Reconstructive Dispersers and Hitting Set Generators

Christopher Umans[⋆]

Computer Science Department
California Institute of Technology
Pasadena CA 91125
`umans@cs.caltech.edu`

Abstract. We give a generic construction of an optimal hitting set generator (HSG) from any good "reconstructive" disperser. Past constructions of optimal HSGs have been based on such disperser constructions, but have had to modify the construction in a complicated way to meet the stringent efficiency requirements of HSGs. The construction in this paper uses existing disperser constructions with the "easiest" parameter setting in a black-box fashion to give new constructions of optimal HSGs without any additional complications.

Our results show that a straightforward composition of the Nisan-Wigderson pseudorandom generator that is similar to the composition in works by Impagliazzo, Shaltiel and Wigderson in fact yields optimal HSGs (in contrast to the "near-optimal" HSGs constructed in those works). Our results also give optimal HSGs that do not use any form of hardness amplification or implicit list-decoding – like Trevisan's extractor, the only ingredients are combinatorial designs and any good list-decodable error-correcting code.

1 Introduction

Ever since Trevisan [1] showed that certain pseudorandom generator (PRG) constructions yield extractors, many of the results in these two areas have been intertwined. In this paper we are concerned with the one-sided variant of a PRG, called a *hitting set generator* (HSG), and the one-sided variant of an extractor, called a *disperser*.

Informally, a HSG construction takes an n-bit truth table of a hard function f and converts it into a collection of poly(n) shorter m-bit strings, with the property that every small circuit D that accepts at least $1/2$ of its inputs, also accepts one of these m-bit strings. The proof that a construction is indeed a HSG typically gives an efficient way to convert a small circuit D on which the construction fails to meet the definition into a small circuit computing f, thus contradicting the hardness of f.

Informally, a disperser takes an n-bit string x sampled from a weak random source with sufficient min-entropy and converts it into a collection of poly(n)

[⋆] Supported by NSF grant CCF-0346991 and an Alfred P. Sloan Research Fellowship.

shorter m-bit strings, with the property that every circuit D that accepts at least $1/2$ of its inputs, also accepts one of these m-bit strings. Trevisan's insight is that a HSG construction whose proof uses D in a black-box fashion *is* a disperser, for the following reason: if there is a circuit D on which the construction fails to meet the disperser definition, then we have a small circuit *relative to* D that describes input x, and it cannot be that every string in a source with sufficiently high min-entropy has such a short description.

Thus we can produce a formal statement to the effect that "every black-box HSG construction yields a disperser with similar parameters." In this paper we consider the reverse question, namely: "under what conditions does a disperser construction yield a HSG construction?"

We will limit ourselves to so-called "reconstructive" dispersers which means, roughly, that the associated proof has the same outline as the one sketched above, and that the conversion in the proof is efficient. At first glance this may seem to be such a strong constraint that the question becomes uninteresting. However, there is an important issue related to the precise meaning of "efficient." It turns out that there are (at least) two possible notions of "efficient;" one is satisfied naturally in several disperser constructions, and the other – which is the one that is actually required for the construction to be a HSG – is far more stringent. The distinction between the two is analogous to the distinction between an error-correcting code being efficiently decodable in the usual sense, and being efficiently *implicitly* decodable in the sense of [2].

To be precise, consider a function $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ (where $m < n$ may be as small as $\text{poly}\log n$) and a circuit $D : \{0,1\}^m \to \{0,1\}$ relative to which E fails to be a disperser; i.e. D accepts at least half of its inputs, but for every x in the weak random source, $E(x,\cdot)$ fails to hit an accepting input of D. If E is equipped with a "reconstructive" proof then the proof should give an efficient way to reconstruct x from D and short advice (the advice may depend on x and auxiliary randomness w used in the proof). The two notions of reconstructivity for E that we discuss are:

explicit reconstructivity. Given oracle access to circuit D, and advice $A(x,w)$, compute in time $\text{poly}(n)$ the string x with non-negligable probability over the choice of w.

implicit reconstructivity. Given oracle access to circuit D, and advice $A(x,w)$, and an index i, compute in time $\text{poly}(m)$ the i-th bit of x with non-negligable probability over the choice of w.

Explicit reconstructivity is naturally satisfied by relatively simple disperser constructions[1] [1, 3, 4], and these construction possess two additional useful features as well (as observed in [5–7]): (1) w has length $O(\log n)$ and (2) A is computable in time $\text{poly}(n)$.

In contrast, obtaining *implicit reconstructivity* has always required significant extra effort: in [4], one needs to use multiple copies of the original disperser with

[1] Of course these are all actually extractor constructions, but an extractor is a disperser.

multiple "strides" and to piece them together in a complex manner; in [8], similar ideas are used, together with an "augmented" low-degree extension; and in [5, 6], the construction of [1] is modified by repeated composition, but at the price of a super-polynomial degradation in the output length.

In this paper, we show that the weaker notion of explicit reconstructivity (together with the two additional properties satisfied by known constructions) is in fact sufficient to construct *optimal* HSGs, thus avoiding the complications of past work.

Our result is shown by analyzing a composition of reconstructive dispersers, similar to the one used in [5, 6]. Our composition has the advantage of being simpler (we believe) and generic (it works for any reconstructive disperser). Most importantly, it produces optimal HSGs for all hardnesses, starting only with reconstructive dispersers for a particular "easy" setting of parameters.

Our results also shed light on two issues: the first concerns the Nisan-Wigderson pseudorandom generator (PRG) [9, 10, 2], which has a non-optimal seed length for sub-exponential hardness assumptions. Two works [5, 6] addressed this deficiency by composing the PRG with itself in a clever way. The result came close to an optimal construction, but fell short. One might have guessed that there was some inherent loss associated with the composition-based approach, or that the combinatorial-design-based constructions were too weak to obtain the optimal result (as subsequent solutions to this problem employed different, algebraic constructions [4, 8]). Our results show that in the end, composing the Nisan-Wigderson PRG with itself *can* be made to work to obtain HSGs, by using a somewhat different composition than those used previously.

Second, until this paper, known constructions of optimal HSGs [4, 8] have made crucial use of implicit list-decoding (cf. [2]) of Reed-Muller codes. Implicit list-decoding of Reed-Muller codes also underlies the hardness amplification results that were a component of earlier (non-optimal) constructions [5, 6, 2]. One may wonder whether some form of implicit list-decoding is in fact *necessary* to construct HSGs. Our results show that it is not, and indeed we can construct optimal HSGs using only combinatorial designs and any good list-decodable code (which, not accidentally, are also the two ingredients needed for Trevisan's extractors).

We remark that "reconstructivity" of various disperser and extractor constructions has emerged as a crucial property of these constructions in a number of applications: error-correcting codes [11, 12], data structures [13], and complexity theory [14]. This suggests that it is worthwhile to formalize and study notions of reconstructivity as we do in this paper.

2 Preliminaries

We use $[n]$ as shorthand for the set $\{1, 2, 3 \dots n\}$. We use U_m for the random variable uniformly distributed on $\{0, 1\}^m$.

Definition 1. *Let Z be a random variable distributed on $\{0,1\}^m$. We say that a function $D : \{0,1\}^m \to \{0,1\}$ "ϵ-catches" Z if $|\Pr[D(U_m) = 1] - \Pr[D(Z) = 1]| > \epsilon$. In the special case that $\Pr[D(Z) = 1] = 0$, we say that D "ϵ-avoids Z".*

In this paper we will always be in the aforementioned special case, since we are discussing one-sided objects (HSGs and dispersers). Replacing "ϵ-avoids" with "ϵ-catches" everywhere in the paper yields the two-sided version of all of the definitions, theorems and proofs, with the exception of one place in the proof of Theorem 4 where we use the one-sidedness critically.

Definition 2. *Let x be a n-bit truth table of a function that requires circuits of size k. An* ϵ-HSG *is a function $H_x : \{0,1\}^t \to \{0,1\}^m$ such that no size m circuit $D : \{0,1\}^m \to \{0,1\}$ ϵ-avoids $H_x(U_t)$.*

This means that every size m circuit D that accepts more than an ϵ fraction of its inputs, also accepts $H_x(y)$ for some y. It has become customary to refer to families of ϵ-HSGs that for every $k = k(n)$ have parameters $t \leq O(\log n)$ and $m \geq k^\delta$ for some constant $\delta > 0$ as *optimal* (because they give rise to hardness vs. randomness tradeoffs that are optimal up to a polynomial).

For reference, we give the standard definition of a (k, ϵ)-disperser before stating the more complicated definition of a reconstructive disperser that we will need for this paper.

Definition 3. *A* (k, ϵ)-disperser *is a function $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ such that for all subsets $X \subseteq \{0,1\}^n$ with $|X| \geq 2^k$, no circuit $D : \{0,1\}^m \to \{0,1\}$ ϵ-avoids $E(X, U_t)$.*

2.1 Reconstructive dispersers

We now define the central object, which we call a reconstructive disperser. It has more parameters than one would prefer, but keeping track of all of these parameters will make the composition much easier to state.

Definition 4. *A* $(n, t, m, d, a, b, \epsilon, \delta)$-reconstructive disperser *is a triple of functions:*

- *the "disperser" function $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$*
- *the "advice" function $A : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^a$*
- *the randomized[2] oracle "reconstruction" procedure $R : \{0,1\}^a \times [n/b] \to \{0,1\}^b$*

that satisfy the following property: for every $D : \{0,1\}^m \to \{0,1\}$ and $x \in \{0,1\}^n$ for which D ϵ-avoids $E(x, U_t)$, we have

$$\forall i \in [n/b] \quad \Pr_w[R^D(A(x,w), i) = x_i] \geq \delta. \tag{1}$$

[2] When we refer to a "randomized" function, we mean a function f that takes an extra argument which is thought of as random bits. We refrain from explicitly writing the second argument; however whenever f occurs within a probability, we understand that the probability space includes the randomness of f.

Here x_i refers to the i-th b-bit block in x. When $b = n$ we drop the second argument to R.

Note that in this definition there is no reference to the parameter "k" that occurs in Definitions 2 and 3. The idea is that if (E, A, R) is a reconstructive disperser, then E must be a (k, ϵ)-disperser for k slightly larger than a, because relative to D, many strings x in the source X have descriptions of size approximately a (via R). Similarly, $E(x, \cdot)$ is an ϵ-HSG when x is the truth table of a function that does not have size k circuits, for k slightly larger than a plus the running time of R. This is because relative to D, and given a bits of advice, R can be used to compute any specified bit of x. See Theorems 2 and 3 for formal statements of these assertions.

The parameter b interpolates between the two types of reconstructivity. When $b = n$, the reconstruction procedure outputs all of x. A reconstruction procedure of this type running in time poly(n) is *explicit* and is often implied by disperser constructions whose proofs use the so-called "reconstruction proof paradigm."

To obtain an optimal HSG, we need b to be small and the running time of the reconstruction procedure to be poly(m) (where $m < n$ may be as small as poly $\log n$). Such a reconstruction procedure is necessarily *implicit*, and it satisfies a much more stringent efficiency requirement that typically does *not* follow from disperser constructions without modification. Note however that it is trivial to decrease b by a multiplicative factor without changing the running time of R (and we will use this fact). The challenge is to decrease b while decreasing the running time of R simultaneously.

There are three quite clean constructions of reconstructive dispersers known for a certain "easy" setting of the parameters.

Theorem 1 ([1, 3, 4]). *For every n and $\epsilon \geq n^{-1}$, and pair of constants $\gamma > \beta > 0$ (and $\gamma > \beta + 1/2$ in the case of [3]), there is a*

$$(n, t = O(\log n), m = n^{\beta}, d = O(\log n), a = n^{\gamma}, b = n, \epsilon, \delta = 1/4)$$

reconstructive disperser (E, A, R) with the running time of E, A, R at most n^c for a universal constant c.

Proof. (sketch) The proofs associated with these constructions all conform to the following outline: (1) given D that ϵ-avoids $E(x, U_t)$, convert it into a randomized *next-bit predictor* with success rate $1/2 + \epsilon/m$; (2) use the next-bit predictor together with the advice $A(x, w)$ to recover x with probability $1/4$ over the choice of w. In some cases the original proof concludes by obtaining a short *list* containing x; in this case we add a random hash of x to the advice string to allow us to correctly select x from the list with probability $1/4$. $\square$

Note that these constructions are not sufficient to produce HSGs directly, because the running time of the reconstruction procedure R is by itself far greater than the trivial upper bound on the circuit complexity of a function whose truth table has size n, so we cannot get the required contradiction.

However, if the running time of R is much smaller than the input length n, reconstructive dispersers *are* hitting set generators when their input is fixed to be a hard function.

Theorem 2. *Let (E, A, R) be a $(n, t, m, d, a, b, \epsilon, \delta = 2/3)$ reconstructive disperser, for which R runs in time T. Let $x \in \{0,1\}^n$ be the truth table of a function that cannot be computed by circuits of size k. There is a universal constant c for which: if $k > c(\log n)(Tm + a)$, then $H_x(\cdot) = E(x, \cdot)$ is an ϵ-HSG.*

Proof. If $E(x, \cdot)$ is not the claimed HSG, then there is a size m circuit $D : \{0,1\}^m \to \{0,1\}$ that ϵ-avoids $E(x, U_t)$. By the definition of reconstructive dispersers we have: $\forall i \in [n/b]$ $\Pr_w[R^D(A(x, w), i) = x_i] \geq 2/3$. We repeat the reconstruction $\Theta(\log n)$ times with independent random w's and take the majority outcome. For a given i, the probability that this fails to produce x_i is less than $1/n$ by Chernoff bounds. By a union bound, the probability that we fail on any i is strictly less than one. Thus we can fix the random bits used in this procedure so that we correctly produce x_i for all i. We hardwire $A(x, w)$ for the chosen w's. The resulting circuit has size $c(\log n)(Tm + a)$ which contradicts the hardness of x. □

We remark that the same argument shows that reconstructive dispersers are indeed a special case of ordinary dispersers (a more efficient conversion is possible but that is not important for this paper):

Theorem 3. *Let (E, A, R) be a $(n, t, m, d, a, b, \epsilon, \delta = 2/3)$ reconstructive disperser. There is a universal constant c for which E is a $(ca(\log n), \epsilon)$-disperser.*

3 Intuition for the composition

In this discussion we focus almost entirely on the advice and reconstruction functions. Getting the parameters of these "right" in the composition gives a reconstructive disperser that is an optimal HSG. From Theorem 2 we can see that the key is for the advice length a to be as short as possible, and for the running time of the reconstruction procedure R to be as small as possible. Specifically, our goal will be to obtain (after several compositions) a reconstructive disperser with input length $N \gg n$, advice length poly(n), and R's running time poly(n), while maintaining an output length of at least $n^{\Omega(1)}$.

Our starting point is the "simple" constructions of Theorem 1. Note that $b = n$ in those constructions so we drop the second argument of the reconstruction procedure R, and then it simply maps $a = n^\gamma$ bits to n bits.

Let as also assume for the purpose of this simplified exposition that $d = 0$, i.e., the advice function A just maps n bits to $a = n^\gamma$ bits. It is useful to think of the advice function A as a procedure that "compresses" an arbitrary n-bit string x into n^γ bits and the reconstruction procedure R as a procedure that "decompresses" the n^γ bit string back to the original n-bit string x. Of course this is information-theoretically impossible, but a slightly relaxed version of this (in which $d = O(\log n)$ rather than 0, and R is given oracle access to "distinguishing" function D) is exactly the definition of A and R in Definition 4.

So we have the ability to compress from n bits down to n^γ bits. Suppose we want to be able to compress from $N \gg n$ down to n^γ bits. A natural thing to do is to divide the N-bit string into N/n substrings of size n, and use A to compress each of the substrings. The resulting string has length $(N/n)n^\gamma$; we can repeat the process $O(\log N/\log n)$ times until we get down to length n^γ. To decompress, we use R to reverse the process. A nice side-effect of this scheme is that when we know *which* n-bit substring of the original N bit string we wish to recover, we only need to invoke R *once* at each level (as opposed to, e.g., N/n times at the bottom level, if we insist on recovering the entire original N bit string). In the formalism of Definition 4, this means that we can take $b = n$, and have R only recover the specified b-bit block of the input string.

The above description specifies an advice function that maps an N-bit string x down to n^γ bits, and a reconstruction procedure that recovers any specified n-bit substring of x from those n^γ bits. A crucial observation is that the new reconstruction procedure has $b = n \ll N$, and it runs in time $\text{poly}(n, \log N/\log n) \ll N$, since it invokes R once for each of the $O(\log N/\log n)$ levels.

We have made only two simplifying assumptions. First, we have assumed that $d = 0$, when in fact it must be $O(\log n)$. When $d = O(\log n)$ the "compression" function A produces $2^d = \text{poly}(n)$ *candidate* compressed versions of x, with the property that with high probability over a choice of candidates, the "decompression" procedure R succeeds. We will carry out the above scheme for all candidates at each level, and ensure that the decompression at each level works with sufficiently high probability so that the overall decompression works with constant probability.

Second, we have ignored the fact that R needs oracle access to a function D with certain properties to succeed. To deal with this, we simply run the disperser E on every n-bit string x that we "compress" in the entire process, and define the disperser E' of the composed object to be the union of these. The reconstruction procedure R' for the composed object is only required to work relative to D that ϵ-avoids the output of E'. Such a D also ϵ-avoids the output of each invocation of E we have performed[3], and so D is exactly what is required to actually allow the "decompressions" to work at all levels.

Both the disperser function E' and the advice function A' of the composed object have asymptotically optimal seed lengths of $O(\log N)$. This is because at each level we need a fresh $O(\log n)$ bit seed for E and A coming from Theorem 1, and there are $O(\log N/\log n)$ levels.

The formal analysis of a single level of this composition is the content of Theorem 4. Corollary 1 and Theorem 5 apply this composition $O(\log N/\log n)$ times to the reconstructive dispersers of Theorem 1 to obtain the final result.

[3] This is the single place where we rely critically on the fact that we are dealing with "ϵ-avoids," rather than "ϵ-catches."

Comparison with [5, 6]. As noted, our composition is similar to the one used in [5, 6], which can also be understood in the language of reconstructive dispersers. If one interprets their composition this way, the crucial difference is that our reconstruction procedure runs on strings of length n at all levels, while in their work, the reconstruction procedure runs on inputs whose lengths increase from one level to the next. In addition, [5, 6] have a layer of hardness amplification at each level, which our composition avoids. The result is that they incur a loss that is superpolynomial if the number of levels is super-constant (and to minimize this loss, they vary parameters from level to level in a sophisticated way). We incur only a loss that is *multiplicative* in the number of levels, which for us is logarithmic in the input length, so we can ignore it altogether.

4 Analysis of the composition

We begin by showing that a simple pairwise independent repetition can increase the success probability of the reconstruction procedure:

Lemma 1 (increasing δ). *Suppose (E, A, R) is a $(n, t, m, d, a, b = n, \epsilon, \delta)$ reconstructive extractor. For every $\alpha > 0$, we can convert (E, A, R) into a*

$$(n, t, m, d' = O(d + \log(pn)), a' = O(pa), b = n, \epsilon, \delta' = 1 - \alpha)$$

reconstructive extractor (E, A', R'), where $p = \delta^{-1}\alpha^{-1}$. The running time of A' is at most poly(p, n) times the running time of A, and the running time of R' is at most poly(p, n) times the running time of R.

Proof. (sketch) A' uses its random bits to output $2p$ invocations of A using pairwise independent seeds, and a random hash of the input x; R' runs R on each of the advice strings, using the hash to select which reconstruction to output. □

Our main composition operation for reconstructive dispersers increases n. More specifically, it roughly squares n, while multiplying each of the two seed lengths (t and d) by a constant. The final object inherits the advice length a from the first reconstructive disperser, and the block length b from the second.

Theorem 4 (composition of reconstructive dispersers). *Suppose (E_1, A_1, R_1) is a $(n_1, t_1, m, d_1, a_1, b_1, \epsilon, \delta_1)$ reconstructive disperser and that (E_2, A_2, R_2) is a $(n_2, t_2, m, d_2, a_2, b_2, \epsilon, \delta_2)$ reconstructive disperser, with $a_2 = b_1$. Set $r = n_1/b_1$. Then (E, A, R) defined as:*

$$E(x_1, \ldots, x_r; w_2, y, p \in [r+1]) = \begin{cases} E_1(A_2(x_1, w_2) \circ \cdots \circ A_2(x_r, w_2), y) & \textit{if } p = r+1 \\ E_2(x_p, y) & \textit{otherwise} \end{cases}$$

$$A(x_1, \ldots, x_r; w_1, w_2) = A_1(A_2(x_1, w_2) \circ \cdots \circ A_2(x_r, w_2), w_1)$$

$$R^D(z; i_1 \in [r], i_2 \in [n_2/b_2]) = R_2^D(R_1^D(z, i_1), i_2)$$

is a

$$(n = (n_1 n_2)/a_2, t = \max(t_1, t_2) + d_2 + \log(r+1), m, \\ d = d_1 + d_2, a = a_1, b = b_2, \epsilon, \delta = \delta_1 + \delta_2 - 1)$$

reconstructive disperser.

A few words of explanation are in order. The input for the composed object is $x = x_1, x_2, \ldots, x_r$, where the x_i coincide with the blocks R will need to output. The function A is very simple: it just concatenates the output of A_2 run on each of $x_1, x_2, \ldots, x_r$, and runs A_1 on the concatenated string. The function R is similarly simple; it reverses the process: it takes the output of A and first uses R_1 to extract the advice associated with x_{i_1}, and then uses R_2 to actually recover (the i_2-th block of) x_{i_1}. Finally E uses part of its seed, p, to decide either to run E_2 on input x_i for some i, or to run E_1 on the concatenated advice string on which A_1 is run.

Proof. (of Theorem 4) Fix D and $x = x_1, x_2, \ldots, x_r$ for which D ϵ-avoids $E(x_1, \ldots, x_r; U_t)$. Also fix $i = i_1 i_2$, where $i_1 \in [r]$ and $i_2 \in [n_2/b_2]$.

From the fact that D ϵ-avoids $E(x_1, \ldots, x_r; U_t)$, we know that

$$\forall w_2 \;\; D \;\; \epsilon\text{-avoids } E_1(A_2(x_1, w_2) \circ \cdots \circ A_2(x_r, w_2), U_{t_1}) \qquad (2)$$

$$\forall p \in [r] \;\; D \;\; \epsilon\text{-avoids } E_2(x_p, U_{t_2}) \qquad (3)$$

From (2) and the definition of reconstructive dispersers, we get that for all w_2:

$$\Pr_{w_1}[R_1^D(A_1(A_2(x_1, w_2) \circ \cdots \circ A_2(x_r, w_2), w_1), i_1) = A_2(x_{i_1}, w_2)] \geq \delta_1. \qquad (4)$$

From (3) and the definition of reconstructive dispersers, we get that:

$$\Pr_{w_2}[R_2^D(A_2(x_{i_1}, w_2), i_2) = (x_{i_1})_{i_2}] \geq \delta_2. \qquad (5)$$

The probability over a random choice of w_1 and w_2 that both events occur is at least $\delta_1 + \delta_2 - 1$. If both events occur, then:

$$\begin{aligned} R^D(A(x_1, \ldots, x_r; w_1, w_2); i_1, i_2) &= R_2^D(R_1^D(A(x_1, \ldots, x_r; w_1, w_2), i_1), i_2) & (6)\\ &= R_2^D(R_1^D(A_1(A_2(x_1, w_2) \circ \cdots \circ A_2(x_r, w_2), w_1), i_1), i_2) & (7)\\ &= R_2^D(A_2(x_{i_1}, w_2), i_2) = (x_{i_1})_{i_2}. & (8) \end{aligned}$$

where (6) just applies the definition of R, (7) applies the definition of A, and (8) follows from our assumption that the events in (4) and (5) both occur.

We conclude that $\Pr_{w_1, w_2}[R^D(A(x; w_1, w_2), i) = x_i] \geq \delta_1 + \delta_2 - 1$, which is what was to be shown. □

We now apply Theorem 4 repeatedly:

Corollary 1. *Fix N, n, ϵ, and constant $\gamma < 1$. Let (E, A, R) be a*

$$(n, t_1 = O(\log n), m, d_1 = O(\log n), a = n^\gamma, b = a, \epsilon, \delta_1 = 1 - (1/\log N))$$

reconstructive disperser with the running time of E, A, R at most n^{c_1} for a universal constant c_1. Then (E', A', R') obtained by $\left(\frac{\log N}{(1-\gamma)\log n}\right)$ applications of Theorem 4 is a $(N, t = O(\log N), m, d = O(\log N), a = n^\gamma, b = a, \epsilon, \delta = 2/3)$ reconstructive disperser with the running time of E' and A' at most N^c, and the running time of R' at most n^c for a universal constant c.

Proof. We claim that after i compositions of (E, A, R) with itself, we obtain a

$$\Big(n_i = n^{i-(i-1)\gamma}, t_i = t_1 + (i-1)(d_1 + \log(n^{1-\gamma}+1)), m,$$
$$d_i = id_1, a, b, \epsilon, \delta_i = i(\delta_1 - 1) + 1\Big)$$

reconstructive disperser (E_i, A_i, R_i). This clearly holds for $i = 1$.

To see that it holds for arbitrary i, consider composing $(E_{i-1}, A_{i-1}, R_{i-1})$ with (E, A, R). By Theorem 4 and our inductive assumption, we get (E_i, A_i, R_i) with parameters:

$$\begin{aligned}
n_i &= n_{i-1}n/a = n^{i-1-(i-2)\gamma}n/n^\gamma = n^{i-(i-1)\gamma}\\
t_i &= \max(t_{i-1}, t_1) + d_1 + \log(n^{1-\gamma}+1)\\
&= [t_1 + (i-1)(d_1 + \log(n^{1-\gamma}+1))] + d_1 + \log(n^{1-\gamma}+1)\\
&= t_1 + i(d_1 + \log(n^{1-\gamma}+1))\\
d_i &= d_{i-1} + d_1 = id_1\\
\delta_i &= \delta_{i-1} + \delta_1 - 1 = (i-1)(\delta_1 - 1) + \delta_1 = i(\delta_1 - 1) + 1
\end{aligned}$$

as claimed. Note that $n_i > n^{i(1-\gamma)}$. Thus when $i = \frac{\log N}{(1-\gamma)\log n}$, we have $n_i \geq N$, and also $d_i = O(\log N)$ and $t_i = O(\log N)$. Also note that for sufficiently large n, $\delta_i = 1 - i(1/\log N) = 1 - \frac{1}{(1-\gamma)\log n} > 2/3$.

Finally, let $T(E_i), T(A_i), T(R_i)$ denote the running times of the functions E_i, A_i, R_i respectively. From the specification of the composition in Theorem 4, we see that

$$\begin{aligned}
T(E_i) &\leq \max(T(E_{i-1}) + n_{i-1}T(A), T(E))\\
T(A_i) &\leq T(A_{i-1}) + n_{i-1}T(A)\\
T(R_i) &\leq T(R_{i-1}) + T(R),
\end{aligned}$$

and then it is easy to verify by induction that $T(E_i), T(A_i) \leq in^{i+c_1}$ and $T(R_i) \leq in^{c_1}$. Plugging in $i = O(\log N/\log n)$ gives the claimed running times. □

5 Optimal hitting set generators

Now, we can use any one of the constructions of Theorem 1 in Corollary 1 to obtain an optimal HSG for arbitrary hardness. Specifically, our HSG is built from the N-bit truth table of a function that is hard for circuits of size k, has a seed length of $O(\log N)$, and outputs $k^{\Omega(1)}$ bits, while running in time poly(N). As usual, we assume $N < 2^{k^\eta}$ for any constant η; as otherwise we could just output the seed.

Theorem 5. *Let x be the N-bit truth table of a function that cannot be computed by circuits of size k, and set $n = k^{1/c}$ for a sufficiently large constant c. Let (E, A, R) be a*

$$(n, t = O(\log n), m = n^\beta, d = O(\log n), a = n^\gamma, b = n, \epsilon, \delta = 1/4)$$

reconstructive disperser from Theorem 1. If (E', A', R') is the reconstructive disperser obtained by applying Lemma 1 with $\alpha = 1/\log N$ and then applying Corollary 1, then $H_x(\cdot) = E'(x, \cdot)$ is an ϵ-HSG against circuits of size $m = k^{\beta/c}$.

Proof. After applying Lemma 1 with $\alpha = 1/\log N$, we have a

$$\left(n, t = O(\log n), m = n^{\beta}, d' = O(\log n), a' = (\log N)n^{\gamma}, b = n, \epsilon, \delta' = 1 - \frac{1}{\log N}\right)$$

reconstructive disperser. By the assumption on N stated before the theorem, we have that $a' \leq n^{\gamma'}$ for some constant $\gamma' < 1$. As noted following Definition 4, we can decrease b from n to a' trivially. Then, after applying Corollary 1 we obtain a $(N, O(\log N), m, O(\log N), n^{\gamma'}, a', \epsilon, 2/3)$ reconstructive disperser (E', A', R'), with the running time of E' and A' at most poly(N) and the running time of R' at most poly(n).

To satisfy Theorem 2 we need $c_1 \log N(n^{c_2}n^{\beta} + n^{\gamma'}) < k$, where c_1 and c_2 are universal constants; we can set c to ensure that this holds. Theorem 2 then states that $E'(x, \cdot)$ is an ϵ-HSG against circuits of size $m = n^{\beta} = k^{\beta/c}$. $\square$

As noted in the introduction, if we use Trevisan's extractor [1] as our starting object, then the entire construction requires only two ingredients: (1) any good list-decodable error-correcting code and (2) combinatorial designs (to obtain the original Trevisan extractor). In particular there is *no* hardness amplification or implicit list-decoding hidden in the construction, precisely because we are able to work with a starting object that only has *explicit reconstructivity* rather than implicit reconstructivity (the latter type of reconstructivity typically has required implicit list-decoding in some form or another).

6 Open problems

We mention briefly two interesting open problems related to this work.

First, is it possible to extend these results to two-sided objects, by giving a similar composition for reconstructive *extractors*? Because implicit reconstructivity of extractors is closely related to efficient *implicit* list-decodability, it is possible that such a result would give a new generic construction of implicitly list-decodable codes (in the sense of [2]) from *any* good list-decodable codes.

Second, is it possible to extend our result to the non-deterministic setting? Here there is an important technical issue of *resiliency* of HSGs discussed in [15]; obtaining a resilient HSG construction would lead to so-called "low-end" uniform hardness vs. randomness tradeoffs for the class AM. One possible route to constructing low-end resilient HSGs against nondeterministic circuits is to construct high-end resilient HSGs (typically an easier task) that possess the features needed to apply the composition in this paper – namely an associated advice function $A(x, w)$ computable in polynomial time, with w having length $O(\log n)$. In the currently known resilient construction [16], w has length much larger than $O(\log n)$.

Acknowledgements. We thank Ronen Shaltiel for his comments on an early draft of this paper.

References

1. Trevisan, L.: Extractors and pseudorandom generators. Journal of the ACM **48** (2002) 860–879
2. Sudan, M., Trevisan, L., Vadhan, S.: Pseudorandom generators without the XOR lemma. JCSS: Journal of Computer and System Sciences **62** (2001)
3. Ta-Shma, A., Zuckerman, D., Safra, S.: Extractors from Reed-Muller codes. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science. (2001)
4. Shaltiel, R., Umans, C.: Simple extractors for all min-entropies and a new pseudo-random generator. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science. (2001)
5. Impagliazzo, R., Shaltiel, R., Wigderson, A.: Near-optimal conversion of hardness into pseudo-randomness. In: Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science. (1999) 181–190
6. Impagliazzo, R., Shaltiel, R., Wigderson, A.: Extractors and pseudo-randomn generators with optimal seed-length. In: Proceedings of the Thirty-second Annual ACM Symposium on the Theory of Computing. (2000)
7. Ta-Shma, A., Umans, C., Zuckerman, D.: Loss-less condensers, unbalanced expanders, and extractors. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing. (2001) 143–152
8. Umans, C.: Pseudo-random generators for all hardnesses. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing. (2002) 627–634
9. Nisan, N., Wigderson, A.: Hardness vs randomness. Journal of Computer and System Sciences **49** (1994) 149–167
10. Impagliazzo, R., Wigderson, A.: P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing. (1997) 220–229
11. Ta-Shma, A., Zuckerman, D.: Extractor codes. IEEE Transactions on Information Theory **50** (2004) 3015–3025
12. Guruswami, V.: Better extractors for better codes? In: STOC. (2004) 436–444
13. Ta-Shma, A.: Storing information with extractors. Inf. Process. Lett. **83** (2002) 267–274
14. Buhrman, H., Lee, T., van Melkebeek, D.: Language compression and pseudo-random generators. In: IEEE Conference on Computational Complexity. (2004) 15–28
15. Gutfreund, D., Shaltiel, R., Ta-Shma, A.: Uniform hardness vs. randomness trade-offs for Arthur-Merlin games. In: 18th Annual IEEE Conference on Computational Complexity. (2003)
16. Miltersen, P.B., Vinodchandran, N.V.: Derandomizing Arthur-Merlin games using hitting sets. In: Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science. (1999) 71–80