Low-end Uniform Hardness vs. Randomness Tradeoffs for AM

Ronen Shaltiel*
Department of Computer Science
University of Haifa
Mount Carmel, Haifa 31905, Israel
ronen@haifa.ac.il

Christopher Umans†
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125
umans@cs.caltech.edu

ABSTRACT

In 1998, Impagliazzo and Wigderson [18] proved a hardness vs. randomness tradeoff for BPP in the *uniform setting*, which was subsequently extended to give optimal tradeoffs for the full range of possible hardness assumptions by Trevisan and Vadhan [29] (in a slightly weaker setting). In 2003, Gutfreund, Shaltiel and Ta-Shma [11] proved a uniform hardness vs. randomness tradeoff for AM, but that result only worked on the "high-end" of possible hardness assumptions.

In this work, we give uniform hardness vs. randomness tradeoffs for AM that are near-optimal for the full range of possible hardness assumptions. Following [11], we do this by constructing a hitting-set-generator (HSG) for AM with "resilient reconstruction." Our construction is a recursive variant of the Miltersen-Vinodchandran HSG [24], the only known HSG construction with this required property. The main new idea is to have the reconstruction procedure operate implicitly and locally on superpolynomially large objects, using tools from PCPs (low-degree testing, self-correction) together with a novel use of extractors that are built from Reed-Muller codes [28, 26] for a sort of locally-computable error-reduction.

As a consequence we obtain gap theorems for AM (and AM ∩ coAM) that state, roughly, that either AM (or AM ∩ coAM) protocols running in time $t(n)$ can simulate all of EXP ("Arthur-Merlin games are powerful"), or else all of AM (or AM ∩ coAM) can be simulated in nondeterministic time $s(n)$ ("Arthur-Merlin games can be derandomized"), for a near-optimal relationship between $t(n)$ and $s(n)$. As in [11], the case of AM ∩ coAM yields a particularly clean theorem that is of special interest due to the wide array of cryptographic and other problems that lie in this class.

Categories and Subject Descriptors: F.2.3 [Theory of Computation]: Tradeoffs Between Complexity Measures

General Terms: Theory, Algorithms

Keywords: Arthur-Merlin games, hardness vs. randomness tradeoff, derandomization, hitting-set generator

*Supported by BSF grant 2004329.

†Supported by NSF CCF-0346991, BSF 2004329, a Sloan Research Fellowship, and an Okawa Foundation research grant.

STOC'07, June 11–13, 2007, San Diego, California, USA.

1. INTRODUCTION

A fundamental question of complexity theory concerns the power of randomized algorithms: Is it true that every randomized algorithm can be simulated deterministically with small (say, subexponential) slowdown? Ideally, is a polynomial slowdown possible – i.e., is BPP = P? The analogous question regarding the power of randomness in Arthur-Merlin protocols is: Is it true that every Arthur-Merlin protocol can be simulated by a nondeterministic machine with small slowdown? Is a polynomial slowdown possible – i.e., does AM = NP? We refer to efforts to answer the first set of questions positively as "derandomizing BPP" and efforts to answer the second set of questions positively as "derandomizing AM". Recent work [13, 20] has shown that derandomizing BPP or AM entails proving certain circuit lower bounds that currently seem well beyond our reach.

The hardness versus randomness paradigm

An influential line of research initiated by [7, 33, 25] tries to achieve derandomization *under the assumption* that certain hard functions exist, thus circumventing the need for proving circuit lower bounds. More precisely, we will work with hardness assumptions concerning the circuit complexity of functions computable in exponential time[1]. Derandomizing BPP can be done with lower bounds against size $s(\ell)$ deterministic circuits while derandomizing AM typically requires lower bounds against size $s(\ell)$ *nondeterministic* circuits, where ℓ is the input length of the hard function. Naturally, stronger assumptions – higher values of $s(\ell)$ – give stronger conclusions, i.e., more efficient derandomization. There are two extremes of this range of tradeoffs: In the "high end" of hardness assumptions one assumes hardness against circuits of very large size $s(\ell) = 2^{\Omega(\ell)}$ and can obtain "full derandomization," i.e., BPP = P [17] or AM = NP [24]. While in the "low-end" one assumes hardness against smaller circuits of size $s(\ell) = \mathrm{poly}(\ell)$ and can conclude "weak derandomization," i.e., simulations of BPP

[1]This type of assumption was introduced by [25] whereas the initial papers [7, 33] relied on cryptographic assumptions. In this paper we are interested in derandomizing AM which cannot be achieved by the "cryptographic" line of hardness versus randomness tradeoffs.

(resp. AM) that run in subexponential deterministic (resp. nondeterministic subexponential) time [4, 26]. Today, after a long line of research [25, 4, 12, 17, 2, 21, 24, 14, 15, 26, 30, 31] we have optimal hardness versus randomness tradeoffs for both BPP and AM that achieve "optimal parameters" in the *non-uniform* setting (see the discussion of non-uniform vs. uniform below).

Pseudorandom generators and hitting set generators

The known hardness versus randomness tradeoffs are all achieved by constructing a *pseudorandom generator* (PRG). This is a deterministic function G which on input m, produces a small set of T m-bit strings in time poly(T), with the property that a randomly chosen string from this set cannot be efficiently distinguished from a uniformly chosen m-bit string. In this paper we are interested in a weaker variant of a pseudorandom generator called a *hitting set generator* (HSG). A function G is a HSG against a family of circuits on m variables, if any circuit in the family which accepts at least $1/3$ of its inputs also accepts one of the m-bit output strings of G (when run with input m).[2] It is standard that given a HSG against deterministic (resp. co-nondeterministic) circuits of size poly(m) one can derandomize RP (resp. AM) in time poly(T) by simulating the algorithm (resp. protocol) on all strings output by the HSG, and accepting if at least one of the runs accepts[3].

The proofs of the aforementioned hardness versus randomness tradeoffs are all composed of two parts: first, they give an efficient way to generate a set of strings (the output of the PRG or HSG) when given access to some function f. Second, they give a *reduction* showing that if the intended derandomization using this set of strings fails, then the function f can be computed by a small circuit, which then contradicts the initial hardness assumption when taking f to be the characteristic function of an EXP complete problem. We now focus on the reduction part. An easy first step is that an input x (to the randomized algorithm or AM protocol) on which the intended derandomization fails gives rise to a small circuit D_x that "catches" the generator, i.e., D_x accepts at least $1/3$ of its inputs, but none of the strings in the generator output. (The obtained circuit D_x is a deterministic circuit when attempting to derandomize BPP and a co-nondeterministic circuit when attempting to derandomize AM). The main part of all the proofs is to then give a reduction that transforms this circuit D_x into a small circuit C that computes f.

Uniform hardness versus randomness tradeoffs

All the aforementioned hardness versus randomness tradeoffs are *nonuniform tradeoffs* because the reduction in the proof is nonuniform: given D_x it only shows the existence of a small circuit C that computes f, but doesn't give an efficient uniform procedure to produce it. (In other words, the reduction relies on nonuniform advice when transforming D_x into C). We remark that all the aforementioned results are "fully black-box" (meaning that they do not use any properties of the hard function f or circuit D_x) and it was shown in [29] that any hardness versus randomness tradeoff that is "fully black box" cannot have a uniform reduction.

A *non-black box* uniform reduction for derandomizing BPP in the low-end was given in [18]. This reduction gives a *uniform* randomized poly-time algorithm (sometimes called a *reconstruction algorithm*) for transforming a circuit D_x that catches the generator into a circuit C that computes the function f. It follows that if the intended derandomization fails, and if furthermore one can *feasibly generate* an input x on which it fails (by a uniform computation), then one can use the uniform reduction construct the circuit C in probabilistic polynomial time, which in turn implies that f is computable in BPP. (This should be compared to the non-uniform setting in which one would get that f is in P/*poly*). An attractive feature of this result is that it can be interpreted as a (low-end) *gap theorem* for BPP that asserts the following: Either randomized algorithms are somewhat weak (in the sense that they can be simulated deterministically in subexponential time on feasibly generated inputs) or else they are very strong (in the sense that they can compute any function in EXP).[4] Obtaining a high-end version of this result is still open. In [29] it was shown how to get a high-end tradeoff in the slightly weaker setting where the hard function f is computable in PSPACE rather than EXP.

Uniform hardness versus randomness tradeoffs for AM

A non-black-box uniform reduction for derandomizing AM in the high-end was given in [11]. It yields gap theorems for both AM and AM$\cap$coAM. The gap theorem for AM is analogous to that of [18] (except that it concerns the high-end and not the low end); it asserts: Either Arthur-Merlin protocols are very weak (in the sense that they can be simulated non-deterministically in polynomial time on feasibly generated inputs) or else they are somewhat strong (in the sense that they can simulate E $=$ DTIME($2^{O(\ell)}$) in time $2^{o(\ell)}$).[5] The gap theorem for AM$\cap$coAM gives the same result with "AM" replaced by "AM $\cap$ coAM." The statement is in fact cleaner for AM$\cap$coAM because it does not mention feasibly generated inputs, and instead applies to *all inputs*.

The result of [11] relies on identifying a certain "resiliency property" of an HSG construction of [24] (constructed for the nonuniform setting) and on "instance checking" [6], which was previously used in this context by [4, 29]. While it gives a high-end result it does not generalize to the low-end because the HSG construction of [24] works only in the high end. We remark that there is an alternative construction (in the nonuniform setting) of [26] that does work in the low-end but does not have the crucial resiliency property.

Our result: low-end uniform hardness versus randomness tradeoffs for AM

In this paper we obtain a resilient HSG (with a uniform reduction proving its correctness) that works over a larger domain of parameters and covers a wide range of hardness assumptions (coming very close to the absolute low-end). Using our result we extend the gap theorems of [11] as fol-

[2] An alternative formulation is to think of G as a function that takes a $t = \log T$ bit "seed" as input and outputs the element in T indexed by the seed.

[3] By [1], HSGs for deterministic circuits also suffice to derandomize two sided error BPP.

[4] To state this result formally one needs a precise definition of "feasibly generated inputs". The actual result also involves "infinitely often" quantifiers which we will ignore in this informal introduction.

[5] The notions of feasibly generated inputs in [11] is incomparable to that in [18] and follows the "pseudo" notion introduced in [19]).

lows (for a formal statement of the two Theorems below see Theorems 4 and 5 in Section 2):

THEOREM (INFORMAL) 1. *Either EXP* $= DTIME(2^{\ell^{O(1)}})$ *is computable by Arthur-Merlin protocols with time* $s(\ell)$ *or for any AM language* L *there is a nondeterministic machine that runs in time exponential in* ℓ *and solves* L *correctly on* feasibly generated inputs *of length* $m = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell)))^2\right)}$.

The second Theorem below achieves a clean statement that works for all inputs (rather than feasibly generated inputs). However, this is only achieved for AM $\cap$ coAM.

THEOREM (INFORMAL) 2. *Either EXP* $= DTIME(2^{\ell^{O(1)}})$ *is computable by Arthur-Merlin protocols with time* $s(\ell)$ *or for any AM* $\cap$ *coAM language* L *there is a nondeterministic (and co-nondeterministic machine) that runs in time exponential in* ℓ *and solves* L *correctly on* all *inputs of length* $m = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell)))^2\right)}$.

Here is how to interpret the parameters: Given a problem in EXP that has "hardness" $s(\ell)$ on inputs of length ℓ one can hope to construct a generator that produces $T = 2^{O(\ell)}$ outputs in time exponential in ℓ where each output is of length $m = s(\ell)^{\Omega(1)}$, and the generator fools circuits of size $m^{O(1)}$.[6] Our construction achieves only slightly worse parameters, only fooling circuits of somewhat smaller size, $m^{\Theta\left(1/(\log \ell - \log\log s(\ell)))^2\right)}$. This constrains the length of inputs to the AM (or AM $\cap$ coAM) language we can handle. Observe that our results are optimal at the high-end when $s(\ell) = 2^{\Omega(\ell)}$, and so they truly extend [11].[7] At the same time our results start working very close to the absolute low-end (which would be $s(\ell) = \ell^{O(1)}$): they produce nontrivial derandomizations as soon as $s(\ell) \geq 2^{\omega(\log^3 \ell)}$. They also give a smooth tradeoff in between the two extremes. It may be helpful to view the consequences for some particular choices of $s(\ell)$ and then express the running time of the nondeterministic machine as a function of the length of *its* input.

- For $s(\ell) = 2^{\Omega(\ell)}$ and EXP replaced with E (the high-end) the nondeterministic machine runs in polynomial time in the length of its input (as is the case in [11]).
- For $s(\ell) = 2^{\ell^\delta}$ and constant $\delta > 0$, the nondeterministic machine runs in time $\exp((\log m)^{O(1/\delta)})$ on inputs of length m.
- For $s(\ell) = 2^{(\log \ell)^a}$ and constant $a > 3$, the nondeterministic machine runs in time subexponential in the length of its input. The $a > 3$ requirement is suboptimal as we can hope to get the same behavior even when $a > 1$ (which is the absolute low-end).

[6]For a discussion on why this is best possible see [16]).
[7]A technicality is that at the high end it is common to replace EXP by E = DTIME($2^{O(\ell)}$) and require that the machine runs in time $2^{O(\ell)}$ rather than $2^{\ell^{O(1)}}$. This is done so that for $s(\ell) = 2^{\Omega(\ell)}$ the running time of the machine is polynomial in the length of its input. This is the way the result is presented in [11] and our results also work in this setting.

Our techniques

There are three new ideas in our construction. First, we use techniques from PCPs (low-degree testing and self-correction) to speed up certain steps in the reduction of [24], so that they run in sublinear time in the size of their input. Although it has long been observed that there is some similarity between aspects of PCP constructions and aspects of PRG and HSG constructions, this seems to be the first time primitives like low-degree testing have proven useful in such constructions. Second, we run both the [24] construction and the associated reduction recursively, in a manner reminiscent of [14, 15, 31] (although the low level details are different). Finally, we introduce a new primitive called *local extractors for Reed-Muller codes*, which are extractors that are computable in sublinear time when run on inputs that are guaranteed to be Reed-Muller codewords. A construction of such an object can be deduced from [26]. They play a crucial role in the improved constructions, and may be of interest in their own right. In Section 5.2 we give a detailed (but not formal) account of our construction and the way the new ideas fit into the proof.

Motivation

Uniform hardness vs. randomness tradeoffs represent some of the most involved proofs of non-trivial relationships between complexity classes, using "current technology." Pushing them to their limits gives new results, but also may expose useful new techniques, as we believe this work does. Moreover, the complexity classes we study, AM and AM $\cap$ coAM, contain a rich array of important problems, from hard problems upon which cryptographic primitives are built, to group-theoretic problems, to graph isomorphism, and indeed all of the class SZK (Statistical Zero Knowledge).

A second motivation is the quest for *unconditional* derandomization results. In [11] it was shown that if one can prove a low-end gap theorem for AM that works for all inputs rather than just feasibly generated inputs, then it follows that AM can be derandomized (in a weak sense) *unconditionally* (the precise details appear in [11]). In this paper we come closer to achieving this goal by achieving a low-end version of [11].

Organization of the paper

In Section 2 we restate our main theorems formally using precise notation. In Section 3 we describe the new "local extractors" and some variants of AM protocols that we will use as sub-protocols. In Section 4 we give the new recursive HSG and the statement of the main technical theorem. In Section 5 we describe the ideas that go into the proof of the main technical theorem. Due to space limitations most proofs are omitted and will appear in the full version.

2. FORMAL STATEMENT OF RESULTS

In this section we formally state Theorems 1 and 2. In order to do so we need to precisely define the notion of "derandomization on feasibly generated inputs".

2.1 Feasibly generated inputs

Following [11] we will use the notions defined in [19]. Loosely speaking, we say that two languages L, M are *indistinguishable* if it is hard to feasibly generate inputs on which they disagree. For this paper it makes sense to allow the procedure trying to come up with such inputs (which is called a *refuter* in the terminology of [19]) to use nondeterminism.

Definition 3 (Distinguishability of languages). *We say that a nondeterministic machine R (the* refuter*) distinguishes between two languages $L, M \subseteq \{0,1\}^*$ on input length n if $R(1^n)$ outputs some $x \in (L \cup M) \setminus (L \cap M)$ on every one of its accepting computation paths.*

With this notation we can formally capture the informal statements in the introduction. More specifically, when given a language $L \in \mathrm{AM}$, a nondeterministic machine M running in time $t(n) < 2^n$ succeeds on feasibly generated inputs if for any refuter R running in time $t(n)$, R does not distinguish L from $L(M)$.[8]

2.2 Formal restatements of Theorems 1 and 2

We are now ready to restate our main theorems in the formal notation. The following Theorem is the formal restatement of Theorem 1.

Theorem 4. *For every function $s(\ell) < 2^\ell$, either:*

- *EXP is computable by Arthur-Merlin protocols running in time $s(\ell)$, or*
- *for any language $L \in AM$ there is a nondeterministic machine M that runs in time $2^{\ell^{O(1)}}$ on inputs of length $m = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell)))^2\right)}$ such that for any refuter R running in time $2^{\ell^{O(1)}}$ and producing inputs of length m, R does not distinguish L from $L(M)$, on infinitely many input lengths.*

The following Theorem is the formal restatement of Theorem 2.

Theorem 5. *For every function $s(\ell) < 2^\ell$, either:*

- *EXP is computable by Arthur-Merlin protocols running in time $s(\ell)$, or*
- *for any language $L \in AM \cap coAM$ there is a nondeterministic machine M that runs in time $2^{\ell^{O(1)}}$ on inputs of length $m = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell)))^2\right)}$ such that L and $L(M)$ are equal on infinitely many input lengths.*

We remark that the two theorems above are also true when replacing EXP with E and then one can replace all the occurrences of $2^{\ell^{O(1)}}$ with $2^{O(\ell)}$.

Following [11] we can also reverse the order of "infinitely often" in Theorem 5 and achieve:

Theorem 6. *For every function $s(\ell) < 2^\ell$, either:*

- *EXP is* infinitely often *computable by Arthur-Merlin protocols running in time $s(\ell)^9$, or*
- *for any language $L \in AM \cap coAM$ there is a nondeterministic machine M that runs in time $2^{\ell^{O(1)}}$ on inputs of length $m = s(\ell)^{\Theta\left(1/(\log \ell - \log\log s(\ell)))^2\right)}$ such that L and $L(M)$ are equal on all input lengths.*

3. PRELIMINARIES

We will be working with nondeterministic and co-nondeterministic circuits. A *nondeterministic circuit* is an ordinary Boolean circuit C with two sets of inputs, x and y. We say that C *accepts* input x if $\exists y\ C(x,y) = 1$ and that C *rejects* input x otherwise. A *co-nondeterministic circuit* has the opposite acceptance criterion: it *accepts* input x if $\forall y\ C(x,y) = 1$ and *rejects* input x otherwise.

3.1 Low degree testing and self correctors

The key to our results is that in many places we work *implicitly* with functions that are supposed to be low-degree polynomials – of course this is the central concept in PCPs as well. Just as with PCPs, we need the ability to locally test whether an implicitly supplied function is of the "correct" form: namely, we need to check whether it is (close to) a low-degree polynomial. As is standard, once we have determined that an implicitly supplied function is close to a low-degree one, we can "access" the nearby low-degree function locally using a self-corrector.

Low-degree testers and *self-correctors* are standard primitives in the PCP literature. In fact for our intended use of these primitives, we do not need delicate control of the parameters; we only need to be able to operate on multivariate functions over a field $\mathbb{F}$ in time poly($|\mathbb{F}|$) (hence making at most that many queries), while handling constant relative distance, and with constant soundness error for both primitives. The formal definitions, and the known results that we will make use of follow:

Definition 7 (low-degree tester). *A* low-degree tester *with parameters h, δ, ϵ is a probabilistic oracle machine M which has oracle access to a function $f : \mathbb{F}^d \to \mathbb{F}$, and for which*

- *if $\deg(f) \le h$ then M^f accepts with probability 1, and*
- *if all polynomials g with $\deg(g) \le h$ satisfy $\Pr_x[f(x) \neq g(x)] \ge \epsilon$, then M^f rejects with probability at least δ.*

Lemma 8 ([9]). *There exists a (non-adaptive) low-degree tester with parameters $h, \delta, \epsilon = 2\delta$, running in poly($|\mathbb{F}|$) time, provided $|\mathbb{F}| > ch$, $\delta < \delta_0$, for universal constants c and δ_0.*

Definition 9 (self-corrector). *A* self-corrector *with parameters h, δ, ϵ is a probabilistic oracle machine M which has oracle access to a function f, and for which*

[8] The statement in [11] uses a formal notation borrowed from [19] that in the situation above reads AM $\subseteq$ [pseudo(NTIME($t(n)$))]–NTIME($t(n)$) where the first occurrence of NTIME($t(n)$) stands for the class of the refuter and the second one for the class of the machine M. We choose not to use this notation as it complicates the statements of our results and makes them less clear. However we stress that our results use exactly the same meaning of feasibly generated inputs as in [11, 19].

[9] The precise meaning of this statement is that for every language $L \in$ EXP there is an Arthur-Merlin protocol that for infinitely many input lengths:

- Fulfills the AM promise (i.e., has noticeable difference between completeness and soundness).
- Computes the language L on that input length.

- *if there exists a polynomial g of total degree h, for which $\Pr_x[g(x) \neq f(x)] < \epsilon$, then*

$$\Pr[M^f(x) = g(x)] > 1 - \delta.$$

Lemma 10 ([5, 22]). *There exists a (non-adaptive) self-corrector with parameters $h, \delta = O(1/(\epsilon|\mathbb{F}|)), \epsilon$, running in poly$(|\mathbb{F}|)$ time, provided $\epsilon < \frac{1}{4}(1 - h/|\mathbb{F}|)$.*

We remark that for both low-degree testers and self-correctors, it is possible decrease the soundness error from a constant to 2^{-t} by repeating the protocol $\Theta(t)$ times.

3.2 Local extractors for subsets

The final object we will use to perform local computations on an implicitly supplied function is what we call a "local extractor for subsets". The notion of "locally computable extractors" was introduced in [23, 32] in the context of encryption in the bounded-storage model. Loosely speaking, it requires that the extractor is computable in time sublinear in the length of its first input. In our construction we require such extractors for very low "entropy thresholds". However, Vadhan [32] proved that it is impossible to have such extractors unless the entropy threshold is very high. For this purpose we introduce a new variant of local extractors in which the first input comes from some prescribed subset (rather than the set $\{0,1\}^n$) and exploit the fact that we intend to run the extractor on inputs that are codewords in an error-correcting code. It turns out that the construction of [26] can be computed in time polynomial in the output when applied on the Reed-Muller code even when shooting for low entropy thresholds. The formal details follow:

Definition 11 (local extractor for subsets). *A (k, ϵ) local C-extractor is an oracle function $E : \{0,1\}^t \to \{0,1\}^m$ for which the following holds:*

- *for every random variable X distributed on C with minentropy*[10] *at least k, $E^X(U_t)$ is ϵ-close to uniform, and*
- *E runs in poly(m, t) time.*

Definition 12 (Reed-Muller code). *Given parameters r, h and a prime power q we define $RM_{r,h,q}$ to be the set of polynomials $p : \mathbb{F}^r \to \mathbb{F}$ over the field with q elements, $\mathbb{F}$, and having degree at most h.*

The construction of [26] gives the following local extractor for the Reed-Muller code (we have made no attempt to optimize the constants):

Lemma 13 (Implicit in [26]). *Fix parameters $r < h$, and let $C = RM_{r,h,q}$ be a Reed-Muller code. Set $k = h^5$. There is an explicit $(k, 1/k)$ local C-extractor E with seed length $t = O(r \log q)$ and output length $m = h = k^{1/5}$.*

The following proposition follows from the definition.

Proposition 14. *Let $E : \{0,1\}^t \to \{0,1\}^m$ be a (k, ϵ) local C-extractor, and let D be a subset of $\{0,1\}^m$. Then at most 2^k elements $x \in C$ satisfy: $\Pr_y[E^x(y) \in D] > \frac{|D|}{2^m} + \epsilon$.*

[10]The minentropy of a random variable X is $\min_{x \in \mathrm{supp}(X)} -\log(\Pr[X = x])$.

We will use local extractors in the following way. We will be interested in the set $\{x : \Pr_y[E^x(y) \in D] = 1\}$, and we would like to be able to check whether some $x \in C$ is in this set by performing a local computation on x. This is not possible in general but a relaxation of this goal is. If we perform the probabilistic test of checking whether $E^x(y) \in D$ for a random y, then we will accept all x in the set, and not accept too many other x, because by the above proposition, the set of $x \in C$ on which this test accepts with high probability is "small" – it has size at most 2^k. This relaxation will turn out to be sufficient for our intended application.

3.3 Commit-and-evaluate protocols

We now define a variant of AM protocols that we will use repeatedly as a subprotocol when constructing standard AM protocols. An i round AM protocol is a protocol in which Arthur and Merlin receive a common input x and at each round Arthur sends public random coins and Merlin replies. At the end of the protocol Arthur outputs a value (not necessarily Boolean), denoted by out(π, M, x), that is a random variable defined relative to a *strategy* M for Merlin; i.e., M is a function that describes Merlin's response given a history of the interaction so far. The running time of the protocol is the running time of Arthur. A protocol may take an auxiliary common input y, which we will variously think of as a "commitment" or an "advice string". In this case we denote the output by out(π, M, x, y). The output $\perp$ (which is intended to be output by Arthur when he detects a dishonest Merlin) is a distinguished symbol disjoint from the set of intended output values.

Definition 15 (AM protocols that output values). *Given an AM protocol π and an input domain I, we say that π with auxiliary input y:*

- *is* PSV *(partially single valued) over I with soundness s if there exists a function g defined over I, such that for all $x \in I$, and all Merlin strategies M*

$$\Pr[out(\pi, M, x, y) \in \{g(x), \perp\}] \geq 1 - s.$$

- conforms *with a function f defined over I with completeness c if for all $x \in I$, there exists a Merlin strategy M for which*

$$\Pr[out(\pi, M, x, y) = f(x)] \geq c.$$

- computes *a function f over domain I with soundness s and completeness c if π with auxiliary input y is PSV over I with soundness s and conforms with f with completeness c.*

We may sometimes omit s and c in which case they are fixed to their default values $s = 1/3$ and $c = 2/3$. We also omit I when it is clear from the context.

We will be interested in protocols that are composed of two phases, and operate over the domain $I = \{0,1\}^n$. The first phase is called the *commit phase*. This is an AM protocol whose input is 1^n, and whose auxiliary input is an advice string α that depends only on n. Loosely speaking, the role of this phase is to generate an auxiliary input to the second phase. The second phase is called the *evaluation phase*. This is an AM protocol whose input is $x \in I$, and whose auxiliary input is the output of the commit phase protocol. The reason we distinguish between two different phases is that we make the additional requirement that there is a function computed by the combined protocol that is completely determined at the end of the commit phase (that is *before* Merlin knows the input x). The exact details appear below.

Definition 16 (commit-and-evaluate). *A* commit-and-evaluate protocol *is a pair of AM protocols* $\pi = (\pi_{commit}, \pi_{eval})$. *Given* π *and an input domain* $I = \{0,1\}^n$, *we say that* π *with advice* α*:*

- conforms *with a function* f *defined over* I *if there exists a Merlin strategy* M_{commit} *for which*

$$\Pr[\pi_{eval} \textit{ with auxiliary input } out(\pi_{commit}, M_{commit}, 1^n, \alpha) \textit{ conforms with } f] = 1.$$

- *is* γ-resilient *over* I *if for all Merlin strategies* M_{commit},

$$\Pr[\pi_{eval} \textit{ with auxiliary input } out(\pi_{commit}, M_{commit}, 1^n, \alpha) \textit{ is } PSV] \geq \gamma.$$

- runs in time $t(n)$ *for some function* t *if both* π_{commit} *and* π_{eval} *run in time bounded by* $t(n)$.

We may sometimes omit γ*, in which case it is fixed to its default value* $\gamma = 2/3$.

We now argue that completeness, soundness and resiliency can be amplified from their default values by parallel repetition.[11]

Proposition 17. *Let* $\pi = (\pi_{commit}, \pi_{eval})$ *be a commit-and-evaluate protocol that is resilient and conforms with* f*, with completeness* 1*, resiliency* 2/3 *and soundness* 1/3*. Furthermore, assume that* π_{commit} *is a one round protocol. Then* π *can be transformed (by parallel repetition) into a commit-and-evaluate protocol* $\pi' = (\pi'_{commit}, \pi'_{eval})$ *that is resilient and conforms with* f*, with completeness* 1*, resiliency* $1-2^{-t}$ *and soundness* 2^{-t}*. The transformation multiplies the running time and the output length of the commit protocol by* $\Theta(t)$*, and the running time of the evaluation protocol by* $\Theta(t^2)$*. The transformation preserves the number of rounds for both the commit protocol and the evaluation protocol.*

Note that after running the commitment protocol π_{commit} it is possible to run the evaluation protocol π_{eval} (with the auxiliary input output by π_{commit}) many times on many different inputs in I.

Note also that a γ-resilient commit-and-evaluate protocol that conforms with f does not necessarily "compute" f in any meaningful way. This is because in the commit phase, Merlin may not cooperate, causing the evaluation phase to receive an auxiliary input leading it to compute a function different from f. To demonstrate the usefulness of this notion we mention the following result from [11] (which follows from the instance-checkability of EXP-complete problems [3, 6]) stated informally:

Theorem (informal) 18. *Let* f *be the characteristic function of an EXP-complete problem. If* f *has a commit-and-evaluate protocol* π *that conforms with* f *resiliently, then there is an AM protocol* π' *that* computes f *and runs in time comparable to that of* π *using one additional round.*

It follows that to construct a (standard) AM protocol for EXP languages it is sufficient to construct commit-and-evaluate protocols that conform resiliently with an EXP-complete problem.

On a more technical level, commit-and-evaluate protocol are useful because the commit phase can be executed *before* the input x is revealed, and following the commit phase it is guaranteed that Merlin is committed to *some* function f. This allows Arthur to make "local tests" on the function f. For concreteness let us demonstrate this approach on low-degree testing (that is testing whether f is close to a low degree polynomial). Consider the following protocol: Arthur and Merlin play the commit phase of the protocol (which determines a function f). Then Arthur sends randomness for a low-degree test which in turn determines queries $x_1, \ldots, x_m$ to f. On each one of the queries x_i, Arthur and Merlin play the evaluation protocol and in the end Arthur checks that the low-degree test passes with the obtained evaluations. Note that no matter how Merlin plays he cannot make Arthur accept a function f that is far from a low degree polynomial, even though Merlin may never explicitly specify f.

4. A RECURSIVE HSG CONSTRUCTION

In this section we present a recursive version of the Miltersen-Vinodchandran (MV) generator [24], that receives a polynomial p (which can be thought of as the encoding of a hard function f) and outputs a multiset of m-bit strings. We also state the theorem asserting that it works as intended.

4.1 The construction

Let $\mathbb{F}$ be a field with q elements. We need one definition before giving the construction.

Definition 19 (Grouping variables and MV lines). *Given a function* $p : \mathbb{F}^r \to \mathbb{F}$ *and a parameter* d *that divides* r *we define* $B = \mathbb{F}^{r/d}$ *and identify* p *with a function from* B^d *to* $\mathbb{F}$.

Given a point $x \in B^d$ *and* $i \in [d]$ *we define the* line passing through x in direction i *to be the function* $L : B \to B^d$ *given by* $L(z) = (x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_d)$*. This is an axis-parallel, combinatorial line, which we call an* MV line *for short.*

Given a function $p : \mathbb{F}^r \to \mathbb{F}$ *and an MV line* L *we define a function* $p_L : B \to \mathbb{F}$ *by* $p_L(z) = p(L(z))$.

Note that if $p : \mathbb{F}^r \to \mathbb{F}$ is a polynomial then $p_L : \mathbb{F}^{r/d} \to \mathbb{F}$ is also a polynomial with degree bounded by that of p. We present our construction in Figure 1. The proof of the following Lemma will appear in the full version.

Lemma 20. *The construction* $RMV_{h,d}(p)$ *runs in time* $q^{O(r)}$ *and outputs at most* $q^{O(r)}$ *strings.*

[11] In the next proposition we only claim amplification for protocols where the commit protocol has one round and the evaluation protocol has perfect completeness. We make these relaxations because all protocols constructed in this paper have these properties. However, we remark that a more careful argument can get the same conclusion without these two assumptions. This follows along the same lines that parallel repetition of multi-round AM protocols amplifies soundness (see for example [10, p.145–148]).

Input A multivariate polynomial $p : \mathbb{F}_q^r \to \mathbb{F}_q$ of degree h.

Output A multiset of m bit strings.

Parameters and requirements We require that r is a power of d and that h is a prime power. We set $q = h^{100}$ and $m = h^{1/100}$.

Ingredients The $(k, 1/k)$ local C-extractor E from Lemma 13 for Reed Muller code $C = \mathrm{RM}_{r/d,h,q}$. Note that $k = h^5$, the extractor uses seed length $O((r/d) \cdot \log q)$ and it outputs m bits.

Operation of $\mathbf{RMV}_{h,d}(p)$:

- Set $B = \mathbb{F}_q^{r/d}$. For every $x \in B^d$ and $i \in [d]$, let $L : B \to B^d$ be the MV-line passing through x in direction i. Note that p_L is an element of the Reed-Muller code $\mathrm{RM}_{r/d,h,q}$. Compute $E^{p_L}(y)$ for all seeds y. Let H_p denote the set of these m bit strings, as L ranges over all MV lines.
- If $r = d$ then output H_p.
- If $r > d$ then for each MV line L make a recursive call to $\mathrm{RMV}_{h,d}(p_L)$. Note that while the dimension of p was r, the dimension of p_L is r/d. Each one of these recursive calls returns a multiset of m bit strings that we will call H_L. Output the union of H_p and H_L as L ranges over all MV lines.

Figure 1: Recursive MV generator $\mathbf{RMV}_{h,d}(p)$

4.2 The main technical theorem

Recall that the proof that a construction is indeed a HSG takes the form of a protocol for computing the hard function if the HSG fails. We will specify a commit-and-evaluate protocol $\pi = (\pi_{\text{commit}}, \pi_{\text{eval}})$ that takes advice $\alpha = D$ (where D is a co-nondeterministic circuit) and attempts to compute the polynomial p. We will prove that whenever D catches the generator $\mathrm{RMV}_{h,d}(p)$ then the protocol π conforms with p resiliently. (Note that this does not mean that π *computes* p. However, in our application we will be able to use π to construct a protocol that *does compute* p). Our main theorem is stated below. In fact, following [11], we prove a slightly stronger statement in which the resiliency of the protocol follows regardless of whether D catches $\mathrm{RMV}_{h,d}(p)$ as long as D rejects few inputs. This will be useful later on.

THEOREM 21. *Let d, h, r, m, q be as in Figure 1. Let $p : \mathbb{F}_q^r \to \mathbb{F}_q$ be a polynomial of degree at most h. Then there is a commit-and-evaluate protocol $\pi = (\pi_{commit}, \pi_{eval})$ with advice $\alpha = D$, where D is a co-nondeterministic circuit of size poly(m), that satisfies:*

Conformity *If D rejects every element of $RMV_{m,d}(p)$ then π conforms with p.*

Resiliency *If D rejects at most a $1/3$-fraction of its inputs then π is resilient.*

Efficiency *π runs in time $h^{O(d \log_d r)}$ and has $\log_d r$ rounds.*

Moreover, π_{eval} has completeness 1, and π_{commit} is a 1-round protocol.

The proof of Theorem 21 is described informally in Section 5. Our main results (Theorems 4 and 5) then follow from Theorem 21 largely using machinery already worked out in [11]. The details will appear in the full version.

5. THE REDUCTION

In this section we describe the reduction that proves Theorem 21.

5.1 Miltersen-Vinodchandran consistency test

We first abstract a certain primitive from the original Miltersen-Vinodchandran construction [24], and prove conformity and resiliency for it. This primitive, together with the three primitives in Sections 3.1 and 3.2 will be the main ingredients in the reduction proving correctness of the new generator. The main point of the abstraction is that the test makes sense when the "lines" of the original MV construction are replaced by what we are calling "MV lines," which are more general. We need one definition first:

DEFINITION 22 (MV PATHS AND S-BOXES). *Given $x \in B^d$ and a set $S \subseteq B$ we define a sequence of d sets $T_1, \ldots, T_d$ called the* MV path *to x using S. Each of these sets contains MV lines as follows: T_i contains all MV lines through points $\{(x_1, \ldots, x_i, s_{i+1}, \ldots, s_d) : s_{i+1}, \ldots, s_d \in S\}$ in direction i. We say that a line L appears in the MV path if $L \in \cup_i T_i$. Note that for $|S| > 1$ there are $\Sigma_{i=1}^d |S|^{i-1} \le |S|^d$ MV lines appearing in the MV path. Given a set $S \subseteq B$, an* S-box *is a function $a : S^d \to \mathbb{F}$.*

Figure 2 describes a test that we call the "MV consistency test". The usefulness of this procedure is captured in the following lemmas: (The proofs follow using the arguments of [24] and will appear in the full version).

LEMMA 23 (CONFORMITY OF MV CONSISTENCY TEST). *Fix a function $p : B^d \to \mathbb{F}$, an $x \in B^d$, and a subset $S \subseteq B$. The MV consistency test passes when given as input x, S, the S-box $a : S^d \to \mathbb{F}$ defined by $a(s_1, \ldots, s_d) = p(s_1, \ldots, s_d)$, and the collection of functions p_L ranging over all MV lines L in the MV path. Furthermore, if L is the single line in T_d, then $p_L(x_d) = p(x)$.*

LEMMA 24 (RESILIENCE OF MV CONSISTENCY TEST). *Let Z be a set of at most K functions where each one is a function from B to $\mathbb{F}$ and assume that for any two functions $g_1, g_2 \in Z$, with $g_1 \ne g_2$, $\Pr_{z \in B}[g_1(z) = g_2(z)] \le \beta$. Then with probability at least γ over the choice of a random subset*

Input A point $x \in B^d$, a subset $S \subseteq B$, and an S-box $a : S^d \to \mathbb{F}$. Also, the following collection of functions: for every line L appearing in the MV path to x using S, a function $g_L : B \to \mathbb{F}$.

Operation Let $T_1, \ldots, T_d$ be the MV path to x using S. The MV consistency test passes if the two tests below pass:

- (agreement with the S-box) For every line L in T_1 and $z \in S$, we check that $g_L(z) = a(L(z))$.
- (agreement at intersection points) For every pair of lines $L_1 \in T_i$ and $L_2 \in T_j$ for some i, j: if $L_1(z_1) = L_2(z_2)$ for some z_1, z_2, we check that $g_{L_1}(z_1) = g_{L_2}(z_2)$.

Figure 2: MV consistency test

$S \subseteq B$ with $|S| \geq (2\log K + \log(1/(1-\gamma)))/\log(1/\beta)$ the following event holds: for every S-box $a : S^d \to \mathbb{F}$ and for every x, there is at most one collection of functions from Z that passes the MV consistency test.

5.2 Overview of the construction and proof

In this section we survey and motivate the main ideas that go into the construction and proof, while highlighting the new ideas in this paper that allow us to improve the previous work of [24, 11].

The original MV generator

We start by describing the original Miltersen-Vinodchandran generator (using some of our language).

Given a polynomial $p : \mathbb{F}^d \to \mathbb{F}$ of degree h the construction sets q and m to be slightly larger than h and (the standard choice is say $m = q = 2h$). For every MV line[12] L it outputs the vector $z_L = (p_L(t))_{t \in \mathbb{F}}$ – the restriction of p to the line L.

Given a co-nondeterministic circuit D such that D rejects every output of the generator we would like to show that there is an commit-and-evaluate protocol π (that receives D as advice) and conforms with p resiliently. We need to make the additional assumption that D rejects very few – say 2^{m^δ} – strings of length m overall. In the context of AM derandomization this can be achieved by amplifying the AM protocol we are attempting to derandomize using dispersers. We stress, as this will be important later on, that this amplification can only achieve a constant $0 < \delta < 1$ efficiently.

We now describe the commit-and-evaluate protocol for evaluating p. In the commit phase Arthur sends a uniformly chosen set S of size $v \approx h^\delta$ and Merlin replies with an S-box that is supposed to be the "correct" S-box $a(s_1, \ldots, s_d) = p(s_1, \ldots, s_d)$. In the evaluation phase the two parties are given a point $x \in \mathbb{F}^d$ and Arthur wants to evaluate $p(x)$. Arthur and Merlin first compute the MV path to x using S (this path has at most v^d MV lines) and for each MV line in the path, Merlin sends Arthur a univariate polynomial $g_L : \mathbb{F} \to \mathbb{F}$ (that is supposed to be the polynomial p_L) by sending its $h+1$ coefficients. Arthur performs the following tests:

Small-set test Arthur asks Merlin to supply witnesses showing that D rejects g_L for all MV lines L on the MV path. (Note that Merlin can do this as D is a co-nondeterministic circuit).

Consistency test Arthur performs the MV consistency test using the polynomials g_L sent by Merlin.

If both the tests pass then Arthur decides that $p(x)$ equals $g_L(x_d)$ where L is the single line in the set T_d.

The conformity and resiliency properties of this protocol follow from Lemmas 23 and 24. More precisely, an honest Merlin can indeed conform to p by following the protocol. A cheating Merlin has the freedom to choose an S-box a that is incorrect and in this case the evaluation protocol does not necessarily conform with p. However the evaluation protocol is (with high probability over the choice of S) PSV as Lemma 24 guarantees that there is at most one collection of functions from the small set $Z = \{z : D \text{ rejects } z\}$ that passes the consistency test. This means that once Merlin commits to the S-box a he cannot make Arthur output two different values on a given input x.

We stress that this argument uses the structure of polynomials in a very weak way. To perform the argument we only need that the set of vectors $C = \{z_L : L \text{ is an MV line}\}$ is an error-correcting code, as is stated precisely in Lemma 24. In the present construction all of the z_L are sequences of $m > h$ evaluations of a degree h univariate polynomial and so the z_L are codewords of a Reed-Solomon code.

We now turn our attention to the running time of the protocol. There are roughly v^d MV lines on the MV path and for each one of them Merlin needs to send $h+1$ coefficients to define each p_L. Thus, overall the time is about hv^d. For Lemma 24 we need to set $v \approx m^\delta \approx h^\delta$ (this comes from the bound we have on the set Z, which in turn comes from the initial amplification of the AM protocol we are derandomizing). Overall the running time is about $h^{\delta d}$. Specifying the polynomial p explicitly requires roughly h^d coefficients and thus the protocol achieves something non-trivial since it runs in time that is only some constant root of h^d.

Goal: achieve the low end

The parameters achieved by the construction outlined above correspond to the "high-end" of hardness assumptions. More specifically when given a Boolean function f over ℓ bits we can encode it as a $d = O(1)$ variate polynomial p (the low-degree-extension of f) with $h, m \approx 2^{\ell/d}$. The protocol[13] above then gives us exactly the kind of parameters one wants; i.e., it runs in time polynomial in the output length, m, of the generator.

[12] For the definitions of MV lines and MV paths to make sense, we set $r = d$ and $B = \mathbb{F}$ for the moment.

[13] We have only argued that the protocol conforms resiliently, so we don't yet have an AM protocol for *computing* p . However, recall that in our intended application, we will have that f is complete for EXP and thus we can transform the AM protocol into one that *computes* f (see Theorem 18).

However, this relationship is only achieved at the "high end", that is when $m = 2^{\Omega(\ell)}$ and in fact the construction fails completely when m becomes significantly smaller. Our goal is to achieve the "low-end" so we must modify the construction of the generator so that we get a running time of $\mathrm{poly}(m)$ for any m, ideally all the way down to $m = \mathrm{poly}(\ell)$.

Reducing the degree h and distinguishing between r, d

A very natural idea (that has been useful in previous works in this area, e.g., [27, 26]) is to encode the function f using a polynomial p with more than a constant number of variables. This will enable the encoding to use smaller degree. Note however that because the number of variables increases when the degree decreases, the running time of the protocol we constructed does not benefit from reducing the degree h, as the gain over the trivial protocol depends only on δ which cannot be smaller than a constant. Thus, at this point it is not clear what we can gain from reducing the degree.

We will attempt to circumvent this problem by achieving the "best of both worlds": having a small degree while keeping the number of variables a constant. To achieve a behavior with that flavor we distinguish between two parameters r (the number of variables) and d (the number of "grouped variables"). More precisely, we now encode the function f as a polynomial $p : \mathbb{F}^r \to \mathbb{F}$ for super-constant r (at the absolute low-end we will use r as small as $\ell/\log \ell$ which allows the degree to go down to $h = \mathrm{poly}(\ell)$). While doing so we keep d as a constant and identify $\mathbb{F}^r$ with B^d, where $B = \mathbb{F}^{r/d}$, as in Definition 19.

We can now run the original MV generator just as before by thinking of p as a function $p : B^d \to \mathbb{F}$. This follows from our observation that we only need the MV lines to form an error-correcting code, and here for every MV line L, the associated function $p_L : \mathbb{F}^{r/d} \to \mathbb{F}$ is a Reed-Muller codeword. In the commit-and-evaluate protocol for p that we already saw, we only need to alter one thing: Merlin will need to supply coefficients for p_L which is now a degree h polynomial in r/d variables and has about $h^{r/d}$ coefficients (as compared to h coefficients previously).

At first glance it may seem that we have made progress and can handle m much smaller than the original MV construction required, but this is not the case. For the p_L to form a code (which is needed for Lemma 24 to apply), we need to output $m > h^{r/d}$ evaluations, and thus overall we do not gain (we were hoping to take m only slightly larger than h, not $h^{r/d}$.). However, intuitively we did make some progress as various quantities in the protocol (such as the size of the S-box and the length of the MV path) depend on d (which is constant) rather than on r.

Reducing m by using local extractors for Reed-Muller codes

We will reduce m by modifying the generator construction further. For each MV line L, instead of outputting enough evaluations of p_L to induce an error-correcting code, we will use an extractor. More precisely, we take E to be an extractor with output length $m \approx h$, and we output the strings $E(z_L, y)$ for all possible seeds y.

Then, in the AM protocol, we can replace the small-set test with a *probabilistic* small-set test: check that D rejects $E(z_L, y)$, for a random y. All of the z_L that formerly passed the small-set test will still do so, since by assumption all of the outputs of the generator (and thus all of the outputs of E run on input z_L) are rejected by D. At the same time, by the extractor property, there can be only a small number (say, 2^{m^2}) of strings that pass the new probabilistic test with reasonable probability. This ensures that Lemma 24 still applies to this modified generator and protocol.

However, our goal was to reduce m and have the protocol run in time $\mathrm{poly}(m)$. But even invoking the extractor once for the probabilistic small-set test takes time linear in its input length $h^{r/d}$, which is much larger than m.

The crucial realization at this point is that we are only ever interested in running the extractor on strings z_L that are evaluations of low-degree polynomials! We can thus replace E with a local extractor for the Reed-Muller code, and consequently reduce the running time of the extractor to $\mathrm{poly}(m)$ when given oracle access to its input.

So, we can do the small-set test in time $\mathrm{poly}(m)$, given oracle access to p_L. For our choice of parameters, the MV consistency test also runs in time $\mathrm{poly}(m)$ given oracle access to p_L. However one hurdle remains: the step in which Merlin sends the coefficients of the polynomials p_L still requires $h^{r/d} \gg m$ time to send the $h^{r/d}$ coefficients of p_L, while we are shooting for $\mathrm{poly}(m)$ time.

Sending the polynomials p_L implicitly

Let us assume at this point that for some reason we already knew that for every L the polynomial p_L has a commit-and-evaluate protocol that conforms with it resiliently and that this protocol runs in time $\mathrm{poly}(m)$. Then instead of having Merlin send the polynomials p_L explicitly, Arthur and Merlin could play the commitment phase of the protocol for p_L, after which Merlin will be able to assist Arthur on evaluating p_L on any input that Arthur wishes.

However, we have now exposed the protocol to the possibility that Merlin may cheat by committing to a function that is not a low-degree polynomial, and then (at least) two things break: the local extractor for Reed-Muller codes may be run with access to an oracle that is not a Reed-Muller codeword, destroying the extractor property needed for the integrity of the small-set test; and, the resiliency of the MV consistency test relies on all of the received functions having large distance.

The solution is to run a low-degree test on each function Merlin commits to, verifying that it is indeed a low-degree polynomial. This test can be done locally, with oracle access to the function, and the fact that Merlin is *committed* to a function (and cannot alter the requested evaluations upon seeing the randomness of the test) ensures the validity of the test.

Let us summarize our current position. *If* we knew that for every MV-line L the polynomial p_L had a $\mathrm{poly}(m)$ time commit-and-evaluate protocol that conformed with it resiliently, then we would be able to produce a commit-and-evaluate protocol that conforms with p resiliently, and more importantly, runs in time $\mathrm{poly}(m)$ (which is our goal).

Using recursion to obtain the protocol for p_L

It is important to note that when trying to construct a protocol for a polynomial p with r variables, we need to assume the existence of a protocol polynomials p_L with a smaller number, r/d, of variables. This will allow us to use recursion. The base case will be the standard MV generator, where $r = d$. For the base case we already showed how to construct an AM protocol that runs in time $\mathrm{poly}(m)$.

To give us the protocol on MV lines needed in the recursive step, we modify the construction of the generator, finally arriving at the construction in Figure 1. In this construction, in addition to the original output of our modified MV generator run on p, we also output all the outputs of our modified MV generator run on the polynomials p_L for all MV lines L, and continue with this recursively. The inputs to the recursive calls are sufficiently smaller than the original input so that we do not increase the set of outputs of the generator by more than a polynomial factor. Now, a circuit D that rejects all the outputs of our generator can be used as advice to play the protocol on all the polynomials p_L that we will ever be interested in.

The final commit-and-evaluate protocol will invoke the protocol now available for MV lines it needs to access, continuing this recursively down to the base case.

We stress that the resiliency property plays a crucial role inside the recursion (in addition to its role as described in Theorem 18). Specifically, the resiliency property of the protocol for p_L says that following the commitment phase, Merlin is committed to some function, and this is what prevents Merlin from cheating when doing the local tests (such as the low degree test). If it wasn't for resiliency then Merlin would be able to choose outputs for p_L *after* seeing the queries of the low degree test which would make the test useless.

Losses suffered in the recursion

While we can reduce m using the ideas outlined above, there are also some costs to using this recursive argument. First, each recursive step in the protocol picks up one additional round and thus we end up with a protocol with $\log_d r$ rounds. Such protocols can be transformed into two round protocols but the running time suffers a blowup which is slightly super-polynomial. The running time also suffers as each recursive step multiples the running time of the protocol by poly(m). When taking these two factors into consideration and transforming to a one round AM protocol we get that this protocol has running time $m^{O(\log_d^2 r)}$ rather than $m^{O(1)}$. This accounts for the slight non-optimality of our main gap theorems.

Acknowledgements We thank Amnon Ta-Shma for helpful discussions.

6. REFERENCES

[1] A. E. Andreev, A. E. F. Clementi, and J. D. P. Rolim. Hitting sets derandomize BPP. In *ICALP*, pages 357–368, 1996.

[2] V. Arvind and J. Kobler. On pseudorandomness and resource-bounded measure. *Theoretical Computer Science*, 255, 2001.

[3] L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[4] 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.

[5] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In Choffrut and Lengauer [8], pages 37–48.

[6] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

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

[8] C. Choffrut and T. Lengauer, editors. *STACS 90, 7th Annual Symposium on Theoretical Aspects of Computer Science, Rouen, France, February 22-24, 1990, Proceedings*, volume 415 of *Lecture Notes in Computer Science*. Springer, 1990.

[9] K. Friedl and M. Sudan. Some improvements to total degree tests. In *ISTCS*, pages 190–198, 1995.

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

[11] D. Gutfreund, R. Shaltiel, and A. Ta-Shma. Uniform hardness versus randomness tradeoffs for Arthur-Merlin games. *Computational Complexity*, 12(3-4):85–130, 2003.

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

[13] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *Proceedings of the 16th Annual Conference on Computational Complexity (CCC-01)*, pages 2–12, 2001.

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

[15] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudo-random generators with optimal seed length. In *Proceedings of the thirty second annual Symposium on Theory of Computing*, pages 1–10, 2000.

[16] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Reducing the seed length in the Nisan-Wigderson generator. To appear in Combinatorica, 2006.

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

[18] R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *39th Annual Symposium on Foundations of Computer Science*, pages 734–743, 1998.

[19] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. In *Proceedings of the 15th Annual IEEE Conference on Computational Colmplexity (COCO-00)*, pages 150–157, 2000.

[20] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[21] A. R. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, Oct. 2002.

[22] R. J. Lipton. Efficient checking of computations. In Choffrut and Lengauer [8], pages 207–215.

[23] C.-J. Lu. Encryption against storage-bounded adversaries from on-line strong extractors. *J. Cryptology*, 17(1):27–42, 2004.

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

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

[26] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the 42nd Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[27] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62:236–266, 2001.

[28] 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.

[29] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th Annual Conference on Computational Complexity*, 2002.

[30] C. Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67:419–440, 2003.

[31] C. Umans. Reconstructive dispersers and hitting set generators. In *APPROX-RANDOM*, pages 460–471, 2005.

[32] S. P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *J. Cryptology*, 17(1):43–77, 2004.

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