requiring that most y''s are "good" for every vertex v, it is sufficient to have that for every vertex v, most y''s are "good" (for different v's, different y''s are good).

3.3.3 The Approximated-Averaging Theorem

We saw that we can construct a randomized algorithm WALK such that its average output over the random bits is a rich neighborhood matrix. However, computing this average requires to try all possible random inputs, and to run WALK on each of them, which requires an extra space of the same length of the random input. Since we would like to use random inputs longer than $O(\log n)$, we cannot afford wasting so much space. We need to use a technique that allows us to compute (or approximate) the average without trying all possible random inputs to the algorithm. This technique, known as samplers [BGG93, Zuc96], uses only slightly more random bits, and with high probability gives a good approximation to the average of a function without using any additional significant amount of space.

Moreover, we wish to use the algorithm WALK in recursion, and in the same time, to reuse the same random bits on all levels. This however raises a dependency problem because the outcome of one level depends on its random bits, so these bits are "no longer random" for the next levels. We need to break this dependency by using the technique of Saks and Zhou [SZ95].

The following Theorem encapsulates the two features. We show in Theorem 3.5 how we can approximate the average of any algorithm (or function) such that our approximation is almost always constant which means that it does not depend on the random bits it gets. This can be done without any space penalty. We start with any randomized algorithm A that uses a random string y' of arbitrary length; we end up with an algorithm B that uses a random string y which is only 3 times longer than y' and an additional random string q which is very short - its length is the logarithm of the estimation error. The output of B is almost always a good approximation for the average of A, and more importantly, it almost only depends on q. It means that for most of the possible values of q, B produces a constant value (that we denote by $\mathcal{B}$) for most of the possible values for y.

Theorem 3.5: Let $\beta < 1$. Let A be an off-line randomized algorithm with random bit complexity $R(\cdot)$ and space complexity $S(\cdot)$; the output of A is in $K = [0,1]^d$ on which we define the L_∞ norm $\|\cdot\|_\infty$, (the dimension d of K may depend on the input to A but for simplicity of notations we denote it by d).

Define $\mathcal{A}(x) = \mathbf{E}_{y' \in \{0,1\}^{R(|x|)}}[A(x;y')]$ to be the average output of A on x when averaging on the off-line random inputs of A. Then there exists an algorithm B such that for every integer $b < R(\cdot)^\beta$, B has random bit complexity $3R(\cdot) + b$ which composed of a long string y of length $3R(\cdot)$ and a short string q of length b, and B has the following properties:

Define $\mathcal{B}(x;q)$ as the value of $B(x;y,q)$ that appears maximal number of times over the $y \in \{0,1\}^{3R(|x|)}$ (where $q \in \{0,1\}^b$), then:

- For every input x and for almost all of the random strings $q \in \{0,1\}^b$, the output of the algorithm $B(x;y,q)$ is constant for almost all random strings $y \in \{0,1\}^{R(|x|)}$.

Formally, for every input x, say that $q \in \{0,1\}^b$ is *good for x* if

$$\Pr_{y \in \{0,1\}^{3R(|x|)}}[B(x;y,q) = \mathcal{B}(x;q)] > 1 - \frac{d}{2^{R(|x|)}}$$

i.e., the output of $B(x;y,q)$ is almost constant over all the possible values of the random string y. Then

$$\forall x, \quad \Pr_{q \in \{0,1\}^b}[q \text{ is good for } x] > 1 - \frac{2d}{2^b}$$

- For every input x and for every random string $q \in \{0,1\}^b$, the "constant" value $\mathcal{B}(x;q)$ is very close to the average $\mathcal{A}(x)$.

  Formally, $\forall x, \forall q \in \{0,1\}^b, \quad \|\mathcal{B}(x;q) - \mathcal{A}(x)\|_\infty < 2^{-b}$

- B has space complexity $O(S(\cdot) + b) + \operatorname{poly}\log R(\cdot)$

We prove this Theorem in Section 3.4.

3.3.4 The Repeated Averaging Lemma

Given a recursive matrix algorithm we show that we can approximate its average in all levels of the recursion using only one off-line random string (instead of using one random string for each different level).

Lemma 3.6: Let $\beta < 1$. Let $K = [0,1]^d$ be a space with L_∞ norm $\|\cdot\|_\infty$. Let A_i for $i \in [p]$, be a sequence of off-line randomized algorithms with input and output in K, space complexity $S(\cdot)$ and off-line randomized complexity $R(\cdot)$. Define

$$\mathcal{A}_i(x) = \mathbf{E}_{y' \in \{0,1\}^{R(|x|)}} A_i(x;y')$$

Then for every integers $b < R(|x|)^\beta$ there exists a randomized algorithm AVER with input from K that outputs a sequence of p elements in K such that

- For every $x_0 \in K$, for all but $p(d2^{-R(|x_0|)} + 2d2^{-b})$ fraction of the $(y,q) \in \{0,1\}^{3R(|x_0|)} \times (\{0,1\}^b)^p$, AVER $(x_0;y,q) = (x_1,\ldots,x_p)$ s.t. for every $i = 1,\ldots,p$

$$\|x_i - \mathcal{A}_i(x_{i-1})\|_\infty \leq 2^{-b}$$

- AVER has $O(p(S(|x_0|) + b) + \operatorname{poly}\log R(|x_0|))$ space complexity and $O(R(|x_0|) + pb)$ off-line randomized complexity. *

Proof: We first define the algorithm AVER :

ALGORITHM 3.3.1 (AVER) *: On input $x_0 \in K$, and parameter b, toss off-line random input $y \in \{0,1\}^{3R(d)}$ and $\vec{q} = (q_1, \ldots, q_p) \in (\{0,1\}^b)^p$. The algorithm outputs the sequence $[x_i]_p$, where*

$$x_{i+1} = B_i(x_i; y, q_i)$$

for $i = 1, \ldots, p$ where B_i is the algorithm that approximates the average of A_i by Theorem 3.5.

Notice that we use the same y in all different levels. The off-line random bit complexity of the algorithm is immediate. The space complexity is also simple. By Proposition 3.1 the space complexity is the sum of the space complexities of all the different recursive levels, and at each such level the space complexity is at most $O(S(|x_0|) + b) + \operatorname{poly}\log R(|x_0|)$. We are left to prove that:

$$\|x_i - \mathcal{A}_i(x_{i-1})\|_\infty \leq 2^{-b}$$

for $i = 1, \ldots, p$.

For every $\vec{q} = (q_1, \ldots, q_p)$, the sequence we see is some $x_1, \ldots, x_p$. The sequence we would like to see is $z_0 = x_0$ and $z_i = \mathcal{B}_i(z_{i-1}, q_i)$ for $i = 1, \ldots p$. Notice that the sequence z is independent of y. We have:

$$\begin{aligned}
\Pr_{y,\vec{q}}\ [\exists i \in [p],\ x_i \neq z_i] &= \Pr_{y,\vec{q}}\ [\bigcup_{i=1}^{p} (\ x_i \neq z_i \wedge [x]_{i-1} = [z]_{i-1})] \\
&= \Sigma_{i=1}^{p}\ \Pr_{y,\vec{q}}[B_i(z_{i-1}; y, q_i) \neq \mathcal{B}_i(z_{i-1}; q_i) \wedge [x]_{i-1} = [z]_{i-1}] \\
&\leq \Sigma_{i=1}^{p}\ \Pr_{y,\vec{q}}[B_i(z_{i-1}; y, q_i) \neq \mathcal{B}_i(z_{i-1}; q_i)]
\end{aligned}$$

But since y and q_i are independent of z_{i-1} it follows by Theorem 3.5 that $\Pr_{y,\vec{q}}\ [\exists i \in [p],\ x_i \neq z_i] \leq p\cdot(d2^{-R(|x_0|)} + 2d2^{-b})$. Hence, with probability at least $1 - p\cdot(d2^{-R(|x_0|)} + 2d2^{-b})$

*We repeat here, at the risk of boring the reader, that in the off-line randomized complexity, p - the number of recursive levels - multiplies only the b, and not the original $R(\cdot)$.

over y and $\vec{q}$ the sequence $x_1, \ldots, x_p$ is exactly the sequence $z_1, \ldots, z_p$. Since $z_i = \mathcal{B}_i(z_{i-1}, q_i)$ we know by Theorem 3.5 that $\|z_i - \mathcal{A}_i(z_{i-1})\|_\infty \leq 2^{-b}$. In particular $\|x_i - \mathcal{A}_i(x_{i-1})\|_\infty \leq 2^{-b}$ for any $i = 1, \ldots, p$. $\square$

3.3.5 The Connectivity Algorithm

We show that in our case getting a sequence that approximates the average is enough for solving the whole problem. Thus, we can use Lemma 3.6 and save off-line random bits. We follow the [NSW92] scheme. In each step we have a graph G_i represented by a Boolean adjacency matrix, and a real neighborhood matrix N_i of it. So we let the L_∞ domain K be the set $\{(G, N)\}$ of all $n \times n$ matrices G, N s.t. G is a Boolean adjacency matrix of an undirected graph, and N is a real neighborhood matrix of G.

If we have at a certain step a graph G_i and a neighborhood matrix N_i of it, then we first shrink G_i according to N_i and then we compute a new neighborhood matrix N_{i+1}. Thus for $x_i = (G_i, N_i)$ we let $A_i(x_i; y')$ be (G_{i+1}, N_{i+1}) where $G_{i+1} =$ SRNK (G_i, s, t, N_i) and $N_{i+1} =$ WALK $(G_{i+1}, m; y')$.

Next we consider the average $\mathcal{A}_i(x) = \mathbf{E}_{y' \in \{0,1\}^{R(|x|)}}[A_i(x; y')]$. On input (G_i, N_i) we have $\mathcal{A}_i(G_i, N_i) = (G_{i+1}, \mathcal{N}_{i+1})$ where $G_{i+1} =$ SRNK (G_i, s, t, N_i) as before, and $\mathcal{N}_{i+1} = \mathbf{E}_{y'}[\text{WALK } (G_{i+1}, m; y')]$ is the average neighborhood matrix. The interesting thing is that we do not really need the exact average, approximating it is also enough:

Lemma 3.7: Let $x_0 = (G, I) \in K$ be an input, where G is an undirected graph with two distinguished vertices s and t, and I is the identity neighborhood matrix. Let $p = \Theta(\frac{\log n}{m})$. For any sequence $x_1, \ldots, x_p \in K$ s.t. $\|x_i - \mathcal{A}_i(x_{i-1})\|_\infty < \frac{1}{n}$, x_p represents a graph with only two vertices s and t, and they are connected in x_p iff they are connected in G.

Proof: We first prove by induction on i that x_i has the same connectivity properties as x_0. The base case $i = 0$ is trivial. Assume for i and let us prove for $i + 1$.

We start with $x_i = (\mathcal{G}'_i, \mathcal{N}'_i) \in K$ that by induction has the same connectivity properties as x_0. Denote $\mathcal{A}_{i+1}(x_i) = (\mathcal{G}_{i+1}, \mathcal{N}_{i+1})$, and $x_{i+1} = (\mathcal{G}'_{i+1}, \mathcal{N}'_{i+1})$.

First we notice that since $\mathcal{N}_i'$ is a neighborhood matrix of $\mathcal{G}_i'$, then $\mathcal{G}_{i+1}$ has also the same connectivity properties as x_0. Next we notice that since $\|x_{i+1} - \mathcal{A}_{i+1}(x_i)\|_\infty < 1$, and $\mathcal{G}_{i+1}, \mathcal{G}_{i+1}'$ are Boolean matrices, it follows that $\mathcal{G}_{i+1} = \mathcal{G}_{i+1}'$. Therefore, $\mathcal{G}_{i+1}'$ also has the same connectivity properties as x_0. Which completes the induction.

The second thing we show is that the graphs $\mathcal{G}_{i+1}'$ represents a shrink by a factor of 2^{m-1} of $\mathcal{G}_i'$. To see this look at $\mathcal{A}_i(x_{i-1}) = (\mathcal{G}_i, \mathcal{N}_i)$. By Lemma 3.4, $\mathcal{N}_i$ is a $(2^{m-1}, \frac{1}{n})$ rich neighborhood matrix of $\mathcal{G}_i$. Since $\|x_i - \mathcal{A}_i(x_{i-1})\|_\infty = \delta < \frac{1}{n}$ it follows that $\mathcal{N}_i'$ is a $(2^{m-1}, \frac{1}{n} - \delta)$ rich neighborhood matrix of $\mathcal{G}_i' = \mathcal{G}_i$. Hence, $\mathcal{G}_{i+1}$ shrinks by a factor of at least 2^{m-1}.

Hence, taking $p = \Theta(\frac{\log n}{m})$ we end up with a graph with only two vertices s and t that are connected iff they are connected in x_0. $\square$

Now we can use Lemma 3.6 by choosing parameters $\beta = \frac{5}{6}$,$d = n^2$,$b = 2\log n$, $m = \log^{2/3} n$ and $p = \Theta(\frac{\log n}{m}) = \Theta(\log^{1/3} n)$. We get an algorithm AVER that given an undirected graph G, and two distinguished vertices s and t, computes with probability at least $1 - 1/n$ a different graph with only two vertices s, t that are connected iff s is connected to t in G, i.e. it solves the s, t connectivity problem. Furthermore, AVER has $O(p \cdot \log n) = O(\log^{4/3} n)$ space complexity and $O(m^2) = O(\log^{4/3} n)$ off-line randomized complexity. We get Theorem 1.3.

3.4 The Proof of the Approximated-Averaging Theorem

In this section we prove Theorem 3.5. Suppose we are given a randomized algorithm A that outputs a value in $[0, 1]$. Using *samplers* we can *estimate* the *expected value* of A with polynomial accuracy. Clearly, there is no way we can avoid the error in the estimate and get perfect accuracy. Thus, the sampler's answer depends on the random bits it uses.

We would like to break this dependency. A natural way to do this, suggested by [SZ95], is by truncating the approximated value to the desired accuracy level. One can then hope that the resulting (truncated) value is *exactly* the truncation of the correct value. Indeed, this is almost true. To make it true with high probability [SZ95] use first random perturbations. We use this technique to prove Theorem 3.5.

3.4.1 Deterministic Sampling

Given a function $f : \{0,1\}^r \to [0,1]$ we want to efficiently approximate the expectation over a randomly chosen $y' \in \{0,1\}^r$ of $f(y')$, i.e., to estimate

$$\mathbf{E}[f] \;=\; \mathbf{E}_{y' \in \{0,1\}^r}[f(y')] \;=\; \frac{1}{2^r}\Sigma_{y' \in \{0,1\}^r} f(y')$$

to a given accuracy, minimizing the number of random bits used.

Definition 3.4: [BR94, Zuc96] An (ϵ, γ) *non-adaptive sampler* SMPL $_f : \{0,1\}^l \to [0,1]$ is a deterministic algorithm that on an input l-bit string y, computes p queries to an oracle function $f : \{0,1\}^r \to [0,1]$, and based on the answers outputs a value v s.t.

$$\Pr_{y \in \{0,1\}^l}[|v - \mathbf{E}[f]| > \epsilon] \leq \gamma$$

In other words, for all but γ fraction of the $y \in \{0,1\}^l$ $|v - \mathbf{E}[f]| \leq \epsilon$.

Clearly a natural way to obtain an (ϵ, γ) non-adaptive sampler is to randomly choose independent sample points. In fact, a simple counting argument shows that choosing $p = O(\frac{1}{\epsilon^2}\log(\frac{1}{\gamma}))$ points is enough. However, this way we use too many random bits: $l = pr$. We would like to reduce l to only $O(r)$ while keeping the number of sample points small with $p = poly(r, \frac{1}{\epsilon})$. Two optimal methods for doing that are known, [BGG93] and [Zuc96].

The space complexity of a non-adaptive sampler is the processing space needed to compute the queries plus the processing space needed to compute the output given the answers to the queries. We remind the reader that the processing space complexity can be much smaller than the size of the output itself. We use a special case of [Zuc96, Theorem 5.5] by setting $m = r, d = p, \alpha = \frac{1}{2}$ and $\gamma = \frac{1}{2^r}$ to prove Lemma 3.8:

Lemma 3.8: [Zuc96] For any constant $\beta < 1$, any positive integer r and any function $f : \{0,1\}^r \to [0,1]$ computable in space s and any $\epsilon > 2^{-r^\beta}$, there is an $(\epsilon, \frac{1}{2^r})$ sampler SMPL $_f : \{0,1\}^{3r} \to [0,1]$ computable in space $s + \log^{O(1)} r$.

We extend this Lemma to $f : \{0,1\}^r \to K$ (where $K = [0,1]^d$ with the L_∞ norm $\|\cdot\|_\infty$). We first need to extend the definition of the sampler to work on norm spaces.

Definition 3.5: An (ϵ, γ) *non-adaptive sampler* SMPL $_f : \{0,1\}^l \to K$ is a deterministic algorithm that on an input l-bit string y, computes p queries to an oracle function $f : \{0,1\}^r \to K$, and based on the answers outputs a value $v \in K$ such that

$$\Pr_{y\in\{0,1\}^l}[\|v - \mathbf{E}[f]\|_\infty > \epsilon] \leq \gamma$$

Lemma 3.9: For any constant $\beta < 1$, any positive integer r, any function $f : \{0,1\}^r \to K$ computable in space s and any $a < r^\beta$, there is a $(2^{-a}, \frac{d}{2^r})$ sampler SMPL $_f : \{0,1\}^{3r} \to K$ computable in space $O(s) + \log^{O(1)} r$.

Proof: Remember that $K = [0,1]^d$. For every $j \in [d]$ let f_j be the jth coordinate of f, let SMPL $_{f_j}$ be the $(2^{-a}, 2^{-r})$ sampler of Lemma 3.8 and define

$$\text{SMPL}_f = (\text{SMPL}_{f_1}, \ldots, \text{SMPL}_{f_d})$$

Let v_j be the jth entry of v. Then,

$$\begin{aligned}\Pr_{y\in\{0,1\}^{3r}}[\|v - \mathbf{E}[f]\|_\infty > 2^{-a}] &= \Pr_{y\in\{0,1\}^{3r}}[\vee_{j\in[d]}|v_j - \mathbf{E}[f_j]| > 2^{-a}] \\ &\leq \sum_{j\in[d]} \Pr_{y\in\{0,1\}^{3r}}[|v_j - \mathbf{E}[f_j]| > 2^{-a}] \leq d2^{-r}\end{aligned}$$

By Lemma 3.8, for every $j \in [d]$, the sampler SMPL $_{f_j}$ runs in space $s + \log^{O(1)} r$ (since f_j is definitely computable in space s) and while computing SMPL $_f$ we need only extra $O(\log d)$ space to point to the jth entry in the output of f, but $s \geq \log d$ since the output is of length at least d. $\square$

3.4.2 Exact Truncated Sampling

Now we construct the off-line randomized algorithm B that approximates the average of A in a way that (w.h.p.) does not depend on the random bits we use. The construction uses the perturbation and truncation scheme of [SZ95]. We start with a definition:

Definition 3.6: [Perturbations and Truncations]: Let x be a real value. The *perturbed* value of x, $x_{-\delta}$, is defined to be $\max\{x-\delta, 0\}$. The *perturbed and truncated* value of x, $\lfloor x_{-\delta} \rfloor_t$, is obtained by truncating the binary expansion of $x_{-\delta}$ after t binary digits. These operators are extended to vectors in $K = [0,1]^d$ ($v_{-\delta}$ and $\lfloor v_{-\delta} \rfloor_t$) by simply applying them entry by entry to the vector v.

We set $a = 2b + 1$ and set a truncation parameter $t = a - b$. For every input x, let $y \in \{0,1\}^{3R(|x|)}$ and $q \in \{0,1\}^b$ and let $\delta = q2^{-a}$. Define B as

$$B(x; y, q) = \lfloor \text{SMPL }_{A(x,\cdot)}(y)_{-\delta} \rfloor_t$$

and we prove:

Lemma 3.10: For every input x, if $a = 2b + 1 < \mathrm{R}(|x|)^\beta$ for some constant $\beta < 1$, then

$$\Pr_{q\in\{0,1\}^b}[\Pr_{y\in\{0,1\}^{3R(|x|)}}[B(x; y, q) = \lfloor \mathcal{A}(x)_{-\delta} \rfloor_t] > 1 - d2^{-r}] > 1 - 2d2^{-b}$$

Proof: A nonnegative real number x is said to be (a, t)-*dangerous* for positive integers $a > t$ if x can be written in the form $2^{-t}I + \rho$ where I is a positive integer and $\rho \in [-2^{-a}, 2^{-a})$, and is said to be (a, t)-*safe* otherwise. E.g., $0.76 = 3/4 + 0.01$ is $(6, 2)$-dangerous and $(7, 2)$-safe. A vector $v \in K$ is (a, t)-*dangerous* if one of its entries is (a, t)-dangerous and it is (a, t)-*safe* if all its entries are (a, t)-safe.

For two positive integers $a \geq b$, we define $\Delta(a, b)$ to be the set of 2^b real numbers $\{q2^{-a} \mid \text{integer } q \in [0, 2^b - 1]\}$. The proofs of the next two lemmas can be found in [SZ95].

Lemma 3.11: [SZ95, Lemma 5.5] Suppose $v \in K$ and $a \geq b$ are positive integers. Let $t = a - b$. Then,

$$\Pr_{\delta\in\Delta(a,b)}[v_{-\delta} \text{ is } (a, t)\text{-dangerous}] \leq 2d2^{-b}$$

Lemma 3.12: [SZ95, Lemma 5.2] Suppose $v \in K$ is (a,t)-safe. Then for any $v' \in K$ such that $\|v - v'\|_\infty \le 2^{-a}$ it holds that $\lfloor v \rfloor_t = \lfloor v' \rfloor_t$.

Thus, by Lemma 3.11 for all but $2d2^{-b}$ fraction of the choices of δ, $\mathcal{A}(x)_{-\delta}$ is (a,t)-safe. Also, by Lemma 3.9, for all but $\frac{d}{2^{R(|x|)}}$ fraction of the choices of y, SMPL $_{A(x,\cdot)}(y)$ is within 2^{-a} to the real average $\mathcal{A}(x)$. By Lemma 3.12 whenever one vector is (a,t)-safe and a second vector is within 2^{-a} distance of the first, then their truncation at position t is equal. This completes the proof of Lemma 3.10. □

Note now that if $\Pr_{y \in \{0,1\}^{3R(|x|)}}[B(x;y,q) = \lfloor \mathcal{A}(x)_{-\delta} \rfloor_t] > 1 - d2^{-r}$, that means that $\lfloor \mathcal{A}(x)_{-\delta} \rfloor_t$ is the value of $B(x;y,q)$ that appears maximal number of times over the random input $y \in \{0,1\}^{3R(|x|)}$, which implies that $\mathcal{B}(x;q) = \lfloor \mathcal{A}(x)_{-\delta} \rfloor_t$, which in turn implies that q is good for x. Then, by Lemma 3.10

$$\Pr_{q \in \{0,1\}^b}[q \text{ is good for } x] > 1 - 2d2^{-b}$$

which proves the first item of Theorem 3.5. To complete the proof of the Theorem one should note that $\|\lfloor \mathcal{A}(x)_{-\delta} \rfloor_t - \mathcal{A}(x)\|_\infty \le 2 \cdot 2^{-t}$ because each of the perturbation and truncation operation introduces an error of at most 2^{-t} which proves the second item of the Theorem, and since B is simply applying perturbation and truncation to the sampler that runs in space $O(s) + \log^{O(1)} r$, everything can be computed in space $O(s+b) + \operatorname{poly}\log R(|x|)$, which completes the proof of Theorem 3.5.

3.5 Pseudo-random Walks

In this section we present algorithm WALK and prove Lemma 3.3. The algorithm receives a graph G and a number m as inputs. Since it is a matrix algorithm, it receives two more indices, s, u, and it should output the value of the (s,u)th entry of the output matrix. Let us denote $k = 2^m$. We start with the following Lemma of Barnes and Feige [BF93].

Lemma 3.13: Suppose G is a connected graph and s is an arbitrary vertex in G. Let k be an integer. Then with probability at least 3/4, a random walk of length $l = l(k) = O(k^3)$ on G from s visits at least k distinct vertices or the whole connected component of s.

For the rest of this section, let us fix the graph G, the starting vertex (the row of the output matrix) s, the parameter m, the number of vertices that we should visit $k = 2^m$ and the length $l = O(k^3)$ given by Lemma 3.13. We will analyze random walks on G, that start from s. Let us start with the intuition behind the formal definitions. We wish to track random walks, starting at s, as long as they pass only through vertices with non-negligible reaching probability. W_τ will be the vertices with "high" reaching probability in the τth step, V_τ are the vertices with "low" reaching probability and $\rho_\tau(v)$ is the probability of reaching v in the τth step when walking only through vertices in W_j. To achieve that, we will make sure that it would be possible to reach W_τ or V_τ only from $W_{\tau-1}$. σ will be the number of steps that we want to analyze. It is at most l, but if at time $\tau < l$ the probability of entering $V_{\tau+1}$ is too large, or, we discover a vertex with at least k neighbors, then we stop. Let us give now the formal definitions. Let $P_{v,w}$ be the probability of a random step to move from v to w, i.e., if (v, w) is an edge of G then $P_{v,w}$ is $\frac{1}{deg(v)}$ and 0 otherwise. Now, define $\rho_0(s) = 1$ and $\forall v \neq s$, $\rho_0(v) = 0$ and for every $\tau \geq 0$ define

$$\begin{aligned}
W_\tau &= \{v : \rho_\tau(v) \geq \frac{1}{2lk}\} \\
V_\tau &= \{v : 0 < \rho_\tau(v) < \frac{1}{2lk}\} \\
\rho_{\tau+1}(v) &= \sum_{w \in W_\tau} \rho_\tau(w) \cdot P_{w,v}
\end{aligned}$$

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

$$\sigma = \min\{l\} \cup \{\tau : \exists v \in W_\tau, |\Gamma(v)| \geq k\} \cup \{\tau : \sum_{v \in V_{\tau+1}} \rho_{\tau+1}(v) > \frac{1}{2l}\}$$

Denote by $W = \bigcup_{\tau=0}^{\sigma} W_\tau$ and get the following Lemma:

Lemma 3.14: $|\Gamma(W)| \geq k$

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

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

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

The set of states of the automaton M is

$$S = \{(\tau, u) : 0 \leq \tau \leq \sigma, u \in W_\tau \cup \{0\}\ \}$$

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

Lemma 3.15: For every $|S| = O(2^{7m})$ and $l = O(2^{3m})$ there exists a $\nu = O(m)$ and a pseudo-random generator $F : \{0,1\}^\nu \times (\{0,1\}^{2\nu})^{\log l} \to (\{0,1\}^\nu)^l$ such that for every automaton Q of size $|S|$, for all but k^{-5} of the choices of $h \in (\{0,1\}^{2\nu})^{\log l}$, the distance between the distribution over the states of $Q(c)$ where $c \in (\{0,1\}^\nu)^l$ is random and $Q(F(x,h))$ where $x \in \{0,1\}^\nu$ is random, is at most k^{-5}.

We use the parameter ν of the generator ($\nu = O(m) = O(\log k)$) to define the alphabet of our automaton as $\{0,1\}^\nu$. Now, it's time to define the transition function of the automaton. For every τ, $0 \leq \tau < \sigma$, and for every $v \in W_\tau$, we look at input strings $\pi \in \{0,1\}^\nu$. Say that a random walk from v that takes its decision according to π takes us to a vertex w. If w is in $W_{\tau+1}$, then the transition function of (τ, v) and π is $(\tau+1, w)$; otherwise, it is $(\tau+1, 0)$

(note that since the degree of every vertex in W_τ is at most 2^ν, we do not face a situation in which we need more than ν bits to take a step from v). For all other states in S, the transition function on every input is a self loop. The starting state of our automaton is of course $(0, s)$.

Now, one should note that by the construction of the automaton M, for every string $\pi \in (\{0,1\}^\nu)^\tau$ where $\tau \leq \sigma$, if π takes M to some state (τ, v) with $v \neq 0$, then a walk on G according to π will end up at vertex v as well. Moreover, for every $\tau \leq \sigma$ and for every $v \in W_\tau$, the probability over $\pi \in (\{0,1\}^\nu)^\tau$ that a walk on M according to π reaches (τ, v) is at least $\rho_\tau(v) - k^{-5}$.

Consider now the pseudo-random generator F of Lemma 3.15. For all but k^{-5} of the choices of $h \in (\{0,1\}^{2\nu})^{\log l}$, for every $\tau \leq \sigma$ and for every $v \in W_\tau$,

$$\Pr_{x \in \{0,1\}^\nu}[M(F(x,h) \text{ passes through } (\tau, v)] \geq \rho_\tau(v) - 2k^{-5}$$

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

Having all this knowledge at our disposal, we can define the algorithm WALK and finish the proof of Lemma 3.3.

Algorithm WALK

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

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

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

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

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

The last claim of Lemma 3.3 is that for every s, for at least $\frac{2}{3}$ of the choices of h, the algorithm will reply 1 for at least k different u's or for the whole connected component of s if it's smaller than k. If the connected component of s is indeed smaller than k, then the walks according to the output of F will traverse it completely for almost every h since F is a pseudo-random generator for automata of such size. In the other case where the connected component of s is at least of size k, then we know that for all but k^{-5} of the possible h's, the probability over the x's that a walk on G according to $F(x, h)$ passes through a vertex v in W is positive. Thus, for every vertex $v \in W$, the algorithm WALK will traverse v. But, by Lemma 3.14, $|\Gamma(W)| \geq k$, and WALK outputs 1 for every u that it either traverses or a neighbor of a traversed vertex v, thus WALK will output 1 for every vertex of $\Gamma(W)$. This concludes the proof of Lemma 3.3. □

Chapter 4

Combinatorial Rectangles

In a general discrepancy problem, we are given a family of sets and want to construct a small sample space that approximates the volume of an arbitrary set in the family. This problem is closely related to other important issues in combinatorial constructions such as the problem of constructing small sample spaces that approximate the independent distributions on many multi-valued random variables [KW84, Lub86, ABI86, CG89, NN93, AGHP92, EGL$^+$92, Sch92, KM93, KK94], and the problem of constructing pseudo-random generators for space bounded computation [Nis92, NZ93, INW94].

More precisely, one can define the following notion of a discrepancy set: Let U be a set and $\mathcal{F}$ be a family of subsets of U. A multiset S of U is said to be an ϵ-discrepancy set for $\mathcal{F}$ if for each element $A \in \mathcal{F}$, the difference between the fraction of points in U that belong to A and the fraction of points in S that belong to A is within ϵ. In this chapter, based on the paper [ASWZ96], motivated by the issue of the construction of space bounded pseudo-random generators, we investigate the problem of constructing small discrepancy sets for a special class of sets called combinatorial rectangles: for positive integers m and d, a *combinatorial rectangle* of type (m, d) (or an (m, d)-*rectangle*) is a subset of $[m]^d$ of the form $R = R_1 \times R_2 \times \ldots \times R_d$, where each $R_i \subseteq [m]$. We use $\mathcal{R}(m, d)$ to denote the family of all (m, d)-rectangles.

It is easy to show that if we select $O(md/\epsilon^2)$ points from $[m]^d$ uniformly at random, then with probability 1/2, the resulting set is an ϵ-discrepancy set for $\mathcal{R}(m, d)$. On the other hand, the problem of finding an explicit construction for such a set of size polynomial in m, d and ϵ^{-1}, is still open.

The discrepancy set problem for rectangles was first studied in the context of number theory and real analysis, where the family of sets considered was the family of geometric rectangles in which each "side" R_i of the rectangle is an interval. [BC87] and [Nie] are good references for this material, which contain sharp existential bounds mainly for small dimensions. The general problem of explicit constructions in high dimensions, and for combinatorial rather than geometric rectangles, was first formulated in [EGL$^+$92]. Their motivation was approximating independent multivalued distributions. They gave a poly size construction for the geometric case, and two quasi-polynomial constructions for the general case: of size $(md/\epsilon)^{\log d}$ (based on Nisan's bounded-space generator [Nis92]) and size $(md/\epsilon)^{\log 1/\epsilon}$ (based on k-wise independence). These become polynomial size (respectively) if either the dimension d or the error ϵ are bounded.

Another source of explicit constructions of small sample spaces for this problem comes from observing that (non)membership in a rectangle can be checked by a DNF formula of size $O(md)$. Thus small sample spaces which approximate such circuits are good discrepancy sets with the same error. Nisan's constant-depth generator [Nis91] (and the improvements in [LVW93]) again give (completely different) quasi-polynomial constructions.

Our main result is an explicit construction of a sample space for this problem of size polynomial in m, d and $\epsilon^{\log \epsilon}$. In other words it is polynomial as long as $1/\epsilon < exp(\sqrt{\log md})$. While reducing the error significantly below the constant bound that follows from previous constructions, it falls short of the natural goal of $1/\epsilon = (md)^{O(1)}$. Our construction is different than previous ones for the discrepancy problem, and follows rather closely the constructions for the related hitting set problem for rectangles (where all we need is that every rectangle of volume at least ϵ is hit by at least one point in the small sample space). Thus, the hitting set problem is the one-sided error version of the discrepancy problem. For this simpler problem a fully polynomial solution, namely an explicit set of size $(m \log d/\epsilon)^{O(1)}$, was given by [LLSZ97]. It is achieved by a sequence of "hitting-preserving" reductions, which we generalize in here to "discrepancy-preserving" reductions. These simplify and reduce (in turn) various parameters of the problem without affecting too much the volume of the sets and the size of the sample needed. Naturally, while in the (one-sided) hitting set problem it suffices to control a lower bound on the volume, we need to keep tight upper and lower bounds throughout these reductions.

Our description (and motivation) of the constructions comes from the perspective of constructing pseudo-random generators, which we explain below. Let l, m, d be positive integers. An (l, m, d)*-generator* G is a function that maps $\{0,1\}^l$ to $[m]^d$. G is said to ϵ*-fool* the (m, d)-rectangles if the multiset image of G in $[m]^d$, which is of size 2^l, is an ϵ-discrepancy set for $\mathcal{R}(m, d)$. Thus to construct a small discrepancy set for $\mathcal{R}(m, d)$, it suffices to construct an (l, m, d)-generator that ϵ-fools the (m, d)-rectangles with small l.

An (m, d)-rectangle can be visualized as a $(2, d, m)$ branching program (definition 2.3) in the natural way. Since in general any pseudo-random generator for non-uniform space bounded computation in the finite state machine model [Nis92] can fool $(m^{\Omega(1)}, d, m)$ branching programs, it fools all (m, d)-rectangles. In particular, Nisan's pseudo-random generator [Nis92] gives an (l, m, d)-generator with $l = \log d(2\log m + \log d + \log \epsilon^{-1})$ that ϵ-fools the (m, d)-rectangles, and the Impagliazzo-Nisan-Wigderson generator [INW94] gives one with $l = \log m + 2\log d(\log d + \log \epsilon^{-1})$. Both of these generators are *efficient* in the sense that they are computable in polynomial time (polynomial in the length of the output) and linear space (linear in the length of the input). Nevertheless, these fall short of the natural lower bound of $l = O(\log m + \log d + \log \epsilon^{-1})$. Our new construction, in this language, is an efficient generator with $l = O(\log m + \log d + \log^2 \epsilon^{-1})$. It is interesting to observe that achieving a bound $l = \log m + O(\log \epsilon^{-1}) + f(d)$ for an arbitrary function f will result in a $o((\log n)^2)$-bit generator which fools all $(O(1), d, m)$ branching programs - one of the main challenges in derandomizing space-bounded computation and a major motivation for our interest in the discrepancy problem.

The rest of this chapter is organized as follows. In Section 4.1, we provide basic notation and definitions; moreover, we formalize the reduction framework in terms of the compositions of function reductions which helps us to clarify certain subtleties in the generator construction. An overview of the construction is given in Section 4.2.1 and the details of the construction are given in the later sections.

4.1 Preliminaries

4.1.1 Basic Notation

For a set U, we let 2^U denote the family of subsets of U and let $\mathcal{M}(U)$ denote the family of multi-subsets of U. With respect to a fixed order of the elements in U, we identify each $S \in \mathcal{M}(U)$ with a nonnegative integer vector indexed by U such that for any element $u \in U$, the entry indexed by u is the number of appearances of u in S. For example, if $U = \{a, b, c\}$ whose elements are in alphabetical order and $S = \{a, c, c, c\} \in \mathcal{M}(U)$, then we have $U = (1, 1, 1)$ and $S = (1, 0, 3)$. Clearly, the inner product $< S, U >$ is the cardinality of S, which we denote by $|S|$.

Let U and V be sets. We say that a matrix is a $U \times V$ matrix if the rows and columns of the matrix are indexed by the elements of U and V, respectively.

For typographical simplicity, we will not specify whether a vector is a row or a column vector in the case that this is easily seen from the context.

All integers are positive unless otherwise specified.

4.1.2 Discrepancy Sets and Reductions

Let U be a set. For a subset $A \subseteq U$, the *volume* of A (in U), denoted $vol(A)$, is defined to be the fraction of elements in U that lie in A, i.e.,

$$vol(A) = vol^U(A) = \frac{< A, U >}{< U, U >}.$$

Let $S \in \mathcal{M}(U)$. The *discrepancy* of A with respect to S (in U) is defined to be

$$disc_S(A) = disc_S^U(A) = |\frac{< A, S >}{< U, S >} - vol(A)|.$$

For $\mathcal{A} \subseteq 2^U$, we define $disc_S(\mathcal{A}) = \max_{A \in \mathcal{A}} disc_S(A)$. We say that S is an *ϵ-discrepancy set* for $\mathcal{A}$ if $disc_S(\mathcal{A}) \leq \epsilon$.

Remark: Here we emphasize the facts that volume is defined only on sets but not on multisets, while discrepancy is defined only on sets but with respect to multisets.

Let U and V be sets. A *reduction* Λ *between* U *and* V is a $U \times V$ nonnegative integral matrix. The *cost* of the reduction, denoted $cost(\Lambda)$, is defined to be the maximum column sum of the matrix; the *image* of the reduction, denoted $image(\Lambda)$, is defined to be ΛV. (Here we emphasize that $image(\Lambda)$ is a multiset.) Clearly, $|image(\Lambda)| \leq cost(\Lambda)|V|$. It is often convenient for us to view such a reduction as a bipartite multigraph on U and V such that there are k edges connecting vertex $u \in U$ and vertex $v \in V$ if and only if the (u, v)-th entry of the reduction is k. The cost of the reduction is thus the maximum degree of any vertex in V, and the image of the reduction is the set of neighbors of V in U counting multiplicity.

Let Λ be a reduction between U and V. Then it is clear that for any $A \in \mathcal{M}(U)$ and any $B \in \mathcal{M}(V)$, we have $A\Lambda \in \mathcal{M}(V)$ and $\Lambda B \in \mathcal{M}(U)$.

Suppose $\mathcal{A} \subseteq 2^U$ and $\mathcal{B} \subseteq 2^V$. A reduction Λ between U and V is said to be $(\mathcal{A}, \mathcal{B}, \delta)$-*discrepancy preserving* if for any $S \in \mathcal{M}(V)$, $disc_{\Lambda S}(\mathcal{A}) \leq disc_S(\mathcal{B}) + \delta$. That is, S is an ϵ-discrepancy set for $\mathcal{B}$ implies that ΛS is an $(\epsilon + \delta)$-discrepancy set for $\mathcal{A}$. Therefore, intuitively, such a reduction reduces the problem of finding a discrepancy set for family $\mathcal{A}$ to the problem of finding a discrepancy set for family $\mathcal{B}$.

Proposition 4.1: Let U and V be sets and let $\mathcal{A} \subseteq 2^U$. For an arbitrary $\mathcal{B} \subseteq 2^V$, suppose Λ is a reduction between U and V that is $(\mathcal{A}, \mathcal{B}, \delta)$-discrepancy preserving, then $image(\Lambda)$ is a δ-discrepancy set for $\mathcal{A}$.

This is because $disc_V(B) = 0$ for any $B \subseteq V$.

4.1.3 Function Reductions

We will be dealing with reductions specified by function families, which we call *function reductions*. One remark on notation: in the case that a function family is a singleton set $\{f\}$, we may use f for simplicity.

Any function f that maps V to U specifies a reduction Λ_f between U and V in a natural way: for $u \in U$ and $v \in V$, $\Lambda_f(u,v) = 1$ if and only if $f(v) = u$. We note that for any $X \subseteq U$, $X\Lambda_f$ is the subset $f^{-1}(X)$ of V and in particular, we have $U\Lambda_f = V$. For a family $\mathcal{F}$ of functions mapping V to U, the function reduction $\Lambda_{\mathcal{F}}$ between U and V specified by $\mathcal{F}$ is defined to be the sum of Λ_f over $f \in \mathcal{F}$, i.e., $\Lambda_{\mathcal{F}} = \sum_{f\in\mathcal{F}} \Lambda_f$. The *image* of $\mathcal{F}$, $image(\mathcal{F})$, is defined to be $image(\Lambda_{\mathcal{F}}) = \Lambda_{\mathcal{F}}V$. It is clear that $cost(\Lambda_{\mathcal{F}}) = |\mathcal{F}|$ and $|image(\mathcal{F})| = |\mathcal{F}||V|$.

Let $\mathcal{A} \subseteq 2^U$ and $\mathcal{B} \subseteq 2^V$. $\mathcal{F}$ is said to be $(\mathcal{A},\mathcal{B},\delta)$-*good* if for each $A \in \mathcal{A}$ the following hold:

- for any $f \in \mathcal{F}$, $A\Lambda_f \in \mathcal{B}$, and
- $|E_{f\in\mathcal{F}}[vol(A\Lambda_f)] - vol(A)| \leq \delta$, where the expectation is over a randomly chosen $f \in \mathcal{F}$.

Lemma 4.2: Let U and V be sets. Suppose $\mathcal{F}$ is a family of functions mapping V to U. For $\mathcal{A} \subseteq 2^U$ and $\mathcal{B} \subseteq 2^V$, if $\mathcal{F}$ is $(\mathcal{A},\mathcal{B},\delta)$-good, then the function reduction $\Lambda_{\mathcal{F}}$ is $(\mathcal{A},\mathcal{B},\delta)$-discrepancy preserving and, consequently, $image(\mathcal{F})$ is a δ-discrepancy set for $\mathcal{A}$.

Proof: Fix any $S \in \mathcal{M}(V)$. We want to show that for any $A \in \mathcal{A}$, $disc_{\Lambda_{\mathcal{F}}S}(A) \leq disc_S(\mathcal{B}) + \delta$.

$$
\begin{aligned}
disc_{\Lambda_{\mathcal{F}}S}(A) &= \left|\frac{< A, \Lambda_{\mathcal{F}}S >}{< U, \Lambda_{\mathcal{F}}S >} - vol(A)\right| \\
&= \left|\frac{\sum_{f\in\mathcal{F}} < A\Lambda_f, S >}{\sum_{f\in\mathcal{F}} < U\Lambda_f, S >} - vol(A)\right| \\
&= \left|\frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}}\left(\frac{< A\Lambda_f, S >}{< V, S >} - vol(A)\right)\right| \\
&\leq \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}}\left|\frac{< A\Lambda_f, S >}{< V, S >} - vol(A\Lambda_f)\right| + \left|\frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}} vol(A\Lambda_f) - vol(A)\right| \\
&\leq \max_{B\in\mathcal{B}} disc_S(B) + |E_{f\in\mathcal{F}}[vol(A\Lambda_f)] - vol(A)| \\
&\leq disc_S(\mathcal{B}) + \delta.
\end{aligned}
$$

Now, Proposition 4.1 concludes the proof since $image(\mathcal{F})$ is $\Lambda_{\mathcal{F}} V$ by definition. $\square$

Lemma 4.2 suggests that in order to construct a small-sized ϵ-discrepancy set for $\mathcal{A} \subseteq 2^U$, it suffices to construct a family $\mathcal{F}$ of functions mapping V to U for some V that is $(\mathcal{A}, \mathcal{B}, \epsilon)$-good for some $\mathcal{B} \subseteq 2^V$, such that both $|\mathcal{F}|$ and $|V|$ are small (thus $|image(\mathcal{F})|$ is small).

4.1.4 The Composition of Families of Functions

Let $V_1, V_2, \ldots, V_k$ be sets, and let $\mathcal{F}_i, 1 \leq i \leq k-1$, be a sequence of families of functions with each $\mathcal{F}_i$ mapping V_{i+1} to V_i. The composition of $\mathcal{F}_i$, denoted $\mathcal{F}^{(k-1)} = \bigodot_{i=1}^{k-1} \mathcal{F}_i$, is defined to be the family of all functions of the form $f_1 \circ f_2 \circ \ldots \circ f_{k-1}$, where $f_i \in \mathcal{F}_i$ for each i and $\circ$ denotes the function composition. The following fact can be easily proved by induction:

Proposition 4.3: Let $V_1, V_2, \ldots, V_k$ be sets and for each $1 \leq i \leq k-1$, let $\mathcal{F}_i$ be a family of functions mapping V_{i+1} to V_i. Then the composition $\mathcal{F}^{(k-1)}$ is a family of functions mapping V_k to V_1 of size $\Pi_{i=1}^{k-1} |\mathcal{F}_i|$ such that $\Lambda_{\mathcal{F}^{(k-1)}} = \Pi_{i=1}^{k-1} \Lambda_{\mathcal{F}_i}$. In words, the function reduction specified by the composition is the product of the function reduction specified by each single family in the composition.

The next lemma will be useful for our generator construction.

Lemma 4.4: Let $V_1, V_2, \ldots, V_k$ be sets and let $\mathcal{A}_i \subseteq 2^{V_i}$ for $1 \leq i \leq k$. Suppose for each $1 \leq i \leq k-1$, $\mathcal{F}_i$ is a family of functions mapping V_{i+1} to V_i that is $(\mathcal{A}_i, \mathcal{A}_{i+1}, \delta_i)$-good. Then the composition $\mathcal{F}^{(k-1)}$ is $(\mathcal{A}_1, \mathcal{A}_k, \delta)$-good where $\delta = \sum_{i=1}^{k-1} \delta_i$.

Proof: We prove by induction on k that $\mathcal{F}^{(k-1)}$ is $(\mathcal{A}_1, \mathcal{A}_k, \delta)$-good. The case where $k = 2$ is trivial. Assume that it holds for $k-1$ and we show for k.

Fix any $A \in \mathcal{A}_1$. We first need to show that for any $f \in \mathcal{F}^{(k-1)}$, $A\Lambda_f \in \mathcal{A}_k$. Let $f = f_1 \circ f_2 \circ \ldots \circ f_{k-1} \in \mathcal{F}^{(k-1)}$ be arbitrary. Then it follows from Proposition 4.3 that $\Lambda_f = \Pi_{i=1}^{k-1} \Lambda_{f_i}$. By the induction hypothesis, we have $A\Pi_{i=1}^{k-2} \Lambda_{f_i} \in \mathcal{A}_{k-1}$. Since $\mathcal{F}_{k-1}$ is $(\mathcal{A}_{k-1}, \mathcal{A}_k, \delta_{k-1})$-good, by definition, $(A\Pi_{i=1}^{k-2} \Lambda_{f_i})\Lambda_{f_{k-1}} \in \mathcal{A}_k$. It remains to show that $|E_{f \in \mathcal{F}^{(k-1)}}[vol(A\Lambda_f)] -$

$vol(A)| \leq \delta$.

$$
\begin{aligned}
&|E_{f=f_1\circ\ldots\circ f_{k-1}\in\mathcal{F}^{(k-1)}}[vol(A\Lambda_f)] - vol(A)| = \\
= \quad & |E_{f_1\circ\ldots\circ f_{k-2}\in\mathcal{F}^{(k-2)}} E_{f_{k-1}\in\mathcal{F}_{k-1}}[vol(A\Pi_{i=1}^{k-2}\Lambda_{f_i}\Lambda_{f_{k-1}})] - vol(A)| \\
= \quad & |E_{f_1\circ\ldots\circ f_{k-2}\in\mathcal{F}^{(k-2)}}[\ (E_{f_{k-1}\in\mathcal{F}_{k-1}}[vol(A\Pi_{i=1}^{k-2}\Lambda_{f_i}\Lambda_{f_{k-1}})] - vol(A\Pi_{i=1}^{k-2}\Lambda_{f_i})) \\
& + (vol(A\Pi_{i=1}^{k-2}\Lambda_{f_i}) - vol(A))\ ]| \\
\leq \quad & E_{f_1\circ\ldots\circ f_{k-2}\in\mathcal{F}^{(k-2)}}[\ |(E_{f_{k-1}\in\mathcal{F}_{k-1}}[vol(A\Pi_{i=1}^{k-2}\Lambda_{f_i}\Lambda_{f_{k-1}})] - vol(A\Pi_{i=1}^{k-2}\Lambda_{f_i})|\ ] \\
& + |E_{f_1\circ\ldots\circ f_{k-2}\in\mathcal{F}^{(k-2)}}[vol(A\Pi_{i=1}^{k-2}\Lambda_{f_i})] - vol(A)| \\
\leq \quad & \delta_{k-1} + \sum_{i=1}^{k-2}\delta_i = \delta
\end{aligned}
$$

where the first term of the last inequality holds since $A\Pi_{i=1}^{k-2}\Lambda_{f_i} \in \mathcal{A}_{k-1}$ by induction, and $\mathcal{F}_{k-1}$ is $(\mathcal{A}_{k-1}, \mathcal{A}_k, \delta_{k-1})$-good; the second term holds because $\mathcal{F}^{(k-2)}$ is $(\mathcal{A}_1, \mathcal{A}_{k-2}, \sum_{i=1}^{k-2}\delta_i)$-good by induction, and $\Lambda_{f_1\circ\ldots\circ f_{k-2}} = \Pi_{i=1}^{k-2}\Lambda_i$. □

4.1.5 Efficiency in Function Computation

For the purposes of constructing pseudo-random generators, we review some facts about the efficiency in the computation of functions.

We say that a function f is in $TS(t(n), s(n))$ if on input of length n, f is computable in time $t^{O(1)}(n)$ and in space $O(s(n))$; and we say that a family of functions is in $TS(t(n), s(n))$ if each function in the family is so.

A family $\mathcal{F}$ of functions is said to be *indexable* if $\mathcal{F}$ can be *identified* with $[|\mathcal{F}|]$ in the sense that each function $f \in \mathcal{F}$ can be indexed by an integer in $[|\mathcal{F}|]$ (or a bit-sequence of length $\lceil \log|\mathcal{F}| \rceil$) so that if f is in $TS(t(n), s(n))$, then given its index, the computation of f can be simulated in $TS(t(n), s(n))$ as well.

It is not difficult to see the following:

Proposition 4.5: Let k be fixed and let $V_1, V_2, \ldots, V_k$ be sets. Suppose for each $1 \leq i \leq k-1$, $\mathcal{F}_i$ is a family of functions mapping V_{i+1} to V_i. If $\mathcal{F}_i$ is in $TS(t_i(n), s_i(n))$ for each

i, then $\bigodot_{i=1}^{k-1} \mathcal{F}_i$ is in $TS(\Pi_{i=1}^{k-1} t_i(n), \sum_{i=1}^{k-1} s_i(n))$. Moreover, if each $\mathcal{F}_i$ is indexable, so is $\bigodot_{i=1}^{k-1} \mathcal{F}_i$.

Given an indexable family $\mathcal{F}$ of functions mapping V to U, the *unification* of $\mathcal{F}$, denoted $G_{\mathcal{F}}$, is defined to be a function mapping $[|\mathcal{F}|] \times V$ to U such that: on input $(\alpha, v) \in [|\mathcal{F}|] \times V$, $G_{\mathcal{F}}$ takes the function $f \in \mathcal{F}$ indexed by α, simulates the computation of $f(v)$ and outputs the result. It follows immediately from the construction that:

Proposition 4.6: Let $\mathcal{F}$ be an indexable family of functions. Then $image(G_{\mathcal{F}}) = image(\mathcal{F})$, and if $\mathcal{F}$ is in $TS(t(n), s(n))$, so is $G_{\mathcal{F}}$.

4.1.6 k-wise Independent Hash Function Family

Let a, b be integers. A family H of functions mapping $[a]$ to $[b]$ is said to be a *k-wise independent hash function family* if for any $u_1, u_2, \ldots, u_k \in [a]$ such that $u_i \neq u_j$ for $1 \leq i < j \leq k$, and any $v_1, v_2, \ldots, v_k \in [b]$,

$$\Pr_{h \in H}[h(u_i) = v_i \text{ for } 1 \leq i \leq k] = 1/b^k.$$

(A pairwise independent hash function family is usually called a *universal hash function family* [CW79].)

It is easy to check that a k-wise independent hash function family is a $(k-1)$-wise independent hash function family. We will need the following well-known fact:

Theorem 4.7: Let k be fixed. Then for any a, b that are integer powers of 2, there is an explicit construction of a k-wise independent hash function family mapping $[a]$ to $[b]$ of size $(\max(a, b))^k$. Moreover, the family is indexable and is in $TS(\log ab, \log ab)$.

4.1.7 Combinatorial Rectangles, Discrepancy Sets and Pseudo-random Generators

For integers m and d, let $U = [m]^d$. A *combinatorial rectangle* of type (m, d) (or an *(m, d)-rectangle*) is a subset of U of the form $R = R_1 \times R_2 \times \ldots \times R_d$, where each $R_i \subseteq [m]$. By definition, the volume of R is thus

$$vol(R) = |R|/|U| = \Pi_{i=1}^{d} |R_i|/m^d.$$

R is said to be PIP, which stands for *pairwise independent projections*, if for any $1 \leq i < j \leq d$, $\frac{|R_i \cap R_j|}{m} = \frac{|R_i|}{m} \cdot \frac{|R_j|}{m}$. We use $\mathcal{R}(m, d)$ to denote the family of all (m, d)-rectangles, and we use PIP-$\mathcal{R}(m, d)$ to denote the family of all PIP (m, d)-rectangles.

Let l, m, d be positive integers. An *(l, m, d)-generator* G is a function that maps $\{0,1\}^l$ to $[m]^d$. (Note that the output length of G is $d\lceil \log m \rceil$.) We call l the *input-length* of G. G is said to *ϵ-fool* the (m, d)-rectangles if G is $(\mathcal{R}(m, d), 2^{\{0,1\}^l}, \epsilon)$-good, or equivalently, $image(G)$ is an ϵ-discrepancy set for $\mathcal{R}(m, d)$; we call such a generator G *efficient* if it is in $TS(d \log m/\epsilon, l)$, i.e., if it is computable in time polynomial in the length of the output over ϵ, and in space linear to the length of the input.

We can see that to efficiently construct a small-sized ϵ-discrepancy set for $\mathcal{R}(m, d)$, where by *efficient construction* we mean that the construction time is polynomial in the size of the output set, it suffices to construct an efficient (l, m, d)-generator that ϵ-fools the (m, d)-rectangles with small l. In the next section we will present such a construction.

4.1.8 The INW Generator for Path Networks

Our construction will make use of a pseudo-random generator introduced in [INW94] which applies to the following communication model.

Suppose there are d processors $p_1, \ldots, p_d$ connected by a path. Each processor p_i receives an input $x_i \in [m]$. They then follow some communication protocol Π in which each processor can send messages to adjacent processors (where the protocol specifies the messages sent by each p_i depending on its input x_i and the messages it has received so far). Eventually the protocol terminates with processor p_d either in an "accept" or a "reject" state. We will call such a protocol an (m, d)-protocol. The *accepting set* $ACC(\Pi)$ of the protocol is the set of inputs $(x_1, \ldots, x_d) \in [m]^d$ which cause p_d to accept. The *complexity* of the protocol is the maximum over all the inputs $(x_1, \ldots, x_d)$ and processors p_i of the number of bits sent by p_i

on input $(x_1, \ldots, x_d)$.

An (l, m, d)-generator G is said to ϵ-fool an (m, d)-protocol Π if

$$|Pr_{y \in \{0,1\}^l}[G(y) \in ACC(\Pi)] - \frac{|ACC(\Pi)|}{m^d}| \leq \epsilon.$$

The following theorem is a restatement of a result in [INW94].

Theorem 4.8: For each positive integer m, d, c and any $0 < \epsilon \leq 1$, there is an explicit construction of an efficient (l, m, d)-generator that ϵ-fools all (m, d)-protocols of complexity at most c with $l = O(\log m + \log d(c + \log d + \log \epsilon^{-1}))$.

4.2 The Generator Construction

In this section we present the construction of our pseudo-random generator for combinatorial rectangles and prove the following:

Theorem 4.9: Let m, d be positive integers and let $0 < \epsilon \leq 1$. Then for some $l = O(\log m + \log d + \log^2 \epsilon^{-1})$, there is an explicit construction of an efficient (l, m, d)-generator that ϵ-fools the (m, d)-rectangles. Consequently, there is an efficient construction of an ϵ-discrepancy set for (m, d)-rectangles of size polynomial in m, d and $\epsilon^{\log \epsilon}$.

In particular, in the case that $\epsilon = 2^{-O(\sqrt{\log md})}$, the size of the discrepancy set in our construction is polynomial in m and d.

4.2.1 The Overview of the Construction

For this discussion, let us fix integers m, d and a real $0 < \epsilon \leq 1$. We want to construct an efficient (l, m, d)-generator that ϵ-fools the (m, d)-rectangles with $l = O(\log m + \log d + \log^2 \epsilon^{-1})$. The starting point of our construction is the pseudo-random generator for communication networks of [INW94].

Any (m, d)-rectangle $R = R_1 \times \ldots \times R_d$ can be naturally visualized as an accepting set of an (m, d)-protocol Π of complexity 1 in the following way (see figure 4.1):

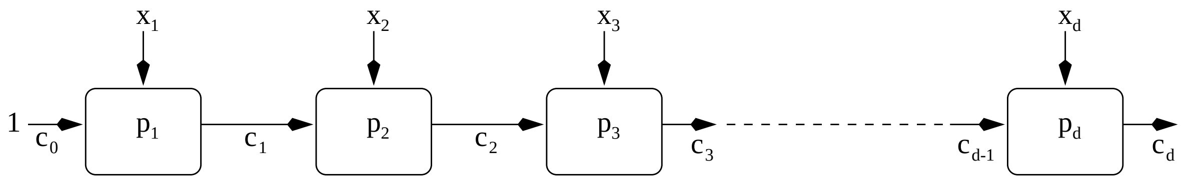

Figure 4.1: The (m,d)-protocol Π that accepts R

Let $p_1, \ldots, p_d$ be d processors in a path network as defined in Section 4.1.8. On input $x = (x_1, \ldots, x_d) \in [m]^d$ to the network, for each $1 \leq i \leq d$ the processor p_i gets as input the i-th coordinate $x_i \in [m]$ and receives a single bit c_{i-1} from its predecessor, and sends a single bit c_i to p_{i+1} where $c_i = c_{i-1} \wedge (x_i \in R_i)$ (we assume here that $c_0 = 1$). So $c_d = 1$ (Π accepts x) if and only if $x \in R$. That is, $R = ACC(\Pi)$. Also it is clear that the complexity of the protocol is 1. Now by Theorem 4.8 we have:

Corollary 4.10: Let m, d be integers, $0 < \epsilon \leq 1$, and let $l = O(\log m + \log d(\log d + \log \epsilon^{-1}))$. Then there is an explicit construction of an efficient (l, m, d)-generator G^* that is $(\mathcal{R}(m,d), 2^{\{0,1\}^l}, \epsilon)$-good.

With respect to what we need, the shortcomings of G^* are that the dependence of l on $\log d$ is not linear and that the dependence of l on d and ϵ^{-1} is multiplicative but not additive. On the other hand, if we apply generator G^* to (m', d')-rectangles for some m', d' where d' depends polynomially *only* on ϵ, then the input-length we need for G^* in this case is $O(\log m' + \log^2 \epsilon^{-1})$. Intuitively, what this observation suggests is that if we could first construct a function family $\mathcal{F}^*$ of "small" size that reduces the problem for (m,d)-rectangles to the problem for (m', d')-rectangles where m' is polynomial in m, d, ϵ^{-1} (thus $\log m'$ is linear in $\log m, \log d$ and $\log \epsilon^{-1}$) and, importantly, d' is polynomial in ϵ^{-1}, then G^* for the latter problem would have short input-length $O(\log m + \log d + \log^2 \epsilon^{-1})$ and so, by composing $\mathcal{F}^*$ and G^*, we could obtain a "small"-sized family $\mathcal{F}$ of functions with short input-length. Therefore, the unification of $\mathcal{F}$ would provide a desired generator.

More precisely, what we will do for our construction is the following: For some $m' = poly(m, d, \epsilon^{-1})$ and $d' = poly(\epsilon^{-1})$, we first construct a family $\mathcal{F}^*$ of functions mapping

$[m']^{d'}$ to $[m]^d$ that is $(\mathcal{R}(m,d), \mathcal{R}(m',d'), 2\epsilon/3)$-good, where the size of $\mathcal{F}^*$ is 2^s for some $s = O(\log d + \log \epsilon^{-1})$. Furthermore, $\mathcal{F}^*$ is indexable and is in $TS(d \log m/\epsilon, \log m + \log d + \log \epsilon^{-1})$. Then we let G^* be the (l', m', d')-generator given by Corollary 4.10 such that G^* is $(\mathcal{R}(m',d'), 2^{\{0,1\}^{l'}}, \epsilon/3)$-good. Thus $l' = O(\log m + \log d + \log^2 \epsilon^{-1})$ and moreover, G^* is in $TS(d' \log m'/\epsilon, l')$ because of its efficiency. Define $\mathcal{F} = \mathcal{F}^* \circ G^*$. By Lemma 4.4, $\mathcal{F}$ is a family of (l', m, d)-generators of size 2^s that is $(\mathcal{R}(m,d), 2^{\{0,1\}^{l'}}, \epsilon)$-good. Moreover, by Proposition 4.5 $\mathcal{F}$ is indexable and is in $TS(d \log m/\epsilon, l')$. Now Proposition 4.6 tells us that $G_{\mathcal{F}}$ is an efficient (l, m, d)-generator that ϵ-fools the (m, d)-rectangles, where $l = l' + s = O(\log m + \log d + \log^2 \epsilon^{-1})$. Thus to accomplish our generator construction as stated in Theorem 4.9, it suffices to build the family $\mathcal{F}^*$.

The family $\mathcal{F}^*$ in our construction is a composition of a sequence of three families of functions $\mathcal{F}_i, 0 \leq i \leq 2$, with each $\mathcal{F}_i$ mapping V_{i+1} to V_i for some V_i. Each function family in the sequence specifies a function reduction that reduces one construction problem to another one with simpler structure. The reduction sequence mainly follows the idea in [LLSZ97] where the problem for general rectangles is first reduced to the problem for PIP rectangles, and then is further reduced to the problem for rectangles whose dimension depends polynomially only on ϵ. One difference between our construction and the one in [LLSZ97] is in the dimension reduction. In [LLSZ97], the error introduced by this reduction can be bounded only from above, which is sufficient for their purposes of constructing (one-sided) hitting sets, but is inadequate for our purposes of (two-sided) discrepancy set construction. An important point in this work is that in the dimension reduction, we reduce the dimension to polynomial in ϵ^{-1} while keeping the error bounded small from both sides. The details of the constructions will be given in the next few subsections. The properties that this sequence $\mathcal{F}_i$ satisfies are summarized below.

For $0 \leq i \leq 3$, each V_i is of the form $[m_i]^{d_i}$ for some m_i, d_i such that, with possible exceptions on $m_0 = m$ and $d_0 = d$, all the other m_i and d_i are integer powers of 2. (Note that the m', d' in the above description are now m_3 and d_3, respectively.)

Family $\mathcal{F}_0$: As a preliminary for the next two constructions, the purpose of this function family is to reduce the problem for (m, d)-rectangles where m, d are arbitrary to the problem for (m', d')-rectangles where m', d' are integer powers of 2.

For $m_1 = O(m_0 d_0/\epsilon)$ and $d_1 = O(d_0)$, $\mathcal{F}_0$ is a family of one single function from V_1 to V_0 that is $(\mathcal{R}(m, d), \mathcal{R}(m_1, d_1), \frac{\epsilon}{3})$-good. Moreover, it is in $TS(d_1 \log m_1, \log m_1 + \log d)$. We will call $\mathcal{F}_0$ the *preliminary reduction function family.*

Family $\mathcal{F}_1$: To accomplish the dimension reduction, it is desirable to deal with PIP rectangles and not general ones. Function family $\mathcal{F}_1$ reduces the problem for general rectangles to the problem for PIP rectangles.

For $m_2 = (\max(m_1, d_1))^2$ and $d_2 = d_1$, $\mathcal{F}_1$ is a family of one single function from V_2 to V_1 that is $(\mathcal{R}(m_1, d_1), PIP\text{-}\mathcal{R}(m_2, d_2), 0)$-good. Moreover, it is in $TS(d_2 \log m_2, \log m_2)$. We will call $\mathcal{F}_1$ the *PIP reduction function family.*

Family $\mathcal{F}_2$: This is the major component of our construction which specifies a function reduction that reduces the PIP rectangle problem to the problem for rectangles whose dimension depends polynomially only on ϵ^{-1}.

For $m_3 = m_2$ and $d_3 = O(\epsilon^{-1} ln^2 \epsilon^{-1})$, $\mathcal{F}_2$ is a family of functions mapping V_3 to V_2 that is $(PIP\text{-}\mathcal{R}(m_2, d_2), \mathcal{R}(m_3, d_3), \frac{\epsilon}{3})$-good. The size of $\mathcal{F}_2$ is $(\max(d_2, d_3))^3$. Moreover, $\mathcal{F}_2$ is in $TS((d_2 + d_3) \log m_3, \log(d_2 d_3))$. We will call $\mathcal{F}_2$ the *dimension reduction function family.*

Furthermore, each function family $\mathcal{F}_i$ is indexable.

It is clear from the parameters chosen above that $m' = m_3 = poly(m, d, \epsilon^{-1}), d' = d_3 = poly(\epsilon^{-1})$, and since $2^s = \Pi_{i=0}^2 |\mathcal{F}_i|$, $s = \log(\max(d_2, d_3))^3 = O(\log d + \log \epsilon^{-1})$. Moreover, by Lemma 4.4, $\mathcal{F}^* = \bigodot_{i=0}^2 \mathcal{F}_i$ is a family of functions mapping $[m']^{d'}$ to $[m]^d$ that is $(\mathcal{R}(m, d), \mathcal{R}(m', d'), 2\epsilon/3)$-good; and by Proposition 4.5, $\mathcal{F}^*$ is indexable and is in $TS(d \log m/\epsilon, \log m + \log d + \log \epsilon^{-1})$. Thus $\mathcal{F}^*$ is what we needed.

In the rest of this section, we present the constructions of the sequence of function families described above. For the clarity of our presentation, we will first give the constructions of $\mathcal{F}_1, \mathcal{F}_2$, assuming that m_1 and d_1 are integer powers of 2. We will then justify this assumption later in Section 4.2.4 by presenting the construction of family $\mathcal{F}_0$.

4.2.2 PIP Reduction Function Family

Let $V_1 = [m_1]^{d_1}$ where both m_1 and d_1 are assumed to be integer powers of 2, and let $V_2 = [m_2]^{d_2}$ where $m_2 = (\max(m_1, d_1))^2$ and $d_2 = d_1$. We construct a family $\mathcal{F}_1$ of functions mapping V_2 to V_1 that is $(\mathcal{R}(m_1, d_1), PIP\text{-}\mathcal{R}(m_2, d_2), 0)$-good. That is, the function family $\mathcal{F}_1$ specifies a function reduction, which we call PIP reduction, that reduces the problem of finding a discrepancy set for (m_1, d_1)-rectangles in V_1 to the problem of finding a discrepancy set for PIP (m_2, d_2)-rectangles in V_2. We want $\mathcal{F}_1$ to be indexable and in $TS(d_2 \log m_2, \log m_2)$ as well.

The construction given here follows from the construction of the PIP-reduction in [LLSZ97]. We present the construction for completeness.

Let H be a pairwise independent hash function family mapping $[d_2]$ to $[m_1]$ of size $(\max(m_1, d_2))^2 = m_2$ obtained by Theorem 4.7. We identify H with $[m_2]$. Define a function $f : V_2 \to V_1$ as follows

$$f(h_1, \ldots, h_{d_2}) = (h_1(1), \ldots, h_{d_2}(d_2)).$$

Let $\mathcal{F}_1 = \{f\}$. Then $\mathcal{F}_1$ is trivially indexable. It is not difficult to check that $\mathcal{F}_1$ is in $TS(d_2 \log m_2, \log m_2)$. Thus it remains to show that $\mathcal{F}_1$ is $(\mathcal{R}(m_1, d_1), PIP\text{-}\mathcal{R}(m_2, d_2), 0)$-good.

Let $R = R_1 \times \ldots \times R_{d_1} \in \mathcal{R}(m_1, d_1)$. For $1 \le i \le d_2 = d_1$, define $R'_i = \{h \in H = [m_2] : h(i) \in R_i\}$. Then $R\Lambda_f = R'_1 \times \ldots \times R'_{d_2} = R'$ is an (m_2, d_2)-rectangle in V_2.

By the definition of H, $\frac{|R'_i|}{m_2} = \Pr_{h \in H}[h(i) \in R_i] = \frac{|R_i|}{m_1}$, which implies that $vol(R') = vol(R)$. Hence, $|E_{f \in \mathcal{F}_1}[vol(R')] - vol(R)| = 0$.

To complete the proof, we now show that R' is PIP. For any $1 \le i < j \le d_2$,

$$\frac{|R'_i \cap R'_j|}{m_2} = \Pr_{h \in H}[h(i) \in R_i \text{ and } h(j) \in R_j] = \frac{|R_i|}{m_1}\frac{|R_j|}{m_1} = \frac{|R'_i|}{m_2}\frac{|R'_j|}{m_2}$$

4.2.3 Dimension Reduction Function Family

The purpose of function family $\mathcal{F}_2$ is to specify a function reduction, which we call dimension reduction, that reduces the problem of finding a discrepancy set for PIP (m_2, d_2)-rectangles in V_2 to the problem of finding a discrepancy set for rectangles whose dimension depends only on ϵ.

Let $V_3 = [m_3]^{d_3}$ where $m_3 = m_2$ and $d_3 = 2^{\lceil \log(8\epsilon^{-1} \ln^2(8\epsilon^{-1})) \rceil}$ (thus $d_3 \geq \frac{8 \ln^2 \frac{8}{\epsilon}}{\epsilon}$). Let H be a 3-wise independent hash function family mapping $[d_2]$ to $[d_3]$ of size $(\max(d_2, d_3))^3$ given by Theorem 4.7. For each $h \in H$, we define a function $f_h : V_3 \to V_2$ as follows:

$$f_h(p_1, \ldots, p_{d_3}) = (p_{h(1)}, \ldots, p_{h(d_2)}).$$

Let $\mathcal{F}_2 = \{f_h : h \in H\}$. Since H is indexable and is in $TS(\log d_2 d_3, \log d_2 d_3)$ by Theorem 4.7, $\mathcal{F}_2$ is indexable and is in $TS((d_2 + d_3)\log m_3, \log d_2 d_3)$. We want to show that $\mathcal{F}_2$ is $(PIP\text{-}\mathcal{R}(m_2, d_2), \mathcal{R}(m_3, d_3), \frac{\epsilon}{3})$-good.

Let m, d be integers and $R = R_1 \times R_2 \times \ldots \times R_d \in \mathcal{R}(m, d)$. For the proof we need the following notations. For $1 \leq i \leq d$, let $\beta_i = \frac{|R_i|}{m}$, $\delta_i = 1 - \beta_i$ and for a subset $S \subseteq [d]$, we denote

$$T_S = \cap_{i \in S} R_i, \quad \gamma(S) = \frac{|T_S|}{m_3}, \quad \pi(S) = \prod_{i \in S} \beta_j,$$

$$\nu(S) = \sum_{i \in S} \delta_i, \quad \mu(S) = \sum_{i,j \in S, i<j} \delta_i \delta_j, \text{ and}$$

$$\tau(S) = \sum_{i,j,k \in S, i<j<k} \delta_i \delta_j \delta_k.$$

Fix any $R = R_1 \times \ldots \times R_{d_2} \in PIP\text{-}\mathcal{R}(m_2, d_2)$. It is easy to verify that for every $h \in H$, $R\Lambda_{f_h} = T_{h^{-1}(1)} \times \ldots \times T_{h^{-1}(d_3)} = R^{(h)} \in \mathcal{R}(m_3, d_3)$. We are left to show that $|E_{f \in \mathcal{F}_2}[vol(R\Lambda_f)] - vol(R)| = |E_{h \in H}[vol(R^{(h)})] - vol(R)| \leq \frac{\epsilon}{3}$.

We want to estimate $|E_{h \in H}[vol(R^{(h)})] - vol(R)|$. The main lemma we will need is the following:

Lemma 4.11: Let m, d, d' be integers and H be a 3-wise independent hash function family mapping $[d]$ to $[d']$. Then for any $R \in PIP\text{-}\mathcal{R}(m,d)$ and for every $h \in H$,

$$vol(R) - \sum_{q\in[d']} \mu(h^{-1}(q)) \leq vol(R^{(h)}) \leq vol(R) + \sum_{q\in[d']} \tau(h^{-1}(q))$$

Let us first assume that the lemma holds and see that the estimation follows. By the lemma, we have that

$$|E_{h\in H}[vol(R^{(h)})] - vol(R)| \leq E_{h\in H}[\sum_{q\in d_3} \mu(h^{-1}(q))] + E_{h\in H}[\sum_{q\in d_3} \tau(h^{-1}(q))]$$

For the estimation, we consider two cases depending on the volume of R. The first case is when $vol(R) \geq \frac{\epsilon}{8}$. Then $\frac{\epsilon}{8} \leq \prod_{i\in[d_2]} \beta_i = \prod_{i\in[d_2]}(1-\delta_i) \leq \exp(-\sum_{i\in[d_2]} \delta_i)$, which implies that $\sum_{i\in[d_2]} \delta_i \leq \ln\frac{8}{\epsilon}$.

$$\begin{aligned}
E_{h\in H}[\sum_{q\in[d_3]} \tau(h^{-1}(q))] &= \sum_{q\in[d_3]} E_{h\in H}[\sum_{i,j,k\in h^{-1}(q), i<j<k} \delta_i\delta_j\delta_k] \\
&= \sum_{q\in[d_3]} \sum_{i,j,k\in[d_2], i<j<k} \Pr_{h\in H}[h(i)=h(j)=h(k)=q]\delta_i\delta_j\delta_k \\
&= \sum_{q\in[d_3]} (\frac{1}{d_3})^3 \sum_{i,j,k\in[d_2], i<j<k} \delta_i\delta_j\delta_k \\
&\leq (\frac{1}{d_3})^2 (\sum_{i\in[d_2]} \delta_i)^3 \\
&\leq \frac{\epsilon^2}{64\ln^4(\frac{8}{\epsilon})} \ln^3(\frac{8}{\epsilon}) \leq \frac{\epsilon}{64}
\end{aligned}$$

where the third equality is by the choice of H as a 3-wise independent hash function family. Similarly, one can show that $E_{h\in H}[\sum_{q\in d_3} \mu(h^{-1}(q))] \leq \frac{\epsilon}{8}$. Therefore $|E_{h\in H}[vol(R^{(h)})] - vol(R)| \leq \frac{\epsilon}{3}$ in this case.

In the second case we have $vol(R) < \frac{\epsilon}{8}$. We let R' be another PIP (m_2, d_2)-rectangle containing R such that $\frac{\epsilon}{8} \leq vol(R') \leq \frac{\epsilon}{4}$. (The existence of such an R' is not difficult to justify.) Since R' contains R, then for every $h \in H$, $R'^{(h)} = R'\Lambda_{f_h}$ contains $R^{(h)}$, and thus $E_{h\in H}[vol(R^{(h)})] \leq E_{h\in H}[vol(R'^{(h)})]$. By the right inequality in Lemma 4.11 and the above analysis, we have that

$$E_{h\in H}[vol(R'^{(h)})] \leq vol(R) + E_{h\in H}[\sum_{q\in[d_3]} \tau(h^{-1}(q))] \leq \frac{\epsilon}{4} + \frac{\epsilon}{64} < \frac{\epsilon}{3}$$

Since volumes are always nonnegative, $|E_{h\in H}[vol(R^{(h)})] - vol(R)| \leq \frac{\epsilon}{3}$ in this case as well.

Now it remains to prove Lemma 4.11.

Proof of Lemma 4.11:

Let $R = R_1 \times R_2 \times \ldots \times R_d$. We first show two preliminary facts.

Proposition 4.12: For any $S \subseteq [d]$,

$$\pi(S) - \mu(S) \leq \gamma(S) \leq \pi(S) + \tau(S).$$

Proof: Let $S \subseteq [d]$ be arbitrary. By definition, $\gamma(S) = \frac{|\cap_{i\in S} R_i|}{m} = 1 - \frac{|\cup_{i\in S} \bar{R}_i|}{m}$. Applying inclusion-exclusion, we have that

$$\gamma(S) \leq 1 - \sum_{i\in S} \frac{|\bar{R}_i|}{m} + \sum_{i,j\in S, i<j} \frac{|\bar{R}_i \cap \bar{R}_j|}{m} = 1 - \nu(S) + \mu(S),$$

where the equality follows from the fact that R is PIP, and that

$$\gamma(S) \geq 1 - \sum_{i\in S} \frac{|\bar{R}_i|}{m} = 1 - \nu(S).$$

Since it is easy to prove by induction on the size of S that

$$1 - \nu(S) + \mu(S) - \tau(S) \leq \pi(S) \leq 1 - \nu(S) + \mu(S),$$

combining this with the above facts about $\gamma(S)$ we draw the conclusion. $\square$

Proposition 4.13: Let x, y, z be nonnegative real numbers such that $x \leq 1$ and $0 \leq x - y \leq z \leq x + y$. Let x', y', z' be nonnegative real numbers such that $0 \leq x' - y' \leq z' \leq x' + y' \leq 1$. Then $xx' - y' - y \leq zz' \leq xx' + y' + y$.

Proof: By assumptions, $(x - y)(x' - y') \leq zz' \leq (x + y)(x' + z')$. Also we have that $(x-y)(x'-y') \geq x(x'-y')-y \geq xx'-y'-y$ and that $(x+y)(x'+y') \leq x(x'+y')+y \leq xx'+y'+y$. The proof is complete. $\square$

Now we prove by induction the following inequalities: for each $1 \leq t \leq d'$,

$$\prod_{q=1}^{t} \pi(h^{-1}(q)) - \sum_{q=1}^{t} \mu(h^{-1}(q)) \leq \prod_{q=1}^{t} \gamma(h^{-1}(q)) \leq \prod_{q=1}^{t} \pi(h^{-1}(q)) + \sum_{q=1}^{t} \tau(h^{-1}(q))$$

The case where $t = 1$ follows from Proposition 4.12. For the inductive step, it is easy to check that Proposition 4.13 establishes the inequalities.

Finally, the lemma follows by substituting d' for t in the above inequalities. □

4.2.4 Preliminary Reduction Function Family

We justify the assumption that m_1 and d_1 are integer powers of 2.

We have $V_0 = [m]^d$ where m, d are arbitrary integers. For some $m_1 = O(md/\epsilon)$ and $d_1 = O(d)$ that are integer powers of 2, we will construct a function family $\mathcal{F}_0$ that is in $TS(d_1 \log m_1, \log m_1 + \log d)$ and is $(\mathcal{R}(m, d), \mathcal{R}(m_1, d_1), \frac{\epsilon}{3})$-good.

The function family we shall construct contains a single function $f_{m,d}$ which is obtained by composing two other functions: f_m and f_d. Let $W = [m_1]^d$. f_m is a function from W to V_0 that is in $TS(d \log m_1, \log m_1)$ and is $(\mathcal{R}(m, d), \mathcal{R}(m_1, d), \frac{\epsilon}{3})$-good, and f_d is a function from V_1 to W that is in $TS(d_1 \log m_1, \log d)$ and is $(\mathcal{R}(m_1, d), \mathcal{R}(m_1, d_1), 0)$-good. Thus $f_{m,d} = f_m \circ f_d$ is as needed.

The Function f_m

Let $c_0 = \lceil \frac{8d}{\epsilon} \rceil$ and let $m_1 = 2^{\lceil \log c_0 m \rceil}$. Define the function $f_m : W \to V_0$ as follows: $f(p_1, \ldots, p_d) = (q_1, \ldots, q_d)$ if and only if for each $1 \leq i \leq d$, $p_i \equiv q_i \pmod{m}$. Clearly, f_m is in $TS(d \log m_1, \log m)$. We want to show that f_m is $(\mathcal{R}(m, d), \mathcal{R}(m_1, d), \frac{\epsilon}{3})$-good.

Fix any $R = R_1 \times \ldots \times R_d \in \mathcal{R}(m, d)$. Let $R'_i = \{p \in [m_1] : (p \pmod{m}) \in R_i\}$. Then $R\Lambda_f = R'_1 \times \ldots \times R'_{d_2} = R' \in \mathcal{R}(m_1, d)$. All that remains is to show that $|vol(R') - vol(R)| \leq \frac{\epsilon}{4}$.

Let c be the unique integer such that $cm \leq m_1 < (c+1)m$. Then it is clear that $c \geq c_0$. Since $\lfloor \frac{m_1}{m} \rfloor |R_i| \leq |R'_i| \leq \lceil \frac{m_1}{m} \rceil |R_i|$, we have that

$$(\frac{cm}{m_1})^d vol(R) \leq vol(R') \leq (\frac{(c+1)m}{m_1})^d vol(R).$$

Now

$$(\frac{cm}{m_1})^d \geq (\frac{c}{c+1})^d \geq (1 - \frac{\epsilon}{8d})^d \geq (\frac{1}{4})^{\frac{\epsilon}{8}} \geq 1 - \frac{\epsilon}{4},$$

which implies that $(1-\frac{\epsilon}{4})vol(R) \leq vol(R')$, and

$$(\frac{(c+1)m}{m_1})^d \leq (\frac{c+1}{c})^d \leq (1+\frac{\epsilon}{8d})^d \leq 2^{\frac{\epsilon}{8}} \leq 1+\frac{\epsilon}{8},$$

which implies that $vol(R') \leq (1+\frac{\epsilon}{8})vol(R)$. Therefore $|vol(R') - vol(R)| \leq \frac{\epsilon}{3}$ and this concludes the proof.

The Function f_d

Let $d_1 = 2^{\lceil \log d \rceil}$ and let $V_1 = [m_1]^{d_1}$. Define the function $f_d : V_1 \to W$ as follows: $f_d(q_1, \ldots, q_{d_1}) = (q_1, \ldots, q_d)$. That is, f_d simply outputs the first d coordinates of the input and discard the rest. Obviously, f_d is in $TS(d_1 \log m_1, \log d)$.

To see that f_d is $(\mathcal{R}(m_1, d), \mathcal{R}(m_1, d_1), 0)$-good, we observe that for any combinatorial rectangle $R = R_1 \times \ldots \times R_d \in \mathcal{R}(m_1, d)$,

$$R\Lambda_{f_d} = R_1 \times \ldots \times R_d \times [m_1]^{d_1-d} \in \mathcal{R}(m_1, d_1)$$

and $R\Lambda_{f_d}$ has the same volume as R has.

Bibliography

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Algorithms*, 7:567–583, 1986.

[AGHP92] Noga Alon, Oded Goldreich, Johan Hastad, and René Peralta. Simple construction of almost k-wise independent random variables. *Algorithms*, 3(3):289–303, 1992.

[AH87] Leonard M. Adleman and M. A. Huang. Recognizing primes in random polynomial time. In *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing (STOC)*, New York City, 1987.

[AKL+79] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovász, and Charles Rackoff. Random walks, universal traversal sequences and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 218–223. IEEE, 1979.

[AKS87] Miklos Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in LOGSPACE. In *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 132–140, 1987.

[Arm98] Roy Armoni. On the derandomization of space-bounded computations. In *Proceedings of the 2nd International Workshop on Randomization and Approxima-*

tion Techniques in Computer Science (RANDOM), Lecture notes in computer science, pages 47–59, Barcelona, Spain, October 1998. Springer-Verlag.

[ASWZ96] Roy Armoni, Michael Saks, Avi Wigderson, and Shiyu Zhou. Discrepancy sets and pseudorandom generators for combinatorial rectangles. In *37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 412–421. IEEE, 1996.

[ATSWZ97] Roy Armoni, Amnon Ta-Shma, Avi Wigderson, and Shiyu Zhou. $SL \subseteq L^{4/3}$. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 230–239, 1997.

[BC87] Jozsef Beck and William W.L. Chen. *Irregularities of distribution.* Cambridge University Press, 1987.

[BF93] Greg Barnes and Uriel Feige. Short random walks on graphs. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 728–737. ACM, 1993.

[BGG93] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Randomness in interactive proofs. *Computational Complexity*, 3(4):319–354, 1993.

[BM84] Manual Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM J. Comput.*, 13(4):850–864, November 1984.

[BNS89] Laszlo Babai, Noam Nisan, and Mario Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing (STOC)*, pages 1–11, Seattle, Washington, 1989.

[BR94] Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–287, Santa Fe, New Mexico, November 1994.

[CG89] Benny Chor and Oded Goldreich. On the power of two-point based sampling. *J. of Complexity*, 5:96–106, 1989.

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

[EGL+92] Guy Even, Oded Goldreich, Michael Luby, Noam Nisan, and Boban Velicković. Approximations of general independent distributions. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 10–16. ACM, 1992.

[INW94] Russel Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 356–364, Montréal, Québec, Canada, May 1994.

[KK94] David Karger and Daphne Koller. (de)randomized construction of small sample spaces in **nc**. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 252–263. IEEE, 1994.

[KM93] Daphne Koller and Nimrod Megiddo. Constructing small sample spaces satisfying given constraints. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 268–277. ACM, 1993.

[KW84] Richard Karp and Avi Wigderson. A fast parallel algorithm for the maximal independent set problem. In *Proceedings of the 16th Annual ACM Symposium on the Theory of Computing (STOC)*. ACM, 1984.

[LLSZ97] Nathan Linial, Michael Luby, Michael Saks, and David Zuckerman. Efficient construction of a small hitting set for combinatorial rectangles in high dimension. *Combinatorica*, 17, 1997.

[LP82] Harry R. Lewis and Cristos H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19:161–187, 1982.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

[LVW93] Michael Luby, Boban Velickovic, and Avi Wigderson. Deterministic approximate counting of depth–2 circuits. In *Proc. of the 2nd ISTCS (Israeli Symposium on Theoretical Computer Science)*, pages 18–24, 1993.

[Nie] Harold Niederreiter. Constructions of low-discrepancy point sets and sequences. manuscript and lecture notes.

[Nis91] Noam Nisan. Pseudo-random bits for constant depth circuits. *Combinatorica*, 11(1):63–71, 1991.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, June 1992.

[Nis94] Noam Nisan. RL $\subseteq$ SC. *Computational Complexity*, 4, 1994.

[NN93] Josepf Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[NSW92] Noam Nisan, Endre Szemeredi, and Avi Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 24–29, 1992.

[NZ93] Noam Nisan and David Zuckerman. More deterministic simulation in logspace. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 235–244, San Diego, CA, May 1993.

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

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

[Sch92] Leonard J. Schulman. Sample spaces uniform on neighborhoods. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 17–25. ACM, 1992.

[Sha81] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. In *Proceedings of the 8th ICALP, Lecture notes in computer science 62*, pages 544–550. Springer-Verlag, 1981.

[SS77] Robert M. Solovay and Volker Strassen. A fast monte-carlo test for primality. *SIAM J. Comput.*, 6:84–85, 1977.

[SZ95] Michael Saks and Shiyu Zhou. $RSPACE(S) \subset DSPACE(S^{\frac{3}{2}})$. In *36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 344–353, Milwaukee, Wisconsin, October 1995. To appear in JCSS.

[Yao82] Andrew C. Yao. Theory and application of trapdorr functions. In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, October 1982.

[Zuc96] David Zuckerman. Randomness-optimal sampling, extractors, and constructive leader election. In *Proceedings of the 28th Annual ACM Symposium on the*

Theory of Computing (STOC), pages 286–295, Philadelphia, Pennsylvania, May 1996.