

# Computational Hardness and Explicit Constructions of Error Correcting Codes

Mahdi Cheraghchi and Amin Shokrollahi and Avi Wigderson

**Abstract**—We outline a procedure for using pseudorandom generators to construct binary codes with good properties, assuming the existence of sufficiently hard functions. Specifically, we give a polynomial time algorithm, which for every integers  $n$  and  $k$ , constructs polynomially many linear codes of block length  $n$  and dimension  $k$ , most of which achieving the Gilbert-Varshamov bound. The success of the procedure relies on the assumption that the exponential time class of  $E \stackrel{\text{def}}{=} \text{DTIME}[2^{O(n)}]$  is not contained in the sub-exponential space class  $\text{DSPACE}[2^{o(n)}]$ .

The methods used in this paper are by now standard within computational complexity theory, and the main contribution of this note is observing that they are relevant to the construction of optimal codes. We attempt to make this note self contained, and describe the relevant results and proofs from the theory of pseudorandomness in some detail.

## I. INTRODUCTION

One of the central problems in coding theory is the construction of codes with extremal parameters. Typically, one fixes an alphabet size  $q$ , and two among the three fundamental parameters of the code (block-length, number of codewords, and minimum distance), and asks about extremal values of the remaining parameter such that there is a code over the given alphabet with the given parameters. For example, fixing the minimum distance  $d$  and the block-length  $n$ , one may ask for the largest number of codewords  $M$  such that there exists a code over the alphabet with  $q$  elements having  $n, M, d$  as its parameters, or in short, an  $[n, \log_q M, d]_q$ -code.

Answering this question in its full generality is extremely difficult, especially when the parameters are large. For this reason, researchers have concentrated on asymptotic assertions: to any  $[n, \log_q M, d]_q$ -code  $C$  we associate a point  $(\delta(C), R(C)) \in [0, 1]^2$ , where  $\delta(C) = d/n$  and  $R(C) = \log_q M/n$ . A particular point  $(\delta, R)$  is called *asymptotically achievable* (over a  $q$ -ary alphabet) if there exists a sequence  $(C_1, C_2, \dots)$  of codes of increasing block-length such that  $\delta(C_i) \rightarrow \delta$  and  $R(C_i) \rightarrow R$  as  $i \rightarrow \infty$ .

Even with this asymptotic relaxation the problem of determining the shape of the set of asymptotically achievable points remains difficult. Let  $\alpha_q(\delta)$  be defined as the supremum of all  $R$  such that  $(\delta, R)$  is asymptotically achievable over a  $q$ -ary alphabet. It is known that  $\alpha_q$  is a continuous function of  $\delta$  [1], that  $\alpha_q(0) = 1$ , and  $\alpha_q(\delta) = 0$  for

$\delta \geq (q-1)/q$ . However, for no  $\delta \in (0, (q-1)/q)$  and for no  $q$  is the value of  $\alpha_q(\delta)$  known.

What is known are lower and upper bounds for  $\alpha_q$ . The best lower bound known is due to Gilbert and Varshamov[2], [3] which states that  $\alpha_q(\delta) \geq 1 - h_q(\delta)$ , where the  $q$ -ary entropy function  $h_q$  is defined as

$$h_q(\delta) \stackrel{\text{def}}{=} -\delta \log_q \delta - (1-\delta) \log_q (1-\delta) + \delta \log_q (q-1).$$

Up until 1982, years of research had made it plausible to think that this bound is tight, i.e., that  $\alpha_q(\delta) = 1 - h_q(\delta)$ . Goppa's invention of algebraic-geometric codes [4], and the subsequent construction of Tsfasman, Vlăduț, and Zink [5] using curves with many points over a finite field and small genus showed however that the bound is not tight when the alphabet size is large enough. Moreover, Tsfasman et al. also gave a polynomial time construction of such codes (which has been greatly simplified since, see, e.g., [6]).

The fate of the binary alphabet is still open. Many researchers still believe that  $\alpha_2(\delta) = 1 - h_2(\delta)$ . In fact, for a randomly chosen linear code  $C$  (one in which the entries of a generator matrix are chosen independently and uniformly over the alphabet) and for any positive  $\varepsilon$  we have  $R(C) \geq 1 - h_q(\delta(C)) - \varepsilon$  with high probability (with probability at least  $1 - 2^{-nc_\varepsilon}$  where  $n$  is the block-length and  $c_\varepsilon$  is a constant depending on  $\varepsilon$ ). However, even though this shows that most randomly chosen codes are arbitrarily close to the Gilbert-Varshamov bound, no explicit polynomial time construction of such codes is known when the alphabet size is small (e.g., for binary alphabets).

In this paper, we use the technology of pseudorandom generators which has played a prominent role in the theoretical computer science research in recent years to (conditionally) produce, for any block-length  $n$  and any rate  $R < 1$ , a list of  $\text{poly}(n)$  many codes of block length  $n$  and designed rate  $R$  (over an arbitrary alphabet) such that a very large fraction of these codes has parameters arbitrarily close to the Gilbert-Varshamov bound. Here,  $\text{poly}(n)$  denotes a polynomial in  $n$ . While it is possible to compute this polynomial explicitly, we will concentrate in this paper on the rough result, and refrain from obtaining the best possible results.

In a nutshell, our approach can be described as follows. We will first identify a boolean function  $f$  of which we assume that it satisfies a certain complexity theoretic assumption. More precisely, we assume that the function cannot be computed by algorithms that require sub-exponential amount of memory. A natural candidate for such a function is introduced below (see Lemma 9). This function is then

M. Cheraghchi and A. Shokrollahi are with the School of Basic Sciences, and the School of Computer Science and Communications, EPFL, CH-1015 Lausanne, Switzerland (email: {mahdi.cheraghchi, amin.shokrollahi}@epfl.ch).

A. Wigderson is with the School of Mathematics, Institute for Advanced Study, Princeton, NJ 08540, USA (email: avi@ias.edu).

extended to produce  $nk$  bits from  $O(\log n)$  bits. This extended function is called a *pseudorandom generator*. The main point about this extended function is that the  $nk$  bits produced cannot be distinguished from random bits by a Turing machine with restricted resources. In our case, the output cannot be distinguished from a random sequence when a Turing machine is used which uses only an amount of space that is polynomially bounded in the length of its input.

These new  $nk$  bits are regarded as the entries of a generator matrix of a code. Varying the base  $O(\log n)$  bits in all possible ways gives us a polynomially long list of codes of which we can show that a majority lies asymptotically on the Gilbert-Varshamov bound, provided the hardness assumption is satisfied.

The rest of the paper is organized as follows. The next section introduces the basic notation we will need from complexity theory. In Section III we give a broad sketch describing the main elements of the construction. Section IV describes the pseudorandom generator in more details. In Section V we apply the pseudorandom generator to achieve our basic construction.

A natural question is whether it is possible to collapse this polynomially long list to one code; this would essentially settle the problem of polynomial time construction of binary codes which asymptotically meet the Gilbert-Varshamov bound. This is an interesting open problem. We will elaborate on it in the last section.

## II. BASIC NOTATION

We begin with the definitions of the terms we will use throughout the paper. For simplicity, we restrict ourselves to the particular cases of our interest and will avoid presenting the definitions in full generality. See [7], [8] for a comprehensive account of coding theoretic notions and [9] for complexity-theoretic notions.

Our main tool in this work is a hardness-based pseudorandom generator. Informally, this is an efficient algorithm that receives a sequence of truly random bits at input and outputs a much longer sequence *looking* random to any distinguisher with bounded computational power. This property of the pseudorandom generator can be guaranteed to hold by assuming the existence of functions that are hard to compute for certain computational devices. This is indeed a broad sketch; Depending on what we precisely mean by the quantitative measures just mentioned, we come to different definitions of pseudorandom generators. Here we will be mainly interested in computational hardness against algorithms with *bounded* space complexity. Hereafter, we will use the shorthand  $\text{DSPACE}[s(n)]$  to denote the class of problems solvable with  $O(s(n))$  bits of working memory and  $\text{E}$  for the class of problems solvable in time  $2^{O(n)}$  (i.e.,  $\text{E} = \bigcup_{c \in \mathbb{N}} \text{DTIME}[2^{cn}]$ , where  $\text{DTIME}[t(n)]$  stands for the class of problems deterministically solvable in time  $O(t(n))$ ).

Certain arguments that we use in this work require *non-uniform* computational models. Hence, we will occasionally

refer to algorithms that receive *advice strings* to help them carry out their computation. Namely, in addition to the input string, the algorithm receives an *advice* string whose content only depends on the *length* of the input and not the input itself. It is assumed that, for every  $n$ , there is an advice string that makes the algorithm work correctly on *all* inputs of length  $n$ . We will use the notation  $\text{DSPACE}[f(n)]/g(n)$  for the class of problems solvable by algorithms that receive  $g(n)$  bits of advice and use  $O(f(n))$  bits of working memory.

*Definition 1:* Let  $S: \mathbb{N} \rightarrow \mathbb{N}$  be a (constructible) function. A boolean function  $f: \{0, 1\}^* \rightarrow \{0, 1\}$  is said to have *hardness*  $S$  if for every algorithm  $A$  in  $\text{DSPACE}[S(n)]/O(S(n))$  and infinitely many  $n$  (and no matter how the advice string is chosen) it holds that

$$|\Pr_x[A(x) = f(x)] - 1/2| < 1/S(n),$$

where  $x$  is uniformly sampled from  $\{0, 1\}^n$ .

Obviously, any boolean function can be trivially computed correctly on at least half of the inputs by an algorithm that always outputs a constant value (either 0 or 1). Intuitively, for a hard function no *efficient* algorithm can do much better. In this work, our central hardness assumption will be the following:

*Assumption 2:* There is a boolean function in  $\text{E}$  with hardness at least  $2^{\varepsilon n}$ , for some constant  $\varepsilon > 0$ .

The term *pseudorandom generator* emphasizes the fact that it is information-theoretically impossible to transform a sequence of truly random bits into a longer sequence of truly random bits, hence the best a transformation with a nontrivial stretch can do is to generate bits that *look* random to a *particular family* of observers. To make this more precise, we need to define *computational indistinguishability* first.

*Definition 3:* Let  $p = \{p_n\}$  and  $q = \{q_n\}$  be families of probability distributions, where  $p_n$  and  $q_n$  are distributed over  $\{0, 1\}^n$ . Then  $p$  and  $q$  are  $(S, \ell, \varepsilon)$ -*indistinguishable* (for some  $S, \ell: \mathbb{N} \rightarrow \mathbb{N}$  and  $\varepsilon: \mathbb{N} \rightarrow (0, 1)$ ) if for every algorithm  $A$  in  $\text{DSPACE}(S(n))/O(\ell(n))$  and infinitely many  $n$  (and no matter how the advice string is chosen) we have that

$$|\Pr_x[A(x) = 1] - \Pr_y[A(y) = 1]| < \varepsilon(n),$$

where  $x$  and  $y$  are sampled from  $p_n$  and  $q_n$ , respectively.

This is in a way similar to computational hardness. Here the *hard task* is *telling the difference* between the sequences generated by different sources. In other words, two probability distributions are indistinguishable if any resource-bounded observer is *fooled* when given inputs sampled from one distribution rather than the other. Note that this may even hold if the two distributions are not statistically close to each other.

Now we are ready to define pseudorandom generators we will later need.

*Definition 4:* A deterministic algorithm that computes a function  $G: \{0, 1\}^{c \log n} \rightarrow \{0, 1\}^n$  (for some constant  $c > 0$ ) is called a (high-end) *pseudorandom generator* if the following conditions hold:

- 1) It runs in polynomial time with respect to  $n$ .

- 2) Let the probability distribution  $G_n$  be defined uniformly over the range of  $G$  restricted to outputs of length  $n$ . Then the family of distributions  $\{G_n\}$  is  $(n, n, 1/n)$ -indistinguishable from the uniform distribution.

An input to the pseudorandom generator is referred to as a *random seed*. Here the length of the output as a function of the seed length  $s$ , known as the *stretch* of the pseudorandom generator, is required to be the exponential function  $2^{s/c}$ .

### III. THE MAIN INGREDIENTS

Our observation is based on the composition of the following facts:

- 1) *Random codes achieve the Gilbert-Varshamov bound:* It is well known that a simple randomized algorithm that chooses the entries of a generator matrix uniformly at random obtains a linear code satisfying the Gilbert-Varshamov bound with overwhelming probability [3].
- 2) *Finding the minimum distance of a (linear) code can be performed in linear space:* One can simply enumerate all the codewords to find the minimum weight codeword, and hence, the distance of the code. This only requires linear amount of memory with respect to the block length.
- 3) *Provided a hardness condition, namely that sub-exponential space algorithms cannot compute all the problems in E, every linear space algorithm can be fooled by an explicit pseudorandom generator:* We will elaborate on this argument in the next section. Roughly speaking, a boolean function satisfying the hardness condition can be used to generate a large number of *pseudorandom* bits from a very short (logarithmically long) truly random seed. Then as we will show later, any randomized algorithm working in linear space is fooled (i.e., its behavior does not considerably change) if we use pseudorandom bits in place of the truly random ones assumed by the algorithm. This will be shown by arguing that any algorithm that is not fooled by the pseudorandom generator can be used to compute the hard function the generator is based on.

In Section V we will see how to compose the statements above to conditionally obtain a small (polynomially large) and explicit family of (linear) codes in which all but a sub-constant fraction of the codes achieve the GV bound. The idea is that the combination of the randomized algorithm we mentioned in the first item above with the linear space algorithm that decides whether the output is a code on the GV bound is a linear space algorithm and has to be fooled by the pseudorandom generator. Hence it follows that the randomized code construction works even if we use pseudorandom bits rather than truly random ones. This would imply an explicit construction of a polynomially large ensemble of good error correcting codes.

### IV. THE PSEUDORANDOM GENERATOR

A pseudorandom generator, as we just defined, extends a truly random sequence of bits into an exponentially long

sequence that *looks* random to any efficient distinguisher. From the definition it is not at all clear whether such an object could exist. In fact the existence of pseudorandom generators (even much weaker than our definition) is not yet known. However, there are various constructions of pseudorandom generators based on unproven (but seemingly plausible) assumptions. The presumed assumption is typically chosen in line with the same guideline, namely, a computational task being *intractable*. For instance, the early constructions of [10] and [11] are based on the intractability of certain number-theoretic problems, namely, integer factorization and the discrete logarithm function. Yao [12] extends these ideas to obtain pseudorandomness from one-way permutations. This is further generalized by [13] who show that the existence of *any* one-way function is sufficient. However, these ideas are mainly motivated by cryptographic applications and often require strong assumptions.

The prototypical pseudorandom generator for the applications in derandomization, which is of our interest, is due to Nisan and Wigderson[14]. They provide a broad range of pseudorandom generators with different strengths based on a variety of hardness assumptions. In rough terms, their generator works by taking a hard function for a certain complexity class, evaluating it in carefully chosen points (related to the choice of the random seed), and outputting the resulting sequence. Then one can argue that an efficient distinguisher can be used to efficiently compute the hard function, contradicting the assumption. Note that for certain complexity classes, hard functions are *provably* known. However, they typically give generators too weak to be applied in typical derandomizations. Here we simply apply the Nisan-Wigderson construction to obtain a pseudorandom generator which is *robust* against space-efficient computations. This is shown in the following theorem:

*Theorem 5:* Assumption 2 implies the existence of a pseudorandom generator as in Definition 4. That is to say, suppose that there is a constant  $\varepsilon > 0$  and a boolean function computable in time  $2^{O(n)}$  that has hardness  $2^{\varepsilon n}$ . Then there exists a function  $G: \{0, 1\}^{O(\log n)} \rightarrow \{0, 1\}^n$  computable in time polynomial in  $n$  whose output (when given uniformly random bits at input) is indistinguishable from the uniform distribution for all algorithms in  $\text{DSPACE}[n]/O(n)$ .

*Proof Sketch [14]:* Let  $f$  be a function satisfying Assumption 2 for some fixed  $\varepsilon > 0$ , and recall that we intend to generate  $n$  pseudorandom bits from a truly random seed of length  $\ell$  which is only logarithmically long in  $n$ .

The idea of the construction is as follows: We evaluate the hard function  $f$  in  $n$  carefully chosen points, each of the same length  $m$ , where  $m$  is to be determined shortly. Each of these  $m$ -bit long inputs is obtained from a particular subset of the  $\ell$  bits provided by the random seed. This can be conveniently represented in a matrix form: Let  $\mathcal{D}$  be an  $n \times \ell$  binary matrix, each row of which having the same weight  $m$ . Now the pseudorandom generator  $G$  is described as follows: The  $i^{\text{th}}$  bit generated by  $G$  is the evaluation of  $f$  on the projection of the  $\ell$ -bit long input sequence to those coordinates indicated by the  $i^{\text{th}}$  row of  $\mathcal{D}$ . Note that because

$f$  is in E, the output sequence can be computed in time polynomial in  $n$ , as long as  $m$  is logarithmically small.

As we will shortly see, it turns out that we need  $\mathcal{D}$  to satisfy a certain *small-overlap* property. Namely, we require the bitwise product of each pair of the rows of  $\mathcal{D}$  to have weight at most  $\log n$ . A straightforward counting argument shows that, for a logarithmically large value of  $m$ , the parameter  $\ell$  can be kept logarithmically small as well. In particular, for the particular choice of  $m \stackrel{\text{def}}{=} \frac{2}{\varepsilon} \log n$ , the matrix  $\mathcal{D}$  exists with  $\ell = O(\log n)$ . Moreover, rows of the matrix can be constructed (in time polynomial in  $n$ ) using a simple greedy algorithm.

To show that our construction indeed gives us a pseudorandom generator, suppose that there is an algorithm  $A$  working in  $\text{DSPACE}[n]/O(n)$  which is able to distinguish the output of  $G$  from a truly random sequence with a bias of at least  $1/n$ . That is, for all large enough  $n$  it holds that

$$\delta \stackrel{\text{def}}{=} |\Pr_y[A^{\alpha(n)}(y) = 1] - \Pr_x[A^{\alpha(n)}(G(x)) = 1]| \geq 1/n,$$

where  $x$  and  $y$  are distributed uniformly in  $\{0,1\}^\ell$  and  $\{0,1\}^n$ , respectively, and  $\alpha(n)$  in the superscript denotes an advice string of linear length (that only depends on  $n$ ). The goal is to transform  $A$  into a space-efficient (and non-uniform) algorithm that approximates  $f$ , obtaining a contradiction.

Without loss of generality, let the quantity inside the absolute value be non-negative (the argument is similar for the negative case). Let the distribution  $D_i$  (for  $0 \leq i \leq n$ ) over  $\{0,1\}^n$  be defined by concatenation of the length- $i$  prefix of  $G(x)$ , when  $x$  is chosen uniformly at random from  $\{0,1\}^\ell$ , with a boolean string of length  $n-i$  obtained uniformly at random. Define  $p_i$  as  $\Pr_z[A^{\alpha(n)}(z) = 1]$ , where  $z$  is sampled from  $D_i$ , and let  $\delta_i \stackrel{\text{def}}{=} p_{i-1} - p_i$ . Note that  $D_0$  is the uniform distribution and  $D_n$  is uniformly distributed over the range of  $G$ . Hence, we have  $\sum_{i=1}^n \delta_i = p_0 - p_n = \delta \geq 1/n$ , meaning that for some  $i$ ,  $\delta_i \geq 1/n^2$ . Fix this  $i$  in the sequel.

Without loss of generality, assume that the  $i^{\text{th}}$  bit of  $G(x)$  depends on the first  $m$  bits of the random seed. Now consider the following randomized procedure  $B$ : Given  $i-1$  input bits  $u_1, \dots, u_{i-1}$ , choose a binary sequence  $r_i, \dots, r_n$  uniformly at random and compute  $A^{\alpha(n)}(u_1, \dots, u_{i-1}, r_i, \dots, r_n)$ . If the output was 1 return  $r_i$ , otherwise, return the negation of  $r_i$ . It is straightforward to show that

$$\Pr_{x,r}[B(G(x)_1^{i-1}) = G(x)_i] \geq \frac{1}{2} + \delta_i. \quad (1)$$

Here,  $G(x)_1^{i-1}$  and  $G(x)_i$  are shorthands for the  $(i-1)$ -bit long prefix of  $G(x)$  and the  $i^{\text{th}}$  bit of  $G(x)$ , respectively, and the probability is taken over the choice of  $x$  and the internal coins of  $B$ .

So far we have constructed a linear-time probabilistic procedure for *guessing* the  $i^{\text{th}}$  pseudorandom bit from the first  $i-1$  bits. By averaging, we note that there is a particular choice of  $r_i, \dots, r_n$ , independent of  $x$ , that preserves the bias given in (1). Furthermore, note that the function  $G(x)_i$  we are trying to guess, which is in fact  $f(x_1, \dots, x_m)$ , does

not depend on  $x_{m+1}, \dots, x_\ell$ . Therefore, again by averaging we see that these bits can also be fixed. Therefore, for a given sequence  $x_1, \dots, x_m$ , one can compute  $G(x)_1^{i-1}$ , feed it to  $B$  (having known the choices we have fixed), and guess  $G(x)_i$  with the same bias as in (1). The problem is of course that  $G(x)_1^{i-1}$  does not seem to be easily computable. However, what we know is that each bit of this sequence depends only on  $\log n$  bits of  $x_1, \dots, x_m$ , followed by the construction of  $\mathcal{D}$ . Hence, having fixed  $x_{m+1}, \dots, x_\ell$ , we can trivially describe each bit of  $G(x)_1^{i-1}$  by a boolean formula (or a boolean circuit) of exponential size (that is, of size  $O(2^{\log n}) = O(n)$ ). These  $i-1 = O(n)$  boolean formulae can be encoded as an additional advice string of length  $O(n^2)$  (note that their descriptions only depend on  $n$ ), implying that  $G(x)_1^{i-1}$  can be computed in linear space using  $O(n^2)$  bits of advice.

All the choices we have fixed so far (namely,  $i, r_i, \dots, r_n, x_{m+1}, \dots, x_\ell$ ) only depend on  $n$  and can be *absorbed* into the advice string as well<sup>1</sup>. Combined with the *bit-guessing* algorithm we just described, this gives us a linear-space algorithm that needs an advice of quadratic length and correctly computes  $f(x_1, \dots, x_m)$  on at least a  $\frac{1}{2} + \delta_i$  fraction of inputs, which is off from  $1/2$  by a bias of at least  $1/n^2$ . But this is not possible by the hardness of  $f$ , which is assumed to be at least  $2^{\varepsilon m} = n^2$ . Thus,  $G$  must be a pseudorandom generator. ■

The above proof uses a function that is completely unpredictable for every efficient algorithm. Impagliazzo and Wigderson [15] improve the construction to show that this requirement can be relaxed to one that only requires a *worst case* hardness, meaning that the function computed by any *efficient* (non-uniform) algorithm needs to differ from the hard function on at least one input. In our application, this translates into the following hardness assumption:

*Assumption 6:* There is a constant  $\varepsilon > 0$  and a function  $f$  in E such that every algorithm in  $\text{DSPACE}[S(n)]/O(S(n))$  that correctly computes  $f$  requires  $S(n) = \Omega(2^{\varepsilon n})$ .

The idea of their result (which was later reproved in [16] using a coding-theoretic argument) is to *amplify* the given hardness, that is, to transform a worst-case hard function in E to another function in E which is hard on average. In our setting, this gives us the following:

*Theorem 7:* Assumption 6 implies Assumption 2 and hence, the existence of pseudorandom generators.

*Proof Idea [16]:* Let a function  $f$  be hard in worst case. Consider the truth table of  $f$  as a string  $x$  of length  $N \stackrel{\text{def}}{=} 2^n$ . The main ingredient of the the proof is a linear code  $\mathcal{C}$  with dimension  $N$  and length polynomial in  $N$ , which is obtained by concatenation of a Reed-Muller code with the Hadamard code. The code is list-decodable up to a fraction  $\frac{1}{2} - \varepsilon$  of errors, for arbitrary  $\varepsilon > 0$ . Moreover, decoding can be done in sub-linear time, that is, by querying the *received word* only at a small number of (randomly chosen) positions.

<sup>1</sup>Alternatively, one can avoid using this additional advice by enumerating over all possible choices and taking a majority vote. However, this does not decrease the total advice length by much.

Then the truth table of the transformed function  $g$  can be simply defined as the encoding of  $x$  with  $\mathcal{C}$ . Hence  $g$  can be evaluated at any point in time polynomial in  $N$ , which shows that  $g \in \text{E}$ . Further, suppose that an algorithm  $A$  can space-efficiently compute  $g$  correctly in a fraction of points non-negligibly bounded away from  $1/2$  (possibly using an advice string). Then the function computed by  $A$  can be seen as a *corrupted* version of the *codeword*  $g$  and can be efficiently *recovered* using the list-decoding algorithm. From this, one can obtain a space-efficient algorithm for computing  $f$ , contradicting the hardness of  $f$ . Hence  $g$  has to be hard on average. ■

While the above result seems to require hardness against non-uniform algorithms (as phrased in Assumption 6), we will see that the hardness assumption can be further relaxed to the following, which only requires hardness against uniform algorithms:

*Assumption 8:* The complexity class  $\text{E}$  is not contained in  $\text{DSPACE}[2^{o(n)}]$ .

*Remark:* A result by Hopcroft et al. [17] shows a deterministic simulation of time by space. Namely, they prove that  $\text{DTIME}[t(n)] \subseteq \text{DSPACE}[t(n)/\log t(n)]$ . However, this result is not strong enough to influence the hardness assumption above. To violate the assumption, a much more space-efficient simulation in the form  $\text{DTIME}[t(n)] \subseteq \text{DSPACE}[t(n)^{o(1)}]$  is required.

Before we show the equivalence of the two assumptions (namely, Assumption 6 and Assumption 8), we address the natural question of how to construct an *explicit* function to satisfy the required hardness assumption (after all, evaluation of such a function is needed as part of the pseudorandom generator construction). One possible candidate (which is a canonical hard function for  $\text{E}$ ) is proposed in the following lemma:

*Lemma 9:* Let  $\mathcal{L}_{\text{E}}$  be the set (encoded in binary)  $\{\langle M, x, t, i \rangle \mid M \text{ is a Turing machine, where given input } x \text{ at time } t \text{ the } i^{\text{th}} \text{ bit of its configuration is } 1\}$ , and let the boolean function  $f_{\text{E}}$  be its characteristic function. Then if Assumption 8 is true, it is satisfied by  $f_{\text{E}}$ .

*Proof:* First we show that  $\mathcal{L}_{\text{E}}$  is complete for  $\text{E}$  under Turing reductions bounded in linear space. The language being in  $\text{E}$  directly follows from the efficient constructions of universal Turing machines. Namely, given a properly-encoded input  $\langle M, x, t, i \rangle$ , one can simply simulate the Turing machine  $M$  on  $x$  for  $t$  steps and decide according to the configuration obtained at time  $t$ . This indeed takes exponential time. Now let  $L$  be any language in  $\text{E}$  which is computable by a Turing machine  $M$  in time  $2^{cn}$ , for some constant  $c > 0$ . For a given  $x$  of length  $n$ , using an oracle for solving  $f_{\text{E}}$ , one can query the oracle with inputs of the form  $\langle M, x, 2^{cn}, i \rangle$  (where the precise choice of  $i$  depends on the particular encoding of the configurations) to find out whether  $M$  is in an accepting state, and hence decide  $L$ . This can obviously be done in space linear in  $n$ , which concludes the completeness of  $\mathcal{L}_{\text{E}}$ . Now if Assumption 8 is true and is not satisfied by  $f_{\text{E}}$ , this completeness result allows one to compute all problems in  $\text{E}$  in sub-exponential time, which

contradicts the assumption. ■

The following lemma shows that this seemingly weaker assumption is in fact sufficient for our pseudorandom generator:

*Lemma 10:* Assumptions 6 and 8 are equivalent.

*Proof:* This argument is based on [18, Section 5.3]. First we observe that, given a *black box*  $C$  that receives  $n$  input bits and outputs a single bit, it can be verified in linear space whether  $C$  computes the restriction of  $f_{\text{E}}$  to inputs of length  $n$ . To see this, consider an input of the form  $\langle M, x, t, i \rangle$ , as in the statement of Lemma 9. The correctness of  $C$  can be explicitly checked when the time parameter  $t$  is zero (that is,  $C$  has to agree with the initial configuration of  $M$ ). Moreover, for every time step  $t > 0$ , the answer given by  $C$  has to be consistent with that of the previous time step (namely, the transition made at the location of the head should be legal and every other position of the tape should remain unchanged). Thus, one can verify  $C$  simply by enumerating all possible inputs and verifying whether the the answer given by  $C$  remains consistent across subsequent time steps. This can obviously be done in linear space.

Now suppose that Assumption 8 is true and hence, by Lemma 9, is satisfied by  $f_{\text{E}}$ . That is, there is a constant  $\varepsilon > 0$  such that every algorithm for computing  $f_{\text{E}}$  requires space  $O(2^{\varepsilon n})$ . Moreover, assume that there is an algorithm  $A$  working in  $\text{DSPACE}[S(n)]/O(S(n))$  that computes  $f_{\text{E}}$ . Using the verification procedure described above, one can (uniformly) simulate  $A$  in space  $O(S(n))$  by enumerating all choices of the advice string and finding the one that makes the algorithm work correctly. Altogether this requires space  $O(S(n))$ . Combined with the hardness assumption, we conclude that  $S(n) = \Omega(2^{\varepsilon n})$ . The converse direction is obvious. ■

Putting everything together, we obtain a very strong pseudorandom generator as follows:

*Corollary 11:* Assumption 8 implies the existence of pseudorandom generators whose output of length  $n$  is  $(n, n, 1/n)$ -indistinguishable from the uniform distribution.

## V. DERANDOMIZED CODE CONSTRUCTION

As mentioned before, the bound given by Gilbert and Varshamov[2], [3] states that, for a  $q$ -ary alphabet, large enough  $n$ , and for any value of  $0 \leq \delta \leq (q-1)/q$ , there are codes with length  $n$ , relative distance at least  $\delta$  and rate  $r \geq 1 - h_q(\delta)$ , where  $h_q$  is the  $q$ -ary entropy function. Moreover, a random linear code (having each entry of its generator matrix chosen uniformly at random) achieves this bound. In fact, for all  $0 \leq r \leq 1$ , in the family of linear codes with length  $n$  and (designed) dimension  $nr$ , all but only a sub-constant fraction of the codes achieve the bound when  $n$  grows to infinity. However, the number of codes in the family is exponentially large ( $q^{nr}$ ) and we do not have an *a priori* indication on which codes in the family are good. Putting it differently, a randomized algorithm that merely outputs a random generator matrix *succeeds* in producing a code on the GV bound with probability  $1 - o(1)$ . However, the number of random bits needed by the algorithm is  $nk \log q$ .

For simplicity, in the sequel we only focus on binary codes, for which no explicit construction approaching the GV bound is known.

The randomized procedure above can be considerably derandomized by considering a more restricted family of codes. Namely, fix a length  $n$  and a basis for the finite field  $\mathbb{F}_m$ , where  $m \stackrel{\text{def}}{=} 2^{n/2}$ . Then over such a basis there is a natural isomorphism between the elements of  $\mathbb{F}_m$  and the elements of the vector space  $\mathbb{F}_2^{n/2}$ . Now for each  $\alpha \in \mathbb{F}_m$ , define the code  $C_\alpha$  as the set  $\{\langle x, \alpha x \rangle \mid x \in \mathbb{F}_m\}$ , where the elements are encoded in binary<sup>2</sup>. This binary code has rate  $1/2$ . Further, it is well known that  $C_\alpha$  achieves the GV bound for all but  $1 - o(1)$  fraction of the choices of  $\alpha$ . Hence in this family a randomized construction can obtain very good codes using only  $n/2$  random bits. Here we see how the pseudorandom generator constructed in the last section can dramatically reduce the amount of randomness needed in all code constructions. First we propose a general framework that can be employed to derandomize a wide range of combinatorial constructions.

*Lemma 12:* Let  $\mathcal{S}$  be a family of combinatorial objects of (binary-encoded) length  $n$ , in which an  $\varepsilon$  fraction of the objects satisfy a property  $P$ . Moreover, suppose that the family is efficiently samplable, that is, there is a polynomial-time algorithm (in  $n$ ) that, for a given  $i$ , generates the  $i^{\text{th}}$  member of the family. Further assume that the property  $P$  is verifiable in polynomial space. Then for every constant  $k > 0$ , under Assumption 8, there is a constant  $\ell$  and an efficiently samplable subset of  $\mathcal{S}$  of size at most  $n^\ell$  in which at least an  $\varepsilon - n^{-k}$  fraction of the objects satisfy  $P$ .

*Proof:* Let  $A$  be the composition of the sampling algorithm with the verifier for  $P$ . By assumption,  $A$  needs space  $n^s$ , for some constant  $s$ . Furthermore, when the input of  $A$  is chosen randomly, it outputs 1 with probability at least  $\varepsilon$ . Suppose that the pseudorandom generator of Corollary 11 transforms  $c \log n$  truly random bits into  $n$  pseudorandom bits, for some constant  $c > 0$ . Now it is just enough to apply the pseudorandom generator on  $c \cdot \max\{s, k\} \cdot \log n$  random bits and feed  $n$  of the resulting pseudorandom bits to  $A$ . By this construction, when the input of the pseudorandom generator is chosen uniformly at random,  $A$  must still output 1 with probability  $\varepsilon - n^{-k}$  as otherwise the pseudorandomness assumption would be violated. Now the combination of the pseudorandom generator and  $A$  gives the efficiently samplable family of the objects we want, for  $\ell \stackrel{\text{def}}{=} c \cdot \max\{s, k\}$ , as the random seed runs over all the possibilities. ■

As the distance of a code is obviously computable in linear space by enumeration of all the codewords, the above lemma immediately implies the existence of a (constructible) polynomially large family of codes in which at least  $1 - n^{-k}$  of the codes achieve the GV bound, for arbitrary  $k$ .

*Remark:* As shown in the original work of Nisan and Wigderson [14] (followed by the hardness amplification of Impagliazzo and Wigderson [15]) all randomized

polynomial-time algorithms (namely, the complexity class BPP) can be fully derandomized under the assumption that  $E$  cannot be computed by boolean circuits of sub-exponential size. This assumption is also sufficient to derandomize probabilistic constructions that allow a (possibly non-uniform) polynomial-time verification procedure for deciding whether a particular object has the desirable properties. For the case of good error-correcting codes, this could work if we knew of a procedure for computing the minimum distance of a linear code using circuits of size polynomial in the length of the code. However, it turns out that (the decision version of) this problem is NP-complete [20], and even the approximation version remains NP-complete [21]. This makes such a possibility unlikely.

However, a key observation, due to Klivans and van Melkebeek [22], shows that the Nisan-Wigderson construction (as well as the Impagliazzo-Wigderson amplification) can be *relativized*. Namely, starting from a hardness assumption for a certain family of *oracle circuits* (i.e., boolean circuits that can use special gates to compute certain boolean functions as *black box*) one can obtain pseudorandom generators secure against oracle circuits of the same family. In particular, this implies that any probabilistic construction that allows polynomial time verification using NP oracles (including the construction of good error-correcting codes) can be derandomized by assuming that  $E$  cannot be computed by sub-exponential sized boolean circuits that use NP oracle gates. However, the result given by Lemma 12 can be used to derandomize a more general family of probabilistic constructions, though it needs a slightly stronger hardness assumption which is still plausible.

## VI. FUTURE WORK

So far, our construction (conditionally) gives a polynomially long family of codes, most of which achieving the GV bound. Though this is an exponential improvement upon the random construction, the question of finding a smaller family and ultimately, a single good code, is yet to be addressed.

Starting with a polynomially large family, it seems a plausible approach to try to (efficiently) combine all the codes in a clever way so as to obtain a single code that *inherits* the average properties of the codes in the family (in particular, the achievement of the GV bound). As the number of codes in such a family is small, this would give an explicit (conditional) construction of binary codes achieving the bound. A similar direction could be an attempt to construct a product of two codes with the same rate that (approximately) preserves the common rate and gives a distance *close enough* to the better of the two.

While our construction can be applied to general linear codes, it seems very appealing to try to delve more into the combinatorial structure of restricted families known to contain good codes. For instance, if it so happens that for a particular family of exponentially many codes, containing a non-negligible fraction of codes on the GV bound, the problem of finding out whether a particular member achieves

<sup>2</sup>These codes are attributed to J. M. Wozencraft (see [19]).

the bound is solvable in polynomial time, then under Assumption 8 we directly obtain an explicit construction of one good code. Note that this will be automatically guaranteed if the distance of the codes in the family concentrates around a central value, as in that case the problem of testing a code is as easy as computing the rank of the generator matrix.

As emphasized in Lemma 12, our result is very general and, besides construction of good error-correcting codes (which is the main focus of this work), can be applied to other interesting construction problems with polynomial-space verification algorithms. We defer a more detailed elaboration to the extended version.

#### REFERENCES

- [1] Y. Manin, "What is the maximum number of points on a curve over  $\mathbb{F}_2$ ?" *J. Fac. Sci. Tokio*, vol. 28, pp. 715–720, 1981.
- [2] E. N. Gilbert, "A comparison of signaling alphabets," *Bell System Technical Journal*, vol. 31, pp. 504–522, 1952.
- [3] R. R. Varshamov, "Estimate of the number of signals in error correcting codes," *Doklady Akademii Nauk SSSR*, vol. 117, pp. 739–741, 1957.
- [4] V. Goppa, "Codes on algebraic curves," *Sov. Math. Dokl.*, vol. 24, pp. 170–172, 1981.
- [5] M. Tsfasman, S. Vlăduț, and T. Zink, "Modular curves, Shimura curves, and Goppa codes better than the Varshamov-Gilbert bound," *Math. Nachrichten*, vol. 109, pp. 21–28, 1982.
- [6] A. Garcia and H. Stichtenoth, "A tower of Artin-Schreier extensions of function fields attaining the Drinfeld-Vladut bound," *Invent. Math.*, vol. 121, pp. 211–222, 1995.
- [7] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam: Elsevier, 1978.
- [8] J. H. van Lint, *Introduction to Coding Theory*. Berlin: Springer-Verlag, 1999.
- [9] C. H. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1994.
- [10] A. Shamir, "On the generation of cryptographically strong pseudorandom sequences," *ACM Transactions on Computer Systems*, vol. 1, no. 1, pp. 38–44, 1983.
- [11] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudorandom bits," *SIAM Journal on Computing*, vol. 13, no. 4, pp. 850–864, 1984.
- [12] A. Yao, "Theory and applications of trapdoor functions," in *Proceedings of the 23<sup>rd</sup> FOCS*, 1982, pp. 80–91.
- [13] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1364–1396, 1999.
- [14] N. Nisan and A. Wigderson, "Hardness vs. randomness," *Journal of Computer and Systems Sciences*, vol. 49, no. 2, pp. 149–167, 1994.
- [15] R. Impagliazzo and A. Wigderson, "P = BPP unless E has sub-exponential circuits: derandomizing the XOR lemma," in *Proceedings of the 29<sup>th</sup> STOC*, 1997, pp. 220–229.
- [16] M. Sudan, L. Trevisan, and S. Vadhan, "Pseudorandom generators without the XOR lemma," *Journal of Computer and Systems Sciences*, vol. 62, no. 2, pp. 236–266, 2001.
- [17] J. Hopcroft, W. Paul, and L. Valiant, "On time versus space," *Journal of the ACM*, vol. 24, pp. 332–337, 1977.
- [18] P. B. Miltersen, "Derandomizing complexity classes," *Book chapter in Handbook of Randomized Computing*, Kluwer Academic Publishers, 2001.
- [19] J. L. Massey, *Threshold decoding*. Cambridge, Massachusetts, USA: MIT Press, 1963.
- [20] A. Vardy, "The intractability of computing the minimum distance of a code," *IEEE Transactions on Information Theory*, vol. 43, no. 6, pp. 1757–1766, 1997.
- [21] I. Dumer, D. Micciancio, and M. Sudan, "Hardness of approximating the minimum distance of a linear code," *IEEE Transactions on Information Theory*, vol. 49, no. 1, pp. 22–37, 2003.
- [22] A. Klivans and D. van Melkebeek, "Graph nonisomorphism has sub-exponential size proofs unless the polynomial-time hierarchy collapses," *SIAM Journal on Computing*, vol. 31, no. 5, pp. 1501–1526, 2002.