Simple Extractors for All Min-Entropies and a New Pseudorandom Generator*

Ronen Shaltiel † Christopher Umans ‡

January 18, 2005

Abstract

A "randomness extractor" is an algorithm that given a sample from a distribution with sufficiently high min-entropy and a short random seed produces an output that is statistically indistinguishable from uniform. (Min-entropy is a measure of the amount of randomness in a distribution). We present a simple, self-contained extractor construction that produces good extractors for all min-entropies. Our construction is algebraic and builds on a new polynomial-based approach introduced by Ta-Shma, Zuckerman, and Safra [TSZS01]. Using our improvements, we obtain, for example, an extractor with output length $m = k/(\log n)^{O(1/\alpha)}$ and seed length $(1 + \alpha) \log n$ for an arbitrary $0 < \alpha \leq 1$, where n is the input length, and k is the min-entropy of the input distribution.

A "pseudorandom generator" is an algorithm that given a short random seed produces a long output that is computationally indistinguishable from uniform. Our technique also gives a new way to construct pseudorandom generators from functions that require large circuits. Our pseudorandom generator construction is *not* based on the Nisan-Wigderson generator [NW94], and turns worst-case hardness *directly* into pseudorandomness. The parameters of our generator match those in [IW97, STV01] and in particular are strong enough to obtain a new proof that $P = BPP$ if E requires exponential size circuits.

Our construction also gives the following improvements over previous work:

- We construct an optimal "hitting set generator" that stretches $O(\log n)$ random bits into $s^{\Omega(1)}$ pseudorandom bits when given a function on $\log n$ bits that requires circuits of size s. This yields a quantitatively optimal hardness versus randomness tradeoff for both RP and BPP and solves an open problem raised in [ISW99].
- We give the first construction of pseudorandom generators that fool *nondeterministic* circuits when given a function that requires large nondeterministic circuits. This technique also give a quantitatively optimal hardness versus randomness tradeoff for AM and the first hardness amplification result for nondeterministic circuits.

*A preliminary version of this paper appeared in the Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, 2001.

†Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. Some of this research was performed at the Institute for Advanced Study, Princeton. This research was also supported in part by the Koshland Scholarship.

‡Computer Science Department, California Institute of Technology, 1200 E. California Blvd., Pasadena, CA 91125. Some of this research was performed while the author was a postdoc at Microsoft Research. This research was supported in part by NSF grant CCF-0346991.

1 Introduction

A central question in Complexity Theory concerns the power of probabilistic algorithms. Such algorithms are allowed to use a string of independent coin tosses in their computation. Two different approaches for obtaining such a string have resulted in two fundamental objects.

Randomness Extractors (defined by [NZ96]) are deterministic machines that extract "pure randomness" from physical sources of "crude randomness". More formally, an extractor takes two inputs: An n bit long string that is sampled from an arbitrary distribution with min-entropy at least k,[1] and a "seed" of $t \ll n$ truly random bits. The extractor should output $m \gg t$ bits that are *statistically close* to truly random bits. Given an extractor E, one can run any probabilistic algorithm using only the crude source of randomness. This is done by taking a sample x from the source, and running the algorithm using $E(x, y)$ as random coins for all 2^t possible seeds y. The final output is the majority vote of the 2^t outputs. It is easy to verify that this algorithm outputs the correct answer with high probability and runs in polynomial time if $t = O(\log n)$.

Extractors have been used in a remarkable variety of settings beyond their intended application. These include complexity theory [Sip88, NZ96, GZ97], algorithms [WZ99], hardness of approximation [Zuc96, Uma99], distributed protocols [Zuc97, RZ98], and coding theory [TSZ04]. A long line of research [NZ96, SZ99, Zuc97, TS96, NTS99, Tre02, RRV02, RRV99, ISW03, RSW00, TSUZ01, LRVW03] has focused on constructing extractors with gradual improvement in various parameters. Typically, one wants to minimize the seed length t and maximize the output length m and achieve this for any relation between the length of the source n and the min-entropy threshold k. A probabilistic argument shows the existence of an "optimal extractor" that matches the lower bounds given in [NZ96, RTS00] and achieves seed length $t = \log(n - k) + O(1)$ and output length $m = k + t - O(1)$. However, all the applications above require extractors that are *explicit* in the sense that they run in polynomial time. For more information on extractors the reader is referred to survey papers [NTS99, Sha02]

Pseudorandom Generators (defined by [BM84, Yao82]) are deterministic machines that stretch a short t bit long "seed" of truly random bits into a long "pseudorandom" string of length m. It is required that no small circuit can distinguish between the distribution of pseudorandom strings and that of truly random strings. The existence of pseudorandom generators implies the existence of functions that cannot be computed by small circuits; thus, in the absence of strong circuit lower bounds, we cannot get unconditional constructions of pseudorandom generators. The "hardness versus randomness paradigm" (initiated by [BM84, Yao82]) suggests basing constructions of pseudorandom generators on the assumption that certain hard functions exist. A weaker notion of PRGs was suggested in [NW94],[2] This notion permits PRG constructions using the assumption that there exists a function family $g : \{0,1\}^{\log n} \to \{0,1\}$ that can be computed in time polynomial in n and is hard for small circuits of size $s = s(n)$ (where s lies between $\log n$ and n).[3] Given a pseudorandom generator one can determistically simulate any prob-

[1] A distribution has min-entropy at least k if it the probability it assigns to every element is at most 2^{-k}.

[2] The notion of pseudorandom generator used in [BM84, Yao82] is stronger than the one used here. It is required that PRGs fool circuits that are *larger* than the running time of the PRG. Such PRGs imply the existence of one-way functions. Following [NW94] we use a weaker notion of PRGs that only fools circuits that are *smaller* than the running time of the PRG. This is often called the "Nisan-Wigderson" setting. A typical choice of parameters in this setting is a PRG that runs in time polynomial in n, stretches $O(\log n)$ bits into $n^{\Omega(1)}$ bits, and fools all circuits of size $n^{\Omega(1)}$ where this size is smaller than the running time of the PRG. As pointed out in [NW94], such PRGs suffice to derandomize probabilistic algorithms.

[3] We remark that a different way to state this assumption is to measure the complexity of the function g in terms of the length of *its* input. In this language the assumption is that there exists a function family in $E = \mathrm{DTIME}(2^{O(n)})$ that cannot be computed by

Additional randomness t	Output length m	which k	Reference
$O(\log n)$	$k^{1-\delta}$	$k > \log^{O(1/\delta)} n$	Corollary 4.6
$(1+\delta)\log n$	$k^{\Omega(\delta)}$	$k > \log^{O(1/\delta)} n$	Corollary 4.7
$(1+\alpha)\log n$	$k/(\log^{O(1/\alpha)} n)$	any k	Corollary 4.9
$\log n + O(1)$	k	any k	optimal

Table 1: Examples of extractors constructed in this paper. Here, $\delta > 0$ is any constant, $0 < \alpha \leq 1$ is an arbitrary function. The results are stated for constant error.

abilistic algorithm by running the algorithm on all 2^t pseudorandom strings and outputting the majority vote. This deterministic simulation runs in polynomial time when $t = O(\log n)$. A long line of research [NW94, BFNW93, Imp95, IW97, STV01, ISW99, ISW03] has focused on constructing PRGs under such an assumption. An important milestone was achieved in [IW97] (and later in [STV01]). They show that if there exist sufficiently hard functions (functions g over $\log n$ bits that are computable in time $n^{O(1)}$ yet hard for circuits of size $s = n^{\Omega(1)}$) then there is a PRG that stretches $t = O(\log n)$ bits into $m = n^{\Omega(1)}$ bits and every polynomial time probabilistic algorithm can be simulated by a polynomial time deterministic algorithm; i.e., $BPP = P$. For more information on pseudorandom generators the reader is referred to survey papers [Gol98, Kab02].

Connections between extractors and PRGs The two areas were recently linked in [Tre02], which showed that Nisan-Wigderson-style PRGs, properly interpreted, are also extractors. For this correspondence, one should think of a PRG construction as receiving an additional input. In addition to the seed the PRG also gets the "hard function" $g : \{0,1\}^{\log n} \to \{0,1\}$ encoded as an n bit long truth table. Trevisan [Tre02] shows that every such construction with certain "black-box properties" yields an extractor.

1.1 Our results

The main contribution of this paper is developing a simple, self-contained, and versatile construction of both extractors and PRGs that achieves good results for a wide range of parameters. We build on a recent new technique introduced in [TSZS01] for building extractors from Reed-Muller codes. By extending this technique and adding some new ideas we are able to construct extractors over a broader parameter range (the extractors in [TSZS01] lose quite a bit of the source randomness and as a consequence only work for sources with high min-entropy). In terms of parameters, our extractors are comparable to the best current constructions, although somewhat inferior in their output length.

Our improvements also allow using Trevisan's connection [Tre02] "the other way" and give a new construction of pseudorandom generators. This construction is the first construction that is not based on the Nisan-Wigderson generator [NW94] and gives an "optimal conversion of hardness into pseudorandomness". In addition, our construction also allows constructing PRGs against nondeterministic circuits from weaker assumptions than previously known. Below, we outline our results in more detail:

circuits of size $s'(n)$ where s' is a function that satisfies $s'(\log n) = s$. We prefer the first setting as it allows a "unified framework" for both extractors and PRGs. However, we also state our results in the second setting to allow easy comparison to previous work.

Extractor constructions: Our extractors are summarized in Table 1. For simplicity we will only discuss the case when the error is constant. Precise statements of the results appear in Section 4.6. Our first extractor achieves a relatively large output length ($m = k^{1-\delta}$ for any constant $\delta > 0$), while retaining the asymptotically optimal seed length of$t = O(\log n)$. This matches the parameters of previous constructions by [Tre02] (which achieved these parameters for $k = n^{\Omega(1)}$) and [ISW03, TSUZ01]. Our second extractor uses a shorter seed length $t = (1 + \alpha)\log n$ where $0 \leq \alpha(n) \leq 1$ is an arbitrary function, and the choice of α affects the output length $m = k^{\Omega(\alpha)}$. A seed length approaching $1 \cdot \log n$ was achieved in [TSZS01] for $k = n^{\Omega(1)}$ while our result works for arbitrary k. Our third extractor improves the dependence of the output length on α and achieves output length $m = k/(\log^{O(1/\alpha)} n)$. Unlike the first two extractors it is not self-contained – it relies on another extractor construction from [TSZS01, NZ96].

In a subsequent work, [LRVW03] have constructed an extractor with seed length $t = O(\log n)$ and output length $m = \Omega(k)$ for arbitrary min-entropy threshold k. This construction (as well as some previous ones; see [Sha02] for more details) achieves better output length than our extractors. However, note that we can achieve $t = (1 + \alpha)\log n$ for small $\alpha > 0$, whereas [LRVW03] can only achieve $t = c\log n$ for some unspecified constant $c > 1$.

Pseudorandom generators constructions: Prior to this paper, all known PRG constructions were based on the original Nisan-Wigderson PRG [NW94] (with the exception of the Blum-Micali-Yao-style PRGs [BM84, Yao82], which are based on a stronger type of hardness assumption). Coming up with an alternate construction has long been an open problem. Our construction does *not* use the NW PRG, and is arguably simpler than previous constructions. In particular, there is no explicit hardness-amplification component: we transform worst-case hardness directly into pseudorandomness. The parameters of our PRG match [IW97, STV01]; that is, given a function $g : \{0,1\}^{\log n} \to \{0,1\}$ that cannot be computed by circuits of size s we construct a PRG with seed length $t = O(\log^2 n/\log s)$ and output length $m = s^{\Omega(1)}$. Consequently, (by setting $s = n^{\Omega(1)}$) we obtain a new proof of the theorem of [IW97] that $BPP = P$ if there exists a function family in E that requires exponential size circuits.

An optimal hitting set generator: A hitting set generator (HSG) is the one-sided variant of a PRG, and the canonical construct for derandomizing RP (instead of BPP). We give the first construction of an *optimal* HSG; that is, a HSG with seed length $t = O(\log n)$ that outputs $m = s^{\Omega(1)}$ bits when given a function $g : \{0,1\}^{\log n} \to \{0,1\}$ that requires size s circuits. Our construction is optimal in the sense made formal in [ISW99, ISW03]; namely, any construction that does significantly better also produces a harder function than the one initially supplied to it. Additionally, by [Tre02] any conversion of hardness into pseudorandomness with a "black-box proof" yields extractors or dispersers. Thus, any such conversion that does significantly better is ruled out by the unconditional lower bounds of [NZ96, RTS00] on extractors and dispersers. The reader is referred to [ISW03] for the precise notions of optimality we are discussing here.

An optimal hardness versus randomness tradeoff for BPP: An optimal HSG immediately gives an optimal hardness versus randomness tradeoff for RP. Using the result of [ACR98], (see also [ACRT99, BF99, GVW00]) that HSGs suffice to derandomize BPP, we use our HSG to obtain an optimal hardness versus randomness tradeoff for BPP. Specifically, we get the following tradeoff: if there exists a family of functions in E requiring circuits of size $s(n)$, then for any time constructible function $t(n)$ BPTIME$(t(n)) \subseteq$ DTIME$(2^{O(s^{-1}(t(n)^{O(1)}))})$. The previous best result (due to [ISW99, ISW03]) obtained the weaker conclusion that BPTIME$(t(n)) \subseteq$ DTIME$(2^{O(s^{-1}(t(n)^{O(\log\log t(n))}))})$. We remark that in a subsequent work, [Uma02] uses our technique to construct optimal PRGs (rather than HSGs) and this gives a more direct proof of the optimal tradeoff for BPP.

PRGs that fool nondeterministic circuits and derandomization of AM: The class AM (introduced by [BM88, GMR89]) is the nondeterministic version of BPP. The $AM \stackrel{?}{=} NP$ problem is the nondeterministic analog of the $BPP \stackrel{?}{=} P$ problem. It was observed in [AK97, GZ97] that PRGs that fool nondeterministic circuits suffice to derandomize AM. It was observed in [KvM02] that existing constructions of PRGs against deterministic circuits can be used against nondeterministic circuits if one assumes that the function g is hard against circuits of size s that use a SAT-oracle. This assumption was relaxed in [MV99]; their construction requires only hardness for nondeterministic circuits.[4] However, their construction only gives an HSG and not a PRG. (HSGs for co-nondeterministic circuits do suffice to derandomize AM as AM coincides with its one-sided error variant [FGM$^+$89]).

We use specific properties of our proof technique to show that both our PRG and HSG, with identical parameters, can be used to fool *nondeterministic circuits* when given a function that is hard for *nondeterministic* circuits. This gives an optimal hardness versus randomness tradeoff for AM and improves and extends previous works by [KvM02] (which relied on a seemingly stronger hardness assumption) and [MV99] (which does not work for low hardness).

Our technique also gives a way to transform a function family in E that is hard on the worst case for small nondeterministic circuits into a function family in E that is hard on average for small nondeterministic circuits. Such transformations are often called "hardness amplification" results. Several such results were proven in the case of *deterministic* circuits [BFNW93, IW97, STV01]. This is the first such result for nondeterministic circuits.[5]

While the constructions in [TSZS01] and the present work are simple, the proofs are more involved. The common thread in the proof techniques of [STV01, TSZS01] and this work is the use of specific properties of error-correcting codes, and ideas for decoding them. In the next section we describe the construction of [TSZS01] and our improvements at a high level.

2 Overview of the Technique

We first outline the relevant prior work on extractors in Section 2.1, then our new ideas in Section 2.2, and finally we describe the additional ideas needed to build PRGs and HSGs in Section 2.3.

2.1 Previous work

The reconstruction proof technique. Trevisan [Tre02] showed that a powerful proof technique that at first seems natural only for PRGs can in fact be used for extractors. The proof technique works by contradiction. One assumes that the extractor's output is not close to uniform for some high min-entropy source X and therefore does not pass some prediction test. That is, there exists an index i and a function f (the

[4] Note that nondeterministic circuits are the nonuniform analog of NP whereas SAT-oracle circuits are the nonuniform analogue of P^{NP}. Furthermore, SAT-oracle circuits are stronger than nondeterministic circuits assuming the polynomial time hierarchy does not collapse. We also remark that the result of [MV99] (as well as ours) can be stated with respect to hardness for "single valued nondeterministic circuits" (see Section 6).

[5] In a subsequent work [SU04] we extend the technique developed in this paper to show that if there are functions in E that require large nondeterministic circuits then there are functions in E that require large circuits that make non-adaptive SAT-queries. This gives a more modular way to prove our results for nondeterministic circuits.

predictor) that is able to predict the i'th output of the extractor given the first $i-1$ outputs. More formally, if we denote the m output symbols of the extractor by $E(x,y)_1, \cdots, E(x,y)_m$ then

$$\Pr_{x \leftarrow X, y}[f(E(x,y)_1, E(x,y)_2, \cdots, E(x,y)_{i-1}) = E(x,y)_i] \tag{1}$$

is noticeably larger than randomly guessing the i'th symbol. (In this discussion, we consider a generalization of extractors that we call "q-ary extractors," which output m symbols from an alphabet of size q rather than m bits; such extractors can be converted to extractors that output bits [TSZS01] – cf., Section 4.5). Then one gives a "reconstruction procedure" based on the predictor f. This procedure is able to reconstruct the string x sampled from the weak random source using f and a short "advice string". More precisely, for many x's (namely those on which the predictor has noticeable advantage) there exists a short "advice string" z such that the reconstruction procedure can reconstruct x using f when given z. If the source has large min-entropy, this is a contradiction because it implies that a large number of strings have short description (whereas there are only a few short descriptions).

The extractor of [TSZS01] Ta-Shma, Zuckerman and Safra [TSZS01] proposed a new extractor construction based on polynomials that uses this proof technique in a new way. Their construction is very simple. One thinks of the string x sampled from the weak random source as a low-degree multivariate polynomial $p_x : F^d \to F$ over a finite field F of size q, and the seed is a random evaluation point $y \in F^d$. The extractor computes m "successive" points $y_1, \cdots, y_m \in F^d$ starting with $y_1 = y$ (the meaning of "successive" is purposely left a bit vague in this presentation and we will elaborate on it later) and outputs m symbols $E(x,y) = (p_x(y_1), \cdots, p_x(y_m))$.

Their proof describes a reconstruction procedure that attempts to reconstruct a source string (viewed as a polynomial p_x) using a predictor f and a short advice string z which contains the value of p_x at a small fixed set of points which we will call the "startup points". The basic idea is to reconstruct p_x step by step, where in each step the reconstruction procedure "learns" the evaluation of p_x at a new point. In the first step the procedure uses $i-1$ successive points from the startup points to predict the value of p_x at the "next" point (using f). In the next step the predictor can be "advanced" by one step and use $i-2$ successive points plus the one just predicted to predict the value of p_x at the next point, and so on, until all of p_x is reconstructed. There are however two complications.

First, the predictor is only correct with a small advantage over random guessing. To overcome this, at each step the predictor is used to predict in parallel all points along a random line L in F^d. (Loosely speaking, this can be done by taking more startup points; more details are below). Because points on a random line are pairwise independent, with high probability the predictor is correct on the same fraction of points in L as its total advantage over the whole space. The collection of values along the line can now be error-corrected since p_x restricted to L is a low-degree univariate polynomial.

Second, the relative number of errors is so large that unique decoding is impossible, and one must use "list-decoding" [Sud97] to obtain a small number of possible options, (that is univariate polynomials), one of which is correct in the sense that it agrees with p_x on the line L. To pin down the correct one, the advice string includes the evaluation of p_x at a random point on L, which with high probability agrees with *only* the correct polynomial in the list.

To summarize, the [TSZS01] reconstruction procedure gets an advice string that enables it to evaluate p_x on $i-1$ successive lines. and the process described above is used to reconstruct p_x by "learning" a new line in each step.

2.2 Improvements of this paper

The key to improving the quality of the [TSZS01] extractor is reducing the length of the advice string. As the length of the advice string depends (amongst other things) on the degree of p_x which we will denote by h, we want to reduce the degree. Recall that $p_x : F^d \to F$ is supposed to encode an n bit long source element x and to encode this amount of information we need $h^d \approx n$. Therefore, when reducing the degree h we must increase the dimension d. It turns out that the straightforward way of increasing the dimension in the construction of [TSZS01] *increases* the length of the advice string (so the best result is obtained when p_x is a polynomial in only two variables, i.e. $d = 2$). Our improvements enable us to increase the dimension d without this deleterious effect.

An algebraic approach. What does "successive" mean? In [TSZS01] (say in two dimensions over finite field F), the successor of (a_1, a_2) is $(a_1 + 1, a_2)$. The rationale is that starting from a random line and taking successive steps, one covers the whole vector-space F^2. The advice string must include the values of p_x on roughly m (the output length) lines, because i in Equation (1) may be as large as $m - 1$. However, this geometric approach succumbs to the "curse of dimensionality" as the dimension is increased: for dimension d, the advice string must include the values of p_x on roughly m $(d - 1)$-dimensional subspaces, and its length becomes huge (namely, it is greater than $mq^{d-1} > mh^{d-1}$, which approaches $n \approx h^d$, the length of the source string).[6]

The main source of our improvement comes from taking an algebraic instead of a geometric approach. The polynomials we wish to learn are defined over the vector-space F^d. Our insight is to view this space as an extension field of F. The multiplicative group of this field is cyclic and has generator g, and for us, the successor of an element is obtained by *multiplication* by g. This indeed has the essential property that by repeatedly taking successors, we cover the whole space. We also use critically that multiplication by g corresponds to a *linear transform* in the vector-space, so that lines get mapped to lines. Replacing the geometric approach by an algebraic approach avoids the geometric structure of F^d and now the dimension does not come into play. Our advice string includes roughly m lines regardless of the dimension, and thus is of length $\approx mh$. To make use of this improvement we also need the following new ideas.

Curves instead of lines. In [TSZS01], each prediction step fails with probability about $1/q$ where q is the size of the field F, and a union bound is used to argue that *no* prediction steps fails during the entire reconstruction process.

Recall that we are increasing the dimension d and decreasing the degree h (which in turn forces us to decrease the field size q in order to have a short seed length). This decreases the number of points on a line and means that it takes many more prediction steps to traverse the whole space and therefore many more events are in the union bound. Decreasing the field size q increases the failure probability of each individual event. Together these effects overwhelm us.

We overcome this by predicting along *degree r curves* instead of lines. Since the collection of points on such curves is r-wise independent, we can use higher moment tail inequalities to argue that the failure probability of each prediction step is *exponentially small* in r. Choosing r large enough permits us to use the union bound even for our much larger collection of events. Using curves instead of lines is also necessary for the improvement described next.

[6]Furthermore, we remark that when successive points are on a line, (as is the case in [TSZS01]), then the m output evaluations are evaluations of some univariate polynomial q with the same degree as p_x. Thus, one has to set m smaller than the degree of the polynomial p_x, as otherwise the predictor f can easily predict the last point by interpolating the polynomial q when given all points but the last one. Our improvement allow us to break this barrier and set the degree of p_x to be much smaller than m.

Interleaved reconstruction procedures. There's an additional cost to using more prediction steps. Recall that in [TSZS01], the advice string must include the value of the polynomial at a random point on the line, *for each prediction step*. Having to include these will blow up the size of the advice string.

To overcome this problem we run two "interleaved" reconstruction procedures. Each uses its own random curve and startup points but we arrange it so that the two curves intersect at a few random points. The two reconstruction procedures work on their own. However, when one needs the value of the polynomial at a random point on its curve, it can use the value *already calculated* by the other reconstruction procedure instead of relying on the advice string. Thus, no additional information is required in the advice string beyond the startup points needed to get the two interleaved reconstruction procedures started.

To conclude, the improvements above allow us to decrease the degree h from about $\sqrt{n}$ to poly $\log n$ and the advice string contains only the startup points and has length roughly mh. This gives extractors that work for every min-entropy threshold k and extract roughly $k/h = k/\text{poly} \log n$ bits from the source.

We stress that almost all of what we have described in the preceding subsections relates to the *proof* that our construction is indeed an extractor. The extractor construction itself remains very simple.

2.3 Constructing PRGs and HSGs

Since the reconstruction proof technique outlined above was originally applied to PRG constructions, it is easy to adapt to that setting. To convert our extractor into a PRG, we fix the "source string" x to be the truth table of a hard function g, use the seed y as before to pick an evaluation point, and output m successive evaluations. If this is *not* a PRG, then there is an *efficient* predictor f, and we wish to use f to produce a *small* circuit C that computes the function $x(j) = x_j$, contradicting the hardness of x.

Two things contribute to the size of C: the length of the advice string (which must be hardwired into C), and the number of prediction steps (since each step invokes f and requires computation to perform the list-decoding). Because the advice string must be small, the improvements we get over [TSZS01] in this area are essential for the PRG construction.

However, the natural adaptation of our extractor into a PRG suffers from an inherent problem of the method of [TSZS01]: It is highly sequential. Specifically, computing x_j at a position j that is "far away" from the startup points (one that takes many successive prediction steps to get to) takes too many steps and makes C too large to derive a contradiction.

The problem is that we can't have a "successor function" defined over F^d in such a way that very few applications of the function can get to *every* point from a fixed starting location. A helpful idea is to allow *several* such successor functions so that short sequences of applications of the different successor functions can reach every point. To achieve this we will take the first successor function to be the one we used for the extractor, the second successor function to be q *applications* of the first, the third to be q^2 applications of the first and so on, where q is the field size. By first taking a few small strides, then a few larger strides, then a few even larger strides, etc., we can reach every point in a small number of steps.

Each one of these successor functions corresponds to a construction very similar to our extractor construction. We show that *at least one* of these constructions must be a PRG, since if none of them are, then we have predictors for all of them and this would give us the predictors with differing strides needed to contradict the hardness of x.

By running all of these "candidate" PRGs with independent seeds, and XOR-ing their output, we obtain the desired single PRG. Each one of the candidate PRGs uses a seed of length $O(\log n)$. Because we need so few candidate generators the seed length of the XORed generator is still relatively short. More precisely, the number of candidate generators is $\log n / \log s$ when x (viewed as a function $x : \{0,1\}^{\log n} \rightarrow \{0,1\}$)

requires circuits of size s. (Note that this is *constant* when x has exponential hardness.) Thus, the seed of the XORed generator is of length $O(\log^2 n/\log s)$ and matches the parameters of [STV01]. We can reduce the seed length to the optimal $O(\log n)$ when constructing a hitting-set generator (HSG). This is done by taking the *union* of the candidate PRGs rather than their exclusive-or. More precisely, the seed of our HSG has two parts: y and i, and when given y and u it runs the i'th candidate on seed y. This corresponds to choosing a random candidate PRG and running it on a random seed.

Generators for nondeterministic circuits. In the nondeterministic setting we construct PRGs and HSGs for nondeterministic circuits based on functions that are hard for nondeterministic circuits. We use the same proof technique as for "ordinary" PRGs; however, to derive a contradiction in the new setting, we must use a *nondeterministic* predictor circuit f to construct a *nondeterministic* circuit that computes the (supposedly hard) function x. The deterministic reconstruction procedure runs the nondeterministic predictor f as a subroutine. The difficulty is that a deterministic procedure with a nondeterministic subroutine does not necessarily yield a nondeterministic procedure. (For example P^{NP} is not likely to be contained in NP). To obtain a nondeterministic circuit the reconstruction procedure can only use the nondeterministic predictor f in a one-sided way: It can efficiently verify that the predictor outputs "one" on a given input, but there's not necessarily an efficient way to verify that the output is "zero". Klivans and van Melkebeek [KvM02] bypassed this problem by imposing a stronger hardness assumption on x – hardness for circuits with SAT-oracle gates. Under this assumption constructions based on [NW94] can be used to fool circuits with SAT-oracle gates, as the proof of Nisan and Wigderson relativizes. Our "ordinary" PRG and HSG constructions also relativize and therefore immediately translate to this framework. Miltersen and Vinodchandran [MV99] gave an alternate construction using a hardness assumption for *single-valued nondeterministic circuits* (SV-circuits), which are (presumably) weaker than both circuits with SAT-oracle gates and nondeterministic circuits.[7] However, their construction gives only HSGs (not PRGs), and works only for the "high end" of possible hardness assumptions, meaning that it requires circuit lower bounds of at least $2^{\sqrt{\log n}}$ for a function g on $\log n$ bits.

Both our PRG and our HSG constructions can be adapted to fool nondeterministic (and co-nondeterministic) circuits using hardness assumptions for SV-circuits. The main observation is that we always use our predictors along random curves. When used this way the fraction of "ones" we expect to find in the output of the predictor along a random curve is close to the fraction in the whole space, and the probability of deviation is very small. The reconstruction receives the fraction of "ones" the predictor outputs over the whole space as non-uniform advice. Then, at every step, we *guess* this fraction of locations along our random curve, and assume the predictor outputs "ones" at these locations and "zeros" elsewhere. We can efficiently verify that the predictor indeed outputs "ones" at the specified locations, and then the only new errors we have introduced are the small number of "ones" we have assumed to be "zeros." However, the number of errors is still small enough to allow the (list-)decoding phase to proceed unchanged. The same technique also gives a hardness amplification result for nondeterministic circuits.

It should be noted that the presentation in this section is over-simplified and the reader is referred to the technical sections for exact details. The actual PRG and HSG constructions also involve a non-standard version of the "low degree extension" encoding that is suitable for our application.

[7] Loosely speaking, nondeterministic circuits are a nonuniform analog of NP, and SV-circuits are a nonuniform analog of $NP \cap coNP$. A precise definition appears as Definition 6.2.

2.4 Outline

The remainder of the paper is organized as follows. In Section 3 we state some previous results. Section 4 is devoted to extractors and contains the core of all of the other constructions that follow. Section 5 contains the PRG and HSG constructions, and Section 6 extends these to the nondeterministic setting. In Section 7 we briefly describe a further application of our ideas to hardness amplification. An important ingredient in our constructions are constructions of "traversing matrices" and in Section 8 we show how to construct such matrices.

3 Preliminaries

We begin with some standard definitions. Whenever we use U_n, we mean the random variable that is uniform on $\{0,1\}^n$. Two distributions on the same domain are *ϵ-close* if the statistical distance between them is at most ϵ; i.e., the probabilities they give to any event differ by at most ϵ. Give a field F, a function $C : F \to F^d$ is a degree r curve if $C(t) = \sum_{i=0}^{r-1} a_i t^i$ for some $a_0, \cdots, a_{r-1} \in F^d$. By $F^{[p]}$ we denote the set of all subsets of F of size p.

We need the following list-decoding bound due to [Sud97]:

Lemma 3.1 ([Sud97]). *Let* $\mathsf{prs}, \mathsf{agr}, \mathsf{deg}$ *be integers. Given* prs *distinct pairs* (x_i, y_i) *in field* F *with* $\mathsf{agr} > \sqrt{2 \cdot \mathsf{deg} \cdot \mathsf{prs}}$*, there are at most* $2\mathsf{prs}/\mathsf{agr}$ *polynomials* g *of degree* deg *such that* $g(x_i) = y_i$ *for at least* agr *pairs. Furthermore, a list of all such polynomials can be computed in time poly*$(\mathsf{prs}, \log|F|)$.

We also need the following tail inequality for t-wise independent random variables:

Lemma 3.2 ([BR94]). *Let* $t > 4$ *be an even integer. Suppose* $X_1, X_2, \ldots, X_n$ *are* t*-wise independent random variables taking values in* $[0,1]$*. Let* $X = \sum X_i$*,* $\mu = E[X]$ *and* $A > 0$*. Then:*

$$\Pr[|X - \mu| \geq A] \leq 8 \cdot \left(\frac{t\mu + t^2}{A^2}\right)^{t/2}$$

4 Extractors

In this section we give our extractor constructions. In Section 4.1 we give definitions of extractors and an generalization of extractors which we call q-ary extractors. In Section 4.2 we present our basic construction. In Sections 4.3 and 4.4 we prove our main theorem: that our construction yields a q-ary extractor. In Section 4.5 we discuss two transformations that convert q-ary extractors into (regular) extractors. This allows us to obtain regular extractors. In Section 4.6 we give a modified construction of q-ary extractors that allows the construction of extractors with small error.

4.1 Extractor preliminaries

A random variable X has min-entropy at least k if $\Pr[X = x] \leq 2^{-k}$ for all x; formally:

Definition 4.1 (min-entropy). *The* min-entropy *of a random variable* X *over* $\{0,1\}^n$*, written* $H_\infty(X)$*, is defined as* $H_\infty(X) = \min_{x \in \{0,1\}^n} \log_2 \frac{1}{\Pr[X=x]}$.

Definition 4.2 (extractor). *A* (k, ϵ) extractor *is a function* $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ *such that for all random variables* X *with* $H_\infty(X) \geq k$*:*

$$E(X, U_t) \text{ is } \epsilon\text{-close to } U_m. \tag{2}$$

An extractor E is *explicit* if E can be computed in time polynomial in n. By [Yao82], to show that (2) above holds, it is sufficient to prove (3):

$$\begin{aligned} &\forall \;\; 1 \leq i \leq m \text{ and all functions } f : \{0,1\}^{i-1} \to \{0,1\}\,, \\ &\qquad \Pr[f(E(X,U_t)_{1\ldots i-1}) = E(X,U_t)_i] \leq \frac{1}{2} + \frac{\epsilon}{m} \end{aligned} \tag{3}$$

and indeed the proofs for many recent extractor constructions follow this route. Property (3) requires that each successive bit of output be "unpredictable" based on the previous bits.

As our construction of extractors is algebraic, we will be working over the finite field with q elements. It will therefore be useful to define a "q-ary" extractor. Such an extractor is required to satisfy an unpredictability property analogous to (3); however, in the larger field we allow the prediction function f to output a small list of possible next elements, instead of just one.[8]

Definition 4.3 (q-ary extractor). *Let* F *be the field with q elements. A* (k, ρ) q-ary extractor *is a function* $E : \{0,1\}^n \times \{0,1\}^t \to F^m$ *such that for all random variables* X *with* $H_\infty(X) \geq k$*:*

$$\begin{aligned} &\forall \;\; 1 \leq i \leq m \textit{ and all functions } f : F^{i-1} \to F^{[\rho^{-2}]}, \\ &\qquad \Pr[E(X,U_t)_i \in f(E(X,U_t)_{1\ldots i-1})] \leq \rho. \end{aligned} \tag{4}$$

In [RRV02, TSZS01] it was shown how to transform q-ary extractors into regular extractors. This transformation does not significantly change the parameters of the q-ary extractor. The precise details of this transformation are given in Section 4.5. This allows us to focus on building good q-ary extractors.

4.2 The basic construction

Our construction is very simple. Following [TSZS01] our first step is to encode the string x from the weak random source with a q-ary Reed-Muller (d-variate polynomial) code. The coordinates of such a code are in one-to-one correspondence with the vectors in the vector-space F^d. Our q-ary extractor uses its truly random bits to pick a random $\vec{v} \in F^d$. It outputs the $\vec{v}$-th symbol of the encoded string and $m-1$ successive symbols. The successor of $\vec{v}$ is $A\vec{v}$, where A is a special $d \times d$ matrix, on which we elaborate below.

More formally, let F be the field with q elements, and fix the dimension d. Let h be an integer such that

$$\binom{h+d-1}{d} \geq \frac{n}{\log q}. \tag{5}$$

For $x \in \{0,1\}^n$, let $\hat{x}$ denote the d-variate polynomial of total degree at most $h-1$ whose coefficients are specified by x. Such a polynomial has $\binom{h+d-1}{d}$ coefficients, so (5) implies that distinct x give rise to distinct $\hat{x}$. Next, we require a matrix A that "generates" $F^d \setminus \{0\}$. That is, for every non-zero vector $\vec{v} \in F^d$,

$$\left\{A^i \vec{v}\right\}_{1 \leq i < q^d} = F^d \setminus \{0\}\,. \tag{6}$$

[8]Allowing predictors to output a list is not really necessary. We use this definition as it enables us to get a slightly shorter seed length for our final extractor.

It is easy to show that such a matrix exists and can be found efficiently. Loosely speaking, the matrix A corresponds to multiplying by a generator of the multiplicative group of $GF(q^d)$.

Lemma 4.4. *Let F be a field with q elements and let d be an integer. There exists an invertible $d \times d$ matrix A with entries in F such that A^{q^d-1} is the identity matrix and for every non-zero $\vec{v} \in F^d$, $\left\{A^i \vec{v}\right\}_{1 \le i < q^d} = F^d \setminus \{0\}$. Furthermore, such an A can be found in time $q^{O(d)}$.*

The proof of Lemma 4.4 appears in Section 8. We can now define our q-ary extractor $E : \{0,1\}^n \times \{0,1\}^{d \log q} \to F^m$. We interpret the second input of the q-ary extractor as a vector $\vec{v} \in F^d$.

$$E(x, \vec{v}) = \hat{x}(A^1 \vec{v}) \circ \hat{x}(A^2 \vec{v}) \circ \cdots \circ \hat{x}(A^m \vec{v}). \tag{7}$$

Our main theorem is the following:

Theorem 4.5 (extractor main). *There exists a universal constant c such that for every n, d, h and prime power q satisfying $\binom{h+d-1}{d} \ge \frac{n}{\log q}$, E is a (k, ρ) q-ary extractor, provided that*

1. $k > cmhd \log q + \log\left(\frac{1}{\rho}\right)$ *and* $q > c(\frac{(hd)^2}{\rho^4})$, *or*

2. $k > cmhd \log^2 q + \log\left(\frac{1}{\rho}\right)$ *and* $q > c\left(\frac{hd \log q}{\rho^4}\right)$.

Moreover, E can be computed in time poly(q^d).

Part (1) of the theorem is used when we are maximizing the output length; part (2) is used when we are minimizing the seed length. The proof of Theorem 4.5 is given in the next two sections. In Section 4.5 we explain how to convert E into a binary extractor and choose the parameters in order to prove the following corollaries:

Corollary 4.6. *For all n, constants $\epsilon, \delta > 0$, and $k \ge \log^{4/\delta} n$ our construction gives an explicit (k, ϵ)-extractor with seed length $t = O(\log n)$ and output length $m = k^{1-\delta}$.*

Corollary 4.7. *For all n, constants $\epsilon > 0$ and $1/8 > \delta > 0$, and $k \ge \log^{4/\delta} n$, our construction gives an explicit (k, ϵ)-extractor with seed length $t = (1 + O(\delta)) \log n$ and output length $m = k^{\delta}$.*

Corollary 4.8. *For all n, k, constants $\epsilon, \delta > 0$ our construction gives an explicit (k, ϵ)-extractor with seed length $t = O(\log n)$ and output length $m = k/(\log n)^{2+\delta}$.*

Corollary 4.9. *For all n, k, constant $\epsilon > 0$ and any $0 < \delta < 1$ (not necessarily a constant) our construction gives an explicit (k, ϵ)-extractor with seed length $t = (1 + \delta) \log n$ and output length $m = k/(\log n)^{O(1/\delta)}$.*

All the Corollaries above are stated for constant $\epsilon > 0$. We address the case of non-constant ϵ in Section 4.6.

4.3 The reconstruction proof paradigm

To prove that E is a q-ary extractor, we use ideas that originated in [Tre02] and are refined in [TSZS01], which one might label the "reconstruction proof paradigm". An important aspect of this paradigm is that to show that a given function E is an extractor it is sufficient to analyze the behavior of E on *fixed* x's.

Definition 4.10 (good strings). *Let $E : \{0,1\}^n \times \{0,1\}^t \to F^m$ be some function. Let $1 \le i^* \le m$ be some index and $f : F^{i^*-1} \to F^{[\rho^{-2}]}$ be some function. A string $x \in \{0,1\}^n$ is ρ-*good *for f with respect to E if*

$$\Pr[E(x, U_t)_{i^*} \in f(E(x, U_t)_{1 \dots i^*-1})] \ge \rho/2.$$

A reconstruction procedure is a randomized procedure that when given oracle access to a predictor f is able to reconstruct x's that are good for f.

Definition 4.11 (reconstruction procedure). *Given a function $E : \{0,1\}^n \times \{0,1\}^t \to F^m$, an (a, ρ)-*reconstruction *for E is a randomized procedure R such that for any $1 \le i^* \le m$, function $f : F^{i^*-1} \to F^{[\rho^{-2}]}$ and $x \in \{0,1\}^n$ such that x is ρ-good for f with respect to E,*

$$\Pr[\exists z \in \{0,1\}^a ,\ R^f(z) = x] \ge 1/2].$$

Note that we place no restrictions on the running time of R and so the particular mode in which it accesses f is not important. We chose the computational flavor of the definition above in order to compare to the computational setup of constructing PRGs. The next Lemma shows that any function E that has a reconstruction procedure is a q-ary extractor.

Lemma 4.12. *If E has an (a, ρ)-reconstruction then E is a (k, ρ) q-ary extractor with $k = a + \log(1/\rho) + 2$.*

Proof. Let X be some distribution over $\{0,1\}^n$ with $H_\infty(X) \ge k$. Let $1 \le i^* \le m$ and fix some "predictor" $f : F^{i^*-1} \to F^{[\rho^{-2}]}$. According to Definition 4.3 we need to show that

$$\Pr[E(X, U_t)_{i^*} \in f(E(X, U_t)_{1 \dots i^*-1})] \le \rho.$$

We define:

$$p = \Pr_{x \leftarrow X}[x \text{ is } \rho\text{-good for } f \text{ with respect to } E].$$

Note that the success probability of f is bounded by $p + \rho/2$, and thus it is sufficient to show that $p \le \rho/2$. We have that for every x that is ρ-good for f, with probability at least $1/2$ there exists a string $z \in \{0,1\}^a$ such that $R^f(z) = x$. It follows that

$$\Pr_{x \leftarrow X}[\exists z \in \{0,1\}^a ,\ R^f(z) = x] \ge p/2.$$

Where the probability above is over the choice of x and the coin tosses of R. There exists a fixing of the coin tosses of R such that the inequality above holds when the probability is only over the choice of x. After this fixing, R has at most 2^a outputs. For every such output w, the probability that $x = w$ is at most 2^{-k}. Thus,

$$\Pr_{x \leftarrow X}[\exists z \in \{0,1\}^a ,\ R^f(z) = x] \le 2^{a-k}.$$

We conclude that $p \le 2^{a-k+1}$ and by our choice of k, $2^{a-k+1} \le \rho/2$. □

Our main task is thus to construct a reconstruction procedure R with a short advice string. To obtain R, we use essentially the framework of [TSZS01] in which the reconstruction runs the given predictor f in many "prediction steps" and performs error correction after each such step. In the next section we describe our reconstruction procedure.

4.4 Proof of the main extractor theorem

In this section we prove Theorem 4.5. The following lemma describes the reconstruction procedure.

Lemma 4.13. *Let n, q, d, h and ρ be as in the statement of Theorem 4.5 (1) (resp. Theorem 4.5 (2)). There is an (a, ρ)-reconstruction for E with $a = O(mhd \log q)$ (resp. $a = O(mhd \log^2 q)$).*

Proof. Fix a function $f : F^{i^*-1} \to F^{\lceil \rho^{-2} \rceil}$ and an x that is ρ-good for f with respect to E. Our goal is to reconstruct x from a short advice string. The predictor function f can "attempt to predict" the evaluation of $\hat{x}$ at u, when given the evaluation of $\hat{x}$ at the points

$$A^{-(i^*-1)}u, A^{-(i^*-2)}u, \cdots, A^{-1}u.$$

This is because $E(x, A^{-i^*}u)_{1 \cdots i^*-1} = (\hat{x}(A^{-(i^*-1)}u), \cdots, \hat{x}(A^{-1}u))$. Thus, if $y = A^{-i^*}u$ is one of the seeds for which f correctly predicts $E(x, y)_{i^*}$ given $E(x, y)_{1 \cdots i^*-1}$ then f computes $\hat{x}$ at u given the previous evaluations.

Recall that A is invertible, and thus $u = A^{i^*}y$ is uniformly distributed when y is uniformly distributed. Consequently, this prediction succeeds on a $\rho/2$-fraction of the points u. The crux of the proof is a randomized choice of low-degree curves with special intersection properties. These curves allow us to error-correct the answer of the predictor gives on points on the curves. The overall argument uses a short advice string z that contains the evaluation of $\hat{x}$ on m successive curves. We then conduct a sequence of "prediction steps" where in each one we learn the evaluation of $\hat{x}$ in at least one new point. In the end we learn the evaluation of $\hat{x}$ on all input points and can interpolate to recover x. We now define the curves.

- Let c' be some constant to be chosen later. Set $r = c'd$ (resp. $r = c'd \log q$)
- Pick $2r$ random points $\vec{y_1}, \vec{y_2}, \ldots \vec{y_{2r}}$ from F^d, and $2r$ random and distinct values $t_1, t_2, \ldots t_{2r}$ from F.
- Let $p_1 : F \to F^d$ be the degree $2r - 1$ polynomial such that $p_1(t_i) = \vec{y_i}$, for $i = 1, 2, \ldots, 2r$.
- Let $p_2 : F \to F^d$ be the degree $2r - 1$ polynomial such that $p_2(t_i) = A\vec{y_i}$ for $i = 1 \ldots r$ and $p_2(t_i) = \vec{y_i}$ for $i = r + 1 \ldots 2r$.

Given a function $p : F \to F^d$ and a $d \times d$ matrix A over F, we use Ap to denote the function $p' : F \to F^d$ defined by $p'(w) = Ap(w)$. It is important to observe that if p is a degree $2r - 1$ curve then $p' = Ap$ is also a degree $2r - 1$ curve. For $1 \le i \le 2q^d$ we define a random variable P_i (over the choice of the t_j's and the $\vec{y_j}$'s). Each such variable is a function $P_i : F \to F^d$. For odd $i = 2j + 1$ we define $P_{2j+1} = A^j p_1$ and for even $i = 2j + 2$ we define $P_{2j+2} = A^j p_2$. Note that each such variable is a degree $2r - 1$ curve. We also view them as multi-sets $\{P_i(w) | w \in F\} \subseteq F^d$ and thus we write $u \in P_i$ to mean that there exists a $w \in F$ such that $P_i(w) = u$. Since A "generates" $F^d \setminus \left\{\vec{0}\right\}$ we have that $\cup_{1 \le i \le 2q^d} P_i = F^d \setminus \left\{\vec{0}\right\}$. Using the fact that A is a non-singular linear transform we have for all i:

- $P_i : F \to F^d$ is a degree $2r - 1$ polynomial.
- The polynomial $\hat{x} \circ P_i$ is a univariate polynomial of degree at most $(2r - 1)(h - 1)$.
- The sequence of points $\{P_i(w)\}_{w \in F}$ is $2r$-wise independent.

- P_{i-1} and P_i intersect at r random distinct positions. More precisely, there exist distinct positions $S = (w_1, \cdots, w_r) \in F$ such that P_{i-1} and P_i agree in all the positions in S. Furthermore, the random variable S is uniformly distributed over all distinct r-tuples in F, and S is independent of P_i. This is because seeing only one of the two curves p_1, p_2 gives no information on the values of $t_1, \cdots, t_{2r}$ used to construct them.

We will set up the reconstruction function by supplying it with an advice string that will allow it to compute the evaluation of $\hat{x}$ on all the points in $\left\{P_1, \cdots, P_{2(i^*-1)}\right\}$. This is done by giving as advice the $(2r-1)(h-1)+1 \leq 2hr$ coefficients of $\hat{x} \circ P_i$ for $1 \leq i \leq 2(i^*-1)$. Thus, the length of the advice string is at most $4mhr \log q \leq a$ as stated. From these evaluations we can use the predictor f to attempt to predict the evaluations of $\hat{x}$ at the points P_i for $i > 2(i^*-1)$. In general, for $i > 2(i^*-1)$ we assume that we already computed the evaluations of $\hat{x}$ at points:

$$P_{i-2(i^*-1)}, \cdots, P_{i-1}$$

we will now show how to use these evaluations to compute the evaluations of $\hat{x}$ at P_i.

We first invoke f once for attempting to predict every point $u \in P_i$. That is we run f on the evaluations of $\hat{x}$ at $\left\{A^{-1}u, \cdots, A^{-(i^*-1)}u\right\}$ to obtain a list of at most ρ^{-2} "candidates" for $\hat{x}(u)$. (Note that $A^{-j}u \in A^{-j}P_i = P_{i-2j}$). We will show that with high probability these predicted values and the evaluations of $\hat{x}$ on P_{i-1} completely determine the values of $\hat{x}$ on P_i. The first step is to show that with high probability, many of the points u in P_i are predicted correctly.

Claim 4.14. *With probability at least $1 - 1/8q^d$ over the coin tosses of R:*

$$\Pr_{\vec{u} \in P_i}[\hat{x}(\vec{u}) \in f(\hat{x}(A^{-i^*+1}\vec{u}), \ldots, \hat{x}(A^{-1}\vec{u}))] \geq \rho/4.$$

Proof. (of Claim 4.14) Let Y_ℓ be the indicator random variable for the event that the set of predicted values for ℓ-th point in P_i contains the evaluation of $\hat{x}$ at that point, and let $Y = \sum_{\ell=1}^{q} Y_\ell$. Since x is ρ-good for f, we have that for every ℓ, $E[Y_\ell] > \rho/2$. By linearity of expectations $\mu = E[Y] > (\rho/2)q$. Since P_i is a $2r$-wise independent set of points, we can apply Lemma 3.2, and get that the probability that $Y \leq (\rho/4)q$ is at most:

$$\Pr[|Y - E[Y]| \geq E[Y]/2] \leq \left(\frac{O(r)}{\rho q}\right)^r < \frac{1}{8q^d},$$

where the final inequality holds using the condition on q in Theorem 4.5 and choosing large enough c' when setting r as $c'd$ (for part 1) and as $c'd \log q$ (for part 2). □

Therefore, with high probability, we have $\mathsf{agr} = (\rho/4)q$ "good" evaluations that agree with the degree $\mathsf{deg} \leq 2rh$ univariate polynomial $\hat{x}_{|P_i}$, out of a total of $\mathsf{prs} = \rho^{-2}q$ pairs. To apply Lemma 3.1 we need to verify that $\mathsf{agr} > \sqrt{2 \cdot \mathsf{deg} \cdot \mathsf{prs}}$ or equivalently that $q > \frac{c''rh}{\rho^4}$ for some constant c''. By our choice of q and r we can meet this requirement by choosing a large enough constant c'. Using Lemma 3.1 we conclude that at most $8\rho^{-3}$ degree $2rh$ polynomials agree with our evaluations on this number of points. We point out that these polynomials depend only on P_i. This is because the set of pairs supplied to Lemma 3.1 depend only on P_i (as well as f and $\hat{x}$, which are both fixed).

Now, P_{i-1} intersects P_i at r random distinct positions S, and the choice of S is independent of P_i. Note that we already know the evaluation of $\hat{x}$ at the points in P_{i-1}. Two different degree $2rh$ polynomials can agree on at most $2rh/q$ fraction of their points, so the probability that an "incorrect" polynomial from among our candidates agrees with $\hat{x} \circ P_i$ on all r random points is at most:

$$(8\rho^{-3})\left(\frac{2rh}{q}\right)^r < \frac{1}{8q^d}$$

where the inequality holds by our choice of r and q for large enough c'.

So, with probability at least $1 - \frac{1}{4q^d}$ over the random coins of R, we learn the evaluations of $\hat{x}$ on the points in P_i successfully.

After $2q^d$ such prediction steps, we have learned $\hat{x}$ on $F^d \setminus \left\{\vec{0}\right\}$. By the union bound, the probability that all steps of this reconstruction are successful is at least $1/2$. These evaluations uniquely determine $\hat{x}$, and the reconstruction function R then outputs x (which can be easily computed from $\hat{x}$). $\square$

Theorem 4.5 now follows using Lemma 4.12.

4.5 From q-ary extractors to regular extractors

We are now left with the task of converting a q-ary extractor into a regular one (Definition 4.2). The standard way to achieve this is to use "list-decodable" error correcting codes. The transformation described in Lemma 4.16 below is essentially the information-theoretic analog of the hard-core bit constructions of Goldreich-Levin [GL89]. In the following definition Δ is the Hamming distance function.

Definition 4.15. *A binary code $C : \{0,1\}^{\bar{k}} \to \{0,1\}^{\bar{n}}$ is (ρ, ℓ)-*list-decodable*, if for all $r \in \{0,1\}^{\bar{n}}$, the set $S_r = \{x : \Delta(C(x), r) \leq (1/2 - \rho)\bar{n}\}$ has size at most ℓ. The code is* efficiently encodable *if C is computable in time poly$(\bar{n})$, and* efficiently list-decodable *if S_r can be computed from r in time poly$(\bar{n}, \ell)$.*

By the Johnson bound (see, e.g., [GS01]), any binary code with relative distance at least $1/2 - \rho^2$ is (ρ, ρ^{-2})-list-decodable.

Lemma 4.16 ([TSZS01]). *Let F be the field with q elements and let $C : \{0,1\}^{\bar{k} = \log q} \to \{0,1\}^{\bar{n}}$ be a (ρ, ρ^{-2})-list-decodable code. If $E : \{0,1\}^n \times \{0,1\}^t \to F^m$ is a (k, ρ) q-ary extractor, then $E' : \{0,1\}^n \times \{0,1\}^{t + \log \bar{n}} \to \{0,1\}^m$ defined by:*

$$E'(x; (y, j)) = C(E(x; y)_1)_j \circ \cdots \circ C(E(x; y)_m)_j$$

is a $(k, 2\rho m)$-extractor.

Proof. Suppose E' is not an extractor. Then there exists some distribution X with min-entropy at least k and a function f violating property (3) with $\epsilon = 2\rho m$. More precisely, there exists an i such that

$$\Pr[f(E'(X, U_t)_{1 \ldots i-1}) = E'(X, U_t)_i] \geq \frac{1}{2} + 2\rho$$

It follows from an averaging argument that for a ρ-fraction of pairs (x, y)

$$\Pr_j[f(E'(x; (y, j))_{1, \cdots, i-1} = E'(x; (y, j))_i] \geq 1/2 + \rho$$

We now design a q-ary predictor f' for E. Given $i - 1$ q-ary inputs $w_1, \cdots, w_{i-1}$, we compute

$$r_j = f(C(w_1)_j, \cdots, C(w_{i-1})_j)$$

for $1 \leq j \leq \bar{n}$. Predictor f' outputs a list of size ρ^{-2} of those codewords that differ from r in at most $(1/2-\rho)\bar{n}$ positions. For a ρ fraction of pairs (x,y) at least a $1/2+\rho$ fraction of the r_js are predicted correctly, and hence this list contains $E(x,y)_i$. The existence of predictor f' contradicts E being a (k,ρ) q-ary extractor. □

There are explicit constructions of codes with the required minimum distance and short blocklength $\bar{n} = (\frac{\log q}{\rho})^{O(1)}$ (see, e.g., [GS00]). Thus, this transformation has minimal effect on the seed length. Even relatively simple codes, like a Reed-Solomon code concatenated with a Hadamard code yield the desired parameters. If we are not optimizing constants in the seed length, we can even afford to use perhaps the simplest binary code, the Hadamard code[9], which has relative distance 1/2.

We now combine Lemma 4.16 with Theorem 4.5 to obtain the first two corollaries stated in Section 4.2. We now restate and prove these corollaries.

Corollary 4.6 (restated). *For all n, constants $\epsilon, \delta > 0$, and $k \geq \log^{4/\delta} n$ our construction gives an explicit (k,ϵ)-extractor with seed length $t = O(\log n)$ and output length $m = k^{1-\delta}$.*

Proof. We choose:

- $h = k^{\delta/2}$
- $d = \log n/(\log(h-1) - \log\log n)$
- $\rho = \epsilon/2m$
- $q = \Theta(\rho^{-4}(hd)^2)$

We verify that the conditions of Theorem 4.5 (1) are met:

$$\binom{h+d-1}{d} > \left(\frac{h-1}{d}\right)^d \geq \left(\frac{h-1}{\log n}\right)^d \geq n$$

By the lower bound on k,

$$O(mhd\log q) + \log(1/\rho) = O(k^{1-\delta}k^{\delta/2}\log n) < k$$

The lower bound on k also gives that $h \geq \log^2 n > d$ and thus the extractor runs in time $q^{O(d)} = (hdm)^{O(d)} = h^{O(d/\delta)} = n^{O(1/\delta)}$.[10] Using the Hadamard code for the conversion from q-ary extractors, the seed length $t = d\log q + \log q = O(\log n)$ is as stated in the corollary. □

One of the advantages of the extractors constructed in [TSZS01] are that they optimize the leading constant in the seed length. By picking parameters appropriately, we can also approach seed length $1 \cdot \log n$.

Corollary 4.7 (restated). *For all n, constants $\epsilon > 0$ and $1/8 > \delta > 0$, and $k \geq \log^{4/\delta} n$, our construction gives an explicit (k,ϵ)-extractor with seed length $t = (1+O(\delta))\log n$ and output length $m = k^{\delta}$.*

Proof. We choose the following parameters:

[9] The Hadamard encoding of a $\log q$-bit string x is $C(x) = \{\sum x_i y_i \bmod 2\}_{y \in \{0,1\}^{\log q}}$.

[10] We remark that in Corollary 4.21 we get an extractor with the same parameters with running time $n^{O(1)}$ where the constant in the running time does not depend on δ.

- $h = \sqrt{k}$
- $d = \log n / (\log(h-1) - \log\log n)$
- $\rho = \epsilon/2m$
- $q = \Theta(\rho^{-4} hd \log n)$

We verify that the conditions of Theorem 4.5(2) are met. As before, $\binom{h+d-1}{d} > n$ and $d < \log n$.

$$O(mhd \log^2 q) + \log(1/\rho) = O(k^{1/2+\delta} \log^2 n) < k$$

holds by the lower bounds on k and δ. Using a code with blocklength $(\log q/\rho)^{O(1)}$ for the conversion from q-ary extractors, the seed length is:

$$t = d \log q + O\left(\log \frac{\log q}{\rho}\right) \leq d \log h + d \cdot O(\log m + \log\log n).$$

By the lower bound on k, we have $h > \log^{1/(2\delta)} n$, so $\log\log n < 2\delta \log h$ and thus $d \log h = (1 + O(\delta)) \log n$ and $d \log\log n = O(\delta \log n)$. Finally, $\log m = 2\delta \log h$, so $d \log m = O(\delta \log n)$, and so altogether, we obtain the stated bound on t. The running time is $q^{O(d)} = (hdm)^{O(d)} = h^{O(d)} = n^{O(1)}$. $\square$

The quality of our extractors can be significantly improved by using a more complex transformation of q-ary extractors to (regular) extractors. Such a transformation was given in [TSZS01].

Theorem 4.17. *[TSZS01] Let F be the field with q elements. For every k, ρ, and m, there is a polynomial time computable function*

$$B : F^m \times \{0,1\}^{O(\log\log q) + \log^{O(1)}(1/\rho)} \to \{0,1\}^{(1-O(\sqrt{\rho}))m - O(\log^* m(\log^* m + \log(1/\rho)))}$$

such that for any (k, ρ) q-ary extractor E with output length m,

$$E'(x; (y, j)) = B(E(x, y), j)$$

is a $(k, O(\rho \log^* m))$-extractor.

The expressions above are a bit complicated. The important thing to notice is that when ρ is not too small, the q-ary extractor is converted into a regular extractor with roughly the same seed length and output length. While this is also the case with Lemma 4.16 the important difference is the relation between the error of the q-ary extractor and the final extractor. In Lemma 4.16 the error of the final extractor is $2\rho m$, as compared to $O(\rho \log^* m)$ in Theorem 4.17. In Corollaries 4.6 and 4.7 we had to choose $\rho = \epsilon/2m$ to get error ϵ when applying Lemma 4.16. As the seed length of the q-ary extractor is at least $d \log(1/\rho) \geq d \log m$ we had to make sure that $d \log m$ was small, i.e., $O(\log n)$. Since $d \approx \log n / \log h$ (so that $\binom{h+d-1}{d} > n$), this forces us to choose h to be $m^{\Omega(1)}$. The effect of this choice is that we extract only a small fraction of the randomness in the source as $m \leq k/h$. However using Theorem 4.17 we can choose $\rho = O(\epsilon/\log^* m)$. This allows us to choose much smaller h, (say $h = \log^{O(1)} n$), and extract a larger fraction of the randomness in the source.

Using Theorem 4.17 on top of our construction we get the following extractor:

Corollary 4.8 (restated). *For all n, k, constants $\epsilon, \delta > 0$ our construction gives an explicit (k, ϵ)-extractor with seed length $t = O(\log n)$ and output length $m = k/(\log n)^{2+\delta}$.*

Proof. We choose $h = \log^{1+\delta/2} n$, $d = \frac{\log n}{\log(h-1)-\log\log n}$, $\rho = \Theta(\epsilon/(\log^* m))$, and $q = \Theta(\frac{(hd)^2}{\rho^4})$. These choices meet the requirements of Theorem 4.5, and give the required parameters. The computations are similar to those made in Corollary 4.6. Note that the running time of the extractor is $q^{O(d)} \leq h^{O(d)} = n^{O(1)}$. □

We can further reduce the seed length to $(1+\delta)\log n$ at the cost of extracting slightly fewer bits.

Corollary 4.9 (restated). *For all n, k, constant $\epsilon > 0$ and any $0 < \delta < 1$ (not necessarily a constant) our construction gives an explicit (k,ϵ)-extractor with seed length $t = (1+\delta)\log n$ and output length $m = k/(\log n)^{O(1/\delta)}$.*

Proof. We choose $h = \log^{\Theta(1/\delta)} n$, $d = \frac{\log n}{\log(h-1)-\log\log n}$, $\rho = \Theta(\epsilon/(\log^* m))$, and $q = \Theta(\frac{hd\log q}{\rho^4})$. The computations we have to make use the fact that $h > \log^{\Theta(1/\delta)} n$, and are similar to those made in Corollary 4.7, with the exception that the running time is $q^{O(d)} = h^{O(d)} = n^{O(1)}$. □

4.6 Extractors with small error

The extractor constructions of the previous section were stated for constant error ϵ, but they can be tuned to give extractors with small (non-constant) error ϵ. However, for very small ϵ, a problem arises. To obtain error ϵ, we must construct a (k,ρ) q-ary extractor with $\rho \leq \epsilon$, which forces $q \geq 1/\rho \geq 1/\epsilon$. One part of the running time of the extractor comes from finding the "generator matrix" A, which takes time $q^{O(d)}$ by brute-force search (we currently do not know of a better method). For very small ϵ, this step takes super-polynomial time, and hence our extractors are not explicit in the usual sense of running in time polynomial in n, the length of the input[11].

In this section we show that with some minor restrictions on d, h and q, we can replace A with another matrix B that can be found in time poly$(h^d, \log q)$. In our constructions we always choose h and d such that $h^d = n^{O(1)}$, and thus this modification allows the entire construction to run in time polynomial in n, even for very small ϵ.

The main observation is the following: The only property of A that we used (in addition to it being non-singular) was that for any nonzero $v \in F^d$, the polynomial $\hat{x}$ is determined by its evaluations on the points $\left\{A^i v | 1 \leq i \leq q^d\right\}$ (which is in fact $F^d \setminus \left\{\vec{0}\right\}$ for the chosen matrix A). In the modified construction we replace the matrix A with a matrix B with the following property: There exists a subset $H \subseteq F$ of size h such that for every nonzero $v \in F^d$, there exists an invertible linear transform T_v such that $T_v \cdot H^d \subseteq \left\{B^i v | 1 \leq i < h^d\right\}$. In words, starting from v and taking consecutive steps according to B traverses a "shifted cube", that is a cube H^d shifted by some invertible linear transform. The polynomial $\hat{x}$ (which is of degree $h-1$) is indeed determined by its evaluations on such a shifted cube. Thus, the proof works when replacing A with B. We show that finding the matrix B can be done in time poly$(h^d, \log q)$.

Lemma 4.18. *Let h, q and d be such that: h is a prime power, q is a power of h, and d and $\log_h q$ are relatively prime. Then there exists an invertible $d \times d$ matrix B with entries from $F = GF(q)$, and a set $H \subseteq F$ with $|H| = h$ such that B^{h^d-1} is the identity matrix and for every nonzero $v \in F^d$ there is an invertible linear transform $T_v : F^d \to F^d$ for which:*

$$T_v \cdot (H^d \setminus \left\{\vec{0}\right\}) = \left\{B^i v | 1 \leq i < h^d\right\}.$$

[11] We remark that we can find A in a pre-processing stage; after this one-time expenditure, the extractor runs in time polynomial in n.

Moreover, B can be found in time poly$(h^d, \log q)$.

The proof of Lemma 4.18 appears in Section 8.

Lemma 4.19. *Let $\hat{x} : F^d \to F$ be a multivariate polynomial with total degree $\leq h - 1$. Let $H \subseteq F$ be a subset of size h, and let $T : F^d \to F^d$ be an invertible linear transform. Then $\hat{x}$ is uniquely determined by its evaluations on the points $T \cdot H^d$.*

Proof. The evaluations of $\hat{x}$ at the points $T \cdot H^d$ are simply the evaluations of the polynomial $p(z) = \hat{x}(Tz)$ at the points H^d. Since $p(z)$ has total degree $< h$, it is uniquely determined by these evaluations, which in turn uniquely determine $\hat{x}$, since $\hat{x}(w) = p(T^{-1}w)$. $\square$

The modified construction: We now present the construction of a q-ary extractor E' that is nearly identical to the extractor E of Section 4.2. The only difference is that E' uses the matrix B of Lemma 4.18 instead of the matrix A of Lemma 4.4. The result is that E' runs in time poly$(h^d, \log q)$ as opposed to poly(q^d). This allows us to set q large enough to handle very small error without blowing up the running time.

$$E'(x, \vec{v}) = \hat{x}(B^1\vec{v}) \circ \hat{x}(B^2\vec{v}) \circ \cdots \circ \hat{x}(B^m\vec{v}) \tag{8}$$

The only difference between the following theorem and Theorem 4.5 is the additional restrictions on q, h, and d.

Theorem 4.20 (q-ary extractors with small error). *There exists a universal constant c such that for every n, d, h and q satisfying $\binom{h+d-1}{d} \geq \frac{n}{\log q}$* and *for which h is a prime power, q is a power of h, and d and $\log_h q$ are relatively prime, E' is a (k, ρ) q-ary extractor, provided that*

1. *$k > cmhd \log q + \log\left(\frac{1}{\rho}\right)$ and $q > c\left(\frac{(hd)^2}{\rho^4}\right)$, or*
2. *$k > cmhd \log^2 q + \log\left(\frac{1}{\rho}\right)$ and $q > c\left(\frac{hd \log q}{\rho^4}\right)$.*

Moreover, E can be computed in time poly$(h^d, \log q)$.

Proof. The proof is almost identical to that of Theorem 4.5 (using B instead of A). The only modification is in the last paragraph of the proof of Lemma 4.13. At this point we have learned the evaluations of $\hat{x}$ on points that include $\left\{B^i p_1(1) | 1 \leq i \leq h^d\right\}$. By Lemma 4.19 these evaluations uniquely determine $\hat{x}$, and the reconstruction function then outputs x, as before. The remainder of the proof is unchanged. $\square$

Using almost the same choice of parameters as in Corollary 4.6 we obtain the following extractor. To meet the additional requirements on h, q, and d, we can choose h and q to be powers of 2, and d to be a prime at most twice the value chosen in Corollary 4.6.

Corollary 4.21. *For all n, constant $\delta > 0$, $\epsilon \geq 2^{-(k^{\delta/4})}$, and $k \geq \log^{4/\delta} n$ our construction gives an explicit (k, ϵ)-extractor with seed length $t = O\left(\log n + \frac{\log n}{\log k} \log\left(\frac{1}{\epsilon}\right)\right)$ and output length $m = k^{1-\delta}$.*

Notice that when $k = n^{\Omega(1)}$ the seed length of this extractor is $O(\log(n/\epsilon))$, which gives the asymptotically optimal dependence on n and ϵ.

It is possible to also get low-error analogs of all our extractors. However, we remark that the additional "divisibility" requirements on d, h and q allow achieving seed length close to $1 \cdot \log n$ only for carefully chosen values of n, k and ϵ.

5 A new pseudorandom generator

In this section we exploit the connection Trevisan noticed between extractors and PRG's "the other way": we build PRGs (and HSGs) using the ideas outlined in Section 2.3.

5.1 Pseudorandom generator preliminaries

In this section the string x plays the role of a "hard function".

Definition 5.1. *We identify the string $x \in \{0,1\}^n$ with the function $x : \{0,1\}^{\log n} \to \{0,1\}$ by setting $x(i) = x_i$. We denote by $S(x)$ the size of the smallest circuit computing function x.*

We now define pseudorandom generators.

Definition 5.2 (PRG). *An ϵ-*PRG for size s *is a function $G : \{0,1\}^t \to \{0,1\}^m$ such that for all size s circuits C:*

$$|\Pr[C(G(U_t)) = 1] - \Pr[C(U_m) = 1]| \le \epsilon \tag{9}$$

As in the case of extractors, by [Yao82] property (9) follows from the next property:

$$\forall\ 1 \le i \le m \text{ and all functions } f : \{0,1\}^{i-1} \to \{0,1\} \text{ with size } s - O(1) \text{ circuits },$$
$$\Pr[f(G(U_t)_{1\ldots i-1}) = G(U_t)_i] \le \frac{1}{2} + \frac{\epsilon}{m}. \tag{10}$$

As in Section 4 we can define the q-ary version of PRGs.

Definition 5.3 (q-ary PRG). *Let F be the field with q elements. A ρ-q-*ary PRG for size s *is a function $G : \{0,1\}^t \to F^m$ such that*

$$\forall\ \ 1 \le i \le m \textit{ and all functions } f : F^{i-1} \to F^{[\rho^{-2}]} \textit{ with size } s \textit{ circuits },$$
$$\Pr[G(U_t)_i \in f(G(U_t)_{1\ldots i-1})] \le \rho. \tag{11}$$

As in Section 4 we will focus on constructing q-ary PRGs and later transform them into (regular) PRGs.

5.2 Overview of changes to the extractor construction

Given a function $x : \{0,1\}^{\log n} \to \{0,1\}$ that cannot be computed by circuits of size s, our goal is to construct a q-ary PRG $G_x : \{0,1\}^t \to F^m$. The construction will be quite similar to the extractor construction $E(x, \cdot) : \{0,1\}^t \to \{0,1\}^m$, and to prove correctess, we will derive a contradiction from the existence of a "predictor" f violating (11) above. As with the extractor this is done by describing a "reconstruction procedure" R. There are two important differences in what we require of R in the PRG setting as compared to the extractor setting. First, R takes an additional input i (as well as the short advice string a which is the same for all i) and should output $x(i)$ as opposed to simply outputting all of x. Second, R should be *efficient*; that is, it should run in time $\ll n$ (as it is trivial to construct a circuit of size n for a function on $\log n$ bits). Our goal will be to run in time poly(m) for some fixed polynomial. Since f also has a small circuit, we can compute $x(i)$ efficiently by evaluating $R^f(a, i)$, which for the proper choice of parameters will give a circuit of size poly(m) that contradict the hardness of the function x. In order to meet these new requirements, we need to make some changes to the construction and the proof.

The encoding: In the extractor setting, $\hat{x}$ is the Reed-Muller encoding of x, and in order to determine $x(i)$ from the encoding, we need to learn enough of $\hat{x}$ to be able to interpolate and find its coefficients. The efficiency demands on R in the PRG setting preclude being able to learn this many evaluations of $\hat{x}$ (we would need n evaluations, but are allowed only poly(m) time). Therefore, we use an alternate "low-degree extension" encoding for the PRG. In this scheme we again encode x as a low-degree polynomial $\hat{x} : F^d \to F$, but we also ensure that there is an efficiently computable function $\ell : [n] \to F^d$ such that $x(i) = \hat{x}(\ell(i))$. Thus, we can determine $x(i)$ by learning only the specified evaluation of $\hat{x}$.

The standard way to produce $\hat{x}$ from x is to pick an arbitrary set $H \subseteq F$ of size h with $h^d = n$ and any efficient one-to-one function $\ell : [n] \to H^d$, and define $\hat{x}$ to be any polynomial with degree at most h in each variable for which $\hat{x}(\ell(i)) = x(i)$. Our reconstruction procedure operates on the cyclic group with generator A (which corresponds to $F^d \setminus \left\{\vec{0}\right\}$), and if we use this standard low-degree extension, we are stuck having to compute the integer j such that $A^j\vec{1} = \ell(i)$ whenever we want to determine $x(i)$. Finding such a j is a discrete-log problem that we don't know how to solve efficiently. Instead, we use a specific embedding of x into $\hat{x}$ that avoids this problem: $x(i)$ is embedded at location $A^{ip}\vec{1} \in F^d$, for a fixed integer p; thus simply knowing i gives us the required exponent. Such an embedding is somewhat delicate: we need to arrange for $\left\{A^{ip}\vec{1}\right\}_{i\in\mathbb{Z}}$ to coincide with H^d; the details on how to achieve such a matrix are in Section 8.

The reconstruction: As explained in the introduction, the main idea is to use *several* predictors with varying strides when performing the reconstruction. This allows us to travel quickly from a fixed point (like $\vec{1}$) to any given point in F^d. To implement this idea we need two new ingredients. First, the curves C_1 and C_2 need to have the intersection properties we used in the extractor setting *for each stride* that we will use for the PRG. This is achieved by generalizing the idea used to obtain the intersections for the extractor setting, which in turn requires slightly larger degree curves. Second, for every $\vec{v} \in F^d$ we need to describe an efficiently computable short sequence of prediction steps (with varying strides) that starts from $\vec{1}$ and reaches $\vec{v}$ – this is used by the reconstruction procedure to rapidly learn $x(i)$ using only the predictor and the initial evaluations of $\hat{x}$ supplied by the advice string.

5.3 The actual construction

Our construction starts with a hard function $x : \{0,1\}^{\log n} \to \{0,1\}$ and encodes it as a low-degree polynomial $\hat{x} : F^d \to F$. Just as before, the major parameters are the field size q, the dimension d and the degree h. As with extractors we will think of F^d as both a vector-space and an extension field of F. However, to produce the non-standard low-degree extension described above, we will additionally require that F have a subfield H of size h (which forces h to be a prime power, and q to be a power of h). We have the following lemma which extends Lemma 4.4.

Lemma 5.4. *Let h, q and d be such that: h is a prime power, q is a power of h, and d and $\log_h q$ are relatively prime. Let F be the field with q elements and H be the subfield of F with h elements. Then there exist invertible $d \times d$ matrices A and B with entries from F that satisfy:*

- *A^{q^d-1} and B^{h^d-1} are the identity matrix.*
- *For any non-zero vector $\vec{v} \in F^d$: $\left\{A^i\vec{v}\right\}_{1\le i<q^d} = F^d \setminus \{0\}$.*
- *For any non-zero vector $\vec{v} \in H^d$: $\left\{B^i\vec{v}\right\}_{1\le i<h^d} = H^d \setminus \{0\}$.*

- $B = A^{(q^d-1)/(h^d-1)}$.

- *A, B can be found in time $q^{O(d)}$.*

The proof of Lemma 5.4 appears in Section 8. For our low-degree-extension of x, we require (compare to (5)):

$$h^d > n. \tag{12}$$

We will "embed" x into a polynomial defined over the vector-space F^d as follows: we want $\hat{x}(B^i\vec{1}) = x(i)$. Here, $\vec{1}$ is the all-ones vector, which is in $H^d \subseteq F^d$ since $1 \in H \subseteq F$, and which serves as a reference vector throughout the construction. Since $h^d - 1 \geq n$, there are enough "slots" to embed all of x, and since B generates $H^d \setminus \{0\}$, we have embedded all of x in a d dimensional cube with sidelength h. Therefore there exists a polynomial $\hat{x}$ over F^d with degree at most $h-1$ in each variable such that $\hat{x}(B^i\vec{1}) = x(i)$. Note that in this section h denotes individual variable degrees and the total degree is at most hd. Once A has been determined, the coefficients of $\hat{x}$ can be computed in time poly(n) using standard methods, and $\hat{x}$ can be evaluated at any point in F^d in time poly$(n, \log q)$.

We now describe d "candidate" PRGs. For $0 \leq j < d$ we define functions $G_x^{(j)} : \{0,1\}^{d \log q} \to F^m$ as follows. We think of the input $\vec{v}$ as a vector in F^d.

$$\begin{aligned}
G_x^{(0)}(\vec{v}) &= \hat{x}(A^1\vec{v}) \circ \hat{x}(A^2\vec{v}) \circ \cdots \circ \hat{x}(A^m\vec{v}) \\
G_x^{(1)}(\vec{v}) &= \hat{x}(A^{q\cdot 1}\vec{v}) \circ \hat{x}(A^{q\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q\cdot m}\vec{v}) \\
G_x^{(2)}(\vec{v}) &= \hat{x}(A^{q^2\cdot 1}\vec{v}) \circ \hat{x}(A^{q^2\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q^2\cdot m}\vec{v}) \\
\vdots\;\; &\;\;\vdots \\
G_x^{(j)}(\vec{v}) &= \hat{x}(A^{q^j\cdot 1}\vec{v}) \circ \hat{x}(A^{q^j\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q^j\cdot m}\vec{v}) \\
\vdots\;\; &\;\;\vdots \\
G_x^{(d-1)}(\vec{v}) &= \hat{x}(A^{q^{d-1}\cdot 1}\vec{v}) \circ \hat{x}(A^{q^{d-1}\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q^{d-1}\cdot m}\vec{v})
\end{aligned} \tag{13}$$

Note that each $G_x^{(j)}$ corresponds to using our q-ary extractor construction with the "successor function" A^{q^j}. Our main theorem will show that at least one of these functions is a q-ary PRG, provided x is a sufficiently hard predicate.

Theorem 5.5 (PRG main). *There exists a universal constant c such that for every n, d, h and q satisfying $h^d > n$ and the conditions of Lemma 5.4, at least one $G_x^{(j)}$ is a ρ-q-ary PRG for size s, provided that $S(x) > s \cdot poly(m, q)$ and $q > max(c\rho^{-4}hd^2 \log^2 q, 2d^4 \log^2 q)$. Furthermore, all the $G_x^{(j)}$s are computable in time $poly(q^d, n)$, given x.*

We prove Theorem 5.5 in Section 5.5. We now show how to construct a *single* (binary) PRG. It will be convenient to fix all parameters as functions of n, m. We require that $\log n \leq m \leq n$ and set:

- $\rho = m^{-3}$

- $h = m$

- $d = \log n / \log m$

- $q = m^{20} > c\rho^{-4} h d^2 \log^2 q$

By these choices the seed length of each "candidate generator" is $O(\log n)$. Our next goal is to transform the q-ary candidates into binary ones, and for this we use a computational analogue of Lemma 4.16. In this case, it is not sufficient that the encoding procedure is efficient. We also require that the code has efficient (list)-decoding.

Lemma 5.6. *Let F be the field with q elements, Let $C : \{0,1\}^{\bar{k}=\log q} \to \{0,1\}^{\bar{n}}$ be an efficiently encodable and efficiently list-decodable (ρ, ρ^{-2})-list-decodable code. If $G_x : \{0,1\}^t \to F^m$ is a ρ-q-ary generator for size $s\bar{n} + m\bar{n}^{O(1)} + (\bar{n}/\rho)^{O(1)}$, then $G'_x : \{0,1\}^{t+\log \bar{n}} \to \{0,1\}^m$ defined by:*

$$G'_x(y, j) = C(G_x(y)_1)_j \circ C(G_x(y))_2)_j) \circ \cdots \circ C(G_x(y))_m)_j$$

is a $2\rho m$-PRG for size s.

Lemma 5.6 follows from the proof of Lemma 4.16 by using the additional efficiency requirements. Using, for example, [GS00], we can obtain such a code with $\bar{n} = \text{poly}(\bar{k}, \rho^{-1})$. However, in our setting $\bar{k} = \log q = O(\log m)$ and we can have $\bar{n}$ as large as $m^{O(1)}$ (we just need to avoid losing too much in the size of the circuit fooled when applying Lemma 4.16). Therefore we can even use the simpler Hadamard code, together with the trivial list-decoding algorithm.[12]

We use Lemma 5.6 to transform each of the $G_x^{(j)}$'s into a binary function ${G'_x}^{(j)} : \{0,1\}^{d \log q + \log \bar{n}} \to \{0,1\}^m$ (and note that $d \log q + \log \bar{n} = O(\log n)$ by our choice of parameters). We conclude that:

Corollary 5.7. *At least one ${G'_x}^{(j)}$ is a $1/m$-PRG for size s provided that $S(x) > sm^{O(1)}$. Furthermore, all the ${G'_x}^{(j)}$'s are computable in time $n^{O(1)}$, given x.*

Our PRG is obtained by "XOR-ing" the d candidate functions with independent seeds:

$$G_x(y_0, \cdots, y_{d-1}) = {G'_x}^{(0)}(y_0) \oplus \cdots \oplus {G'_x}^{(d-1)}(y_{d-1}) \tag{14}$$

It is standard that "XOR-ing" many candidates where one of them is a PRG for size s indeed produces a PRG for size s; for a proof, see [ISW03]. The seed length of G_x is $O(d \log n) = O(\frac{\log^2 n}{\log m})$. This matches the parameters of the PRG construction of [STV01].

Corollary 5.8. *For any s, if there exists a function $g : \{0,1\}^{\log n} \to \{0,1\}$ that is computable in time $n^{O(1)}$ with $S(g) \geq s$ then there exists a $1/m$-PRG for size m with seed length $t = O(\log^2 n / \log s)$ and output length m, for $m = s^{\Omega(1)}$. Furthermore, this generator can be computed in time $n^{O(1)}$.*

The most important implication of this corollary is a new proof of the Impagliazzo-Wigderson Theorem [IW97], which states that $BPP = P$ if there exist a function family in E that requires exponential size circuits. More precisely given $g \in E$ that requires size $2^{\Omega(\ell)}$ circuits on inputs of length ℓ, and a BPP algorithm that runs in time $t(n) = n^c$ for some constant c, we choose $\ell = c' \log n$ where c' is a large enough constant so that our PRG construction based on g with input length ℓ fools circuits of size $s = n^c$. The seed length is $t = O(\ell^2 / \log 2^{\Omega(\ell)}) = O(\log n)$ and thus in time polynomial in n we can run the algorithm over all outputs of the PRG and take the majority vote. This gives a deterministic polynomial time algorithm.

[12]The trivial list-decoding algorithm is to go over all codewords, encode them and choose the ones that are sufficiently close to the received word.

5.4 Hitting set generators and an optimal hardness vs. randomness tradeoff

Hitting set generators (HSGs) are designed to derandomize RP.

Definition 5.9 (HSG). *A function $H : \{0,1\}^t \to \{0,1\}^m$ is an ϵ-*HSG *for size s circuits if for all size s circuits $C : \{0,1\}^m \to \{0,1\}$*

$$\Pr[C(U_m) = 1] \geq \epsilon \Rightarrow \Pr[C(H(U_t)) = 1] > 0.$$

One of the ${G'_x}^{(j)}$'s is a PRG and therefore if we choose a random candidate, we will hit this PRG with positive probability. We define:

$$H_x(y, j) = {G'_x}^{(j)}(y)$$

It is standard that choosing a random candidate from a collection of functions where one of them is a PRG produces an HSG; for a proof see [ISW99]. Very few bits (at most $\log d \leq \log\log n + O(1)$) are needed to choose j and thus we get an *optimal* HSG.

Corollary 5.10. *For any s, if there exists a function $g : \{0,1\}^{\log n} \to \{0,1\}$ that is computable in time $n^{O(1)}$ with $S(g) \geq s$ then there exists a $1/m$-HSG for size m with seed length $t = O(\log n)$ and output length $m = s^{\Omega(1)}$. Furthermore, this generator can be computed in time $n^{O(1)}$.*

This improves upon the best previous results by [ISW99] which has $m = s^{\Omega(1/\log\log\log n)}$. In [ACR98] (see also [ACRT99, BF99, GVW00]) it was shown how to derandomize two-sided error probabilistic algorithms using an HSG. Applying this result, we extend the Impagliazzo-Wigderson Theorem [IW97] to any hardness assumption.

Corollary 5.11. *If there exists a function family $g = \{g_\ell\} \in E$ that require size $s(\ell)$ circuits, then for any time constructible function $t(n)$, BPTIME$(t(n)) \subseteq$ DTIME$(2^{O(s^{-1}(t(n)^{O(1)}))})$.*

Actually, in our setup we can use a simpler construction from [ISW99]. Given many candidate generators where at least one of them is pseudorandom, [ISW99] showed how to derandomize two-sided error probabilistic algorithms by conducting a "tournament of generators".

5.5 Proof of the main PRG theorem

In this section we prove Theorem 5.5. Let n, q, d, and h be as in the statement of Theorem 5.5. We fix a string $x \in \{0,1\}^n$, and let $\hat{x} \in F^d$ be the encoding of x described in the Section 5.3. Assume for the purpose of contradiction that *no* $G_x^{(j)}$ is a ρ-q-ary PRG. Then by definition we have integers $i^{(j)}$ and next-element predictors $f^{(j)} : F^{i^{(j)}-1} \to F^{[\rho^{-2}]}$ violating property (11) for each $G_x^{(j)}$, respectively. By the symmetry of our PRGs, we can assume that $i^{(j)} = m$ for all j, as predictor $f^{(j)}$ can simply ignore its first $m - i^{(j)}$ inputs. In other words, we have that for all j:

$$\Pr[G_x^{(j)}(U_t)_m \in f(G_x^{(j)}(U_t)_{1\ldots m-1})] > \rho$$

Each of these predictors can be implemented by a size s circuit. As in the proof of Theorem 4.5 our task is to use these predictors and a short advice string to reconstruct x. However, in this setup reconstructing x means constructing a small circuit that computes $x(i)$ given input i.

It will be helpful to abstract the process used in the proof of Theorem 4.5 to learn a new curve and present it as a procedure. In the remainder of this section $\deg(\hat{x}) \leq d(h-1)$ denotes the total degree of polynomial $\hat{x}$.

Procedure Learn Next Curve

- Input:
  - next curve $C : F \to F^d$: a degree v polynomial
  - reference points $R \subseteq F$: a set of r distinct elements from F
  - stride j: an integer in $[0 \ldots (d-1)]$
  - input evaluations $\{a_t^\ell\}_{t \in F, \ell \in [1 \ldots (m-1)]}$ and $\{b_t\}_{t \in R}$: elements of F whose *intended values* are $a_t^i = \hat{x}(A^{-iq^j} C(t))$ and $b_t = \hat{x}(C(t))$.
- Output:
  - output evaluations $\{c_t\}_{t \in F}$: elements of F whose *intended values* are $c_t = \hat{x}(C(t))$.
- Action:
  - For each $t \in F$, compute $f^{(j)}(a_t^{m-1}, a_t^{m-2}, \ldots, a_t^1)$, which gives a set S_t of ρ^{-2} values.
  - Apply Lemma 3.1 on the $\mathsf{prs} = q\rho^{-2}$ pairs $\{(t,e)\}_{t \in F, e \in S_t}$ (assuming the agreement is $\mathsf{agr} = \rho q/4$) to obtain a list of at most $8\rho^{-3}$ degree $\deg(\hat{x}) \cdot v$ univariate polynomials $p(t)$ that contains all polynomials such that $p(t) \in S_t$ for at least $\mathsf{agr} = \rho q/4$ values of t. If this list is empty, fail. Note that to apply Lemma 3.1 we will need to satisfy $\mathsf{agr} > \sqrt{2 \cdot \mathsf{deg} \cdot \mathsf{prs}}$ or equivalently that $q > 32\deg(\hat{x}) \cdot v/\rho^4$.
  - If the list contains a unique polynomial $p(t)$ for which $p(t) = b_t$ for all $t \in R$, output $\{p(t)\}_{t \in F}$; otherwise fail.

We say that **Learn Next Curve** *succeeds* (on a curve, reference points and stride) if its output evaluations are the intended values when its input evaluations are the intended values. We now argue that for a random next curve C and a random set of reference points $R \subseteq F$, the procedure succeeds with high probability.

Lemma 5.12. *Let $n, q, d,$ and h be as in the statement of Theorem 5.5. Let v be such that*

$$q > 32 deg(\hat{x}) \cdot v/\rho^4.$$

For all strides $0 \leq j \leq d-1$,

$$\Pr_{C,R}[\textbf{Learn Next Curve}\ \textit{succeeds}] \geq 1 - \left(\frac{O(v)}{\rho q}\right)^{v/2} - (8\rho^{-3})\left(\frac{v \cdot deg(\hat{x})}{q}\right)^r,$$

where $C : F \to F^d$ is a uniformly chosen degree v curve, and $R \subseteq F$ is a uniformly chosen subset of F of size r that is independent from C.

Proof. The argument is almost identical to that in Section 4.4. We first argue that with high probability $\hat{x}(C(t))$ is predicted correctly for many t's.

Claim 5.13. *With probability at least* $1 - \left(\frac{O(v)}{\rho q}\right)^{v/2}$ *over the choice of* C*:*

$$\frac{|\{t : \exists e \in S_t,\ e = \hat{x}(C(t))\}|}{q} \geq \frac{\rho}{4}$$

Proof. (of Claim 5.13) The proof is identical to that of Claim 4.14. We argue that the collection of points $\{C(t)\}_{t \in F}$ is v-wise independent (over the choice of C), and then use Lemma 3.2. □

It follows that with probability at least $1 - \left(\frac{O(v)}{\rho q}\right)^{v/2}$ the procedure applies Lemma 3.1 with enough "correct pairs", and therefore one of the polynomials in the list is the polynomial $p(t) = \hat{x}(C(t))$. Two polynomials of degree $v \cdot \deg(\hat{x})$ can agree on at most a $v \cdot \deg(\hat{x})/q$ fraction of their points, so the probability that an "incorrect" polynomial from the list agrees with p on r random points is at most

$$(8\rho^{-3}) \left(\frac{v \cdot \deg(\hat{x})}{q}\right)^r.$$

If $\hat{x}(C(t))$ is predicted correctly for enough t's so that $p(t)$ appears in the list, and no "incorrect" polynomials in the list agree on the r random points in R, the procedure succeeds. □

As in the previous section, if $C : F \to F^d$ is a degree v curve and A is a $d \times d$ matrix, then we denote by AC the function defined by $AC(t) = A \cdot C(t)$ and recall that this is also a degree v curve. In some contexts we also use AC to denote the multi-set $\{AC(t) | t \in F\} \subseteq F^d$, and we adopt the shorthand $A(C_1 \cup C_2)$ for $AC_1 \cup AC_2$. Given two curves C_1 and C_2 we use $[C_1 \cap C_2]$ to denote the set $\{t \in F | C_1(t) = C_2(t)\}$.

Lemma 5.14. *Let* $n, q, d,$ *and* h *be as in the statement of Theorem 5.5. There exist degree* $v = O(d^2 \log q)$ *curves* C_1 *and* C_2 *for which the following hold:*

- $C_1(1) \neq \vec{0}$.
- *for all* $1 \leq i < q^d$ *and all* $0 \leq j \leq d-1$, $[A^{i+q^j} C_1 \cap A^i C_2]$ *and* $[A^i C_1 \cap A^i C_2]$ *are of size at least* r.
- *for all* $1 \leq i < q^d$ *and all* $0 \leq j \leq d-1$, **Learn Next Curve** *succeeds given* next curve $A^{i+q^j} C_1$, reference points $[A^{i+q^j} C_1 \cap A^i C_2]$ *and* stride j, *and*
- *for all* $1 \leq i < q^d$ *and all* $0 \leq j \leq d-1$, **Learn Next Curve** *succeeds given* next curve $A^i C_2$, reference points $[A^i C_1 \cap A^i C_2]$ *and* stride j.

Proof. We pick C_1 and C_2 randomly with certain intersection properties, and apply Lemma 5.12 to argue that with high probability each invocation of **Learn Next Curve** listed above succeeds. A union bound then shows that with non-zero probability *all* such invocations succeed, and the lemma follows.

Set $r = c' d \log q$ for some constant c' to be chosen later. Set $u = d + 1$, and pick ur random points from F^d:

$$\vec{y_{11}}, \vec{y_{12}}, \ldots \vec{y_{1r}}, \vec{y_{21}}, \vec{y_{22}}, \ldots \vec{y_{2r}}, \quad \ldots \quad \vec{y_{u1}}, \vec{y_{u2}}, \ldots \vec{y_{ur}}$$

and ur random and distinct values from F:

$$t_{11}, t_{12}, \ldots t_{1r}, t_{21}, t_{22}, \ldots t_{2r}, \quad \ldots \quad t_{u1}, t_{u2}, \ldots t_{ur}.$$

We define the degree $ur - 1$ polynomial C_1 so that $C_1(t_{ij}) = \vec{y_{ij}}$ for all i, j; similarly, we define the degree $v = ur - 1$ polynomial C_2 so that $C_2(t_{1j}) = \vec{y_{1j}}$ for all j and $C_2(t_{ij}) = A^{q^{i-2}} \vec{y_{ij}}$ for $i \geq 2$ and all j. The curves C_1 and C_2 have the following properties for all i:

- The functions A^iC_1 and A^iC_2 are degree $ur-1$ polynomials from F to F^d.
- The functions $\hat{x} \circ A^iC_1$ and $\hat{x} \circ A^iC_2$ are univariate polynomials of degree $\deg(\hat{x})(ur-1)$.
- For $j = 0, 1, 2, \ldots, u-2$ the sets $A^{i+q^j}C_1$ and A^iC_2 intersect at r random positions. More precisely, the random variable $[A^{i+q^j}C_1 \cap A^iC_2]$ is of size at least r. Let S denote the first r elements of $[A^{i+q^j}C_1 \cap A^iC_2]$. The random variable S is uniformly distributed over all distinct r tuples in F, and furthermore, S is independent of $A^{i+q^j}C_1$.
- The sets A^iC_1 and A^iC_2 intersect at r random positions. More precisely, the random variable $[A^iC_1 \cap A^iC_2]$ is of size at least r. Let S' denote the first r elements of $[A^iC_1 \cap A^iC_2]$. The random variable S' is uniformly distributed over all distinct r tuples in F, and furthermore, S' is independent of A^iC_2.

Therefore, by Lemma 5.12, for each *individual* invocation of **Learn Next Curve** listed in the statement of the lemma, the procedure succeeds with probability at least:

$$1 - \left(\frac{\Omega(v)}{\rho q}\right)^{v/2} - (8\rho^{-3})\left(\frac{v \cdot \deg(\hat{x})}{q}\right)^r,$$

which is at least $1 - \frac{1}{8dq^d}$ by our choice of parameters for large enough c'. By the union bound, *all* $2dq^d$ invocations of **Learn Next Curve** listed in the statement of the lemma succeed simultaneously with probability at least $3/4$. The probability that $C_1(1) = \vec{0}$ is q^{-d}. Thus, the lemma holds. $\square$

Using **Learn Next Curve** with the "good" curves C_1 and C_2, we can now construct a small circuit that when given input i produces $x(i)$. The basic step involves *two* invocations of **Learn Next Curve** to learn the evaluation of $\hat{x}$ at the points $A^i(C_1 \cup C_2)$, for some i. Specifically, we first invoke **Learn Next Curve** with next curve A^iC_1, reference points $[A^{i-q^j}C_2 \cap A^iC_1]$ and stride j; then we invoke **Learn Next Curve** with next curve A^iC_2, reference points $[A^iC_1 \cap A^iC_2]$ and stride j. We will call this two-step process *interleaved learning* of $A^i(C_1 \cup C_2)$ using stride j. Notice that to supply **Learn Next Curve** with the correct input evaluations for interleaved learning of $A^i(C_1 \cup C_2)$ using stride j, we need to know the evaluation of $\hat{x}$ at points:

$$\bigcup_{k=1}^{m-1} A^{i-kq^j}(C_1 \cup C_2).$$

By Lemma 5.14 we have that for every i and stride j the interleaved learning of $A^i(C_1 \cup C_2)$ succeeds when supplied with the correct inputs.

Let $p = (q^d - 1)/(h^d - 1)$. Recall that by Lemma 5.4 we have that $B = A^p$. By our encoding, we know that $x(i) = \hat{x}(B^i\vec{1}) = \hat{x}(A^{ip}\vec{1})$. Also, since A generates $F^d \setminus \left\{\vec{0}\right\}$, we know that $C_1(1) = A^a\vec{1}$ for some integer a between 0 and $q^d - 1$. Thus we need to "travel" $b = ip - a \pmod{q^d - 1}$ steps from curve C_1 to reach curve A^bC_1, and then we output the evaluation of $\hat{x}$ at $A^bC_1(1) = A^{ip-a}A^a\vec{1} = A^{ip}\vec{1}$. Our circuit will be supplied with A, C_1, C_2, a, and the evaluation of $\hat{x}$ at:

$$\bigcup_{k=1}^{m-1} A^k(C_1 \cup C_2)$$

as non-uniform advice.

We now use **Learn Next Curve** in d phases. Write $b' = (b - mq^{d-1}) \bmod (q^d - 1)$ in its q-ary representation: $b' = \sum_{j=0}^{d-1} b_j q^j$. We maintain the invariant that after phase j, we have learned $\hat{x}$ at $A^w(C_1 \cup$

C_2) for an integer w for which w and b' agree on the least significant $j+1$ digits of their q-ary representation. Specifically, we execute the following sequence:

Phase 0**:** Perform interleaved learning of $A^{m+k}(C_1 \cup C_2)$ using stride 0, for $k = 0, 1, \ldots, mq - m - 1$. Notice that the non-uniform advice provides the needed input evaluations.

Phase 1**:** Perform interleaved learning of $A^{mq+b_0+kq}(C_1 \cup C_2)$ using stride 1, for $k = 0, 1, \ldots, mq-m-1$. Notice that the values learned in phase 0 provide the needed input evaluations.

Phase 2**:** Perform interleaved learning of $A^{mq^2+b_0+qb_1+kq^2}(C_1 \cup C_2)$ using stride 2, for $k = 0, 1, \ldots, mq-m-1$. Notice that the values learned in phase 1 provide the needed input evaluations.

$\vdots$

Phase j**:** Perform interleaved learning of $A^{mq^j+\sum_{t=0}^{j-1} b_t q^t + kq^j}(C_1 \cup C_2)$ using stride j, for $k = 0, 1, \ldots, mq-m-1$. The values learned in phase $j-1$ provide the needed input evaluations.

$\vdots$

Phase $d-1$**.** Perform interleaved learning of $A^{mq^{d-1}+\sum_{t=0}^{d-2} b_t q^t + kq^{d-1}}(C_1 \cup C_2)$ using stride $d-1$, for $k = 0, 1, \ldots, b_{d-1}$. The values learned in phase $d-2$ provide the needed input evaluations. The last value of k yields $A^{mq^{d-1}+b'}(C_1 \cup C_2)$, and note that $mq^{d-1} + b' \equiv b \pmod{q^d - 1}$. Since A^{q^d-1} is the identity matrix we have learned $\hat{x}(A^b C_1(1)) = x(i)$, which we output.

Notice that we have invoked **Learn Next Curve** $O(mqd)$ times. Each invocation requires poly(m, q) computation time and invokes a predictor (with a circuit of size s) q times. The total computation time is therefore $O(mqd(sq + \text{poly}(m, q)))$, and the non-uniform advice has size $O(mdhur \log q)$ so altogether the circuit has size $s \cdot \text{poly}(m, q)$. Therefore, if x has hardness greater than this value, we have a contradiction, implying that some $G_x^{(j)}$ must be an ϵ-q-ary PRG. This concludes the proof of Theorem 5.5.

6 Pseudorandom generators for nondeterministic circuits

Just as BPP is a randomized version of P, the class AM (defined in [Bab85, BM88]) is a randomized version of NP. To derandomize BPP (ideally, prove $BPP = P$), we can use PRGs that "fool" small (deterministic) circuits. Such PRGs are built from functions that require large non-uniform (deterministic) complexity, and indeed PRGs imply the existence of such hard functions. To derandomize AM (ideally, prove $AM = NP$), we can use PRGs that "fool" small *nondeterministic* circuits.[13] As usual, such PRGs imply the existence of functions that require large non-uniform *nondeterministic* complexity, and we therefore construct such PRGs assuming the existence of functions that require large non-uniform *nondeterministic* complexity. However, the precise meaning of "non-uniform nondeterministic complexity" is important here, and a number of definitions have been utilized in previous work.

[13] It is known that AM coincides with its one-sided error version [FGM+89], and therefore even HSGs that "fool" small co-nondeterministic circuits suffice.

6.1 Previous work

As discussed in the introduction, PRGs and HSGs have been constructed before from a variety of non-uniform nondeterministic hardness assumptions. Over time, the assumptions have been getting progressively weaker.

Klivans and van Melkebeek [KvM02] observed that the proofs of the NW PRG and the hardness amplification constructions relativize, and therefore functions that are worst-case hard for *circuits with SAT oracle gates* suffice for constructing PRGs that fool circuits with SAT oracle gates, which in turn derandomize AM. This circuit model is the non-uniform analog of P^{NP} while nondeterministic circuits are the non-uniform analog of NP. One can then ask whether anything can be done with the presumably weaker assumption that there exist functions that are worst-case hard for nondeterministic circuits.

Miltersen and Vinodchandran [MV99] used novel techniques to show that functions that are worst-case hard for *single-valued nondeterministic circuits* suffice to build a HSG that derandomizes AM. Although this circuit model is a non-uniform analogue of $NP \cap coNP$, this hardness assumption is equivalent to worst-case hardness for nondeterministic circuits which are a non-uniform analogue of NP. Thus, their result derandomizes AM under a presumably weaker assumption than [KvM02].

However, as noted in the introduction the [MV99] HSG does not give an optimal hardness vs. randomness tradeoff for AM; in fact it fails altogether if the hard function has hardness less than $2^{\sqrt{\log n}}$ (on $\log n$ bits inputs). In this section we construct PRGs and HSGs that fool nondeterminstic and co-nondeterministic circuits using the [MV99] hardness assumption (i.e., there exist functions that are worst-case hard for single-valued nondeterministic circuits), and as a consequence obtain an *optimal* hardness vs. randomness tradeoff for AM (just as we did for BPP). Our PRGs are also the first PRGs (as opposed to HSGs) to fool nondeterministic and co-nondeterministic circuits using only the [MV99] hardness assumption.

We also mention an earlier result in which Arvind and Köbler [AK97] showed that the NW PRG [NW94] works in the nondeterministic setting when given a function x that is hard *on average* for nondeterministic circuits. In the standard (deterministic) setting such "average-case" hardness assumptions were weakened to worst-case hardness assumptions via "hardness amplification" transformations [BFNW93, Imp95, IW97, STV01] which convert worst-case hardness into average-case hardness. However, these transformations were not known to transform worst case *nondeterministic* hardness into average case *nondeterministic* hardness. Our techniques also address this problem; in Section 7 we give the first hardness amplification transformation for nondeterministic circuits.

6.2 Definition of nondeterministic circuits

To state our result we need to briefly review some definitions of nondeterministic circuits.

Definition 6.1 (nondeterministic circuit). *A* nondeterministic circuit C *(resp.* co-nondeterministic circuit C*) is a an ordinary circuit with a single output gate and two sets of inputs:* $x_1, x_2, \ldots, x_n$ *and* $y_1, y_2, \ldots, y_m$*. The function* $f : \{0,1\}^n \to \{0,1\}$ *computed by* C *is defined by* $f(x) = 1$ *iff* $\exists y\ C(x,y) = 1$ *(resp.* $\forall y\ C(x,y) = 1$*).*

Notice that if f is computed by a nondeterministic circuit of size s, then $\neg f$ is computed by a co-nondeterministic circuit of size s, and vice versa.

Definition 6.2 (SV nondeterministic circuits and machines). *A* single-valued (SV) nondeterministic circuit C *is an ordinary circuit with a single output gate, a single "flag" input* z*, and two sets of inputs:* $x_1, x_2, \ldots, x_n$ *and* $y_1, y_2, \ldots, y_m$*. A function* f_n *on* n *bits is computed by* C *if*

$$C(0,x,y) = 1 \Rightarrow C(1,x,y) = f(x),$$

and for all x, $\exists y$ $C(0,x,y) = 1$. The size of an SV circuit is the size of the underlying circuit C. We use $S_{SV}(f)$ to denote the smallest SV nondeterministic circuit that computes f.

A single-valued (SV) nondeterministic machine *M computing a function family $f = \{f_n\}$ is defined in the same way, with M replacing C above. We say that a function family f is computed in SV-nondeterministic time $t(n)$ if there is an SV nondeterministic machine that computes f and runs in time $t(n)$.*

Loosely speaking, when the when the first input of C is "zero", the output says whether the circuit "accepted" the "nondeterministic guess" $y_1, y_2, \ldots, y_m$. The requirement is that for every accepted nondeterministic guess, the circuit outputs the correct value when its first input is "1". Notice that if f is computed by a SV-nondeterministic circuit of size s, then $\neg f$ is also computed by a SV-nondeterministic circuit of size s. We also remark that the predicates computed in SV-nondeterministic polynomial time are precisely those in $NP \cap coNP$.

The relationship between functions computed by SV-nondeterministic circuits and functions computed by nondeterministic (and co-nondeterministic) circuits is somewhat tricky. It is believed that SV-nondeterministic circuits are weaker than nondeterministic (or co-nondeterministic) circuits (as otherwise $coNP \subseteq NP/poly$ and the polynomial time hierarchy collapses). Nevertheless, the next easy lemma shows that a hardness assumption for nondeterministic circuits is *equivalent* to a hardness assumption for SV-nondeterministic circuits. It is more convenient to work with SV-nondeterministic circuits because the set of functions they compute is closed under composition.

Lemma 6.3. *For a function f we denote $\tilde{f}(x,b) = \begin{cases} f(x) & b = 0 \\ \neg f(x) & b = 1 \end{cases}$*

1. *$f(x)$ computable by a size $\Theta(s)$ SV-nondeterministic circuit $\Leftrightarrow$ f computable by a size $\Theta(s)$ nondeterministic circuit* and *f computable by a size $\Theta(s)$ co-nondeterministic circuit*
2. *$f(x)$* not *computable by a size $\Theta(s)$ SV-nondeterministic circuit $\Rightarrow$ $\tilde{f}$* not *computable by a size $\Theta(s)$ nondeterministic circuit.*
3. *$f(x)$* not *computable by a size $\Theta(s)$ SV-nondeterministic circuit $\Rightarrow$ $\tilde{f}$* not *computable by a size $\Theta(s)$ co-nondeterministic circuit.*
4. *If $f \in E$ then $\tilde{f} \in E$.*

Proof. For the part (1), let $C(z,x,y)$ be a SV-nondeterministic circuit computing f. Then $C'(x,y) = C(0,x,y) \wedge C(1,x,y)$ is a nondeterministic circuit for f; similarly $C''(x,y) = \neg C(0,x,y) \vee C(1,x,y)$ is a co-nondeterministic circuit for f. In the other direction, given nondeterministic and co-nondeterministic circuits for f, $C'(x,y)$ and $C''(x,y)$, respectively, the circuit $C(z,x,y)$ defined by:

$$\begin{aligned} C(0,x,y) &= C'(x,y) \vee \neg C''(x,y) \\ C(1,x,y) &= C'(x,y) \end{aligned}$$

is a SV-nondeterministic circuit for f.

For part (2) we prove the contrapositive. Notice that if $C(x,b;y)$ is a nondeterministic circuit for $\tilde{f}$, then $C'(x,y) = C(x,0;y)$ is a nondeterministic circuit for f and $C''(x,y) = C(x,1;y)$ is a nondeterministic circuit for $\neg f$ which implies a co-nondeterministic circuit for f. Applying part (1), we obtain a SV-nondeterministic circuit for f.

The proof of part (3) is almost identical to the proof of part (2), and the part (4) is trivial. $\square$

6.3 Our results

We now define objects analogous to those in Section 5 for nondeterministic circuits.

Definition 6.4 (PRG against nondeterministic circuits). *An* ϵ-PRG for nondeterministic size s *is a function* $G : \{0,1\}^t \rightarrow \{0,1\}^m$ *such that for all size* s *nondeterministic circuits* C*:*

$$|\Pr[C(G(U_t)) = 1] - \Pr[C(U_m) = 1]| \leq \epsilon \tag{15}$$

As in the case of PRGs for deterministic circuits, by [Yao82] property (15) follows from the next property:[14]

$$\begin{aligned} &\forall \;\; 1 \leq i \leq m \text{ and all functions } f : \{0,1\}^{i-1} \rightarrow \{0,1\} \text{ with size } s - O(1) \\ &\qquad\qquad \text{nondeterministic or co-nondeterministic circuits,} \\ &\qquad\qquad\qquad\qquad \Pr[f(G(U_t)_{1\ldots i-1}) = G(U_t)_i] \leq \frac{1}{2} + \frac{\epsilon}{m}. \end{aligned} \tag{16}$$

Our construction and results translate to the nondeterministic setup with exactly the same parameters. The only thing we need to change is the proof; the additional arguments used for the nondeterministic setup are outlined in Section 6.4. We first state our results which are analogous to those in Sections 5.3 and 5.4.

Let x be a function $x : \{0,1\}^{\log n} \rightarrow \{0,1\}$ and let ${G'_x}^{(j)} : \{0,1\}^{d \log q + \log \bar{n}} \rightarrow \{0,1\}^m$ be the functions defined by (13) after applying the transformation described in Lemma 5.6 with $\rho = \epsilon/(8m)$ to each $G_x^{(j)}$ (using, e.g., the Hadamard code, so $\bar{n} = q$). The next theorem is analogous to Theorem 5.5.

Theorem 6.5 (PRG against nondeterministic circuits: main theorem). *There exists a universal constant* c *such that for every choice of* n, d, h, q *satisfying* $h^d > n$ *and the conditions of Lemma 5.4, at least one* ${G'_x}^{(j)}$ *is an* ϵ*-PRG against nondeterministic circuits of size* s*, provided that* $S_{SV}(x) > s \cdot poly(m, q)$ *and* $q > max(c(m/\epsilon)^4 hd^2 \log^2 q, 2d^4 log^2 q)$*. Furthermore, all the* ${G'_x}^{(j)}$*s are computable in time* $poly(q^d, n)$ *with oracle access to* x*.*

Note that Theorem 6.5 refers to the binary versions of the candidate generators whereas Theorem 5.5 refers to the q-ary versions. In the deterministic setup this makes no difference, however in the nondeterministic setup we do not know in general how to convert from q-ary to binary, and rely on particular properties of our construction.

By fixing the parameters in the same way as in Section 5.3, we obtain the following corollary, which is analogous to Corollary 5.7.

Corollary 6.6. *At least one* ${G'_x}^{(j)}$ *is a* $1/m$*-PRG against nondeterministic circuits of size* s *provided that* $S_{SV}(x) > sm^{O(1)}$*. Furthermore, all the* ${G'_x}^{(j)}$*'s are computable in (deterministic) time* $n^{O(1)}$*, given* x*.*

By using XOR to combine the generators as in Section 5.3, we obtain the following PRG against nondeterministic circuits. This corollary is analogous to Corollary 5.8.[15]

[14] The argument of [Yao82] converts a distinguishing function C that violates (15) into a predictor function f that violates (16). The argument shows that there exist constants $1 \leq i \leq m$, $a_i, \cdots, a_m \in \{0,1\}$ and $b \in \{0,1\}$ such that $f(x_1, \cdots, x_{i-1}) = C(x_1, \cdots, x_{i-1}, a_i, \cdots, a_m) \oplus b$. In the nondeterministic setting note that if C is computable by a size s nondeterministic then f is computable by a circuit of roughly the same size. Yet, if $b = 0$ then this circuit is nondeterministic, and if $b = 1$ then this circuit is a co-nondeterministic circuit. Thus, to obtain the relation between distinguishers and predictors we need to guarantee that the PRG fools both nondeterministic and co-nondeterministic predictor circuits.

[15] In the corollary above we allow both x and the generator to be computable in SV-nondeterministic time $n^{O(1)}$ rather than deterministic time $n^{O(1)}$. This is because the application we have in mind is derandomizing AM and in this setup the generator is run by a nondeterministic machine so we can allow it to be computable nondeterministically. However, the assumption that g is computable in SV-nondeterministic time $n^{O(1)}$ could be replaced by "g is computable in (deterministic) time $n^{O(1)}$". This is a stronger assumption, and it "buys" a stronger conclusion: the generator will run in (deterministic) time $n^{O(1)}$.

Corollary 6.7. *For any s, if there exists a function $g : \{0,1\}^{\log n} \to \{0,1\}$ that is computable in SV-nondeterministic time $n^{O(1)}$ with $S_{SV}(g) \geq s$ then there exists a $1/m$-PRG against nondeterministic circuits of size m with seed length $t = O(\log^2 n/\log s)$ and output length $m = s^{\Omega(1)}$. Furthermore, this generator can be computed in SV-nondeterministic time $n^{O(1)}$.*

As noted above, the notions of one-sided error and two-sided error coincide for AM and so hitting set generators against co-nondeterministic circuits suffice to derandomize it.

Definition 6.8 (HSG against nondeterministic circuits). *A function $H : \{0,1\}^t \to \{0,1\}^m$ is an ϵ-*HSG for nondeterministic size s *if for all size s nondeterministic or co-nondeterministic circuits $C : \{0,1\}^m \to \{0,1\}$*

$$\Pr[C(U_m) = 1] \geq \epsilon \Rightarrow \Pr[C(H(U_t)) = 1] > 0.$$

Combining the candidate generators into an HSG as in Section 5.4 gives the following corollary, analogous to Corollary 5.10.

Corollary 6.9. *For any s, if there exists a function $g : \{0,1\}^{\log n} \to \{0,1\}$ that is computable in SV-nondeterministic time $n^{O(1)}$ with $S_{SV}(g) \geq s$ then there exists a $1/m$-HSG against nondeterministic circuits of size m with seed length $t = O(\log n)$ and output length $m = s^{\Omega(1)}$. Furthermore, this generator can be computed in SV-nondeterministic time $n^{O(1)}$.*

Finally, the HSG can be used to derandomize AM, giving the following optimal tradeoff (compare to Corollary 5.11):

Corollary 6.10. *If there exist a function family $g = \{g_\ell\} \in NE \cap coNE$ that requires size $s(\ell)$ SV nondeterministic circuits, then for every time constructible function $t(n)$, AMTIME$(t(n)) \subseteq NTIME(2^{O(s^{-1}(t(n)^{O(1)}))})$.*

This extends the previous results by Miltersen and Vinodchandran [MV99] to general $s(\ell)$.

6.4 Proof of the main theorem for nondeterministic circuits

In this section we prove Theorem 6.5. The proof follows the outline of the proof of Theorem 5.5. We first explain why we need to modify the proof of Theorem 5.5 for it to work in the nondeterministic setting.

The proof of Theorem 5.5 constructs a small (deterministic) circuit C that computes x when given (deterministic) circuits that compute the predictors $f^{(j)}$'s. In the nondeterministic setting each one of the $f^{(j)}$'s has either a small nondeterministic circuit or a small co-nondeterministic circuit, and we want to construct a small SV nondeterministic circuit C computing x. Suppose that $f^{(j)}$ is a nondeterministic circuit, and that in the course of its computation, C wishes to evaluate $f^{(j)}$ on input a. If $f^{(j)}(a) = 1$ then there is a short proof that shows this, and C can use this short proof to justify its computation. However, if $f^{(j)}(a) = 0$ we cannot assume that there is a short proof of this fact. In this case, C cannot evaluate $f^{(j)}(a)$ as part of its computation. More generally, the problem is that an SV nondeterministic circuit cannot use a nondeterministic circuit as a black box.

To solve this problem we would like to find a short proof that $f^{(j)}(a) = 0$. We will use the fact that at each prediction step, C runs the predictor $f^{(j)}$ on points a that are on a random curve[16]. With high probability the fraction of a's on the curve for which $f^{(j)}(a) = 1$ and the fraction p of a's in F^d on which $f^{(j)}(a) = 1$ differ by at most some small δ. If C verifies that a $(p - \delta)$ fraction of the a's on the curve have

[16]This presentation is oversimplified and confuses between binary predictors and q-ary predictors. The actual proof deals with this problem.

$f^{(j)}(a) = 1$ (which can be done within an SV nondeterministic computation) then it can be sure that almost all remaining a's on the curve have $f^{(j)}(a) = 0$. In particular, if C assumes that *all* remaining a's have $f^{(j)}(a) = 0$ then it agrees with $f^{(j)}$ on at least a $(1-2\delta)$ fraction of the points in the curve. Thus we can view C as having access to a predictor that makes slightly more errors than $f^{(j)}$. By choosing the parameters appropriately C can perform the list-decoding phase even with slightly more errors, and the proof goes on essentially unchanged. We also need to provide p to C as non-uniform advice.

In the remainder of the section we formally prove Theorem 6.5. Recall that the "candidate PRGs" $G_x'^{(j)} : F^d \times [\bar{n}] \to \{0,1\}^m$ have been obtained from the $G_x^{(j)}$ defined in (13) using an (ρ, ρ^{-2})-efficiently list-decodable error-correcting code, with encoding function $E : \{0,1\}^{\log q} \to \{0,1\}^{\bar{n}}$, and that $\rho = \epsilon/(8m)$. (For clarity, we are splitting the seed of the $G_x'^{(j)}$ into two parts – the first being the vector in F^d and the second being the index into the codeword of the error-correcting code). We assume for the purpose of contradiction that no $G_x'^{(j)}$ is an ϵ-PRG against nondeterministic circuits. Then we have next-bit predictors for all $G_x'^{(j)}$. Without loss of generality we assume that all these predictors predict the last bit given $m-1$ previous bits. That is, we assume that there exist functions $f^{(j)} : \{0,1\}^{m-1} \to \{0,1\}$ violating property (16) for each $G_x'^{(j)}$, respectively. Furthermore, each of these predictor functions can be implemented by a size s nondeterministic or co-nondeterministic circuit.

Using these predictors, we produce a small SV nondeterministic circuit that computes $x(i)$ from input i. Our algorithm here differs from the algorithm used in the proof of Theorem 5.5 *only* by some additional steps at the beginning of **Learn Next Curve**, and a corresponding modification of the proof of Lemma 5.12. For clarity we present the modified version of **Learn Next Curve** in its entirety; the new actions are marked with "+".

Let $p_0^{(j)} = \Pr_{y \in F^d, z \in [\bar{n}]}[f^{(j)}(G_x'^{(j)}(y,z)_{1\ldots m-1}) = 0]$ be the fraction of points on which the j-th predictor predicts 0, and let $p_1^{(j)} = 1 - p_0^{(j)}$ be the fraction of points on which it predicts 1. The new procedure will make use of values $n_j \in [q\bar{n}]$ and $\beta_j \in \{0,1\}$ for $j = 0, 1, \ldots, d-1$ defined as follows:

$$\beta_j = \begin{cases} 1 & \text{if } f^{(j)} \text{ is computed by a size } s \text{ nondeterministic circuit} \\ 0 & \text{if } f^{(j)} \text{ is computed by a size } s \text{ co-nondeterministic circuit} \end{cases} \tag{17}$$

$$n_j = \left\lceil q\bar{n}\left(p_{\beta_j}^{(j)} - \frac{\epsilon}{4m}\right)\right\rceil. \tag{18}$$

These values will ultimately be supplied as non-uniform advice to the circuit. For all j, let $D^{(j)}$ be the nondeterministic or co-nondeterministic circuit computing $f^{(j)}$.

Procedure Nondeterministic Learn Next Curve

- Input:
  - next curve $C : F \to F^d$: a degree v polynomial
  - reference points $R \subseteq F$: a set of elements of F
  - stride j: an integer in $[0 \ldots (\ell-1)]$
  - input evaluations $\{a_t^i\}_{t \in F, i \in [1\ldots(m-1)]}$ and $\{b_t\}_{t \in R}$: elements of F whose *intended values* are $a_t^i = \hat{x}(A^{-iq^j}C(t))$ and $b_t = \hat{x}(C(t))$.
- Output:
  - output evaluations $\{c_t\}_{t \in F}$: elements of F whose *intended values* are $c_t = \hat{x}(C(t))$.

- Action:
  + Guess a set T of n_j distinct pairs $(t_i, z_i) \in F \times [\bar{n}]$ and a "witness" string w_i for each.
  + Check that this is a "good guess"; i.e.,

$$\forall (t_i, z_i) \in T \quad f^{(j)}(E(a_{t_i}^{m-1})_{z_i}, E(a_{t_i}^{m-2})_{z_i}, \ldots E(a_{t_i}^{1})_{z_i}; w_i) = \beta_j.$$

    If it is not, halt and output "bad guess."
  + For all $t \in F$ and all $z \in [\bar{n}]$, set $r_z^t = \begin{cases} \beta_j & \text{if } (t,z) \in T \\ 1-\beta_j & \text{otherwise} \end{cases}$.
  + For all $t \in F$, set S_t to be the list of ρ^{-2} codewords that differ from r^t in at most $(1/2-\rho)\bar{n}$ places.
  - Apply Lemma 3.1 on the $q\rho^{-2}$ pairs $\{(t,e)\}_{t\in F, e\in S_t}$ to obtain a list of at most $8\rho^{-3}$ degree $\deg(\hat{x})v$ univariate polynomials $p(t)$ that contains all polynomials for which $p(t) \in S_t$ for at least $\rho q/4$ values of t. If this list is empty, fail.
  - If the list contains a unique polynomial $p(t)$ for which $p(t) = b_t$ for all $t \in R$, output $\{p(t)\}_{t\in F}$; otherwise fail.

We say that **Nondeterministic Learn Next Curve** *succeeds* (on a curve, reference points and stride) if its output evaluations are the intended values when its input evaluations are the intended values *for all "good" guesses, and there is at least one such "good" guess*. As in the proof of Theorem 5.5, we argue that for a random next curve C and a random set of reference points $R \subseteq F$, the procedure succeeds with high probability.

Lemma 6.11. *For all strides j,*

$$\Pr_{C,R}[\textbf{Nondeterministic Learn Next Curve } \textit{succeeds}] \geq 1 - O(2^{-v/2}) - (8\rho^{-3})\left(\frac{v \cdot deg(\hat{x})}{q}\right)^r,$$

where $C : F \to F^d$ is a uniformly chosen degree $v \leq \left(\frac{\epsilon}{4m}\right)^2 (q/4)$ curve, and $R \subseteq F$ is a uniformly chosen subset of F of size r.

Proof. Fix j, and suppose that the input evaluations are the intended values. We first argue that the fraction of points on which $f^{(j)}$ predicts β_j along curve C is close the fraction of points on which $f^{(j)}$ predicts β_j in the whole space. Define the random variable

$$X_t = \Pr_{z\in[\bar{n}]}[f^{(j)}(E(a_t^{m-1})_z, E(a_t^{m-2})_z, \ldots E(a_t^1)_z) = \beta_j],$$

and let $X = \sum_{t\in F} X_t$. Notice that $E[X] = qp_{\beta_j}^{(j)}$. By Lemma 3.2, we have:

$$\Pr[|X - E[X]| \geq \frac{\epsilon}{4m}q] \leq O\left(2^{-v/2}\right). \tag{19}$$

This implies that with probability at least $1 - O\left(2^{-v/2}\right)$,

$$\bar{n}q\left(p_{\beta_j}^{(j)} - \frac{\epsilon}{4m}\right) \leq \left|\left\{(t,z) : f^{(j)}(E(a_t^{m-1})_z, \ldots E(a_t^1)_z) = \beta_j\right\}\right| \leq \bar{n}q\left(p_{\beta_j}^{(j)} + \frac{\epsilon}{4m}\right) \tag{20}$$

By integrality, the size of the set in the above inequality is at least n_j. If the "bad" event in (19) does not occur, two important observations hold: (1) there exists at least one "good guess" of T and the witness strings w_i, and (2) for any such "good guess,"

$$\Pr_{t,z}[r_z^t = f^{(j)}(E(a_t^{m-1})_z, \ldots E(a_t^1)_z)] \geq 1 - \frac{\epsilon}{2m},$$

since the set T identifies all but an $\epsilon/(2m)$ fraction of the points on which $f^{(j)}$ predicts β_j.

Now we argue that $f^{(j)}$ is correct along curve C on almost the same fraction of points as the fraction of points in the whole space on which $f^{(j)}$ is correct. Define the random variable

$$Y_t = \Pr_{z \in [\bar{n}]}[f^{(j)}(E(a_t^{m-1})_z, \ldots, E(a_t^1)_z) = E(\hat{x}(C(t)))_z],$$

and define $Y = \sum_{t \in F} Y_t$. Notice that $E[Y] > q(1/2 + \epsilon/m)$ since we are assuming that $f^{(j)}$ violates property (16). By Lemma 3.2, we have:

$$\Pr[|Y - E[Y]| \geq \frac{\epsilon}{4m} q] \leq O\left(2^{-v/2}\right). \tag{21}$$

If neither the event in (19) nor the event in (21) occurs, then we have:

$$\Pr_{t,z}[r_z^t = E(\hat{x}(C(t)))_z] \geq \frac{1}{2} + \frac{\epsilon}{4m}.$$

By an averaging argument we have that for at least an $\epsilon/(8m)$ fraction of the t's, $\Pr_z[r_z^t = E(\hat{x}(C(t)))_z] \geq 1/2 + \epsilon/(8m)$. For these t, the relative Hamming distance between r^t and $E(\hat{x}(C(t)))$ is at most $1/2 - \rho$, so S_t contains $\hat{x}(C(t))$. The remainder of the proof of Lemma 5.12 now goes through unchanged. $\Box$

The remainder of the proof of Theorem 5.5 (following Lemma 5.12) goes through unchanged. Lemma 5.14 now shows that there exist curves C_1 and C_2 for which **Nondeterministic Learn Next Curve** succeeds on all steps. In the present context, this also means that on these curves **Nondeterministic Learn Next Curve** is a bone fide SV-nondeterministic "subroutine" (on "bad" curves there may be no "good guess," violating the requirements of Definition 6.2). Using this procedure repeatedly to compute $x(i)$ as in the proof of Theorem 5.5 results in an SV-nondeterministic circuit computing x of size $s \cdot \text{poly}(m, q, \bar{n})$, a contradiction. This concludes the proof of Theorem 6.5.

7 Hardness amplification for deterministic and nondeterministic circuits

A critical component of previous PRG constructions has been *hardness amplification*. Hardness amplification is an efficient transformation that takes a function $x : \{0,1\}^{t=\log n} \to \{0,1\}$ that is worst-case hard for size s circuits, and produces a function $x' : \{0,1\}^{t'} \to \{0,1\}$ that cannot be computed correctly on even a $1/2 + \epsilon$ fraction of its inputs by size s' circuits. One hopes for t' not much larger than t, and s' not much smaller than s.

Our deterministic and nondeterministic PRG constructions are in fact hardness amplification transformations when their output is truncated after 1 bit. For example in the deterministic case, starting with a function $x : \{0,1\}^{t=\log n} \to \{0,1\}$ for which $S(x) > s$, let $G : \{0,1\}^{t'=O(\log^2 n/\log s)} \to \{0,1\}$ be the ϵ-PRG built from x in Corollary 5.8, whose output is truncated after 1 bit. We claim that G cannot be computed correctly on even a $1/2 + \epsilon$ fraction of its inputs by size $s' = s^{\Omega(1)}$ circuits. If there was such a circuit

P, then it would constitute a predictor for the function $\hat{G}(y) = y \circ G(y)$. Standard minor modifications to the proof that G is an ϵ-PRG reveal that just as with a predictor for G, a predictor for $\hat{G}$ can be used to construct a small circuit computing x correctly on every input, and contradicting the hardness of x.[17]

Currently, the hardness amplification transformation for deterministic circuits that achieves the best parameters is [STV01]; they obtain $s' = s^{\Omega(1)}$ and $t' = O(t)$. The construction sketched above fails to match these parameters because our t' may be as large as t^2. However, in this particular setting, we can get away *without* XORing the candidate binary PRGs $G_x'^{(i)}$ (from Section 5.3) to obtain G. In fact, taking $G = G_x'^{(0)}$ is sufficient for the following reason. If G is not average-case hard, then the predictor P we must use to obtain a contradiction has (by definition):

$$\Pr_{y,j}[P(y,j) = C(\hat{x}(Ay))_j] > \frac{1}{2} + \epsilon.$$

Defining $P^{(i)}(y,j) = P(A^{q^i-1}y, j)$ we obtain similar predictors for all $G_x'^{(i)}$ from the single predictor P. Using these predictors, the proof arrives at a contradiction as before. Since $G_x'^{(0)}$ has seed length $t' = O(\log n)$, this hardness amplification transformation essentially matches the parameters of [STV01] for deterministic circuits.

For nondeterministic circuits, we obtain a new result. Hardness amplification transformations were known for circuits with SAT oracle gates [KvM02], using the fact that known deterministic hardness amplification transformations (e.g., [STV01]) relativize. However, no such transformations were known that transform worst-case hardness for nondeterministic circuits (Definition 6.1) into average-case hardness for nondeterministic circuits. Our PRG against nondeterministic construction gives the first hardness amplification transformation for nondeterministic circuits. The argument is the same as the one for deterministic circuits outlined above. For clarity we state the result for nondeterministic circuits in the following theorem.

Theorem 7.1. *For every function $g : \{0,1\}^t \to \{0,1\}$ such that $S_{SV}(g) \geq s$ there is a function $h : \{0,1\}^{O(t)} \to \{0,1\}$ such that for every nondeterministic or co-nondeterministic circuit C of size $s' = s^{\Omega(1)}$ computing a function $f : \{0,1\}^{O(t)} \to \{0,1\}$:,*

$$\Pr_y[f(y) = h(y)] \leq \frac{1}{2} + \frac{1}{s'}.$$

Furthermore, h can be computed in (deterministic) time $2^{O(t)}$ given oracle access to g.

We remark that in a subsequent work [SU04] we show how to apply the argument of [STV01] *directly* for the nondeterministic case. This gives a simpler and more modular proof of the theorem above.

8 Constructions of "traversing matrices"

In the previous sections we used matrices A and B with entries in $F = GF(q)$ that "traverse" subsets of F^d as components in our extractor and PRG constructions. In this section we show how to construct such matrices and prove Lemma 4.4, Lemma 4.18 and Lemma 5.4. Throughout the section, we will need a representation of the field $GF(q)$ in order to perform field arithmetic. For our purposes it is sufficient that when $q = p^c$ for a prime p, such a representation can be found deterministically in time poly(p, c) [Sho90].

[17] The literature refers to such a PRG as a "strong" PRG, and many PRG constructions in fact produce strong PRGs.

The basic idea is to view the vector space $F^d = GF(q)^d$ as the extension field $GF(q^d)$ and use the additional multiplicative structure to obtain the matrices. We fix a basis for $GF(q^d)$ as a vector-space over F, and let g be a generator for the multiplicative group $GF(q^d)^*$. The function $T : F^d \to F^d$ that given a vector $\vec{v}$ interprets it as a field element v and outputs $g \cdot v$ is an invertible linear transform, and we can pick A to be the matrix such that $Av = T(v)$. We start by restating and proving Lemma 4.4

Lemma 4.4 (restated). *Let F be a field with q elements and let d be an integer. There exists an invertible $d \times d$ matrix A with entries in F such that A^{q^d-1} is the identity matrix and for every non-zero $\vec{v} \in F^d$, $\left\{A^i\vec{v}\right\}_{1\le i<q^d} = F^d \setminus \{0\}$. Furthermore, such an A can be found in time $q^{O(d)}$.*

Proof. (of Lemma 4.4) The field $GF(q^d)$ is a vector space of dimension d over $F = GF(q)$ and is thus isomorphic to F^d. Let g be a generator of the multiplicative group of $GF(q^d)$ (which is cyclic). Multiplication with a fixed element in the field corresponds to a linear transform in the vector-space, so the linear transform A corresponding to multiplication by g satisfies (6). We can find g by exhaustive search[18] in time $q^{O(d)}$. □

Our construction of extractors for small error required a more sophisticated version which is restated below.

Lemma 4.18 (restated). *Let h, q and d be such that: h is a prime power, q is a power of h, and d and $\log_h q$ are relatively prime. Then there exists an invertible $d \times d$ matrix B with entries from $F = GF(q)$, and a set $H \subseteq F$ with $|H| = h$ such that B^{h^d-1} is the identity matrix and for every nonzero $v \in F^d$ there is an invertible linear transform $T_v : F^d \to F^d$ for which:*

$$T_v \cdot (H^d \setminus \left\{\vec{0}\right\}) = \left\{B^i v | 1 \le i < h^d\right\}.$$

Moreover, B can be found in time poly$(h^d, \log q)$.

The proof of Lemma 4.18 uses the proof technique of the following more general Lemma which we used for our PRG construction.

Lemma 5.4 (restated). *Let h, q and d be such that: h is a prime power, q is a power of h, and d and $\log_h q$ are relatively prime. Let F be the field with q elements and H be the subfield of F with h elements. Then there exist invertible $d \times d$ matrices A and B with entries from F that satisfy:*

- *A^{q^d-1} and B^{h^d-1} are the identity matrix.*
- *For any non-zero vector $\vec{v} \in F^d$: $\left\{A^i\vec{v}\right\}_{1\le i<q^d} = F^d \setminus \{0\}$*
- *For any non-zero vector $\vec{v} \in H^d$ $\left\{B^i\vec{v}\right\}_{1\le i<h^d} = H^d \setminus \{0\}$.*
- *$B = A^{(q^d-1)/(h^d-1)}$.*
- *A, B can be found in time $q^{O(d)}$.*

[18]More precise bounds, and significant improvements in certain cases can be found in [Sho92] and [Shp96].

Proof. (of Lemma 5.4) We first need a polynomial of degree d *with coefficients in* H that is irreducible over F. Let $c = \log_h q$ and let $p(z)$ be a polynomial of degree d that is irreducible over $H = GF(h)$. Let α be a root of p and notice that the field $GF(h^c)[\alpha]$ contains both F and $GF(h^d)$. Furthermore, $GF(h^c)[\alpha]$ is contained in $GF(h^{cd})$. However, no *proper* subfield of $GF(h^{cd})$ can contain both $F = GF(h^c)$ and $GF(h^d)$ because c and d are relatively prime. Therefore $GF(h^c)[\alpha] = GF(h^{cd})$, which implies that p is irreducible over $F = GF(h^c)$, as desired. We now construct $GF(q^d)$ by considering its elements to be polynomials over F modulo $p(z)$. Let $\left\{1, z, z^2, \ldots z^{d-1}\right\}$ be the standard basis for $GF(q^d)$ over F. In this basis, the set H^d is exactly the following subset of $GF(q^d)$:

$$\left\{\sum_{i=0}^{d-1} \beta_i z^i | \beta_i \in H\right\}.$$

Since $p(z)$ has all coefficients in H, this subset is closed under multiplication. It follows that H^d is isomorphic to $GF(h^d)$. Moreover, $H^d \setminus \{0\}$ is a subgroup of order $h^d - 1$ of the multiplicative group of $GF(F^d)$. Therefore, if we find A corresponding to a generator of the multiplicative group of $GF(F^d)$ (as in Lemma 4.4), then $B = A^{(q^d-1)/(h^d-1)}$ generates the unique subgroup of order $h^d - 1$, whose elements are $H^d \setminus \{0\}$. As before A can be found by exhaustive search. □

We now prove Lemma 4.18:

Proof. (of Lemma 4.18) We use the technique of Lemma 5.4. We have shown that under the conditions of the lemma, there is a polynomial p of degree d with coefficients in H that is irreducible over F. Such a polynomial can be found in time $h^{O(d)}$ by exhaustive search. We construct $GF(q^d)$ using p, and then as noted above the set H^d is a subfield. We let B be a matrix (with entries in H) that corresponds to multiplying by a generator of the multiplicative group of this subfield. Such a matrix can be found (by exhaustive search for a generator of the multiplicative group) in time $h^{O(d)}$.

Finally, let A be a matrix that corresponds to multiplying by a generator of the multiplicative group of $GF(q^d)$. We have $v = A^j\vec{1}$ for some j, since A generates $F^d \setminus \{0\}$. Note that A and B commute (since B is a power of A), and that $\vec{1}$ (the all-ones vector) is in H^d. Using these two facts, we have:

$$\left\{B^i v | 1 \le i \le h^d\right\} = \left\{B^i A^j \vec{1} | 1 \le i \le h^d\right\} = \left\{A^j B^i \vec{1} | 1 \le i \le h^d\right\} = A^j \cdot (H^d \setminus \left\{\vec{0}\right\}).$$

□

9 Acknowledgments

We thank Henry Cohn, Venkat Guruswami, Valentine Kabanets, Omer Reingold, Muli Safra, Amnon Ta-Shma, Salil Vadhan, Avi Wigderson and David Zuckerman for helpful discussions. We are especially grateful to the authors of [TSZS01] for explaining their result to us. We thank the anonymous referees for their very detailed comments and suggestions that improved the presentation significantly.

References

[ACR98] A. E. Andreev, Andrea E. F. Clementi, and J. D. P. Rolim. A new general derandomization method. *Journal of the ACM*, 45(1):179–213, January 1998.

[ACRT99] A. E. Andreev, A. E. F. Clementi, J. D. P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28(6), 1999.

[AK97] V. Arvind and J. Köbler. On resource-bounded measure and pseudorandomness. In *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 235–249, 1997.

[Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, 1985.

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In *Theoretical aspects of computer science, 16th annual symposium*, 1999.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[BM88] L. Babai and S. Moran. Arthur-merlin games: A randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[BR94] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994.

[FGM+89] M. Furer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos. On completeness and soundness in interactive proof systems. In S. Micali, editor, *Randomness and Computation*, pages 429–442, Greenwich, Connecticut, 1989. Advances in Computing Research, vol. 5, JAI Press.

[GL89] O. Goldereich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[Gol98] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, Algorithms and Combinatorics, 1998.

[GS00] V. Guruswami and M. Sudan. List decoding algorithms for certain concatenated codes. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 2000.

[GS01] V. Guruswami and M. Sudan. Extensions to the Johnson bound. Manuscript, February 2001.

[GVW00] O. Goldreich, S. Vadhan, and A. Wigderson. Simplified derandomization of BPP using a hitting set generator. Technical Report TR00-004, Electronic Colloquium on Computational Complexity, January 2000.

[GZ97] O. Goldreich and D. Zuckerman. Another proof that BPP subseteq PH (and more). Technical Report TR97-045, Electronic Colloquium on Computational Complexity, 1997.

[Imp95] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 181–190, 1999.

[ISW03] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Reducing the seed length in the nisan-wigderson generator. Manuscript, a preliminary version appeared in STOC00 under the title "extractors and pseudorandom generators with optimal seed length, 2003.

[IW97] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[Kab02] V. Kabanets. Derandomization: a brief overview. *Bulletin of the European Association for Theoretical Computer Science*, 76:88–103, 2002.

[KvM02] A. R. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31:1501–1526, 2002.

[LRVW03] C. J. Lu, Omer Reingold, S. Vadhan, and A. Wigderson. Extractors: Optimal up to constant factors. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 2003.

[MV99] P. B. Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 71–80, 1999.

[NTS99] N. Nisan and A. Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58:148–173, 1999.

[NW94] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[RRV99] R. Raz, O. Reingold, and S. Vadhan. Error reduction for extractors. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[RRV02] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. *JCSS: Journal of Computer and System Sciences*, 65, 2002.

[RSW00] 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*, 2000.

[RTS00] J. Radhakrishnan and A. Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, February 2000.

[RZ98] A. Russell and D. Zuckerman. Perfect-information leader election in $\log^* n + O(1)$ rounds. *Journal of Computer and System Sciences*, 1998. To appear. Preliminary version in *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 576–583.

[Sha02] R. Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

[Sho90] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. *Mathematics of Computation*, 54:435–447, 1990.

[Sho92] V. Shoup. Searching for primitive roots in finite fields. *Mathematics of Computation*, 58:369–380, 1992.

[Shp96] I. Shparlinski. On finding primitive roots in finite fields. *Theoretical Computer Science*, 157:273–275, 1996.

[Sip88] M. Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36(3):379–383, 1988.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the xor lemma. *JCSS: Journal of Computer and System Sciences*, 62, 2001.

[SU04] R. Shaltiel and C. Umans. Pseudorandomness for approximate counting and sampling. Technical Report TR04-086, Electronic Colloquium on Computational Complexity (ECCC), 2004.

[Sud97] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13, 1997.

[SZ99] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM Journal on Computing*, 28(4):1433–1459, August 1999.

[Tre02] L. Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2002.

[TS96] A. Ta-Shma. On extracting randomness from weak random sources. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 276–285, 1996.

[TSUZ01] A. Ta-Shma, C. Umans, and D. Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 143–152, 2001.

[TSZ04] A. Ta-Shma and D. Zuckerman. Extractor codes. *IEEE Transactions on Information Theory*, 50(12):3015–3025, 2004.

[TSZS01] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller codes. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001.

[Uma99] C. Umans. Hardness of approximating Σ_2^p minimization problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 465–474, 1999.

[Uma02] C. Umans. Pseudo-random generators for all hardnesses. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 627–634, 2002.

[WZ99] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. *Combinatorica*, 19(1):125–138, 1999.

[Yao82] A. C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[Zuc96] D. Zuckerman. On unapproximable versions of NP-complete problems. *SIAM Journal on Computing*, 25:1293–1304, 1996.

[Zuc97] D. Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11:345–367, 1997.