Extracting Randomness: How and Why
A survey

Noam Nisan*
Institute of Computer Science
Hebrew University
Jerusalem, Israel

Abstract

Extractors are boolean functions that allow, in some precise sense, extraction of randomness from somewhat random distributions. Extractors, and the closely related "Dispersers", exhibit some of the most "random-like" properties of explicitly constructed combinatorial structures. In turn, extractors and dispersers have many applications in "removing randomness" in various settings, and in making randomized constructions explicit.

This manuscript surveys extractors and dispersers: what they are, how they can be designed, and some of their applications. The work described is due to of a long list of research papers by various authors – most notably by David Zuckerman.

1 Introduction

During the last two decades the use of randomization in the design of algorithms has become commonplace. There are many examples of randomized algorithms for various problems which are better than any known deterministic algorithm for the problem. The randomized algorithms may be faster, more space-efficient, use less communication, allow parallelization, or may be simply simpler than the deterministic counterparts. We refer the reader e.g. to [MR95] for a textbook on randomized algorithms.

Despite the wide spread use of randomized algorithms, in almost all cases it is not at all clear whether randomization is really necessary. As far as we know, it may be possible to convert *any* randomized algorithm to a deterministic one without paying any penalty in time, space, or other resources. In many cases, in fact, we do know how to convert a randomized algorithm to a deterministic one – "De-randomizing" the algorithm. This notion of "De-randomization" has also become quite common. By now, a standard technique for designing a deterministic algorithm is to first design a randomized one (which in many cases is easier to do) and then De-randomize it.

This state of affairs in algorithmic design is parallel in the design of various combinatorial objects (such as graphs or hypergraphs with certain properties). The "probabilistic method" is many times used to non-constructively prove the existence of these sought after objects. Again, in many cases it is known how to "De-randomized" these probabilistic proofs, and achieve an explicit construction. We refer the reader to [AS92a] for a survey of the probabilistic method.

Derandomization Techniques

It is possible to roughly categorize the techniques used for De-randomization according to their generality. On one extreme are techniques which relate very strongly to the problem and algorithm at hand. These usually rely on a sophisticated understanding of the structure of the problem, and are not applicable to different ones. On the other extreme are completely general De-randomization results – e.g. converting *any* polynomial time randomized algorithm to a deterministic one. This may be done with a sufficiently good pseudo-random generator, but with our current understanding of computational complexity such results *always* rely on unproven assumptions.

In the middle range of generality lie various techniques that apply to certain "types" of randomized algorithms. Algorithms that use their randomness in a "similar" way be De-randomized using similar techniques. Two main strategies for De-randomization are commonly employed. The first is the construction of a "small sample space" for the algorithm. Instead of choosing a truly random string, we fix a small set of strings – the "small sample space" – and then take a string from the sample space instead of choosing it completely at random. This requires, of course, a proof that this sample space is good enough as a replacement for a truly random string. The second strategy is to adap-

*This work was supported by BSF grant 92-00043 and by a Wolfeson award administered by the Israeli Academy of Sciences.

tively "construct" a replacement for the random string – e.g. by gradually improving some conditional probability. In many cases both strategies are combined, and a replacement string is constructed in the small sample space.

It is probably fair to say that there are only two or three basic types of tools which are commonly used in the construction of small sample spaces for De-randomizations:

1. Pairwise (and k-wise) independence and Hashing.
2. Small Bias Spaces.
3. Expanders.

We refer the reader, again, to [AS92a, MR95] for further information as well as for references.

Dispersers and Extractors

This survey explores a fourth general type of tool: a family a graphs called *Dispersers and Extractors*. These graphs have certain strong "random-like" properties – hence they can be used in many cases where "random-like" properties are needed. Roughly speaking, these graphs convert a "somewhat random" distribution into an almost random distribution. This is done using a small additional number of truly random bits.

There are several explicit constructions known for these dispersers and extractors, with various parameters, and there are many examples where these dispersers are used for the purposes of de-randomization – of algorithms or in explicit constructions.

The roots of the research surveyed here lie mostly in the work on "somewhat random sources" done in the late 1980's, by Vazirani, Santha and Vazirani, Vazirani and Vazirani, Chor and Goldreich, and others [SV86, Vaz87a, Vaz86, Vaz87b, VV85, CG88]. The direct development of the constructions and applications of extractors and dispersers came first in papers written by Zuckerman in the early 1990's [Zuc90, Zuc91], and then in a sequence of papers by various authors [NZ93, WZ93, SZ94, SSZ95, TaS96, Zuc93, Zuc96]. Dispersers were first defined (with somewhat different parameters) by Sipser [Sip88], while extractors were defined by Nisan and Zuckerman [NZ93].

Organization of the Survey

In section 2 we provide the basics: go over some preliminaries, define extractors and dispersers, discuss simple lower and upper bounds, and list known results. In section 3 we give an overview of some of the constructions of extractors and dispersers. In section 4, we go over some of the main applications.

2 Basics

2.1 Preliminaries

Let us first define some notions which will be used throughout this survey.

Probability Distributions

We will constantly be discussing probability distributions, the distances between them, and the randomness hidden in them. In this subsection we give the basic definitions used to allow such a discussion.

A probability distribution X over a (finite) space Λ simply assigns to each $a \in \Lambda$ a positive real $X(a) > 0$, with the property that $\sum_{a\in\Lambda} X(a) = 1$. For a subset $S \subseteq \Lambda$ we denote $X(S) = \sum_{a\in S} X(a)$. The uniform distribution U on Λ is defined as $U(a) = 1/|\Lambda|$ for all $a \in \Lambda$.

We usually identify a random variable with its probability distribution (we make the distinction only where necessary). We use capitals $X, Z, \ldots$ to denote such random variables and probability distributions. We use small letters $a, x, z \ldots$ to denote elements in the probability space. Unless stated otherwise x is distributed according to X, z according to Z, etc.

Statistical Distance

Definition 2.1 *Let X, Y be two distributions over the same space Λ. The* statistical distance *between them is given by:* $d(X,Y) = \frac{1}{2}|X - Y|_1 = \frac{1}{2}\Sigma_{a\in\Lambda}|X(a)-Y(a)| = max_{S\subseteq\Lambda}|X(S)-Y(S)|$.

It is easy to verify that the statistical distance is indeed a metric. We say X is ϵ-close to Y if $d(X,Y) \leq \epsilon$. We say X is ϵ quasi-random if it is ϵ-close to uniform.

Min-Entropy

We will need to measure the "amount of randomness" that a given distribution X has in it. The Shannon entropy of X certainly springs to mind, but it turns out that we will require a different notion.

Definition 2.2 *The min-entropy of a distribution X is $H_\infty(X) = min_a(-log_2(X(a)))$.*

It is easy to verify that $H_\infty(X)$ is always bounded from above by the Shannon entropy $H(X)$. Equality holds whenever X is uniform over a subset $S \subseteq \Lambda$, in which case $H_\infty(X) = H(X) = \log_2|S|$. It is useful to think of a distribution X with $H_\infty(X) = k$ as a generalization of being uniform over a set of size 2^k.

2.2 Extractors and Dispersers

Extractors and dispersers are very similar to each other, yet, in the literature, dispersers have been usually defined as graphs [Sip88, SSZ95], while extractors as functions [NZ93, SZ94, TaS96, Zuc96].

We will define both extractors and dispersers both as functions and as graphs, taking the view that it is the same combinatorial object, viewed in two different, useful, ways.

Graph Definitions

Extractors and dispersers are certain types of bipartite (multi-)graphs. Throughout this survey the left hand side of the graph will have $N = 2^n$ vertices and the right hand side of the graph $M = 2^m$ vertices. Vertices will be numbered by integers which we identify with their binary representation. Thus the left hand side of the graph is always $[N] = \{1...N\} = \{0,1\}^n$, and the right hand side is always $[M] = \{1...M\} = \{0,1\}^m$. The graphs will usually be highly imbalanced $n > m$. Furthermore, all vertices on the left-hand side will have the same degree, $D = 2^d$, which is usually very small $d << m$.

Given a graph $G = ([N],[M],E)$; for each vertex $a \in [N]$, we denote its neighbor set by $\Gamma(a) = \{z \in [M] | (a,z) \in E\}$. For a subset $A \subseteq [N]$ we denote its neighbor set by $\Gamma(A) = \bigcup_{a \in A} \Gamma(a)$. For a distribution X on $[N]$, we denote by $\Gamma(X)$ the probability distribution induced on $[M]$ by first choosing uniformly $x \in [N]$ according to X and then choosing uniformly a random neighbor $z \in \Gamma(x)$.

Definition 2.3 *A (multi-)graph $G = ([N],[M],E)$ is a (k,ϵ)-disperser if for any $A \subseteq [N], |A| \geq K = 2^k$, $|\Gamma(A)| \geq (1-\epsilon)M$.*

Definition 2.4 *A (multi-)graph $G = ([N],[M],E)$ is a (k,ϵ)-extractor if for any distribution X on $[N]$, with $H_\infty(X) \geq k$, we have that $\Gamma(X)$ is ϵ-close to uniform on $[M]$.*

Remark 2.1 *This definition is slightly different from the one in [NZ93, SZ94], who require that the random choice of the edge originating at x is also almost independent from $\Gamma(X)$. The difference is minor, though, and we prefer this definition.*

It is easy to see that any (k,ϵ)–extractor is also an (k,ϵ)–disperser (consider the uniform distribution on A).

The name "disperser" is natural from this point of view: the graph disperses any set of size K, to almost all the right hand side.

Function Definitions

The definitions below are equivalent to the previous definitions, but take the functional view. Recall that the left hand side of our graphs is $\{1...N\} = [N] = \{0,1\}^n$, the right hand side $\{1...M\} = [M] = \{0,1\}^m$. The left degree is $D = 2^d$, so the edges originating at each vertex on the left hand side will be labeled by $\{1...D\} = [D] = \{0,1\}^d$. Our functions will take a name of a vertex on the left-hand side and an index of an edge and will return the name of the vertex on the right hand side: $G : [N] \times [D] \to [M]$. For $A \subseteq [N]$ We will denote $\Gamma(A) = \{z = G(x,y) \ : \ x \in A, \ y \in [D]\}$. Similarly, for a distribution X on $[N]$, $\Gamma(X)$ will denote the distribution of $G(x,y)$ induced by choosing x according to X, and y uniformly in $[D]$.

Definition 2.5 *A function $G : [N] \times [D] \to [M]$ is called a (k,ϵ)-disperser if for any $A \subseteq [N], |A| \geq K = 2^k$, we have that $|\Gamma(A)| \geq (1-\epsilon)M$.*

Definition 2.6 *A function $G : [N] \times [D] \to [M]$ is called a (k,ϵ)-extractor if for any distribution X on $[N]$, $H_\infty(X) \geq k$, we have that $\Gamma(X)$ is ϵ-close to uniform.*

The name "extractor" is natural from this point of view: the function extracts m-bits of randomness from the n-bits given to it, as long as these n bits have k min-entropy. In order to do this it uses d truly random bits as well.

Explicitness

We will be constructing expanders and dispersers. Let us define what an explicit construction is. Let us recall that (in theoretical computer science) defining a combinatorial object as explicitly constructed only makes sense for a *family* of objects. In our case, we will be taking n as the parameter of the family, have $N = 2^n$, and view $M = 2^m, K = 2^k, D = 2^d, \epsilon$ as depending on it. We simply require that the function G is in polynomial time:

Definition 2.7 *A family of functions $G = \{G_n : [N] \times [D] \to [M]\}$ ($N = 2^n, D = D(n), M = M(n)$), is called explicit if there is a deterministic Turing machine that for every $x \in [N]$ and $y \in [D]$, outputs $G_n(x,y)$ in polynomial time (in the length of the input x, y).*

2.3 Behavior of Parameters

In this section we discuss the basic relations between the different parameters of extractors and dispersers. Let us recall first the five different parameters we are interested in, and their intuitive meanings, both in the graph view and in the function view.

- $N = 2^n$. N is the number of vertices in the left hand side of G. n is the number of bits in the string x from which randomness should be extracted.
- $M = 2^m$. M is the number of vertices on the right hand side of G. m is the length of the near-random output z.
- $D = 2^d$. D is the left-degree of G. d is the length of the truly random input y of G.
- $K = 2^k$. K is the size of the set which should dispersed. k is the min-entropy of x.
- ϵ. The error parameter.

We view n as a parameter, m as being a second parameter anywhere in the range $1 \le m \le n$, and then we try to have k and ϵ as small as possible. Our main goal, for each choice of these parameters, is to get d to be as small as possible.

Non-Explicit Existence

Rather standard applications of the probabilistic method show that for all $n \ge m \ge 1, \epsilon > 0$, There exist extractors with $k = m$ and $D = O(n/\epsilon^2)$ and dispersers with $D = O(n/\epsilon)$ (I.e. in both cases $d = O(\log n + \log \epsilon^{-1})$). This calculation was done, for certain dispersers, in [Sip88].

Lower Bounds

A trivial lower bound, for any n, m and $\epsilon < 1/2$, is $d \ge m - k - 1$. We will mostly consider the case where $k \ge m$ so this does not help us. In [NZ93] a lower bound of $d \ge \min(m, \Omega(\log n + \log \epsilon^{-1}))$ was proved for all $n, m, k \le n-1, \epsilon < 1/2$.

Expanders vs. Dispersers and Extractors

Extractors and Dispersers resemble expanders in that the requirement imposed in both cases is that the each subset of vertices have enough neighbors. A major difference is the strongly imbalanced sizes of the different sides of the graph in dispersers and extractors. A more important difference is that we can obtain much stronger parameters with extractors and dispersers. Such strong parameters cannot be obtained by using the eigenvalue methods used in all expander constructions. On the other hand, due to these strong requirements, extractors and dispersers cannot have constant degree like expanders can.

Before getting into known constructions of extractors and dispersers, it is perhaps instructive to see the best construction of dispersers which is implied by expanders. This construction is due to [AKS87] (implicitly) and appears explicitly in [IZ89, CW89].

Take any expander on M vertices with degree $C = 2^c$, and pick any $l = \theta(m)$. The vertices on the right hand side of our disperser will simply be the vertices of the expander. The vertices on the left hand side correspond to paths of length l on the expander (thus $n = m + lc$). Each path is connected to all the vertices on it (thus $D = l$). The analysis showing that this is a disperser proceeds by analyzing the random walk on the expander, showing that it misses any set $W \subseteq [M], |W| \ge M/2$ with exponentially small probability. This construction gives dispersers with the optimal $d = O(\log n + \log \epsilon^{-1})$, but works only for cases where $k > n/2$. This limit is reached since the important parameter in the above analysis is the second largest eigenvalue λ of the expander, which is constrained by the lower bound $\lambda \ge \Omega(\sqrt(C))$ ([Alo85], see [CW89]). In the rest of this survey we will be interested in constructions which work for smaller values of k.

Known Results

The first explicit construction of extractors came in [NZ93], and relied on techniques developed in [Zuc90, Zuc91]. This construction had $d = polylog(n)$ for $k \ge n/polylog(n)$. In [SZ94] and the final version of [Zuc91], $d = O(\log n)$ was obtained for $k = \Omega(n)$. In [SZ94] $d = polylog(n)$ was obtained for $k \ge \sqrt{n}$, with "nearly-logarithmic" d for larger values of k. In [SSZ95] a disperser with $d = O(\log n)$ was obtained for any $k = n^{\Omega(1)}$. In [TaS96], $d = polylog(n)$ (and $m = k$) was obtained for all values of k. Also in [TaS96] "nearly-logarithmic" d was obtained for all $k = n^{\Omega(1)}$.

In the following table we list the currently known best explicitly constructible extractors and dispersers for various parameters.

k	d	m	ref.
$\Omega(n)$	$O(log(n/\epsilon))$	$\Omega(k)$	[Zuc96]
$\Omega(n^\gamma)$	$O(logn)$, $\epsilon = 1/2$	n^δ	[SSZ95] Disperser
$\Omega(n^\gamma)$	$O(log(n/\epsilon) \cdot log^{(k)} n)$	n^δ	[TaS96]
any k	$poly(log(n/\epsilon))$	k	[TaS96]

There are several open problems regarding improvement in parameters:

1. Build an extractor or disperser, for small k, with $d = O(\log n)$.
2. Build an extractor or disperser with $D = O(n/poly(\epsilon))$.

3 A construction

In this section we provide an overview of a construction of extractors. We will also present the main ingredients used in known constructions and indicate how they are put together. We will describe the construction itself precisely, but provide only sketches of proofs for their validity.

All constructions for extractors and dispersers build, as a starting point, an extractor with very weak parameters, but which can be obtained easily. This is described in section 3.1. In section 3.2. we describe how extractors can be composed. In section 3.3. we show how such a composition may be applied to build a single composite extractor. Finally, section 3.4. sketches the different ways in which these elements can be combined to obtain some known constructions of extractors and dispersers.

3.1 The mother of all extractors

A useful way to think about extractors is to consider the edges of the extractor as hashing the vertices on the left to vertices on the right. The requirement is that large enough sets are well hashed. Viewed this way we can construct an extractor from a family of hash functions as follows.

A Construction based on Hashing

Let H be a family of functions $h : [N] \rightarrow [L]$. The extractor defined by H is given by $G(x, h) = h(x) \circ h$. Thus $D = |H|$, and $M = DL$.

Now we define the property that we require from the family of hash functions in order for the graph to be an extractor.

Definition 3.1 *$H = \{h : [N] \mapsto [L]\}$ is a family of hash functions with collision error δ, if for any $x_1 \neq x_2 \in [N]$, $Prob_{h \in H}[h(x_1) = h(x_2)] \leq (1 + \delta)/L$.*

The following lemma is a variant of the of leftover hash lemma of Impagliazzo, Levin, and Luby [ILL89], stated in our terms:

Lemma 3.1 *[ILL89] Let H be a family of hash functions from $[N]$ to $[L]$ with collision error δ. Then, the extractor defined from H is an (k, ϵ)-extractor for $K = 2^k = O(L/\delta)$ and $\epsilon = O(\sqrt{\delta})$.*

Proof: (sketch) We need to show that for every distribution X on $[N]$, with $H_\infty(X) \geq \log(L/\delta)$, the distribution of $h \circ h(x)$ is ϵ-close to uniform (where x is chosen according to X and h uniformly in H). It turns out that the notion of collision probability is a convenient tool for this proof. The collision probability of a distribution X is defined to be $col(X) = \sum_a X(a)^2$. With this definition the following three steps, which imply the lemma, can be shown.

1. If $H_\infty(X) \geq K$ then $col(X) \leq 1/K$.
2. If $col(X)$ is small and H has a small collision error, then the collision probability of $Z = (h \circ h(x))$ is close to the collision probability of the uniform distribution (on $H \times [L]$).
3. If the collision probability of Z is close to the collision probability of the uniform distribution (on the domain of Z) then Z is close to uniform.

□

Universal Hashing

The first family of hash functions which can be used for extractors are Carter-Wegman universal hash functions.

Definition 3.2 *$H = \{h : [N] \rightarrow [L]\}$ is called a universal family of hash functions, if for any $x_1 \neq x_2 \in [N]$, and for any $w_1, w_2 \in [L]$, $Prob_{h \in H}$ ($h(x_1) = w_1 \wedge h(x_2) = w_2$) $= \frac{1}{L^2}$.*

It is clear that a universal family of hash functions, has, in particular, 0 collision error. For all ranges of $1 \leq l \leq n$, there exist constructions of universal families H of hash functions of size $|H| = poly(N = 2^n)$.

Using these families and the construction presented above, we get, for every $n \leq m \leq 2n$ extractors with high degree $d = O(n)$, and with $k = m - d + O(\log \epsilon^{-1})$ which is optimal for k (for this choice of n, m, d). This degree is much higher than we desire, and the output size is is larger than what we are usually interested in (usually we want $m < n$).

We did get something useful though: the min-entropy on the left hand side is really extracted in full since the output randomness, m, is essentially the sum of the input min-entropy, k, and the extra randomness, d.

This construction is sufficient, as a starting point, for the composition of extractors described in section 3.2. A better construction (which is easier to use) is implied by better families of hash functions, described next.

Tiny Families of Hash Functions

Universal families of hash functions have 0-collision error, and, in fact, 0 collision error does imply that their size $|H|$ must be large $|H| \geq N$. On the other hand, our purposes do not quite require 0-collision error but only small collision error. This will allow us to reduce the size of the family of hash functions, and hence, reduce the extractor degree. It turns out that "tiny families of hash functions" [SZ94, GW94] can be built using small-biased distributions.

Lemma 3.2 *[SZ94, GW94] For all $1 \le L = 2^l \le N = 2^n$ and $\epsilon > 0$, There exist explicit families H of hash functions from $[N]$ to $[L]$, with ϵ collision error, and size with $|H| = poly(n, \epsilon, L)$.*

This translates to,

Corollary 3.3 *For every $M = 2^m \le N = 2^n$ and $\epsilon > 0$, there exists an explicit (k, ϵ)-extractor with $D = poly(n, \epsilon, M)$ and $k = m - d + O(\log \epsilon^{-1})$.*

Notice that this extractor is almost optimal, except that D is also polynomial in M. Thus, for small M, i.e. with size $M \le poly(n)$, we get an optimal extractor.

A useful way to view this is that we manage to multiply the number of truly random bits from d to $m = (1 + c)d$ $(c > 0)$ using a source with enough min-entropy to supply this increase.

3.2 Composing extractors

Another useful way to look at extractors is to consider them as multipliers of pure randomness (from d bits to m bits), as long as they are also given a source with enough min-entropy. They take a very short random string y, and output a long near-random string z. Of course, the extra randomness in z really comes from the source, X, Viewed this way, it is natural to take the output of one extractor and feed it into another extractor, compounding their "randomness multiplying" powers.

Composing Extractors

Definition 3.3 *Let $G_1 : [N_1] \times [D_1] \to [M_1]$, and let $G_2 : [N_2] \times [D_2] \to [M_2 = D_1]$ be extractors. Define $G_1 \circ G_2 : [N_1] \times [N_2] \times [D_2] \to [M_1]$ to be $G_1(x_1, G_2(x_2, y))$.*

Intuitively, it is clear that if X_1 has as much min-entropy in it as G_1 requires and X_2 as much min-entropy as G_2 requires then the output of $G_2 \circ G_1$ will indeed be close to random. A delicate issue to consider is the independence of the random variables X_1 and X_2: if they are indeed independent then this intuitive claim is true. If they are allowed to be correlated then the previous statement may not hold. The following key definition, first formulated in [CG88], turns out to suffice instead of total independence.

Definition 3.4 *Let X_1, X_2 be (possibly correlated) random variables taking values, respectively, in $[N_1]$ and $[N_2]$. We say that (X_1, X_2) is a block-wise (k_1, k_2) source if*

1. *$H_\infty(X_1) \ge k_1$.*
2. *For every fixed value x_1 of X_1, $H_\infty(X_2 | X_1 = x_1) \ge k_2$. ("$X_2 | X_1 = x_1$" denotes the marginal distribution on X_2 conditioned on the event $X_1 = x_1$.)*

Lemma 3.4 *Let $G_1 : [N_1] \times [D_1] \to [M_1]$ be a (k_1, ϵ_1)-extractor, and Let $G_2 : [N_2] \times [D_2 = M_1] \to [M_2]$ be a (k_2, ϵ_2)-extractor. Let X_1, X_2 be a block-wise (k_1, k_2) source, and let y be chosen in random. Then the output of $G_1 \circ G_2(x_1, x_2, y)$ is $(\epsilon_1 + \epsilon_2)$-close to uniform.*

Proof: (sketch) Denote $w = G_2(x_2, y)$, and consider this random variable W. Fix any value x_1. Conditioned on $X_1 = x_1$, the distribution of W is ϵ_2 close to uniform (since G_2 is an extractor and by condition 2 of being a block-wise source). However this is a true for every value x_1, thus the joint distribution of (X_1, W) is ϵ_2 close to the product distribution $X_1 \times Uniform$. Thus the distribution of $G_1(x_1, w)$ is ϵ_2-close to the distribution of $G_1(x_1, w')$, where w' is chosen uniformly in $[D_2 = M_1]$, independently from X_1. This last distribution is ϵ_1-close to uniform since G_1 is an extractor and by condition 1 of being a block-wise source. The lemma follows. □

This idea can of course be extended to allow composition of an arbitrary number of extractors. The truly random input, y_i, to each extractor is obtained from the output of the previous one, and they each get a separate "block" as the somewhat random input, x_i. The requirement that is needed for the output to be close to uniform is:

Definition 3.5 *[CG88] Let $B_1, ..., B_t$ be correlated random variables taking values, respectively, in $[N_1], ..., [N_t]$. We say that $(B_1, ..., B_t)$ is a block-wise $(k_1, ..., k_t)$ source if for each $1 \le i \le t$ and for every fixed sequence of values $x_1 ... x_{i-1}$, we have that $H_\infty(B_i | B_1 = x_1, ..., B_{i-1} = x_{i-1}) \ge k_i$.*

Having as input such a block-wise $(k_1, ..., k_t)$ source we can combine t extractors $G_1 ... G_t$ (with $D_{i-1} = M_i$ for all $1 < i \le t$, and G_i being a (k_i, ϵ_i)-extractor) to output m_1 near-random bits starting from D_t truly random ones. A useful way to view this is that $G_1 \circ \cdots \circ G_t$ multiplies the number of random bits by the product of the "randomness-multiplying" capabilities of each extractor. Of course, this multiplication requires an extra block-wise source, and not just any source with enough min-entropy.

3.3 Getting a Block-wise Source

By composing, as described above, the basic extractors constructed in section 3.1, we can directly extract essentially all the randomness from block-wise source using only a very small number of truly random bits. Our problem, though, is to extract randomness from an arbitrary distribution with enough min-entropy. The solution is to first convert a general distribution X into a block-wise source $B_1 ... B_t$, and then proceed using this composition.

The first step in getting a block-wise source from a general distribution is to get the first block $B = B_1$. It turns out that once we can do this, in an appropriate way, the other blocks can be obtained in essentially the same way. What we need is for B to have enough min-entropy, even though it is much shorter than the original string (so that enough min-entropy is left for further blocks.)

In [NZ93] it is shown that choosing a subset of the bits of X in a pairwise independent way suffices for this. Other ways of sampling a subset of the bits of X behave similarly.

Given an n bit string $x_1 ... x_n$, and a subset $S \subseteq \{1...n\}$, denote x_S to be the string obtained by concatenating the bits x_i for all $i \in S$ (in the natural order).

We will choose a subset S of $\{1...n\}$ of a given size l, in a pairwise independent way. We take the meaning of this to be performing the following procedure: use any distribution on sets $S \subseteq \{1...n\}$ such that for every $1 \le i < j \le n$: $Pr[i \in S \, and \, j \in S] = 1/l^2$. (The expected size of S is l, and it is known that, w.h.p, the actual size is close to l.) This may be achieved in a sample space of size $O(n^2)$, thus the description of S requires only $O(\log n)$ bits. If the resulting set is of size larger than l then delete all indices except for the first l ones; if it is smaller than l then add the first indices not in S to pad to size l.

Getting a Block Given a string $x_1 ... x_n$ and an index of a set S (of size l in a pairwise independent sample space), we output x_S.

Lemma 3.5 *[NZ93] For every distribution X and for for almost all choices of S, the distribution of X_S is close to some distribution W with $H_\infty(W) \ge \tilde{\Omega}(\frac{l}{n} H_\infty(X))$.*

Proof: (*intuition only*) Consider the entropy $H(X)$ of the source instead of the min-entropy. We associate with each bit i of X its conditional entropy $p_i = H(X_i | X_1 ... X_{i-1})$. A standard fact regarding conditional entropies implies that $\sum_i p_i = H(X)$. Since S is chosen in a pairwise independent way, w.h.p., $\sum_{i \in S} p_i$ is close to its expectation $l/n \cdot H(X)$. Again a standard calculation with conditional entropies will show that $H(X_S) \ge \sum_{i \in S} p_i$, which provides the lower bound required.

The above argument can indeed be, quite easily, made formal and does show that for almost all choices of S, $H(X_S) \ge \Omega(l/n \cdot H(X))$. The problem is that we require a lower bound on $H_\infty(X_S)$, which we can not obtain in a similar way since the "conditional min-entropies" of distributions do not behave as nicely as the conditional entropies.

The actual proof of this lemma proceeds by essentially carrying out the above argument seperately for each possible value of x, and then combining all the x's back together. Unfortunately, this argument is quite cumbersome and delicate. $\Box$

We can use the above lemma to convert any random source into a block-wise source. Let us start with a general random source X with k min-entropy. We start by extracting, as in the previous lemma, a block B_1 of length $l_1 << m$. Notice that for any value b_1 of B_1, $H_\infty(X \mid B_1 = b_1) \ge H_\infty(X) - \log Pr[B_1 = b_1]$. We expect $Pr[B_1 = b_1]$ to be about 2^{-l_1}. Values of b_1 which take significantly smaller probability can be safely ignored since they happen so rarely. This means that even conditioned on (almost) any value of B_1, X has sufficient min-entropy. Hence, choosing another subset of entries S_2 (independently from the first S), we can extract one more block B_2, which has high min-entropy even conditioned on the history b_1. Actually, as long as $l_1 + \ldots + l_{i-1} << m$, we can go on and extract another block B_i, which has high min-entropy even conditioned on the history.

Thus we get:

Lemma 3.6 *[NZ93] Let X be a distribution on $[N]$ with $H_\infty(X) \ge k$. Then for any choice of $l_1 ... l_t$ where $\sum l_i < k/2$, the above procedure extracts a source $B = B_1 \circ \ldots \circ B_t$ which is close to a block-wise $(k_1, ..., k_t)$ source with $k_i = \tilde{\Omega}(k l_i / n)$.*

Notice that this lemma gives non-trivial parameters as long as $k \ge \Omega(\sqrt{n})$. Next we shortly mention a different idea which works even for smaller values of k.

An Alternative Method

This alternative method for effectively getting a block-wise source was introduced in [Zuc91], was first used for the construction of dispersers in [SSZ95], and was extended to allow construction of extractors in [TaS96]. The intuition can again be best obtained by considering the entropy $H(X)$ instead of the min-entropy.

Let us again be given a source $X = (X_1 ... X_n)$ from which we aim to get a block-wise $(k_1 ... k_t)$ source. If we denote $e_i = H(X_1 ... X_i)$, then we have for all i, $0 \le e_i - e_{i-1} \le 1$. We also have

$e_0 = 0$ and $e_n \geq k$. For any value $k_1 \leq k$, there thus always exists an index i_1 such that $k_1 \leq H(X_1...X_{i_1}) \leq k_1 + 1$ – this is our first block, B_1. Now, the conditional entropy of X conditioned on B_1 is $H(X_{i_1+1}...X_n|X_1...X_{i_1}) = H(X) - H(X_1...X_{i_1}) \geq k - k_1$. We can thus continue and find i_2 such that $k_2 \leq H(X_{i_1+1}...X_{i_2}|X_1...X_{i_1}) \leq k_2 + 1$. This process can be continued as long as $\sum_j k_j < k$, and gives our block-wise source.

As before, this indeed gives blocks which have high entropy conditioned on the previous ones, but cannot directly give a real block-wise source, which requires high min-entropy. Again, the solution is to mimic the above idea for each input value $x_1...x_n$ seperately – which requires non-trivial details.

An important aspect omitted is how these $i_1, ...i_t$ are explicitly found. This indeed cannot be done without knowing the source X. However, it turns out that for dispersers we can simply try all the choices, and combine them together (i.e. take the union of edges that each choice implies). As long as t is relatively small, the total number of choices is manageble. Two more tricks can be used. In [SSZ95] a "universal" set of choices for $i_1...i_t$ with smaller (for the intersting cases – polynomial) size is presented, thus achieving optimal dispersers (for certain ranges of parameters). In [TaS96] a method by which these many choices for $i_1...i_t$ can be "merged" is presented, thus achieving near-optimal constructions of extractors.

3.4 Constructions of Extractors

In subsection 3.3 we saw how to convert any random source with high min-entropy into a block-wise source; in subsection 3.3 we saw how to extract randomness from such sources, using a composition of basic extractors; and in subsection 3.1 we saw how to build these basic extractors. Let us see several ways by which these ingridients can be put together in order to get good extractors.

A Construction of Good Extractors

We first discuss the basic strategy used in [Zuc90, Zuc91, NZ93, SZ94] and present it essentially in the form found in [SZ94].

Choose $t = O(\log n)$ blocks each of size $O(k/\log n)$ as described in subsection 3.3. This gives a (near) block-wise source. We now compose t extractors each which multiplies the number of random bits by a constant factor – as presented in subsection 3.1. We start with $O(\log n)$ bits, which after going through the composed t extractors get multiplied all the way to the min-entropy present in the final block, $\tilde{O}(k/n \cdot k/\log n)$ – which is our output ($m = \tilde{\Omega}(k^2/n)$).

The total number d of truly random bits used by this construction is the sum of the following: (A) $O(\log n)$ bits for the random input of the first extractor; (B) $O(\log n)$ bits for each of the t blocks – used to choose the pairwise independent set used to get the block. All together this gives $d = O(\log^2 n)$. As remarked in subsection 3.3. this can work as long as $k = H_\infty(X) \geq \tilde{\Omega(\sqrt{n})}$.

For $k = H_\infty(X) = \Omega(n)$, we can choose the sizes of the t blocks more carefully, making sure that the final block has length $\Omega(n)$, and thus we can extract $m = \Omega(n)$ bits.

Reducing d

We now show a simple idea from [SZ94], showing that by composing two of the extractors just designed we can reduce d even further.

Let us first choose a block of length $k/2$ and a second block of length $n' = 2^{O(\sqrt{\log n})}$. Choose an extractor for the second block as described above – taking into account that now the length is n' (instead of n). Thus, using $O(\log^2 n') = O(\log n)$ bits we can extract many more random bits. A simple calculation will show that the number of bits extracted is more than the $O(\log^2 n)$ bits required for extraction of randomness from the first block. The total number of bits required is twice $O(\log n)$ bits for getting the two blocks, and $O(\log n)$ bits for random input to the second block.

The error parameter ϵ of this construction depends on the smaller block and is $1/poly(n')$, which is larger than $1/poly(n)$. By carefully controlling the composition of many extractors of differing lengths, [SZ94, Zuc96] show that $d = O(\log n + \log \epsilon^{-1})$ suffices for any ϵ as long as $k = \Omega(n)$ (and even slightly less).

Increasing m

As observed in [WZ93], a simple repetition can increase the value of m. After using d_1 bits to extract m_1 bits from a source X with $H_\infty(X) \geq k$, we can use d_2 fresh random bits extract m_2 more bits from X, as long as we use an extractor that can extract random bits from sources with minentropy $k - m_1$. These m_2 bits are independent from the first m_1 bits since, even conditioned on any particular value of the first m_1 bits, X still has minentropy of at least $k - m_1$, and thus the extractor produces random bits.

The above idea uses $d_1 + d_2$ bits to extract $m_1 + m_2$ bits and can of course be repeated until $k - \sum m_i$ is too small. Repeating the previous construction $O(1)$ times allows extracting $m = k(1 - \eta)$ random bits using only $d = O(\log n)$ truly random ones for any $\eta > 0$ and $k = \Omega(n)$ [Zuc96].

Decreasing k

This is done using the "alternative method" sketched in section 3.3.

4 Applications

In this section we list some of the applications of extractors and dispersers. We use extractors and dispersers with different parameters. We will continue working with the parameters used so far, $N = 2^n$, $M = 2^m$, $K = 2^k$, $D = 2^d$, and ϵ. In many cases the parameters of the extractor or disperser used will be derived directly from the application in mind. In these cases we will "cleverly" define the application already with the appropriate names for parameters. All theorems thus implicitly start with "let G be any extractor with parameters N, M, K, D, ϵ, then...".

4.1 Simulating BPP using defective random sources

While randomized algorithms seem useful, the following question must be addressed before they can actually be used: How do computers get random bits? Currently some pseudo-random generator is used, usually without any proof that it "works". In many cases, these pseudo-random bits are not good enough as a replacement for the truly random bits needed by the algorithm.

An alternative solution is to rely on some physical source which produces some "real" randomness. A popular example of such sources are Zener Diodes, which (can be made to) produce quantum mechanical noise – thus are random since should be truly impossible to predict physically.

The problem is that although these sources contain a substantial amount of randomness, they do not output truly unbiased and independent random bits – thus they cannot be used directly in randomized algorithms.

A natural idea is to, deterministically, convert this source into truly random bits. For certain types of sources this can indeed be done. E.g. [Blu86] shows how it can be done if the source is a (known) Markov chain. For more general sources it can be shown that this cannot be done [SV86]. Instead, we may use the somewhat random source to indirectly simulate a given randomized algorithm.

The kind of simulation we have in mind is a black-box simulation: one that does not require us to understand the randomized algorithm in question. let us define this "universal" type of simulation explicitly for randomized algorithms which have one-sided error (RP-type algorithms.)

A black box simulation of RP

We want to run a randomized algorithm which has one-sided error and requires m random bits; we are given a random source which has distribution X on $[N]$, from which we can get a single element $x \in [N]$. From x we generate, deterministically, D different strings $z_1 ... z_D \in [M]$, ($M = 2^m$,) and then run the original algorithm on each z_i. We accept if the original algorithm accepted for at least one of the z_i's. We consider D to be the time of the simulation (thus counting only invocations of the original algorithm as a single time unit, each). We call the simulation polynomial if D is polynomial in n[1].

Definition 4.1 *The above procedure is a black box simulation of RP using source X with error α if for every set $W \subseteq [M]$, $|W| \geq M/2$, $Pr_x[exists\ i : z_i \in W] \geq 1 - \alpha$.*

If the original RP algorithm accepted then for at least half the possible $z \in [M]$ it accepts – call this set W. In this case the simulation, using X, accepts with probability at least $1 - \alpha$. If, on the other hand, the original algorithm rejected, then for no $z \in [M]$ does it accept (since the error is one-sided), and thus the simulation will reject. The analogous procedure for simulating BPP, does not accept whenever one of the z_i's causes the original algorithm to accept but rather accepts according to some other condition which depends on the answers obtained for all z_i's – usually their majority vote.

Our usual goal would be to obtain this simulation without knowing the exact distribution of the somewhat random source X. Our simulation should only rely on the fact that X has some given property, which is the model of "somewhat randomness". Clearly, physicists should try to produce physical source where this property is as strong as possible, while computer scientists should aim to rely on a property which is as weak as possible.

Several models of "somewhat random" sources were considered in the literature. Santha and Vazirani [SV86], Vazirani [Vaz87a, Vaz86, Vaz87b] and Vazirani and Vazirani [VV85] studied a class of sources, called "slightly random sources", and showed that BPP can be simulated given such a source. Chor and Goldreich [CG88] generalized this model, and showed that BPP can be simulated even using the more general source. Many authors studied other restricted classes of random sources (e.g. [CW89, CGH+85, LLS89]). Finally, Zuckerman [Zuc90] suggested a general model generalizing

[1] Thus, in order for the whole simulation to be polynomial in terms of the original algorithm it is also required that $n = poly(m)$, and that $z_1 ... z_D$ can be computed in polynomial time – which is the definition used in [CW89].

all the previous models. His model was simply all random sources having high min-entropy. Later [Zuc91] he showed that BPP can be simulated given such a source (with enough min-entropy).

In a very basic sense, Zuckerman's model is the most general source we can think off.

Lemma 4.1 *If a black box RP simulation (in time D) requiring m bits can be obtained, with error α, using source X, then X is $O(\alpha)$-close to some distribution X' with $H_\infty(X') \geq m - \log D - 1$.*

Proof: (sketch) Consider the set S of all the elements x which get a probability higher than $2^m/(2D)$. This total probability cannot be greater than α since otherwise we can have W be the set of all elements z which are never produced by an element of S, a set which has size $|W| \geq M - |S| \cdot D \geq M/2$. The distribution X' is obtained by zeroing the probability of elements in S and correcting the total probability to 1. □

Using extractors a simulation good for all sources with enough min-entropy is easily obtained.

Theorem: *RP (and BPP) with m bits can be simulated (in time D) using any random source X with $H_\infty(X) \geq k$. (The error is $exp(k - H_\infty(X)$.)*

With current constructions, we get a polynomial time simulation (i.e. $D = poly(n)$) for RP as long as $H_\infty(X) \geq n^\delta$ for any $\delta > 0$ [SSZ95]; a polynomial time simulation for BPP as long as $H_\infty(X) = \Omega(n)$ (or even slightly less) [Zuc93]; and a quasi-polynomial time simulation for BPP for any $H_\infty(X)$ [TaS96].

Proof: (for RP) Get x from the distribution, and set $z_i = G(x,i)$ for $i = 1...D$, where G is a disperser with $\epsilon < 1/2$. Let us calculate the probability that this simulation fails, i.e. that all $z_i \notin W$. Let us look at the set S of these "bad" x's – it cannot be larger than K, since the neighbor set of any set of size K is larger than $M/2$ and thus intersects W. The total probability assigned by X to this set of "bad" x's can be at most $|S| \cdot 2^{-H_\infty(X)}$. □

A similar argument for BPP uses an extractor.

4.2 Deterministic amplification

It turns out that extractors and dispersers can be used to decrease the probability of error of randomized algorithms in several settings. We describe some of these settings in this subsection.

4.2.1 Basic Amplification

Our goal now is to convert a BPP (or RP) algorithm that uses m random bits and has error, say, $1/4$, into one that errs with probability at most 2^{-t}. A trivial solution is to run the original algorithm $O(t)$ times using independent random bits, and take the majority vote – this requires $O(tm)$ random bits. We want to achieve this using as few random bits as possible.

This problem, known as the "deterministic amplification" problem, was extensively studied [KPS85, CG89, IZ89, CW89]. Using expanders, this can be done using only $n + O(t)$ random bits [AKS87, IZ89, CW89]. Sipser [Sip88] defined dispersers as a tool which implies stronger RP amplification. Using extractors, we obtain BPP amplification.

Theorem: *If L is accepted by a BPP algorithm using m random bits and error $1/4$, then L is also accepted by a BPP algorithm using n random bits and having error K/N.*

Using current constructions we can use only $n = (1+\alpha)(m+t)$ bits to get error 2^{-t}, for any fixed $\alpha > 0$ [Zuc93, Zuc96].

Proof: Use an extractor G with $\epsilon < 1/4$.

The Algorithm: Choose randomly $x \in [N]$. For any $z \in \Gamma(x)$ run the machine accepting L with z as the random string, and decide according to the majority of the results.

Correctness: Denote $W \subseteq [M]$ the set of witnesses z leading to the wrong answer by the machine accepting L. Thus $|W| \leq M/4$. Let $B \subseteq [N]$ be the set of "bad" x's – those with most of their neighbors in W. The simulation arrives at the wrong result iff a "bad" x is chosen. Since G is an extractor, $|B| < K$, since otherwise a uniform distribution on B would, on one hand have min-entropy k, but on the other hand, $\Gamma(B)$ is far from uniform since it gives weight of at least $1/2$ to a set W of size at most $1/4$. □

It can easily be seen that if we want to amplify an RP algorithm we can use dispersers instead of extractors.

4.2.2 Oblivious Sampling

The above simulation may be generalized to give what is called an oblivious sampler. These are sampling procedures that give a good estimate for the expected value of a real valued function on some finite domain.

Definition 4.2 *[BR94] An oblivious (δ, ϵ)-sampler is a deterministic function that for each $x \in [N]$ produces a set $\Gamma(x) = \{z_1 ... z_D\} \subseteq [M]$*

such that for every function $f : [M] \rightarrow [0,1]$, (where $[0,1]$ is the real interval between 0 and 1),

$$Pr_x[|\frac{\sum_{i\in[D]} f(z_i)}{D} - \frac{\sum_{z\in[M]} f(z)}{M}| \geq \epsilon] \leq \delta$$

.

For given ϵ, δ, and m, our wish is to have D be polynomial and n be as small as possible. Oblivious samplers were constructed in [BR94] who used them for interactive proof systems. The best known results to date, which use extractors, appears in [Zuc96].

Theorem: *There exist explicitly constructible (δ, ϵ) oblivious samplers where $\delta = 2K/N$.*

Using the best constructions of extractors [Zuc96], we get oblivious samplers with $n = (1+\alpha)(m + \log\delta^{-1})$ and $D = poly(m, \epsilon^{-1}, \log\delta^{-1})$ (for any $\alpha > 0$).

Proof: Take an extractor G and use the $\Gamma(x)$ from the extractor. Denote the expected value of f by $e = (\sum_{z\in[M]} f(z))/M$. The proof follows by considering the set $S_<$ of x's such that $(\sum_{z\in\Gamma(x)} f(z))/D < e - \epsilon$, and separately $S_>$, the set of x's such that $(\sum_{z\in\Gamma(x)} f(z))/D > e+\epsilon$. These sets must each be of size less than K, since otherwise the uniform distribution on one of them has high enough min-entropy and thus gives near (to within ϵ) uniform distribution on the z's. This cannot be true since an ϵ-close to uniform distribution on the z's gives an ϵ-close estimate of e, in contradiction to each element in $S_<$ (resp, $S_>$) erring on e by at least ϵ. $\square$

4.2.3 Approximating Clique

Zuckerman [Zuc93] showed how deterministic amplification obtained by extractors implies that

Theorem: *[Zuc93] Approximating $log(Clique(G))$ to within an constant factor is $\tilde{NP}$–hard.*

In fact Zuckerman showed that, for any constant j, approximating the j'th iterated log of clique to within any constant is $\tilde{NP}$–hard. For the precise statement see [Zuc93, SZ94].

The proof itself builds on the known hardness results for approximating clique [AS92b, ALM+92, FGL+91], and is best understood when viewing it as deterministic amplification for PCP systems.

4.2.4 Time vs. Space

Sipser [Sip88] defined dispersers in order to obtain the following theorem (which became a theorem only with the recent constructions of [SSZ95]). Again, the dispersers are needed for deterministic amplification.

Theorem: *If $P \neq RP$ then there is some $0 < \alpha$ s.t. for any (nice) function $t = t(n)$, $Time(t) \subseteq ioSpace(t^{1-\alpha})$, where $ioSpace(s)$ is the class of languages solvable by algorithms that for infinitely many inputs use at most space s.*

This is an unexpected connection between the question of the power of randomness and the (seemingly unrelated) question of time vs. space.

4.3 Explicit Graphs With Random Properties

In this section be present some explicit constructions of certain kinds of graphs. In these cases, the "random-like" properties of extractors and dispersers suffice as a replacement for using random graphs, and thus convert a non-constructive proof to a construction.

4.3.1 Super concentrators

DEFINITION 4.3 *[Pip77] Let $H = (V, E)$ be a directed graph with a specified subset $I \subset V$ of nodes called input nodes, and a disjoint subset, $O \subseteq V$, called output nodes. Assume $|I| = |O| = N$. H is called a super-concentrator if for any sets $W \subseteq I$, $Z \subseteq O$ of size K each, there are at least K vertex-disjoint paths from W to Z.*

The parameters of interest are, first, the *size* which is the number of edges in H, and second, the *depth* which is the length of the longest directed path in it.

Gabber and Galil [GG81] constructed, using expanders, the first explicit linear size super concentrators. The graph they construct has $O(log(N))$ depth. For depth 2, non-constructive proofs show that super concentrators of depth 2 and size $O(N \cdot log^2(N))$ exists [Pip82]. Meshulam [Mes84] showed an explicit depth 2, super concentrator of size $O(N^{1+1/2})$. [WZ93] give a construction based on extractors.

Theorem: *(following [WZ93]) There is an explicit super concentrator of depth 2 and size $N \cdot O(\sum_{k=1}^{n} D_k 2^k/M_k)$, where D_k, M_k are the parameters for dispersers with $K = 2^k$.*

Current explicit extractors [TaS96] imply depth 2 super concentrators of size $N \cdot 2^{polyloglog(N)}$.

Proof:

Meshulam [Mes84] showed that H is a super concentrator of depth 2, iff for any $1 \leq K' \leq N$ and any two sets $W \subseteq I, Z \subseteq O$ of size K' each, there are at least K' common neighbors. We will build a depth 2 graph with this property.

Build the graph as follows:

- I and O each have $N = 2^n$ vertices.
- The other vertices are partitioned into disjoint sets C_k for $1 \leq k \leq n$. $|C_k| = 4K$.
- For each k, we partition C_k into $4K/M_k$ equal-sized sets each of size M_k and put a disperser between I and each set, as well as between O and each set. (We take an disperser with $\epsilon = 1/4$.)

Now take any two sets $W \subseteq I$ and $Z \subseteq O$ of size K' each. Let $K = 2^k$ be the largest power of 2 which is less or equal to K'. Consider only edges connecting W and Z to C_k. In each one of the $4K/M_k$ different subsets of C_k, we have that $\Gamma(W), \Gamma(Z) \geq 3M_k/4$ (since we put a disperser there). It follows that in each of these subsets, $|\Gamma(W) \cap \Gamma(Z)| \geq M_k/2$. Taking all $4K/M_k$ subsets together, the intersection size is at least $2K \geq K'$. □

Wigderson and Zuckerman showed that such a construction would also imply an explicit linear size super concentrator of small depth. With [TaS96], of depth $polyloglog(N)$.

4.3.2 Highly expanding graphs

We now consider expanders with very strong expansion properties.

Definition 4.4 *A graph H on N vertices is called a K-expander if any two sets W, Z of vertices, $|W| = |Z| = K$, have a common neighbor.*

Since any two sets of size K have a common neighbor, a set of size K must have at least $N - K$ neighbors and therefore the degree of the graph is at least $(N - K)/K$. We would like to explicitly build K-expanding graphs with degree as close as possible to N/K. The eigenvalue methods for constructing expanders give such expanders with degree $O(N^2/K^2)$ which is quadratic in the above lower bound and trivial for $K \leq \sqrt{N}$. [WZ93] show how dispersers can be used to construct expanders, for all values of K, with near-optimal degree.

Theorem: *(following [WZ93]) There are constructible K-expanding graph with N vertices, and maximum degree $O(ND^2/M)$.*

Using the extractors of [TaS96] we get maximum degree of $\frac{N}{K} \cdot exp(polyloglogN)$ for any value of K. Let us remark that the construction takes time polynomial in N. It does not allow computing whether an edge exists between two given vertices in time polynomial in n.

Proof: We use a disperser G with $\epsilon = 1/5$.

Let $B \subseteq [M]$ be the set of z's of degree greater than $2DN/M$. Notice that $|B| \leq M/2$ since the total number of edges in the disperser is DN. Now we connect a vertex $x_1 \in [N]$ to a vertex $x_2 \in [N]$ if they share a neighbor not in B, i.e. if there exist $y_1, y_2 \in [D]$ such that $G(x_1, y_1) = G(x_2, y_2) \notin B$.

Now for every set $W \subseteq [N], |W| \geq K$ we have, in the disperser, $\Gamma(W) \geq 4M/5$, and thus $|\Gamma(W) \cap \Gamma(Z)| \geq 3M/5 \geq |B|$. Thus, in the expander we have just built, W and Z will share a neighbor.

The number of edges is bounded from above $D \cdot 2DN/M$.

□

Pippenger [Pip87] showed that good explicit highly expanding graphs, yield good algorithms for "sorting in rounds" and for "selecting in rounds".

4.4 Pseudo-random Generators

Extractors can also be used to construct pseudo-random generators which fool certain classes of algorithms. The following theorem of [NZ93] improves on previous results of [AKS87].

Theorem: *[NZ93] There exists a pseudo-random generator which converts $O(S)$ truly random bits into $poly(S)$ bits which look random to any algorithm which runs in space S. The generator runs in $O(S)$ space and $poly(S)$ time.*

Proof: (sketch) We will build a generator that, for any t, converts $n + td$ bits into tm bits which look random (to within ϵ) to space $S = n - k - O(\log \epsilon^{-1})$ algorithms. Current extractors G with $k, m = \Omega(n)$ imply, for all $t \geq S$, conversion of $O(t)$ bits to $\Omega(t \cdot S/\log S)$ bits. To get any $poly(S)$ factor gain in the number of bits simply compose the pseudo-random generators. A more careful type of iteration with improved parameters can be found in [AW95].

The Pseudo-random Generator

Input: x, $y_1 ... y_t$.

Output: $G(x, y_1), ..., G(x, y_t)$.

for each $i = 0...t$, consider the probability distribution A_i on the internal state of the algorithm after im random bits $z_1...z_i$ have been read by the algorithm versus the distribution A'_i in the case that pseudo-random bits $G(x, y_1)...G(x, y_i)$ have been read. The proof that this generator indeed looks random to space S algorithms proceeds by induction on i of the following claim:

Claim: The distribution A_i is close to the distribution A'_i.

Proof: Fix a typical internal state a of the algorithm. We will show that, conditioned on a being the state reached after $(i-1)m$ bits were read, A_i and A'_i are close. The claim will follow by averaging over all a's and noticing that the probabilities of getting to a, in A_{i-1} vs. A'_{i-1}, are close by the induction hypothesis.

Let us see what is the conditional distribution on X conditioned upon a being the state reached after $(i-1)m$ bits were read. The space bound S on the algorithm implies that there are at most 2^S different possible values of a, thus we would expect that the probability of getting to a is about 2^{-S} (otherwise a can be ignored). Looking at the conditional probabilities we have that $H_\infty(X|a) \geq n - S \geq k$. Therefore, the distribution of $G(x, y_i)$ is nearly uniform under this conditioning, since G is an extractor. The distributions A_i and A'_i, under this conditioning, are completely determined by z and $G(x, y_i)$, resp., and thus are close to each other.

□

Acknowledgements

I thank Amnon Ta-Shma for his help in writing this survey. I thank David Zuckerman, Avi Wigderson, and, again, Amnon Ta-Shma for many helpful discussions.

References

[AKS87] Ajtai, Komlos, and Szemeredi. Deterministic simulation in LOGSPACE. In *ACM Symposium on Theory of Computing (STOC)*, 1987.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science*, IEEE, pages 14–23, 1992.

[Alo85] Alon. Expanders, sorting in rounds and superconcentrators of limited depth. In *ACM Symposium on Theory of Computing (STOC)*, 1985.

[AS92a] N. Alon and J. H. Spencer. *The Probabilistic Method.* John Wiley and Sons, 1992.

[AS92b] S. Arora and S. Safra. Probabilistic checking of proofs; a new characterization of NP. In *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science*, pages 2–13, 1992.

[AW95] R. Armoni and A. Wigderson. manuscript. 1995.

[Blu86] M. Blum. Independent unbiased coin flips from a correlated biased source: a finite markov chain. *Combinatorica*, 6(2):97–108, 1986.

[BR94] Bellare and Rompel. Randomness-efficient oblivious sampling. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.

[CG89] Chor and Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5, 1989.

[CGH+85] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky. The bit extraction problem and t-resilient functions. In *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, pages 396–407, 1985.

[CW89] A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *Proceedings of the 30th Annual IEEE Symposium on the Foundations of Computer Science*, pages 14–19, 1989.

[FGL+91] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science*, IEEE, pages 2–12, 1991.

[GG81] Gabber and Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22, 1981.

[GW94] O. Goldreich and A. Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, ACM, pages 574–583, 1994.

[ILL89] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, ACM, pages 12–24, 1989.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the 30th Annual IEEE Symposium on the Foundations of Computer Science*, IEEE, pages 248–253, 1989.

[KPS85] R. Karp, N. Pippernger, and M. Sipser. A time randomness tradeoff. In *AMS Conference on Probabilistic Computational Complexity*, 1985.

[LLS89] Lichtenstein, Linial, and Saks. Some extremal problems arising from discrete control processes. *Combinatorica*, 9, 1989.

[Mes84] R. Meshulam. A geometric construction of a superconcentrator of depth 2. *Theoretical Computer Science*, 32:215–219, 1984.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[NZ93] N. Nisan and D. Zuckerman. More deterministic simulation in logspace. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, ACM, pages 235–244, 1993.

[Pip77] Pippenger. Superconcentrators. *SICOMP: SIAM Journal on Computing*, 1977.

[Pip82] Pippenger. Superconcentrators of depth 2. *JCSS: Journal of Computer and System Sciences*, 24, 1982.

[Pip87] N. Pippenger. Sorting and selecting in rounds. *SIAM Journal on Computing*, 16:1032–1038, 1987.

[Sip88] Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36, 1988.

[SSZ95] M. Saks, A. Srinivasan, and S. Zhou. Explicit dispersers with polylog degree. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, ACM, 1995.

[SV86] M. Santha and U. Vazirani. Generating quasi-random sequences from slightly random sources. *J. of Computer and System Sciences*, 33:75–87, 1986.

[SZ94] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, 1994.

[TaS96] A. TaShma. On extracting randomness from weak random sources. In *STOC 1996*, 1996.

[Vaz86] U. Vazirani. *Randomness, Adversaries and Computation.* PhD thesis, University of California, Berkeley, 1986.

[Vaz87a] U. Vazirani. Efficiency considerations in using semi-random sources. In *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, ACM, pages 160–168, 1987.

[Vaz87b] U. Vazirani. Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources. *Combinatorica*, 7(4):375–392, 1987.

[VV85] U. Vazirani and V. Vazirani. Random polynomial time is equal to slightly-random polynomial time. In *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, IEEE, pages 417–428, 1985.

[WZ93] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, ACM, pages 245–251, 1993.

[Zuc90] D. Zuckerman. General weak random sources. In *Proceedings of the 31st Annual IEEE Symposium on the Foundations of Computer Science*, pages 534–543, 1990.

[Zuc91] D. Zuckerman. Simulating BPP using a general weak random source. In *Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science*, pages 79–89, 1991.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *Proceedings of the 8th Structures in Complexity Theory*, IEEE, pages 305–312, 1993.

[Zuc96] D. Zuckerman. Randomness-optimal sampling, extractors, and constructive leader election. In *STOC*, 1996.