2-Source Dispersers for Sub-Polynomial Entropy and Ramsey Graphs Beating the Frankl-Wilson Construction

Boaz Barak *
Department of Computer Science
Princeton University
boaz@cs.princeton.edu

Anup Rao †
Department of Computer Science
University of Texas at Austin
arao@cs.utexas.edu

Ronen Shaltiel ‡
University of Haifa
Mount Carmel
Haifa, Israel
ronen@cs.haifa.ac.il

Avi Wigderson §
Institute for Advanced Study
Princeton
New Jersey
avi@math.ias.edu

ABSTRACT

The main result of this paper is an explicit disperser for two independent sources on n bits, each of entropy $k = n^{o(1)}$. Put differently, setting $N = 2^n$ and $K = 2^k$, we construct explicit $N \times N$ Boolean matrices for which no $K \times K$ submatrix is monochromatic. Viewed as adjacency matrices of bipartite graphs, this gives an explicit construction of K-Ramsey *bipartite* graphs of size N.

This greatly improves the previous bound of $k = o(n)$ of Barak, Kindler, Shaltiel, Sudakov and Wigderson [4]. It also significantly improves the 25-year record of $k = \tilde{O}(\sqrt{n})$ on the special case of Ramsey graphs, due to Frankl and Wilson [9].

The construction uses (besides "classical" extractor ideas) almost all of the machinery developed in the last couple of years for extraction from independent sources, including:

- Bourgain's extractor for 2 independent sources of some entropy rate $< 1/2$ [5]
- Raz's extractor for 2 independent sources, one of which has any entropy rate $> 1/2$ [18]
- Rao's extractor for 2 independent block-sources of entropy $n^{\Omega(1)}$ [17]
- The "Challenge-Response" mechanism for detecting "entropy concentration" of [4].

The main novelty comes in a bootstrap procedure which allows the Challenge-Response mechanism of [4] to be used with sources of less and less entropy, using recursive calls to itself. Subtleties arise since the success of this mechanism depends on restricting the given sources, and so recursion constantly changes the original sources. These are resolved via a new construct, in between a disperser and an extractor, which behaves like an extractor on sufficiently large subsources of the given ones.

This version is only an extended abstract, please see the full version, available on the authors' homepages, for more details.

Categories and Subject Descriptors

G.2.2 [**Mathematics of Computing**]: Discrete Mathematics—*Graph algorithms*

General Terms

Theory, Algorithms

Keywords

Dispersers, Ramsey Graphs, Independent Sources, Extractors

*Supported by a Princeton University startup grant.

†Most of this work was done while the author was visiting Princeton University and the Institute for Advanced Study. Supported in part by an MCD fellowship from UT Austin and NSF Grant CCR-0310960.

‡This research was supported by the United States-Israel Binational Science Foundation (BSF) grant 2004329.

§This research was supported by NSF Grant CCR-0324906.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
STOC'06, May21–23, 2006, Seattle, Washington, USA.
Copyright 2006 ACM 1-59593-134-1/06/0005 ...$5.00.

1. INTRODUCTION

This paper deals with randomness extraction from weak random sources. Here a weak random source is a distribution which contains some entropy. The extraction task is to design efficient algorithms (called *extractors*) to convert this entropy into useful form, namely a sequence of independent unbiased bits. Beyond the obvious motivations (potential use of physical sources in pseudorandom generators and in derandomization), extractors have found applications in a variety of areas in theoretical computer science where randomness does not seem an issue, such as in efficient constructions of communication networks [24, 7], error correcting codes [22, 12], data structures [14] and more.

Most work in this subject over the last 20 years has focused on what is now called *seeded* extraction, in which the extractor is given as input not only the (sample from the) defective random source, but also a few truly random bits (called the *seed*). A comprehensive survey of much of this body of work is [21].

Another direction, which has been mostly dormant till about two years ago, is (seedless, deterministic) extraction from a few *independent* weak sources. This kind of extraction is important in several applications where it is unrealistic to have a short random seed or deterministically enumerate over its possible values. However, it is easily shown to be impossible when only one weak source is available. When at least 2 independent sources are available extraction becomes possible in principle. The 2-source case is the one we will focus on in this work.

The rest of the introduction is structured as follows. We'll start by describing our main result in the context of Ramsey graphs. We then move to the context of extractors and disperser, describing the relevant background and stating our result in this language. Then we give an overview of the construction of our dispersers, describing the main building blocks we construct along the way. As the construction is quite complex and its analysis quite subtle, in this proceedings version we try to abstract away many of the technical difficulties so that the main ideas, structure and tools used are highlighted. For that reason we also often state definitions and theorems somewhat informally.

1.1 Ramsey Graphs

Definition 1.1. *A graph on N vertices is called a K-*Ramsey Graph *if it contains no clique or independent set of size K.*

In 1947 Erdős published his paper inaugurating the *Probabilistic Method* with a few examples, including a proof that *most* graphs on $N = 2^n$ vertices are $2n$-Ramsey. The quest for constructing such graphs explicitly has existed ever since and lead to some beautiful mathematics.

The best record to date was obtained in 1981 by Frankl and Wilson [9], who used intersection theorems for set systems to construct N-vertex graphs which are $2^{\sqrt{n \log n}}$-Ramsey. This bound was matched by Alon [1] using the *Polynomial Method*, by Grolmusz [11] using low rank matrices over rings, and also by Barak [2] boosting Abbot's method with almost k-wise independent random variables (a construction that was independently discovered by others as well). Remarkably all of these different approaches got stuck at essentially the same bound. In recent work, Gopalan [10] showed that other than the last construction, all of these can be viewed as coming from low-degree symmetric representations of the OR function. He also shows that any such symmetric representation cannot be used to give a better Ramsey graph, which gives a good indication of why these constructions had similar performance. Indeed, as we will discuss in a later section, the $\sqrt{n}$ entropy bound initially looked like a natural obstacle even for our techniques, though eventually we were able to surpass it.

The analogous question for bipartite graphs seemed much harder.

Definition 1.2. *A bipartite graph on two sets of N vertices is a K-*Ramsey Bipartite Graph *if it has no $K \times K$ complete or empty bipartite subgraph.*

While Erdős' result on the abundance of $2n$-Ramsey graphs holds as is for bipartite graphs, until recently the best explicit construction of bipartite Ramsey graphs was $2^{n/2}$-Ramsey, using the Hadamard matrix. This was improved last year, first to $o(2^{n/2})$ by Pudlak and Rődl [16] and then to $2^{o(n)}$ by Barak, Kindler, Shaltiel, Sudakov and Wigderson [4].

It is convenient to view such graphs as functions $f : (\{0,1\}^n)^2 \to \{0,1\}$. This then gives exactly the definition of a disperser.

Definition 1.3. *A function $f : (\{0,1\}^n)^2 \to \{0,1\}$ is called a* 2-source disperser for entropy k *if for any two sets $X, Y \subset \{0,1\}^n$ with $|X| = |Y| = 2^k$, we have that the image $f(X, Y)$ is $\{0,1\}$.*

This allows for a more formal definition of *explicitness*: we simply demand that the function f is computable in polynomial time. Most of the constructions mentioned above are explicit in this sense.[1]

Our main result (stated informally) significantly improves the bounds in both the bipartite and non-bipartite settings:

Theorem 1.4. *For every N we construct polynomial time computable bipartite graphs which are $2^{n^{o(1)}}$-Ramsey. A standard transformation of these graphs also yields polynomial time computable ordinary Ramsey Graphs with the same parameters.*

1.2 Extractors and Dispersers from independent sources

Now we give a brief review of past relevant work (with the goal of putting this paper in proper context) and describe some of the tools from these past works that we will use. We start with the basic definitions of k-sources by Nisan and Zuckerman [15] and of extractors and dispersers for independent sources by Santha and Vazirani [20].

Definition 1.5 ([15], see also [8]). *The* min-entropy *of a distribution X is the maximum k such that for every element x in its support,* $\Pr[X = x] \leq 2^{-k}$. *If X is a distribution on strings with min-entropy at least k, we will call X a k-source* [2].

To simplify the presentation, in this version of the paper we will assume that we are working with entropy as opposed to min-entropy.

Definition 1.6 ([20]). *A function $f : (\{0,1\}^n)^c \to \{0,1\}^m$ is a c-source (k, ϵ)* extractor *if for every family of c independent k-sources $X_1, \cdots, X_c$, the output $f(X_1, \cdots, X_c)$*

[1] The Abbot's product based Ramsey-graph construction of [3] and the bipartite Ramsey construction of [16] only satisfy a weaker notion of explicitness.

[2] It is no loss of generality to imagine that X is uniformly distributed over some (unknown) set of size 2^k.

is a ϵ-close [3] *to uniformly distributed on m bits. f is a* disperser *for the same parameters if the output is simply required to have a support of relative size $(1-\epsilon)$.*

To simplify the presentation, in this version of the paper, we will assume that $\epsilon = 0$ for all of our constructions.

In this language, Erdős' theorem says that most functions $f : (\{0,1\}^n)^2 \to \{0,1\}$ are dispersers for entropy $1 + \log n$ (treating f as the characteristic function for the set of edges of the graph). The proof easily extends to show that indeed most such functions are in fact extractors. This naturally challenges us to find explicit functions f that are 2-source extractors.

Until one year ago, essentially the only known explicit construction was the Hadamard extractor Had defined by $\mathsf{Had}(x, y) = \langle x, y\rangle (\mod 2)$. It is an extractor for entropy $k > n/2$ as observed by Chor and Goldreich [8] and can be extended to give $m = \Omega(n)$ output bits as observed by Vazirani [23]. Over 20 years later, a recent breakthrough of Bourgain [5] broke this "1/2 barrier" and can handle 2 sources of entropy $.4999n$, again with linear output length $m = \Omega(n)$. This seemingly minor improvement will be crucial for our work!

Theorem 1.7 ([5]). *There is a polynomial time computable 2-source extractor $f : (\{0,1\}^n)^2 \to \{0,1\}^m$ for entropy $.4999n$ and $m = \Omega(n)$.*

No better bounds are known for 2-source extractors. Now we turn our attention to 2-source dispersers. It turned out that progress for building good 2-source dispersers came via progress on extractors for more than 2 sources, all happening in fast pace in the last 2 years. The seminal paper of Bourgain, Katz and Tao [6] proved the so-called "sum-product theorem" in prime fields, a result in arithmetic combinatorics. This result has already found applications in diverse areas of mathematics, including analysis, number theory, group theory and ... extractor theory. Their work implicitly contained dispersers for $c = O(\log(n/k))$ independent sources of entropy k (with output $m = \Omega(k)$). The use of the "sum-product" theorem was then extended by Barak et al. [3] to give extractors with similar parameters. Note that for linear entropy $k = \Omega(n)$, the number of sources needed for extraction c is a constant!

Relaxing the independence assumptions via the idea of repeated condensing, allowed the reduction of the number of independent sources to $c = 3$, for extraction from sources of any linear entropy $k = \Omega(n)$, by Barak et al. [4] and independently by Raz [18].

For 2 sources Barak et al. [4] were able to construct dispersers for sources of entropy $o(n)$. To do this, they first showed that if the sources have extra structure (*block-source* structure, defined below), even extraction is possible from 2 sources. The notion of block-sources, capturing "semi independence" of parts of the source, was introduced by Chor and Goldreich [8]. It has been fundamental in the development of seeded extractors and as we shall see, is essential for us as well.

Definition 1.8 ([8]). *A distribution $X = X_1, \ldots, X_c$ is a c-block-source of (block) entropy k if every block X_i has entropy k even conditioned on fixing the previous blocks $X_1, \cdots, X_{i-1}$ to arbitrary constants.*

[3] The error is usually measured in terms of ℓ_1 distance or variation distance.

This definition allowed Barak et al. [4] to show that their extractor for 4 independent sources, actually performs as well with only 2 independent sources, as long as both are 2-block-sources.

Theorem 1.9 ([4]). *There exists a polynomial time computable extractor $f : (\{0,1\}^n)^2 \to \{0,1\}$ for 2 independent 2-block-sources with entropy $o(n)$.*

There is no reason to assume that the given sources are block-sources, but it is natural to try and reduce to this case. This approach has been one of the most successful in the extractor literature. Namely try to partition a source X into two blocks $X = X_1, X_2$ such that X_1, X_2 form a 2-block-source. Barak et al. introduced a new technique to do this reduction called the *Challenge-Response mechanism*, which is crucial for this paper. This method gives a way to "find" how entropy is distributed in a source X, guiding the choice of such a partition. This method succeeds only with small probability, dashing the hope for an extractor, but still yielding a disperser.

Theorem 1.10 ([4]). *There exists a polynomial time computable 2-source disperser $f : (\{0,1\}^n)^2 \to \{0,1\}$ for entropy $o(n)$.*

Reducing the entropy requirement of the above 2-source disperser, which is what we achieve in this paper, again needed progress on achieving a similar reduction for extractors with more independent sources. A few months ago Rao [17] was able to significantly improve all the above results for $c \geq 3$ sources. Interestingly, his techniques do not use arithmetic combinatorics, which seemed essential to all the papers above. He improves the results of Barak et al. [3] to give $c = O((\log n)/(\log k))$-source extractors for entropy k. Note that now the number c of sources needed for extraction is constant, even when the entropy is as low as n^δ for any constant δ!

Again, when the input sources are block-sources with sufficiently many blocks, Rao proves that 2 independent sources suffice (though this result does rely on arithmetic combinatorics, in particular, on Bourgain's extractor).

Theorem 1.11 ([17]). *There is a polynomial time computable extractor $f : (\{0,1\}^n)^2 \to \{0,1\}^m$ for 2 independent c-block-sources with block entropy k and $m = \Omega(k)$, as long as $c = O((\log n)/(\log k))$.*

In this paper (see Theorem 2.7 below) we improve this result to hold even when only one of the 2 sources is a c-block-source. The other source can be an arbitrary source with sufficient entropy. This is a central building block in our construction. This extractor, like Rao's above, critically uses Bourgain's extractor mentioned above. In addition it uses a theorem of Raz [18] allowing seeded extractors to have "weak" seeds, namely instead of being completely random they work as long as the seed has entropy rate $> 1/2$.

2. MAIN NOTIONS AND NEW RESULTS

The main result of this paper is a polynomial time computable disperser for 2 sources of entropy $n^{o(1)}$, significantly improving both the results of Barak et al. [4] ($o(n)$ entropy). It also improves on Frankl and Wilson [9], who only built Ramsey Graphs and only for entropy $\tilde{O}(\sqrt{n})$.

THEOREM 2.1 (MAIN THEOREM, RESTATED). *There exists a polynomial time computable* 2*-source disperser* $\mathsf{D} : (\{0,1\}^n)^2 \to \{0,1\}$ *for entropy* $n^{o(1)}$.

The construction of this disperser will involve the construction of an object which in some sense is stronger and in another weaker than a disperser: a *subsource somewhere extractor*. We first define a related object: a *somewhere extractor*, which is a function producing several outputs, one of which must be uniform. Again we will ignore many technical issues such as error, min-entropy vs. entropy and more, in definitions and results, which are deferred to the full version of this paper.

DEFINITION 2.2. *A function* $f : (\{0,1\}^n)^2 \to (\{0,1\}^m)^\ell$ *is a* 2*-source* somewhere extractor *with* ℓ *outputs, for entropy* k*, if for every* 2 *independent* k*-sources* X, Y *there exists an* $i \in [\ell]$ *such the* i*th output* $f(X,Y)_i$ *is a uniformly distributed string of* m *bits.*

Here is a simple construction of such a somewhere extractor with ℓ as large as $\mathrm{poly}(n)$ (and the p in its name will stress the fact that indeed the number of outputs is that large). It will nevertheless be useful to us (though its description in the next sentence may be safely skipped). Define $\mathsf{pSE}(x,y)_i = \mathsf{V}(\mathsf{E}(x,i), \mathsf{E}(y,i))$ where E is a "strong" logarithmic seed extractor, and V is the Hadamard/Vazirani 2-source extractor. Using this construction, it is easy to see that:

PROPOSITION 2.3. *For every* n, k *there is a polynomial time computable somewhere extractor* $\mathsf{pSE} : (\{0,1\}^n)^2 \to (\{0,1\}^m)^\ell$ *with* $\ell = \mathrm{poly}(n)$ *outputs, for entropy* k*, and* $m = \Omega(k)$.

Before we define subsource somewhere extractor, we must first define a subsource.

DEFINITION 2.4 (SUBSOURCES). *Given random variables* Z *and* $\hat{Z}$ *on* $\{0,1\}^n$ *we say that* $\hat{Z}$ *is a* deficiency d subsource *of* Z *and write* $\hat{Z} \subseteq Z$ *if there exists a set* $A \subseteq \{0,1\}^n$ *such that* $(Z|Z \in A) = \hat{Z}$ *and* $\Pr[Z \in A] \geq 2^{-d}$.

A *subsource* somewhere extractor guarantees the "somewhere extractor" property only on subsources X', Y' of the original input distributions X, Y (respectively). It will be extremely important for us to make these subsources as large as possible (i.e. we have to lose as little entropy as possible). Controlling these entropy *deficiencies* is a major technical complication we have to deal with. However we will be informal with it here, mentioning it only qualitatively when needed. We discuss this issue a little more in Section 6.

DEFINITION 2.5. *A function* $f : (\{0,1\}^n)^2 \to (\{0,1\}^m)^\ell$ *is a* 2*-source* subsource somewhere extractor *with* ℓ *outputs for entropy* k*, if for every* 2 *independent* k*-sources* X, Y *there exists a subsource* $\hat{X}$ *of* X*, a subsource* $\hat{Y}$ *of* Y *and an* $i \in [\ell]$ *such the* i^{th} *output* $f(\hat{X}, \hat{Y})_i$ *is a uniformly distributed string of* m *bits.*

A central technical result for us is that with this "subsource" relaxation, we can have much fewer outputs – indeed we'll replace $\mathrm{poly}(n)$ outputs in our first construction above with $n^{o(1)}$ outputs.

THEOREM 2.6 (SUBSOURCE SOMEWHERE EXTRACTOR). *For every* $\delta > 0$ *there is a polynomial time computable subsource somewhere extractor* $\mathsf{SSE} : (\{0,1\}^n)^2 \to (\{0,1\}^m)^\ell$ *with* $\ell = n^{o(1)}$ *outputs, for entropy* $k = n^\delta$*, with output* $m = \sqrt{k}$.

We will describe the ideas used for constructing this important object and analyzing it in the next section, where we will also indicate how it is used in the construction of the final disperser. Here we state a central building block, mentioned in the previous section (as an improvement of the work of Rao [17]). We construct an extractor for 2 independent sources one of which is a block-sources with sufficient number of blocks.

THEOREM 2.7 (BLOCK SOURCE EXTRACTOR). *There is a polynomial time computable extractor* $\mathsf{B} : (\{0,1\}^n)^2 \to \{0,1\}^m$ *for* 2 *independent sources, one of which is a* c*-block-sources with block entropy* k *and the other a source of entropy* k*, with* $m = \Omega(k)$*, and* $c = O((\log n)/(\log k))$.

A simple corollary of this block-source extractor B, is the following weaker (though useful) somewhere block-source extractor SB. A source $Z = Z_1, Z_2, \cdots, Z_t$ is a *somewhere* c*-block-source* of block entropy k if for some c indices $i_1 < i_2 < \cdots < i_c$ the source $Z_{i_1}, Z_{i_2}, \cdots, Z_{i_c}$ is a c-block-source. Collecting the outputs of B on every c-subset of blocks results in that somewhere extractor.

COROLLARY 2.8. *There is a polynomial time computable somewhere extractor* $\mathsf{SB} : (\{0,1\}^n)^2 \to (\{0,1\}^m)^\ell$ *for* 2 *independent sources, one of which is a somewhere* c*-block-sources with block entropy* k *and* t *blocks total and the other a source of entropy* k*, with* $m = \Omega(k)$*,* $c = O((\log n)/(\log k))$*, and* $\ell \leq t^c$.

In both the theorem and corollary above, the values of entropy k we will be interested in are $k = n^{\Omega(1)}$. It follows that a block-source with a constant $c = O(1)$ suffices.

3. THE CHALLENGE-RESPONSE MECHANISM

We now describe abstractly a mechanism which will be used in the construction of the disperser as well as the subsource somewhere extractor. Intuitively, this mechanism allows us to identify parts of a source which contain large amounts of entropy. One can hope that using such a mechanism one can partition a given source into blocks in a way which make it a block-source, or alternatively focus on a part of the source which is unusually condensed with entropy - two cases which may simplify the extraction problem.

The reader may decide, now or in the middle of this section, to skip ahead to the next section which describes the construction of the subsource somewhere extractor SSE, which extensively uses this mechanism. Then this section may seem less abstract, as it will be clearer where this mechanism is used.

This mechanism was introduced by Barak et al. [4], and was essential in their 2-source disperser. Its use in this paper is far more involved (in particular it calls itself recursively, a fact which creates many subtleties). However, at a high level, the basic idea behind the mechanism is the same:

Let Z be a source and Z' a part of Z (Z projected on a subset of the coordinates). We know that Z has entropy k, and want to distinguish two possibilities: Z' has no entropy (it is fixed) or it has at least k' entropy. Z' will get a pass or fail grade, hopefully corresponding to the cases of high or no entropy in Z'.

Anticipating the use of this mechanism, it is a good idea to think of Z as a "parent" of Z', which wants to check if this "child" has sufficient entropy. Moreover, in the context of the initial 2 sources X, Y we will operate on, think of Z as a part of X, and thus that Y is independent of Z and Z'.

To execute this "test" we will compute two sets of strings (all of length m, say): the *Challenge* $C = \mathsf{C}(Z', Y)$ and the *Response* $R = \mathsf{R}(Z, Y)$. Z' fails if $C \subseteq R$ and passes otherwise.

The key to the usefulness of this mechanism is the following lemma, which states that what "should" happen, indeed happens after some restriction of the 2 sources Z and Y. We state it and then explain how the functions C and R are defined to accommodate its proof.

LEMMA 3.1. *Assume Z, Y are sources of entropy k.*

1. *If Z' has entropy $k' + O(m)$, then there are subsources $\hat{Z}$ of Z and $\hat{Y}$ of Y, such that*
$$\Pr[\hat{Z}' \text{ passes}] = \Pr[\mathsf{C}(\hat{Z}', \hat{Y}) \subsetneq \mathsf{R}(\hat{Z}, \hat{Y})] \geq 1 - n^{O(1)} 2^{-m}$$
2. *If Z' is fixed (namely, has zero entropy), then for some subsources $\hat{Z}$ of Z and $\hat{Y}$ of Y, we have*
$$\Pr[Z' \text{ fails}] = \Pr[\mathsf{C}(\hat{Z}', \hat{Y}) \subseteq \mathsf{R}(\hat{Z}, \hat{Y})] = 1$$

Once we have such a mechanism, we will design our disperser algorithm assuming that the challenge response mechanism correctly identifies parts of the source with high or low levels of entropy. Then in the analysis, we will ensure that our algorithm succeeds in making the right decisions, at least on subsources of the original input sources.

Now let us explain how to compute the sets C and R. We will use some of the constructs above with parameters which don't quite fit.

The response set $\mathsf{R}(Z, Y) = \mathsf{pSE}(Z, Y)$ is chosen to be the output of the somewhere extractor of Proposition 2.3. The challenge set $\mathsf{C}(Z', Y) = \mathsf{SSE}(Z', Y)$ is chosen to be the output of the subsource somewhere extractor of Theorem 2.6.

Why does it work? We explain each of the two claims in the lemma in turn (and after each comment on the important parameters and how they differ from Barak et al. [4]).

1. Z' has entropy. We need to show that Z' passes the test with high probability. We will point to the output string in $\mathsf{C}(\hat{Z}', \hat{Y}')$ which avoids $\mathsf{R}(\hat{Z}, \hat{Y})$ with high probability as follows. In the analysis we will use the union bound on several events, one associated with each (poly(n) many) string in $\mathsf{pSE}(\hat{Z}, \hat{Y})$. We note that by the definition of the response function, if we want to fix a particular element in the response set to a particular value, we can do this by fixing $\mathsf{E}(Z, i)$ and $\mathsf{E}(Y, i)$. This fixing keeps the restricted sources independent and loses only $O(m)$ entropy. In the subsource of Z' guaranteed to exist by Theorem 2.6 we can afford to lose this entropy in Z'. Thus we conclude that one of its outputs is uniform. The probability that this output will equal any fixed value is thus 2^{-m}, completing the argument. We note that we can handle the polynomial output size of pSE, since the uniform string has length $m = n^{\Omega(1)}$ (something which could not be done with the technology available to Barak et al. [4]).
2. Z' has no entropy. We now need to guarantee that in the chosen subsources (which we choose) $\hat{Z}, \hat{Y}$, all strings in $C = \mathsf{C}(\hat{Z}', \hat{Y})$ are in $\mathsf{R}(\hat{Z}, \hat{Y})$. First notice that as Z' is fixed, C is only a function of Y. We set $\tilde{Y}$ to be the subsource of Y that fixes all strings in $C = \mathsf{C}(Y)$ to their most popular values (losing only ℓm entropy from Y). We take care of including these fixed strings in $\mathsf{R}(Z, \tilde{Y})$ one at a time, by restricting to subsources assuring that. Let σ be any m-bit string we want to appear in $\mathsf{R}(Z, \tilde{Y})$. Recall that $\mathsf{R}(z, y) = \mathsf{V}(\mathsf{E}(z, i), \mathsf{E}(y, i))$. We pick a "good" seed i, and restrict Z, $\tilde{Y}$ to subsources with only $O(m)$ less entropy by fixing $\mathsf{E}(Z, i) = a$ and $\mathsf{E}(\tilde{Y}, i) = b$ to values (a, b) for which $\mathsf{V}(a, b) = \sigma$. This is repeated successively ℓ times, and results in the final subsources $\hat{Z}, \hat{Y}$ on which $\hat{Z}'$ fails with probability 1. Note that we keep reducing the entropy of our sources ℓ times, which necessitates that this ℓ be tiny (here we could not tolerate $poly(n)$, and indeed can guarantee $n^{o(1)}$, at least on a subsource – this is one aspect of how crucial the subsource somewhere extractor SSE is to the construction.

We note that initially it seemed like the Challenge-Response mechanism as used in [4] could not be used to handle entropy that is significantly less than $\sqrt{n}$ (which is approximately the bound that many of the previous constructions got stuck at). The techniques of [4] involved partitioning the sources into t pieces of length n/t each, with the hope that one of those parts would have a significant amount of entropy, yet there'd be enough entropy left over in the rest of the source (so that the source can be partitioned into a block source).

However it is not clear how to do this when the total entropy is less than $\sqrt{n}$. On the one hand we will have to partition our sources into blocks of length significantly more than $\sqrt{n}$ (or the adversary could distribute a negligible fraction of entropy in all blocks). On the other hand, if our blocks are so large, a single block could contain all the entropy. Thus it was not clear how to use the challenge response mechanism to find a block source.

4. THE SUBSOURCE SOMEWHERE EXTRACTOR SSE

We now explain some of the ideas behind the construction of the subsource somewhere extractor SSE of Theorem 2.6. Consider the source X. We are seeking to find in it a somewhere c-block-source, so that we can use it (together with Y) in the block-source extractor of Theorem 2.8. Like in previous works in the extractor literature (e.g. [19, 13]) we use a "win-win" analysis which shows that either X is already a somewhere c-block-source, or it has a condensed part which contains a lot of the entropy of the source. In this case we proceed recursively on that part. Continuing this way we eventually reach a source so condensed that it must be a somewhere block source. Note that in [4], the challenge response mechanism was used to find a block source also, but there the entropy was so high that they could afford to use

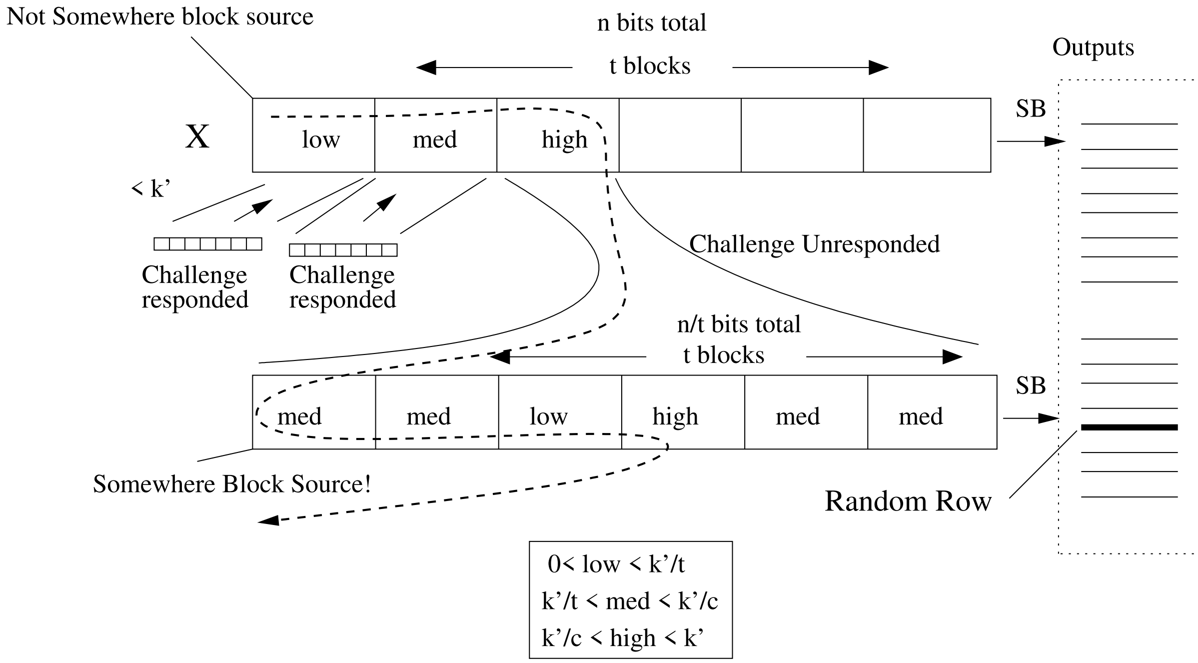

Figure 1: Analysis of the subsource somewhere extractor.

a tree of depth 1. They did not need to recurse or condense the sources.

Consider the tree of parts of the source X evolved by such recursion. Each node in the tree corresponds to some interval of bit locations of the source, with the root node corresponding to the entire source. A node is a child of another if its interval is a subinterval of the parent. It can be shown that some node in the tree is "good"; it corresponds to a somewhere c-source, but we don't know which node is good. Since we only want a somewhere extractor, we can apply to each node the somewhere block-source extractor of Corollary 2.8 – this will give us a random output in every "good" node of the tree. The usual idea is output all these values (and in seeded extractors, merge them using the externally given random seed). However, we cannot afford to do that here as there is no external seed and the number of these outputs (the size of the tree) is far too large.

Our aim then will be to significantly prune this number of candidates and in fact output only the candidates on *one* path to a *canonical* "good" node. First we will give a very informal description of how to do this (Figure 1). Before calling SSE recursively on a subpart of a current part of X, we'll use the "Challenge-Response" mechanism described above to check if "it has entropy".[4] We will recurse only with the *first* (in left-to-right order) part which passes the "entropy test". Thus note that we will follow a single path on this tree. The algorithm SSE will output only the sets of strings produced by applying the somewhere c-block-extractor SB on the parts visited along this path.

Now let us describe the algorithm for SSE. SSE will be initially invoked as $\mathsf{SSE}(x, y)$, but will recursively call itself with different inputs z which will always be substrings of x.

[4]We note that we ignore the additional complication that SSE will actually use recursion also to compute the challenge in the challenge-response mechanism.

Algorithm: $\mathsf{SSE}(z, y)$

Let $\mathsf{pSE}(.,.)$ be the somewhere extractor with a polynomial number of outputs of Proposition 2.3.

Let SB be the somewhere block source extractor of Corollary 2.8.

Global Parameters: t, the branching factor of the tree. k the original entropy of the sources.

Output will be a set of strings.

1. If z is shorter than $\sqrt{k}$, return the empty set, else continue.

2. Partition z into t equal parts $z = z_1, z_2, \ldots, z_t$.

3. Compute the response set $\mathsf{R}(z, y)$ which is the set of strings output by $\mathsf{pSE}(z, y)$.

4. For $i \in [t]$, compute the challenge set $\mathsf{C}(z_i, y)$, which is the set of outputs of $\mathsf{SSE}(z_i, y)$.

5. Let h be the smallest index for which the challenge set $\mathsf{C}(z_h, y)$ is not contained in the response set (set $h = t$ if no such index exists).

6. Output $\mathsf{SB}(z, y)$ concatenated with $\mathsf{SSE}(z_h, y)$.

Proving that indeed there are subsources on which SSE will follow a path to a "good" (for these subsources) node, is the heart of the analysis. It is especially complex due to the fact that the recursive call to SSE on subparts of the current part is used to generate the Challenges for the Challenge-Response mechanism. Since SSE works only on a subsources we have to guarantee that restriction to these does not hamper the behavior of SSE in past and future calls to it.

Let us turn to the highlights of the analysis, for the proof of Theorem 2.6. Let k' be the entropy of the source Z at some place in this recursion. Either one of its blocks Z_i has entropy k'/c, in which case it is very condensed, since its size is n/t for $t \gg c$), or it must be that c of its blocks form a c-block source with block entropy k'/t (which is sufficient for the extractor B used by SB). In the 2nd case the fact that $\mathsf{SB}(z,y)$ is part of the output of of our SSE guarantees that we are somewhere random. If the 2nd case doesn't hold, let Z_i be the leftmost condensed block. We want to ensure that (on appropriate subsources) SSE calls itself on that ith subpart. To do so, we fix all Z_j for $j < i$ to constants z_j. We are now in the position described in the Challenge-Response mechanism section, that (in each of the first i parts) there is either no entropy or lots of entropy. We further restrict to subsources as explained there which make all first $i-1$ blocks *fail* the "entropy test", and the fact that Z_i still has lots of entropy after these restrictions (which we need to prove) ensures that indeed SSE will be recursively applied to it.

We note that while the procedure SSE can be described recursively, the formal analysis of fixing subsources is actually done globally, to ensure that indeed all entropy requirements are met along the various recursive calls.

Let us remark on the choice of the branching parameter t. On the one hand, we'd like to keep it small, as it dominates the number of outputs t^c of SB, and thus the total number of outputs (which is $t^c \log_t n$). For this purpose, any $t = n^{o(1)}$ will do. On the other hand, t should be large enough so that condensing is faster than losing entropy. Here note that if Z is of length n, its child has length n/t, while the entropy shrinks only from k' to k'/c. A simple calculation shows that if $k^{(\log t)/\log c} > n^2$ then a c block-source must exist along such a path before the length shrinks to $\sqrt{k}$. Note that for $k = n^{\Omega(1)}$ a (large enough) constant t suffices (resulting in only logarithmic number of outputs of SSE). This analysis is depicted pictorially in Figure 1.

5. THE FINAL DISPERSER D

Following is a rough description of our disperser D proving Theorem 2.1. The high level structure of D will resemble the structure of SSE - we will recursively split the source X and look for entropy in the parts. However now we must output a *single* value (rather than a set) which can take both values 0 and 1. This was problematic in SSE, even knowing where the "good" part (containing a c-block-source) was! How can we do so now?

We now have at our disposal a much more powerful tool for generating challenges (and thus detecting entropy), namely the subsource somewhere disperser SSE. Note that in constructing SSE we only had essentially the somewhere c-block-source extractor SB to (recursively) generate the challenges, but it depended on a structural property of the block it was applied on. Now SSE does not assume any structure on its input sources except sufficient entropy [5].

Let us now give a high level description of the disperser D. It too will be a recursive procedure. If when processing some part Z of X it "realizes" that a subpart Z_i of Z has entropy, but not *all* the entropy of Z (namely Z_i, Z is a 2-block-source) then we will halt and produce the output of D. Intuitively, thinking about the Challenge-Response mechanism described above, the analysis implies that we can either pass or fail Z_i (on appropriate subsources). But this means that the outcome of this "entropy test" **is** a 1-bit disperser!

To capitalize on this idea, we want to use SSE to identify such a block-source in the recursion tree. As before, we scan the blocks from left to right, and want to distinguish three possibilities.

low Z_i has low entropy. In this case we proceed to $i+1$.

medium Z_i has "medium" entropy (Z_i, Z is a block-source). In which case we halt and produce an output (zero or one).

high Z_i has essentially all entropy of Z. In this case we recurse on the condensed block Z_i.

As before, we use the Challenge-Response mechanism (with a twist). We will compute challenges $\mathsf{C}(Z_i, Y)$ and responses $\mathsf{R}(Z, Y)$, all strings of length m. The responses are computed exactly as before, using the somewhere extractor pSE. The Challenges are computed using our subsource somewhere extractor SSE.

We really have 4 possibilities to distinguish, since when we halt we also need to decide which output bit we give. We will do so by deriving three tests from the above challenges and responses: $(C_H, R_H), (C_M, R_M), (C_L, R_L)$ for high, medium and low respectively, as follows. Let $m \geq m_H >> m_M >> m_L$ be appropriate integers: then in each of the tests above we restrict ourselves to prefixes of all strings of the appropriate lengths only. So every string in C_M will be a prefix of length m_M of some string in C_H. Similarly, every string in R_L is the length m_L prefix of some string in R_H. Now it is immediately clear that if C_M is contained in R_M, then C_L is contained in R_L. Thus these tests are monotone, if our sample fails the high test, it will definitely fail all tests.

Algorithm: $\mathsf{D}(z, y)$

Let $\mathsf{pSE}(.,.)$ be the somewhere extractor with a polynomial number of outputs of Proposition 2.3.

Let $\mathsf{SSE}(.,.)$ be the subsource somewhere extractor of Theorem 2.6.

Global Parameters: t, the branching factor of the tree. k the original entropy of the sources.

Local Parameters for recursive level: $m_L \ll m_M \ll m_H$.

Output will be an element of $\{0, 1\}$.

1. If z is shorter than $\sqrt{k}$, return 0.
2. Partition z into t equal parts $z = z_1, z_2, \ldots, z_t$.
3. Compute three response sets R_L, R_M, R_H using $\mathsf{pSE}(z, y)$. R_j will be the prefixes of length m_j of the strings in $\mathsf{pSE}(z, y)$.
4. For each $i \in [t]$, compute three challenge sets C_L^i, C_M^i, C_H^i using $\mathsf{SSE}(z_i, y)$. C_j^i will be the prefixes of length m_j of the strings in $\mathsf{SSE}(z_i, y)$.
5. Let h be the smallest index for which the challenge set C_L is not contained in the response set R_L, if there is no such index, output 0 and halt.
6. If C_H^h is contained in R_H and C_H^h is contained in R_M, output 0 and halt. If C_H^h is contained in R_H but C_H^h is not contained in R_M, output 1 and halt.

[5] There is a catch – it only works on subsources of them! This will cause us a lot of head ache; we will elaborate on it later.

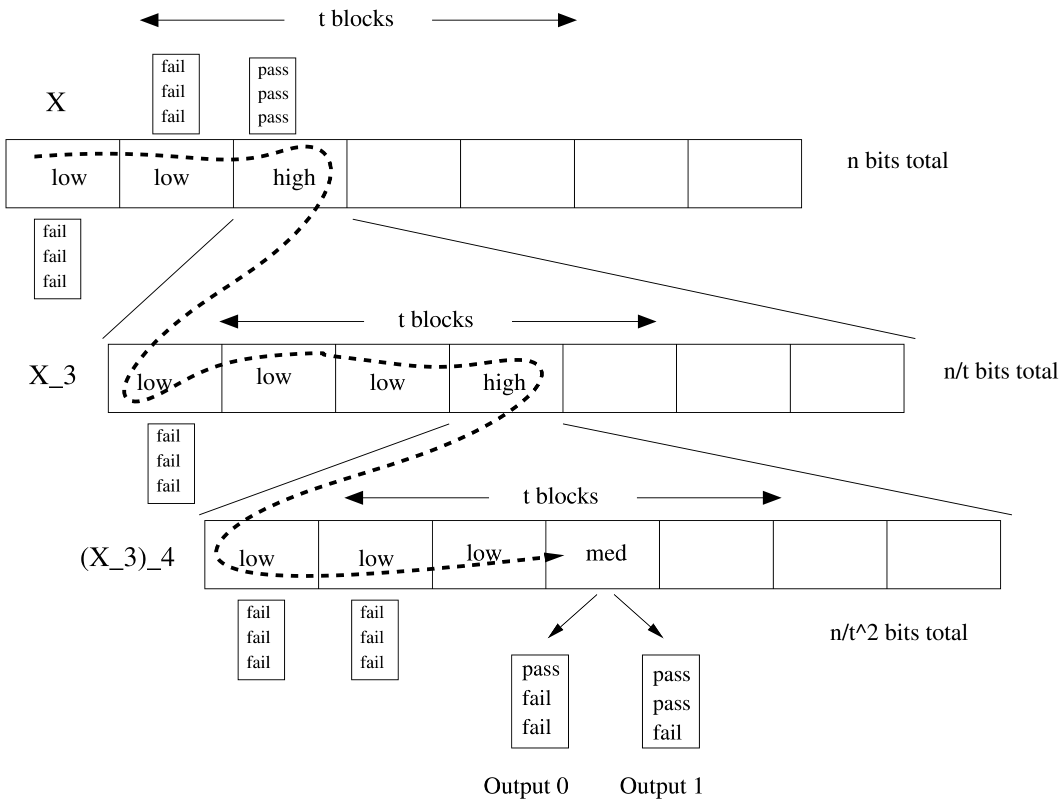

Figure 2: Analysis of the disperser.

7. Output $\mathsf{D}(z_h, y)$,

First note the obvious monotonicity of the tests. If Z_i fails one of the tests it will certainly fail for shorter strings. Thus there are only four outcomes to the three tests, written in the order (low, medium, high): (pass, pass, pass), (pass, pass, fail), (pass, fail, fail) and (fail, fail, fail). Conceptually, the algorithm is making the following decisions using the four tests:

1. (fail, fail, fail): Assume Z_i has low entropy and proceed to block $i + 1$.

2. (pass, fail, fail): Assume Z_i is medium, halt and output 0.

3. (pass, pass, fail): Assume Z_i is medium, halt and output 1.

4. (pass, pass, pass): Assume Z_i is high and recurse on Z_i.

The analysis of this idea (depicted in Figure 2).turns out to be more complex than it seems. There are two reasons for that. Now we briefly explain them and the way to overcome them in the construction and analysis.

The first reason is the fact mentioned above, that SSE which generates the challenges, works only on a subsources of the original sources. Restricting to these subsources at some level of the recursion (as required by the analysis of of the test) causes entropy loss which affects both definitions (such as these entropy thresholds for decisions) and correctness of SSE in higher levels of recursion. Controlling this entropy loss is achieved by calling SSE recursively with smaller and smaller entropy requirements, which in turn limits the entropy which will be lost by these restrictions. In order not to lose all the entropy for this reason alone, we must work with special parameters of SSE, essentially requiring that at termination it has almost all the entropy it started with.

The second reason is the analysis of the test when we are in a medium block. In contrast with the above situation, we cannot consider the value of Z_i fixed when we need it to fail on the Medium and Low tests. We need to show that for these two tests (given a *pass* for High), they come up both (pass, fail) and (fail, fail) each with positive probability.

Since the length of Medium challenges and responses is m_M, the probability of failure is at least $\exp(-\Omega(m_M))$ (this follows relatively easily from the fact that the responses are somewhere random). If the Medium test fails so does the Low test, and thus (fail, fail) has a positive probability and our disperser D outputs 0 with positive probability.

To bound (pass, fail) we first observe (with a similar reasoning) that the low test fails with probability at least $\exp(-\Omega(m_L))$. But we want the medium test to pass at the same time. This probability is at least the probability that low fails minus the probability that medium fails. We already have a bound on the latter: it is at most $\mathrm{poly}(n)exp(-\ell m_M)$. Here comes our control of the different length into play - we can make the m_L sufficiently smaller than m_M to yield this difference positive. We conclude that our disperser D outputs 1 with positive probability as well.

Finally, we need to take care of termination: we have to ensure that the recurrence *always* arrives at a medium subpart, but it is easy to chose entropy thresholds for low, medium and high to ensure that this happens.

6. RESILIENCY AND DEFICIENCY

In this section we will breifly discuss an issue which arises in our construction that we glossed over in the previous sections. Recall our definition of subsources:

DEFINITION 6.1 (SUBSOURCES). *Given random variables* Z *and* $\hat{Z}$ *on* $\{0,1\}^n$ *we say that* $\hat{Z}$ *is a* deficiency d subsource *of* Z *and write* $\hat{Z} \subseteq Z$ *if there exists a set* $A \subseteq \{0,1\}^n$ *such that* $(Z|A) = \hat{Z}$ *and* $\Pr[Z \in A] \geq 2^{-d}$.

Recall that we were able to guarantee that our algorithms made the right decisions only on subsources of the original source. For example, in the construction of our final disperser, to ensure that our algorithms correctly identify the right high block to recurse on, we were only able to guarantee that there are subsources of the original sources in which our algorithm makes the correct decision with high probability. Then, later in the analysis we had to further restrict the source to even smaller subsources. This leads to complications, since the original event of picking the correct high block, which occurred with high probability, may become an event which does not occur with high probability in the current subsource. To handle these kinds of issues, we will need to be very careful in measuring how small our subsources are.

In the formal analysis we introduce the concept of *resiliency* to deal with this. To give an idea of how this works, here is the actual definition of somewhere subsource extractor that we use in the formal analysis.

DEFINITION 6.2 (SUBSOURCE SOMEWHERE EXTRACTOR). *A function* $\mathsf{SSE} : \{0,1\}^n \times \{0,1\}^n \to (\{0,1\}^m)^\ell$ *is a* subsource somewhere extractor *with* nrows *output rows,* entropy threshold k*,* deficiency def*,* resiliency res *and* error ϵ *if for every* (n,k)*-sources* X, Y *there exist a deficiency* def *subsource* X^{good} *of* X *and a deficiency* def *subsource* Y^{good} *of* Y *such that for every deficiency* res *subsource* X' *of* X^{good} *and deficiency* res *subsource* Y' *of* Y^{good}*, the random variable* $\mathsf{SSE}(X', Y')$ *is* ϵ*-close to a* $\ell \times m$ *somewhere random distribution.*

It turns out that our subsource somewhere extractor does satisfy this stronger definition. The advantage of this definition is that it says that once we restrict our attention to the good subsources $X^{\text{good}}, Y^{\text{good}}$, we have the freedom to further restrict these subsources to smaller subsources, as long as our final subsources do not lose more entropy than the resiliency permits.

This issue of managing the resiliency for the various objects that we construct is one of the major technical challenges that we had to overcome in our construction.

7. OPEN PROBLEMS

Better Independent Source Extractors A bottleneck to improving our disperser is the block versus general source extractor of Theorem 2.7. A good next step would be to try to build an extractor for one block source (with only a constant number of blocks) and one other independent source which works for polylogarithmic entropy, or even an extractor for a constant number of sources that works for sub-polynomial entropy.

Simple Dispersers While our disperser is polynomial time computable, it is not as explicit as one might have hoped. For instance the Ramsey Graph construction of Frankl-Wilson is extremely simple: For a prime p, let the vertices of the graph be all subsets of $[p^3]$ of size $p^2 - 1$. Two vertices S, T are adjacent if and only if $|S \cap T| \equiv -1 \bmod p$. It would be nice to find a good disperser that beats the Frankl-Wilson construction, yet is comparable in simplicity.

8. REFERENCES

[1] N. Alon. The shannon capacity of a union. *Combinatorica*, 18, 1998.

[2] B. Barak. A simple explicit construction of an $n^{\tilde{o}(\log n)}$-ramsey graph. Technical report, Arxiv, 2006. `http://arxiv.org/abs/math.CO/0601651`.

[3] B. Barak, R. Impagliazzo, and A. Wigderson. Extracting randomness using few independent sources. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 384–393, 2004.

[4] B. Barak, G. Kindler, R. Shaltiel, B. Sudakov, and A. Wigderson. Simulating independence: New constructions of condensers, Ramsey graphs, dispersers, and extractors. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 1–10, 2005.

[5] J. Bourgain. More on the sum-product phenomenon in prime fields and its applications. *International Journal of Number Theory*, 1:1–32, 2005.

[6] J. Bourgain, N. Katz, and T. Tao. A sum-product estimate in finite fields, and applications. *Geometric and Functional Analysis*, 14:27–57, 2004.

[7] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 659–668, 2002.

[8] 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.

[9] P. Frankl and R. M. Wilson. Intersection theorems with geometric consequences. *Combinatorica*, 1(4):357–368, 1981.

[10] P. Gopalan. Constructing ramsey graphs from boolean function representations. In *Proceedings of the 21th Annual IEEE Conference on Computational Complexity*, 2006.

[11] V. Grolmusz. Low rank co-diagonal matrices and ramsey graphs. *Electr. J. Comb*, 7, 2000.

[12] V. Guruswami. Better extractors for better codes? *Electronic Colloquium on Computational Complexity (ECCC)*, (080), 2003.

[13] C. J. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: Optimal up to constant factors. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 602–611, 2003.

[14] P. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57:37–49, 1 1998.

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

[16] P. Pudlak and V. Rodl. Pseudorandom sets and explicit constructions of ramsey graphs. *Submitted for publication*, 2004.

[17] A. Rao. Extractors for a constant number of polynomially small min-entropy independent sources. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, 2006.

[18] R. Raz. Extractors with weak random seeds. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 11–20, 2005.

[19] O. Reingold, R. Shaltiel, and A. Wigderson. Extracting randomness via repeated condensing. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 22–31, 2000.

[20] M. Santha and U. V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33:75–87, 1986.

[21] R. Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the European Association for Theoretical Computer Science*, 77:67–95, 2002.

[22] A. Ta-Shma and D. Zuckerman. Extractor codes. *IEEE Transactions on Information Theory*, 50, 2004.

[23] U. Vazirani. Towards a strong communication complexity theory or generating quasi-random sequences from two communicating slightly-random sources (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 366–378, 1985.

[24] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. *Combinatorica*, 19(1):125–138, 1999.