UNIFORM HARDNESS VERSUS RANDOMNESS TRADEOFFS FOR ARTHUR-MERLIN GAMES

Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma

Abstract. Impagliazzo and Wigderson proved a uniform hardness vs. randomness "gap theorem" for BPP. We show an analogous result for AM: Either Arthur-Merlin protocols are very strong and everything in $\mathrm{E} = \mathrm{DTIME}(2^{O(n)})$ can be proved to a sub-exponential time verifier, or else Arthur-Merlin protocols are weak and every language in AM has a polynomial time nondeterministic algorithm such that it is infeasible to come up with inputs on which the algorithm fails. We also show that if Arthur-Merlin protocols are not very strong (in the sense explained above) then $\mathrm{AM} \cap co\mathrm{AM} = \mathrm{NP} \cap co\mathrm{NP}$.
Our technique combines the nonuniform hardness versus randomness tradeoff of Miltersen and Vinodchandran with "instance checking". A key ingredient in our proof is identifying a novel "resiliency" property of hardness vs. randomness tradeoffs.
Keywords. Derandomization, Arthur-Merlin games.
Subject classification. 68Q15.

1. Introduction

One of the most basic goals of computational complexity is understanding the relative power of probabilistic complexity classes such as BPP, MA and AM. In particular, a long line of research is aimed at showing that randomness does not add substantial computational power. Much research is aimed at achieving this by using the mildest possible unproven assumptions.

1.1. Nonuniform hardness versus randomness tradeoffs. One very fruitful sequence of results uses the "Hardness versus Randomness" paradigm, first suggested by Blum & Micali (1984) and Yao (1982). Nisan & Wigderson (1994) extended this paradigm. Their approach is to show that one can take a function that is computable in exponential time but cannot be computed by small circuits and use it to construct a *pseudo-random generator* that "stretches" a short string of truly random bits into a long string of "pseudo-random bits" that cannot be distinguished from uniform by small circuits. Such generators allow deterministic simulation of probabilistic classes. Loosely speaking, these constructions differ in:

- The type of circuits "fooled" by the generator. To derandomize BPP and MA one needs to "fool" deterministic circuits, and to derandomize AM one needs to fool co-nondeterministic circuits.

- The "stretch" of the generator. Generators with polynomial "stretch" (t bits to t^c bits) are called "low-end" generators and give subexponential time deterministic simulation (e.g., BPP $\subseteq$ SUBEXP $= \cap_{\delta>0}$DTIME(2^{n^δ}) or AM $\subseteq$ NSUBEXP $= \cap_{\delta>0}$NTIME(2^{n^δ})). Generators with exponential stretch (t bits to $2^{\Omega(t)}$ bits) are called "high-end" generators and give polynomial time deterministic simulation (e.g., BPP $=$ P or AM $=$ NP).

- The precise assumption on the hard function. Typically "High-end" generators require lower bounds on larger circuits (circuits of size $2^{\Omega(n)}$) whereas "Low-end" generators may require only super-polynomial lower bounds. Generators that fool co-nondeterministic circuits typically require hardness for co-nondeterministic circuits.

Today, after a long line of research (Babai *et al.* 1993; Blum & Micali 1984; Impagliazzo *et al.* 1999, 2000; Impagliazzo & Wigderson 1997; Klivans & van Melkebeek 1999; Miltersen & Vinodchandran 1999; Nisan & Wigderson 1994; Shaltiel & Umans 2001; Sudan *et al.* 1999; Umans 2002; Yao 1982), we have powerful and elegant constructions of "low-end" and "high-end" generators that derandomize BPP, MA and AM using "necessary assumptions" (i.e., assumptions that are implied by the existence of pseudo-random generators against nonuniform circuits). The reader is referred to a recent survey paper on derandomization for more details (Kabanets 2002).

All the above mentioned Hardness vs. Randomness tradeoffs give generators which fool some *nonuniform* class of circuits and require a *uniformly computable* function that is hard against a *non-uniform* class of circuits. In fact, every generator against a non-uniform class of circuits implies such a function.

We would like to mention that the nonuniform assumptions used in the tradeoffs mentioned above can be replaced by assumptions involving only uniform classes. It was shown by Karp & Lipton (1980) (with improvements in Babai *et al.* (1991b)) that if EXP $\neq$ PH then there is a function in exponential time that cannot be computed by polynomial size circuits (both deterministic and nondeterministic).

1.2. Uniform derandomization. It seems strange that when trying to derandomize *uniform* classes such as BPP, MA and AM one constructs generators which fool *nonuniform* circuits. It is natural to consider a weaker notion of pseudo-random generators which fool only *uniform machines* (either deterministic or nondeterministic). However, Such generators only suffice to derandomize BPP, MA and AM in a weak sense: It is infeasible to come up with inputs on which the suggested derandomization fails.[1]

We now explain why generators against uniform machines do not suffice for "full derandomization". Suppose that we use such a generator to derandomize say BPP. If the derandomization fails (almost everywhere) then there exist a sequence of inputs $\{x_n\}$ on which the deterministic simulation is incorrect. There is no guarantee that these inputs can be feasibly computed by a uniform machine. Thus, we cannot argue that there is a uniform machine which distinguishes the output of the generator from uniform. (Indeed, this is where nonuniformity usually comes in. These inputs are hardwired to the BPP-algorithm to create a distinguishing circuit.) Nevertheless, if we only require the derandomization to succeed on

[1]We remark that by Impagliazzo *et al.* (2002) even such weaker generators imply circuit lower bounds.

all "feasibly generated" inputs then it *is* guaranteed to work. Because if it fails then there is an efficient uniform machine that can generate the "bad" inputs. These inputs can be served as distinguishers, resulting in a uniform distinguisher for the generator. Following Kabanets (2000) we refer to derandomization that is guaranteed only to succeed on feasibly generated inputs as "pseudo-setting derandomization". We now define this notion more precisely.

Let $\mathcal{C}$ be a class of algorithms, we say that two languages L_1, L_2 are $\mathcal{C}$-different if there exists an algorithm $REF \in \mathcal{C}$, called a *refuter*, that on almost all input lengths produces instances in the symmetric difference of L_1 and L_2. $\mathcal{C}$ may be an arbitrary uniform class, and can be, e.g., a deterministic, probabilistic or a non-deterministic class. For a class of languages $\mathcal{C}$ and a class of algorithms $\mathcal{C}'$, the complexity class [io-pseudo($\mathcal{C}'$)]–$\mathcal{C}$ denotes all languages which are $\mathcal{C}'$-similar (i.e., not $\mathcal{C}'$-different) to some language in $\mathcal{C}$ [2]. Notice that the stronger the refuter is, the smaller the pseudo-class is. Precise definitions, as well as some other related notions are given in Section 3. So in the context of derandomization, a deterministic simulation that places, for example, BPP in the class [io-pseudo($\mathcal{C}'$)]–DTIME(t(n)), means that the simulation works in DTIME(t(n)), and it succeeds on all inputs that can be generated by algorithms in the class $\mathcal{C}'$.

We remark that all the hardness versus randomness tradeoffs mentioned above require hardness against *nonuniform circuits* even when attempting to fool *uniform machines*. Namely, even if we only want derandomization in the pseudo-setting, we still need to assume nonuniform hardness when working with the current constructions. This is because their correctness proofs use *nonuniform reductions*. (More precisely, the proof of correctness of these tradeoffs follow by showing a reduction from a machine that distinguishes the generator from random into a circuit that computes the hard function. The reductions for all the tradeoffs above use nonuniform advice.) In some sense (see Trevisan & Vadhan (2002) for precise details) every "black-box" construction of generators must rely on nonuniform reductions, and hence on hardness against nonuniform circuits.

Impagliazzo & Wigderson (1998) (see also Trevisan & Vadhan (2002)) construct a generator that relies on a *uniform* reduction only. Thus it is guaranteed to fool *uniform* deterministic machines under a weaker (and uniform) assumption: hardness against efficient uniform probabilistic machines (the previous nonuniform tradeoff of Babai *et al.* (1993) required hardness against small circuits). This hardness versus randomness tradeoff has a nice interpretation. It gives a "gap theorem" for BPP: Loosely speaking, either BPP = EXP (i.e., BPP is as strong as possible) or else every language in BPP has a subexponential time deterministic algorithm in the pseudo-setting. More formally, if BPP $\neq$ EXP then for every $\delta > 0$, BPP $\subseteq$ [io-pseudo(BPP)]–TIME(2^{n^δ}) [3].

We mention that more "gap theorems" for other classes (such as ZPP,MA, BPE, ZPE) were given in Impagliazzo *et al.* (2002); Kabanets (2000), by using the easy-witness method of Kabanets (2000).

[2]Here, as in Kabanets (2000) the "io" stands for infinitely often. One can also define "almost everywhere" versions of "pseudo".

[3]Impagliazzo & Wigderson (1998) use different notations and their statement is actually slightly stronger than the one we give here. The suggested derandomization works for a random input with high probability. We elaborate on this later on.

1.3. Our Results. In this paper we prove nondeterministic analogues of the results of Impagliazzo & Wigderson (1998). That is, we replace the class BPP by the class AM.

1.3.1. Hitting-set generators against co-nondeterministic machines.. We construct a "hitting" generator that fools *uniform* co-nondeterministic machines using a uniform assumption: hardness against efficient uniform Arthur-Merlin games.[4]

THEOREM 1.1. *For every $c > 0$ there exists a generator G with exponential stretch that runs in polynomial time in its output length, and:*

(i) If $E \not\subseteq AMTIME(2^{\beta n})$ for some $\beta > 0$ then G is $[io]$– $\frac{1}{2}$-hitting against $coNTIME(n^c)$.

(ii) If $E \not\subseteq \{io\text{–}AMTIME\}\ (2^{\beta n})$ for some $\beta > 0$ then G is $\frac{1}{2}$-hitting against $coNTIME_{\frac{1}{2}}(n^c)$.

By exponential stretch we mean that the generator needs a seed of length $O(\log m)$ to produce an output of length m. See Section 3 for the exact definition of the classes involved. For the time being the reader can think of $\{\text{io–AMTIME}\}\ (2^{\beta n})$ as an i.o. analogue of AM (although they are not quite the same) and of $co\text{NTIME}_{\frac{1}{2}}$ as the class of co-nondeterministic machines which answer "one" on at least half of the inputs. We later rephrase Theorem 1.1 in a more general and formal way, see Theorem 5.5.

1.3.2. Uniform Gap Theorems for AM. We give an AM analogue of the Impagliazzo & Wigderson (1998) gap theorem for BPP. A technicality is that while Impagliazzo & Wigderson (1998) works only for the low-end setting (see Trevisan & Vadhan (2002)), our result works only for the high-end setting.[5] We prove:

THEOREM 1.2. *If $E \not\subseteq AMTIME(2^{\beta n})$ for some $\beta > 0$, then for all $c > 0$,*

$$AM \subseteq [io\text{-}pseudo(NTIME(n^c))]\text{–}NP$$

Precise definitions appear in Section 3. In words our result can be stated as a gap theorem for AM as follows: Either Arthur-Merlin protocols are very strong and everything in E can be proved to a sub-exponential time verifier, or else Arthur-Merlin protocols are very weak and Merlin can prove nothing that cannot be proven in the pseudo-setting by standard NP proofs.

Our second gap theorem concerns the class AM $\cap$ *co*AM which we believe is of special interest as it contains the well studied class SZK (Statistical Zero-Knowledge) (Aiello & Hastad 1991), and therefore contains some very natural problems that are not known to be in NP, e.g., Graph non-isomorphism (Goldreich *et al.* 1991), approximations of shortest and closest vectors in a lattice (Goldreich & Goldwasser 1998) and Statistical Difference (Sahai & Vadhan 1997). For AM $\cap$ *co*AM we can completely get rid of the [io-pseudo]–quantifier and show:

[4]A "hitting" generator is a one-sided version of a "pseudo-random" generator. We say that a generator G is ϵ-hitting for a Boolean function h if there is an output of G which is accepted by h, or if h accepts less than an ϵ-fraction of the inputs (of each input length). We say that G hits a complexity class, if it hits every language L in the class, where L is viewed as the membership function $L(x) = 1$ iff $x \in L$.

[5]We remark that as observed in Trevisan & Vadhan (2002) the results of Impagliazzo & Wigderson (1998) can be adapted to the high-end setting if one assumes a hard function computable in polynomial space instead of exponential time.

THEOREM 1.3.

(i) If $E \not\subseteq$ AMTIME$(2^{\beta n})$ for some $\beta > 0$ then $AM \cap coAM \subseteq$ [io]–NP $\cap$ [io]–coNP.

(ii) If $E \not\subseteq$ {io–AMTIME} $(2^{\beta n})$ for some $\beta > 0$ then $AM \cap coAM = NP \cap coNP$.

We chose to state the assumption as hardness against AMTIME$(2^{\beta n})$ to make it similar to the assumption of Theorem 1.2. However, in both Theorems we can start with hardness against AMTIME$(2^{\beta n}) \cap co$AMTIME$(2^{\beta n})$.

We mention that the fact that we have both *infinitely often* and *almost everywhere* versions for the case of AM $\cap$ *co*AM is not trivial and indeed some technical work is needed for showing that. The construction of Impagliazzo & Wigderson (1998), as well as our Theorem 1.2 do not have both versions.

Our last gap theorem concerns derandomization under the assumption that there is a hard function in *nondeterministic* exponential-time. In nonuniform tradeoffs, when moving from BPP to AM one can allow the hard function to be in NE $\cap$ *co*NE. This only affects the complexity of the generator, instead of being computable in *deterministic* polynomial time, it is now computable in NP $\cap$ *co*NP. Since the application of the generator to derandomize AM already uses nondeterminism, this still gives the same result.

We cannot prove a high-end version of Theorem 1.2 and Theorem 1.3 that is based on a hard function in NE $\cap$ *co*NE, however, we can prove the following [6]:

THEOREM 1.4. *If NE $\cap$ coNE $\not\subseteq$ AMTIME$(2^{\beta n})$ for some $\beta > 0$, then:*

(i) AM $\subseteq$ [io-pseudo(NTIME(n^c))]–NTIME(2^{n^ϵ}) for every $c, \epsilon > 0$.

(ii) AM $\cap$ coAM $\subseteq$ [io]–$NTIME(2^{n^\epsilon})$ $\cap$ [io]–coNTIME(2^{n^ϵ}) for every $\epsilon > 0$.

Note the unusual hardness-randomness tradeoff in Theorem 1.4. While the hardness assumption correspond to the high-end setting (namely, a strong assumption), the conclusions corresponds to the low-end setting (namely, weak derandomization). Nevertheless, Theorem 1.4 involves nondeterministic classes only (with or without randomness), and thus has a nice interpretation in terms of the relative power of randomness in the context of proof systems: either randomness is helpful and every proof that requires exponentially long witnesses can be replaced by a much more efficient Arthur-Merlin game, or else, randomness is relatively weak and every (polynomial-time) Arthur-Merlin game can be replaced by a proof that does not use randomness while paying at most a subexponential cost in efficiency. (See also Remark 7.8 in Section 7.4).

1.3.3. Comparison To Previous Work. We now compare our results to previous work on derandomizing AM.

- Nonuniform hardness vs. randomness tradeoffs for AM were given in Klivans & van Melkebeek (1999); Miltersen & Vinodchandran (1999); Shaltiel & Umans (2001). Miltersen & Vinodchandran (1999) prove that if $NE \cap coNE \not\subseteq$ NTIME$(2^{\beta n})/2^{\beta n}$ for some constant β then AM = NP. This high-end result was extended in Shaltiel & Umans (2001); Umans (2002) to the low-end setting.

[6] Actually, we prove something slightly stronger in Section 7.4. The reason we stated a weak version here is to make it comparable to Theorem 1.2 and Theorem 1.3.

- Using the easy witness method of Kabanets (2000), Lu (2001) showed a derandomization of AM under a uniform assumption. Specifically, he showed that if NP $\not\subseteq$ [io-pseudo(NP)]–DTIME(2^{n^ϵ}) for some $\epsilon > 0$ then AM = NP. Impagliazzo *et al.* (2002) were able to remove the [io-pseudo]–quantifier from Lu (2001) and obtain the same conclusion using an assumption on exponential classes, namely, if NE $\cap$ *co*NE $\not\subseteq$ [io]–DTIME($2^{2^{\epsilon n}}$) for some $\epsilon > 0$, then AM = NP.

Our result is a uniform version of the non-uniform result of Miltersen & Vinodchandran (1999) and uses their construction. We soon explain our technique and the technical difficulty in this transition. The results of Lu (2001) and Impagliazzo *et al.* (2002) are incomparable to ours and use different techniques. We would like to stress that our results give an analogue for AM of the derandomization result of Impagliazzo & Wigderson (1998), while previous results do not. We also stress that our technique is very different from that of Impagliazzo & Wigderson (1998). We elaborate on our technique in Section 2.

1.4. Uniform generators and explicit construction of combinatorial objects. Our main motivation for constructing generators for uniform machines is obtaining a gap theorem for AM. However, such generators are useful in other contexts as well, as we now explain.

Some combinatorial objects can be easily constructed by probabilistic algorithms, yet no deterministic algorithm which explicitly constructs them is known. Klivans & van Melkebeek (1999) showed that the hardness versus randomness paradigm can sometimes be used to obtain conditional, deterministic explicit constructions of such objects. The main observation is that if the property we are looking for can be checked in *co*NP (i.e., checking whether a given object has the property can be done in *co*NP), then any generator which "fools" co-nondeterministic circuits, must have an element with the desired property as one of its outputs (or else the *co*NP algorithm is a distinguisher for the generator). We observe that as we only want to fool *uniform* machines we can use our construction to achieve the same conclusion under weaker *uniform* assumptions - namely under the assumption of Theorem 1.1.

In Section 6 we identify a general setting in which our construction can be used to derandomize probabilistic combinatorial constructions. We demonstrate this with matrix rigidity. The rigidity of a matrix M over a ring S, denoted $R^S_M(r)$, is the minimal number of entries that must be changed in order to reduce the rank of M to r or below. Valiant (1977) proved that almost all matrices have large rigidity. On the other hand, known explicit constructions do not achieve this rigidity (Friedman 1990; Pudlak & Vavrin 1991; Razborov 1989). Klivans & van Melkebeek (1999) proved that under the assumption that E requires exponential-size circuits with SAT-oracle gates, matrices with the required rigidity can be explicitly constructed. Miltersen & Vinodchandran (1999) relaxed the hardness assumption to nondeterministic circuits of exponential-size. We further relax the assumption to hardness against Arthur-Merlin protocols in which Arthur runs in exponential time [7].

THEOREM 1.5. *If there exists some constant $\beta > 0$ such that $E \not\subseteq$ {io–AMTIME} ($2^{\beta n}$) then there exists an explicit construction algorithm that for every large enough n constructs*

[7]Note that this is indeed a weaker assumption, because by standard inclusions of probabilistic classes in non-uniform classes we have that AMTIME($t(n)$) $\subseteq$ NTIME($poly(t(n))$)/$poly(t(n))$.

in time polynomial in n *a matrix* M_n *over* $S_n = \mathbb{Z}_{p(n)}[x]$, *such that* $R^{S_n}_{M_n} = \Omega((n-r)^2/\log n)$, *where* $p(n)$ *is polynomially bounded.*

Another application of Theorem 1.1 was recently given in Barak *et al.* (2003) to achieve certain "bit-commitment" protocols.

1.5. Organization. In Section 2 we give an overview of the ideas used in the papers. In Section 3 we describe the tools and set up the definitions and notations that we use in the technical parts. In Section 4 we present the Miltersen-Vinodchandran generator, prove its correctness and its useful properties. In particular we show that it has a resilient reconstruction algorithm. In Section 5 we prove Theorem 1.1 that under uniform assumptions gives us our generators against uniform co-non-deterministic machines. In Section 6 we apply Theorem 1.1 in the context of explicit constructions to achieve Theorem 1.5 about an explicit construction of rigid matrices. In Section 7 we prove our uniform gap theorems for AM (Theorem 1.2, Theorem 1.3 and Theorem 1.4). We conclude in Section 8 with some open problems and motivation for further research.

2. Overview of the technique

In this section we explain the main ideas in the paper on an "intuitive level". In this presentation it is easier to be imprecise with respect to "infinitely often".

2.1. Previous work. We start with an overview of relevant previous work.

Reconstruction algorithms. All current generators constructions under the hardness vs. randomness paradigm exploit the "reconstruction method" which we now explain. Let f be a hard function on which a generator $G = G_f$ is based. A circuit D is distinguishing if it distinguishes the "pseudo-random bits" of G_f from uniformly distributed bits. A reconstruction algorithm R gets a distinguishing circuit D for G_f and a short "advice string" a (that may depend on f and D) and outputs a small circuit $C = R(D, a)$ that computes the function f. Reconstruction algorithms serve as "proofs of correctness" for hardness versus randomness tradeoffs: If f is hard against small circuits and a reconstruction algorithm exists, then it must be the case that the generator G_f is pseudo-random (or else if G_f is not pseudo-random, there exists a small distinguisher D, and $C = R(D, a)$ is a small circuit for f, contradicting the hardness of f). As we previously explained this argument is essentially nonuniform. There is not necessarily an efficient way to come up with the distinguisher D or the advice string a.

Impagliazzo & Wigderson (1998): Using the reconstruction method to find the advice string in a uniform way. Impagliazzo and Wigderson made the simple observation that if the reconstruction algorithm R is *efficient*, then an algorithm which is trying to compute the hard function can "run" it. Furthermore, all known hardness vs. randomness tradeoffs for BPP use efficient reconstruction algorithms.

Let us use this observation to sketch a proof of the contra-positive of Impagliazzo & Wigderson (1998). Recall that this means that if BPP does not have a subexponential-time simulation in the pseudo-setting then BPP = EXP. Suppose that indeed BPP does not have a subexponential time deterministic algorithm. This means that BPP can not be derandomized by any generator. In particular, if we take an EXP–complete function f and use a nonuniform tradeoff to construct a generator G_f then G_f fails to derandomize BPP. Hence there exists a distinguisher D and an advice string a such that $C = R(D, a)$ computes the EXP–complete function f. The uniform pseudo-setting guarantees that D can be uniformly and efficiently generated. (This follows as in this setting there is a uniform refuter which generates inputs on which the derandomization fails.) The problem we are left with is how to uniformly find the advice string a. The key idea of Impagliazzo & Wigderson (1998) is to exploit specific learning properties of a particular reconstruction algorithm of Babai *et al.* (1993); Nisan & Wigderson (1994), as well as properties of the function f, to gradually and efficiently reconstruct the advice a *uniformly*. Once we can uniformly get the correct advice a, C can be generated efficiently (in probabilistic polynomial-time) and therefore the EXP–complete function f and the class EXP itself are in BPP, i.e., EXP = BPP.

Babai *et al.* (1991b): A (possibly dishonest) prover supplies the non-uniformity. Babai *et al.* (1991b) show that EXP $\neq$ MA implies EXP $\not\subseteq$ P/*Poly*. (Together with Arvind & Kobler (1997); Babai *et al.* (1993); Goldreich & Zuckerman (1997) this gives a gap theorem for MA: If EXP $\neq$ MA then by Babai *et al.* (1991b) EXP $\not\subseteq$ P/*Poly* and such a hardness assumption suffices to conclude that MA $\subseteq$ NSUBEXP.) Our technique for AM uses this approach. We now present the Babai *et al.* (1991b) argument in more detail. We prove the contra-positive.

We use the terminology of *instance checkers* from Blum & Kannan (1995). An instance checker for a function f, is a probabilistic polynomial-time oracle machine that when given oracle access to f and input x outputs $f(x)$, and when given oracle access to any $f' \neq f$ either outputs $f(x)$ or rejects with probability almost one. By Babai *et al.* (1991b), every EXP–complete function f has an instance checker. Let f be an EXP–complete function and assume EXP $\subseteq$ P/*Poly*. To show that $f \in$ MA consider the following proof system: On input x, Merlin sends Arthur a polynomial size circuit for $f \cap \{0,1\}^{|x|}$. Arthur simulates the instance checker on the given input x, using the circuit as the oracle. It is easy to check that the protocol is sound and complete. For the discussion below, it is important to point out the crucial fact in the proof: by sending the circuit Merlin *commits* himself to a specific function.

2.2. Our technique. Our technique is an integration between the reconstruction method and the instance checking techniques. We now present it on an intuitive level. Again it is convenient to be imprecise with respect to infinitely often. It is also easier to present the technique for the "low-end" setting, we will point out exactly where we have to switch to the "high-end". The presentation is sometimes oversimplified, and the reader can refer to the technical parts for the exact details.

First attempt. Our aim is to construct a generator G_f that is based on an EXP–complete function f and fools co-nondeterministic machines, assuming that f does not have efficient

Arthur-Merlin protocols. Such generators suffice for derandomization of AM in the pseudo-setting (we give more details on this later). Our starting point is known constructions of generators that fools co-nondeterministic circuits under nonuniform hardness assumptions. Such generators are constructed in Miltersen & Vinodchandran (1999); Shaltiel & Umans (2001). As usual, the correctness of these constructions is proved by presenting a reconstruction algorithm $C = R(D, a)$. In contrast to generators against deterministic circuits where C, D are deterministic circuits, in the constructions of Miltersen & Vinodchandran (1999); Shaltiel & Umans (2001), the circuit D is a co-nondeterministic circuit and the circuit C is a nondeterministic *single-valued* circuit. Informally, a single valued circuit is a nondeterministic circuit in which each computation path can take one value from $\{0, 1, quit\}$ such that all non-quitting paths take the *same* value (which we call the "output" of the circuit).

Now suppose that a co-nondeterministic distinguisher for G_f exists for every input length. Further assume that Arthur can somehow get hold of it (we later explain how this can be done). Arthur wants to compute f with the help of Merlin (this would contradict the hardness assumption). Let us try to follow the arguments of Babai *et al.* (1991b) together with the reconstruction algorithm R. Consider the following protocol for f: Merlin sends the advice string a. Then Arthur computes the circuit $C = R(D, a)$. As we cannot trust Merlin and we don't know if he sent the correct advice, we ask Arthur to run the instance checker for f using C as the oracle. Since C is a nondeterministic circuit, Arthur cannot "run" the circuit by himself. Therefore, each time Arthur wants to compute the function on some input, he asks Merlin to provide him with a non-quitting computation path for that input.

This argument fails because not every nondeterministic circuit is necessarily single-valued. A dishonest prover may send an advice string a such that $C = R(D, a)$ is not single valued. (We are only guaranteed that for the "correct" a, C is single valued.) Consider a circuit that for every input has a non-quitting path that evaluates to 1 and another that evaluates to 0. The prover can evaluate the circuit to any value he wishes, and is not committed to any specific function.

Resilient reconstruction algorithms. The new approach suggested in this paper is to study the behavior of reconstruction algorithms when given an "incorrect" advice string a. We cannot hope that the reconstruction algorithm R outputs a circuit C that computes f when given a wrong advice a. We can however hope that the circuit C is a nondeterministic *single valued* circuit for some function f'. We say that a reconstruction algorithm R is *resilient*, if it outputs a nondeterministic single-valued circuit given *any* (possibly wrong) advice a.

If R is a resilient reconstruction algorithm then even if Merlin is dishonest and sends an incorrect string a, the constructed circuit $C = R(D, a)$ is single-valued. Thus, even a dishonest Merlin has to commit himself to some function f' (the one defined by the single-valued circuit C). With that we can continue with the argument of Babai *et al.* (1991b), and Arthur can use the instance checker to validate his result.

While the reconstruction algorithm of Shaltiel & Umans (2001) does not seem to be resilient, we show that the "hitting-set" generator of Miltersen & Vinodchandran (1999) has such a resilient *probabilistic* reconstruction algorithm. The Miltersen-Vinodchandran generator only works in the high-end setting, and this is why all our results work only in the high-end setting.

In order to complete the argument we have to show how Arthur gets hold of the distinguisher. This is done in different ways according to the statement we prove (namely Theorem 1.1,Theorem 1.2, or Theorem 1.3), as we explain below.

A generator against co-nondeterministic machines. This is the easiest case. If the generator fails to hit uniform co-nondeterministic machines then there is a machine that is a distinguisher for the generator on every input length. This machine is uniform and can be part of Arthur's machine. This gives Theorem 1.1.

Gap theorems for AM. If the generator fails to derandomize a language L in AM, then for every input length n there is an instance x_n on which the derandomization failed. It is standard that x_n gives rise to a co-nondeterministic distinguisher for G_f. The problem we face now is how to find the "refuting" input $x_n \in \{0,1\}^n$. As we explained earlier this exact problem appears in most uniform derandomization works Impagliazzo & Wigderson (1998); Kabanets (2000); Lu (2001); Trevisan & Vadhan (2002). Previous works did not solve the problem, but rather weakened the result by requiring the derandomization to succeed in the pseudo-setting. In our case this means that if G_f fails to derandomize $L \in$ AM in the pseudo-setting, then there exists a uniform machine (the refuter) that on input n presents x_n that defines a co-nondeterministic machine D_n that distinguishes G_f. Arthur can use this refuter to obtain D_n. This gives Theorem 1.2.

Surprisingly, in the case of AM $\cap$ *co*AM (Theorem 1.3) we do not have to settle for pseudo-setting derandomization. Instead of using the refuter, we now ask the prover to supply us with a correct refuting input x_n. Naturally,, we should be wary of dishonest provers supplying incorrect inputs x_n (namely, inputs on which the derandomization hasn't failed) that do not lead to distinguishing algorithms D_n. It turns out that the Miltersen-Vinodchandran generator is even more resilient than what we required above. Namely, it is resilient not only in a, but also has the following resiliency property in D: Whenever D answers "zero" on few inputs (regardless of being a distinguisher or not), for every a, $R(D,a)$ is single-valued. This added resiliency is the basis for our improved result for AM $\cap$ *co*AM. It means that we can trust Merlin to send x_n as long as he can prove that D_n answers "zero" on few inputs.

In the case of AM $\cap$ *co*AM, Merlin can prove to Arthur that x_n *isn't* in the language. This means that the AM protocol will accept x_n with very low probability, which translates into a guarantee that D_n answers "zero" on few inputs.

2.3. A note on "infinitely often". The appearance of "infinitely often" is an unavoidable technicality in hardness versus randomness tradeoffs. We have so far ignored this technicality and we strongly suggest the reader to ignore these issues at a first reading. Nevertheless, for the reader that do want to dwell into the technical details, we feel that some explanation of how we handle "infinitely often" is in place.

Usually, hardness versus randomness tradeoffs come in two versions according to the positioning of the "infinitely often". It is helpful to state the tradeoff in the contra-positive.

1. If the derandomization of the randomized class fails almost everywhere then the function is easy almost everywhere.

2. If the derandomization of the randomized class fails infinitely often then the function is easy infinitely often.

Most proofs work in an "input length to input length basis". That is, for every input length of f on which the generator based on f fails the proof uses a distinguisher to show that f is easy on that length.[8] This usually suffices to achieve both versions above. However, in a uniform tradeoff there is a subtlety concerning the second version. suppose that there are infinitely many lengths n on which a given function $D : \{0,1\}^* \to \{0,1\}$ is a distinguisher for a generator. It is important to observe that using a hard function with input length ℓ, the generator outputs $m(\ell) >> \ell$ bits. Thus, the same length ℓ is used against many different input lengths of D - the input lengths between $m(\ell - 1)$ and $m(\ell)$. This poses a problem, when trying to use the distinguisher D to show that f is easy on length ℓ we have to choose a length n in the range above and might miss the "interesting" lengths on which D is a distinguisher. Nonuniform tradeoffs can bypass this problem by hard-wiring the "interesting n" to the circuit. We do not know how to overcome this difficulty in Theorem 1.2. We overcome this difficulty in Theorem 1.1, and Theorem 1.3 by having Merlin send the "good" length n. In these cases we use additional properties of the distinguisher D to argue that a dishonest prover cannot cheat by sending "bad" lengths.

3. Preliminaries

The density of a set $S \subseteq \{0,1\}^n$, denoted $\rho(S)$, is $\rho(S) = \frac{|S|}{2^n}$. The density of a circuit D over Boolean inputs of length m is $\rho(D) = \rho(\{y \in \{0,1\}^m \mid D(y) = 1\})$. For a language L and an input x, $L(x)$ is one if the input is in the language and zero otherwise. For a language L and an input length n, we define $L_n = L \cap \{0,1\}^n$. The notation $z \leftarrow U_m$ denotes picking z uniformly at random from $\{0,1\}^m$. In this notation $\rho(L_n) = \Pr_{x \leftarrow U_n}[L_n(x) = 1]$. For a class of languages C we define the class

$$[\mathrm{io}]\text{–}C = \{L : \exists M \in C \text{ s.t. for infinitely many } n,\ L_n = M_n\}$$

3.1. Complexity Classes. We denote EXP $=$ DTIME$(2^{poly(n)})$, E $=$ DTIME$(2^{O(n)})$ and SUBEXP $= \bigcap_{\epsilon>0}$ DTIME(2^{n^ϵ}). We let NEXP, NE and NSUBEXP be their nondeterministic analogs respectively. We define the non-standard class coNTIME$_\gamma(T(n))$ to be the class of languages L solvable by a coNTIME(n) machine, and $\rho(L_n) \geq \gamma$ for every $n \in N$.

[8]We remark that the proof of Impagliazzo & Wigderson (1998) has a different structure and requires that the generator fails on all input lengths.

Definition 3.1 (nondeterministic circuits). *A nondeterministic Boolean circuit $C(x, w)$ gets x as an input and w as a witness. We say $C(x) = 1$ if there exists a witness w such that $C(x, w) = 1$, and $C(x) = 0$ otherwise. A co-nondeterministic circuit is defined similarly with $C(x) = 0$ if there exits a witness w such that $C(x, w) = 0$ and $C(x) = 1$ if $C(x, w) = 1$ for all witnesses w.*

Next, we define classes of languages that have Arthur-Merlin games (or protocols).

Definition 3.2 (AM). *A language L belongs to $AMTIME_{\epsilon(n)}(TIME = t(n), COINS = m(n))$ if there exists a constant-round public-coin interactive protocol (P, V) such that the verifier uses at most $m(n)$ random coins, the protocol takes at most $t(n)$ time, and,*

- *(Completeness) For every $x \in L$ the verifier V always accepts when interacting with P, and,*
- *(Soundness) For every $x \notin L$ and every possibly dishonest prover P^*, the probability V accepts when interacting with P^* is at most $\epsilon(n)$.*

If ϵ is omitted then its default value is $1/3$. If we are not interested in the number of coins we omit it. The class AM denotes the class $\bigcup_{c>0} AMTIME_{\frac{1}{2}}(n^c)$.

The original definition of Babai & Moran (1988) has two-sided error, but it was shown in Furer *et al.* (1989) that this is equivalent to the one-sided version. Also, by the results of Babai & Moran (1988) and Goldwasser & Sipser (1989), a language has a constant-round interactive proof of complexity $t(n)$, if and only if it has a one round protocol of complexity $poly(t(n))$, where Arthur sends his public random coins to Merlin and Merlin answers. We will use this equivalence in the following way. We assume that protocols we want to derandomize have one-sided error and are one round, public coins protocols. Yet, when constructing protocols for hard functions we construct two-sided error multi-round protocols.

We will need a non-standard infinitely often version of the class AMTIME, in which the soundness condition holds for every input length but the completeness holds only for infinitely many input lengths. We denote this class by {io–AMTIME} .

Definition 3.3 ({io–AMTIME}). *A language L belongs to {io–AMTIME} $_{\epsilon(n)}(TIME = t(n), COINS = m(n))$ if there exists a constant-round public-coin interactive protocol (P, V) such that the verifier uses at most $m(n)$ random coins, the protocol takes at most $t(n)$ time, and the completeness condition in Definition 3.2 holds for infinitely many input lengths and the soundness condition of Definition 3.2 holds for all input lengths.*

Remark 3.4. *It is instructive to compare [io]–AM and {io–AM}. For a language L to be in [io]–AM there should be a language $M \in AM$ such that infinitely often M agrees with L. In particular, for every input length, M should define some language such that there is a non-negligible gap between the acceptance probability of inputs in the language and outside it. In contrast, the {io–AM} definition does not impose any restriction on positive instances* *of lengths that are not in the good infinite sequence, however, false proofs cannot be given even for these input lengths.*

This strange "io" notion comes in when trying to reduce between different problems. Suppose there is a linear time (or polynomial time) Karp-reduction from problem A to problem B. This means that if B is in AM then A is in AM. However, suppose that B is only known to be in [*io*]*–AM. It does not follow that A is also in* [*io*]*–AM. Nevertheless, replacing* [*io*]*–AM by* {*io–AM*} *(and requiring some additional properties of B) the conclusion does follow. See Lemma 3.13.*

3.2. Single valued proofs. The notion of proofs (e.g., NP proofs or interactive proofs) is asymmetric in nature, the prover can prove membership in a language but is unable to give false proofs of membership. The symmetric version of such proofs is where the prover can prove membership or non-membership in a language and can not give false proofs. It is not hard to see that if a language L has such a symmetric proof system, then both L and $\overline{L}$ have a one-sided proof system. Nevertheless, as we extensively use this notion we explicitly define it. We begin with non-deterministic circuits:

Definition 3.5 (Nondeterministic SV circuits). *A nondeterministic SV (single-valued) circuit $C(x,w)$ has three possible outputs: 1,0 and* quit *such that all non-quit paths are consistent, i.e., for every input $x \in \{0,1\}^n$, either $\forall_w C(x,w) \in \{0, quit\}$ or $\forall_w C(x,w) \in \{1, quit\}$. We say that $C(x) = b \in \{0,1\}$ if there exists at least one w such that $C(x,w) = b$, and then we say that w is a proof that $C(x) = b$. When no such w exists we say that $C(x) = quit$.*

*We say that C is a nondeterministic TSV (*total *single-valued) circuit if for all $x \in \{0,1\}^n$ $C(x) \neq quit$, i.e., C defines a total function on $\{0,1\}^n$. Otherwise, we say that it is a nondeterministic PSV (*partial *single-valued) circuit.*

It is easy to see that a Boolean function f has a nondeterministic TSV circuit of size $O(s(n))$ if and only if f has both a nondeterministic and a co-nondeterministic circuits of size $O(s(n))$. We next define single valued AM protocols. We remind the reader that for a language L we let $L(x)$ be one if $x \in L$ and zero otherwise.

Definition 3.6 (SV–AM protocols). *A language L has a SV–AMTIME$_{\epsilon(n)}$(TIME $= t(n)$, COINS $= m(n)$) protocol if there exists a constant-round public-coin interactive protocol (P, V) such that on input $(x, b) \in \{0,1\}^{n+1}$ the verifier uses at most $m(n)$ random coins, the protocol takes at most $t(n)$ time, and,*

(i) (Completeness) For every x, when interacting with the honest prover P, the verifier V accepts $(x, L(x))$ with probability at least $1 - \epsilon(n)$.

(ii) (Soundness) For every x and every possibly dishonest prover P^*, the probability V accepts $(x, 1 - L(x))$ is at most $\epsilon(n)$.

If soundness holds for every input length, but completeness holds only for infinitely many input lengths we say that L has a {*SV–io–AMTIME*}$_{\epsilon(n)}(t(n), m(n))$ *protocol.*

Clearly, if L has a SV–AMTIME$_{\epsilon(n)}(t(n), m(n))$ protocol (or, respectively, an {SV–io–AMTIME} $_{\epsilon(n)}(t(n), m(n))$ protocol) then $L \in$ AMTIME$_{\epsilon(n)}(t(n), m(n))$ (resp. $L \in$ {io–AMTIME} $_{\epsilon(n)}(t(n), m(n))$).

As usual if we are not interested in the number of coins we omit it. If the ϵ is omitted then its default value is 1/3.

3.3. Generators. A generator is a function $G : \{0,1\}^k \to \{0,1\}^m$ for $m > k$. We think of G as "stretching" k bits into m bits. We say a generator G is:

- ϵ-hitting for a class $\mathcal{A}$ if for every function $h : \{0,1\}^m \to \{0,1\}$ in $\mathcal{A}$ such that $\Pr_{z \leftarrow U_m}[h(z) = 1] > \epsilon$ there exists a $y \in \{0,1\}^k$ such that $h(G(y)) = 1$.
- ϵ-pseudo-random for $\mathcal{A}$, if for every function $h : \{0,1\}^m \to \{0,1\}$ in $\mathcal{A}$, it holds that

$$|\Pr_{y \leftarrow U_k}[h(G(y)) = 1] - \Pr_{z \leftarrow U_m}[h(z) = 1]| \;<\; \epsilon$$

Note that every ϵ-pseudo-random generator is also ϵ-hitting.

We will be interested in $\mathcal{A}$'s such as functions computed by deterministic circuits of some size $t(m)$, nondeterministic circuits, co-nondeterministic circuits, etc. When G is not hitting (pseudo-random) for $\mathcal{A}$ we call a function $h \in \mathcal{A}$ that violates the condition above an ϵ-distinguisher for G.

We often think of a generator G as a sequence $G_k : \{0,1\}^k \to \{0,1\}^{m=m(k)}$ defined for every $k \in N$. Given an $h : \{0,1\}^m \to \{0,1\}$ we can try and fool it by choosing the smallest k such that $m(k) \geq m$, and using G_k. When considering a sequence $h = h_m$ of functions. We can define two notions of hitting (pseudo-random) generators according to whether the generator succeeds almost everywhere or just infinitely often.

Definition 3.7. *Let $h = \{h_m\}$ be a sequence of functions $h_m : \{0,1\}^m \to \{0,1\}$.*

- *$G : \{0,1\}^k \to \{0,1\}^{m(k)}$ is ϵ-hitting (pseudo-random) for h if for every $m \in N$, taking k to be the smallest number such that $m(k) \geq m$, $G_k : \{0,1\}^k \to \{0,1\}^{m(k)}$ is ϵ-hitting (pseudo-random) for h_m.*
- *$G : \{0,1\}^k \to \{0,1\}^{m(k)}$ is [io]–ϵ-hitting (pseudo-random) for h if for infinitely many input lengths $m \in N$, taking k to be the smallest number such that $m(k) \geq m$, $G_k : \{0,1\}^k \to \{0,1\}^{m(k)}$ is ϵ-hitting (pseudo-random) for h_m.*

Current PRG constructions, under the hardness vs. randomness paradigm, take a hard function $f : \{0,1\}^\ell \to \{0,1\}$ and use it to build a PRG $G_f : \{0,1\}^{k(\ell)} \to \{0,1\}^{m(\ell)}$. We say that a construction G is a black-box generator if for every function $f : \{0,1\}^\ell \to \{0,1\}$ it defines a function $G_f : \{0,1\}^{k(\ell)} \to \{0,1\}^{m(\ell)}$, and furthermore it is possible to compute G_f in time polynomial in its output length when given oracle access to f. We remark that all existing constructions are black-box generators. If G is black-box then we sometimes denote it by $G_\ell : \{0,1\}^{k(\ell)} \to \{0,1\}^{m(\ell)}$, meaning that when G_f is given access to a Boolean function

f on ℓ bits it constructs from it a function $G_f : \{0,1\}^{k(\ell)} \to \{0,1\}^{m(\ell)}$. We use the notation G_f when we want to emphasize that we work with a specific function $f : \{0,1\}^{\ell} \to \{0,1\}$. When we want to emphasize both the specific function f and the input length ℓ that we are currently working with, we use $G_{f,\ell}$. We sometimes add $m = m(\ell)$ as a subscript to emphasize the output length of the generator.

3.4. Pseudo-Classes. In this section we define the notion of *uniform* indistinguishability which is sometimes called "the pseudo setting". For the purposes of this paper, we define only indistinguishability with respect to nondeterministic observers. Indistinguishability with respect to other observers (e.g. deterministic, probabilistic) can be similarly defined. The following definitions and notations are adopted from Kabanets (2000).

DEFINITION 3.8. *We say that two languages $L, M \subseteq \{0,1\}^*$ are NTIME($t(n)$) distinguishable a.e. (almost everywhere), if there exists a nondeterministic length-preserving procedure REF (which we call a* refuter*), that runs in time $t(n)$, such that for all but finitely many n's, R on input 1^n has at least one accepting computation path, and on every accepting path it outputs an instance x such that $x \in L \triangle M$ (where $L \triangle M$ is the symmetric difference between L and M). If this holds only for infinitely many n's, we say that L and M are NTIME($t(n)$)-distinguishable i.o. (infinitely often).*

If L and M are not NTIME($t(n)$)-distinguishable a.e. (resp. i.o.), we say that they are NTIME($t(n)$)-indistinguishable i.o. (resp. a.e.).

DEFINITION 3.9. *Given a complexity class $\mathcal{C}$ of languages over $\{0,1\}^*$ we define the complexity classes:*

$$\begin{aligned}
[\mathit{pseudo}(\mathit{NTIME}(t(n)))]\text{–}\mathcal{C} &= \{L : \exists M \in \mathcal{C} \text{ s.t. } L \text{ and } M \text{ are} \\
&\qquad \mathit{NTIME}(t(n))\text{-indistinguishable a.e.}\} \\
[\mathit{io\text{-}pseudo}(\mathit{NTIME}(t(n)))]\text{–}\mathcal{C} &= \{L : \exists M \in \mathcal{C} \text{ s.t. } L \text{ and } M \text{ are} \\
&\qquad \mathit{NTIME}(t(n))\text{-indistinguishable i.o.}\}
\end{aligned}$$

We remark that if the refuters have unlimited computational power, then the definition [io-pseudo($\mathcal{C}$)]– coincides with the standard notion of [io]–$\mathcal{C}$.

REMARK 3.10. *We choose to use the notion of Kabanets (2000) with nondeterministic refuters. This notion is incomparable to that of Impagliazzo & Wigderson (1998). The notion of refuter used in Impagliazzo & Wigderson (1998) concerns average case complexity. In Impagliazzo & Wigderson (1998) a refuter (relative to some samplable distribution μ) is a probabilistic algorithm which outputs a counterexample x with non-negligible probability. Thus, if two languages L and M are indistinguishable (relative to some samplable distribution) then they agree with high probability on a random input.*

Our results can work relative to such a probabilistic refuter. However we have to use refuters which output a counterexample with high probability (significantly larger than a 1/2). It is important to observe that it doesn't immediately follow that one can amplify the success probability of a refuter (that is convert a refuter which outputs a counterexample with *non-negligible probability into one which outputs a counterexample with high probability). The obvious strategy for performing this amplification requires sampling many candidates x and the ability to check whether a given input is a counterexample. This was done by Impagliazzo & Wigderson (1998) in the scenario of BPP but seems harder for AM.*[9]

3.5. Instance Checking. Blum & Kannan (1995) introduced the notion of instance checkers. We give a slight variation on their definition.

DEFINITION 3.11. *An instance checker for a language L, is a probabilistic polynomial-time oracle machine $IC^O(y, r)$ whose output is $0, 1$ or fail and such that,*

- *For every input y, $Pr_r[IC^L(y, r) = L(y)] = 1$.*
- *For every input $y \in \{0, 1\}^\ell$ and every oracle L', $Pr_r[IC^{L'}(y, r) \notin \{L(y), fail\}] < 2^{-\ell}$.*

It follows from Arora & Safra (1998); Babai *et al.* (1991b) that:

THEOREM 3.12. *For every complete problem in* E*, under linear-time reductions, there is a constant c and an instance checker for the problem that makes queries of length exactly $c\ell$ on inputs of length ℓ.*

The next Lemma allows us to use a fixed function f and a fixed instance checker in the constructions. We prove:

LEMMA 3.13. *There is a function f that is* E*-complete under linear-time reductions, and the following holds,*

- *There is a constant c and an instance checker for f that makes queries of length exactly $c\ell$ on inputs of length ℓ.*
- *If $f \in \bigcap_{\beta>0} AMTIME(2^{\beta n})$ then $E \subseteq \bigcap_{\beta>0} AMTIME(2^{\beta n})$.*
- *If f has a $\{SV\text{–}io\text{–}AMTIME\}$ $(2^{O(\beta n)})$ protocol for every $\beta > 0$, then so does every language in E. In particular, if f has such a protocol then*

$$E \subseteq \bigcap_{\beta>0} \{io\text{–}AMTIME\}\ (2^{O(\beta n)})$$

.

PROOF. Let f be the characteristic function of the following language: $\{(M, x, c) : M$ is a (padded) description of a deterministic machine that accepts x in time at most $c\}$. By "padded description" we mean that M is a string with a (possibly empty) prefix of zeros, followed by a description of a machine that starts with the bit 1. It is easy to verify that

[9]One of the reasons is that BPP algorithms can run a given deterministic circuit whereas AM protocols cannot run a given co-nondeterministic circuit. Loosely speaking, to check that a given x is a counterexample one converts x into a "distinguisher circuit" to some generator and checks that the circuit is indeed a distinguisher. However, in the case of AM this circuit is co-nondeterministic and thus, it seems hard to perform this check by an Arthur-Merlin protocol.

this language is complete in E under linear-time reductions. An important property of this function is that there is a simple mapping reduction from instances of input length n to instances of input lengths larger than n, just by padding the description of the machine. We say that this reduction embeds instances of length n into instances of length $m > n$.

The first two items follow directly from the fact that f is E-complete (together with Theorem 3.12). we prove the third item. Let $g \in$ E. Then there exists a linear-time reduction from g to f mapping inputs of length ℓ to inputs of length $d\ell$, for some constant d. Consider the following protocol for g: on input $y \in \{0,1\}^{\ell}$ and $b \in \{0,1\}$, apply the reduction from g to f mapping y to $y' \in \{0,1\}^{d\ell}$. Next, the prover specifies an input length between $d\ell$ and $d(\ell+1)$. Embed y' into an instance y'' of the specified length, and then run the {SV–io–AMTIME} $(2^{O(\beta n)})$ protocol for f on (y'', b) and answer accordingly.

To see correctness observe that by the properties of {SV–io–AMTIME} protocols, for every y, no prover can convince the verifier to accept the wrong answer $1 - g(y)$ with probability larger than $1/3$ (because soundness always holds, in particular for inputs of length $|y''|$). Furthermore, there are infinitely many input lengths ℓ for which the (honest) prover can find a good input length between $d\ell$ and $d(\ell+1)$ where the {SV–io–AMTIME} $(2^{O(\beta n)})$ protocol for f works well, and therefore on these input lengths the prover can convince the verifier to accept $g(y)$ with probability at least $2/3$. □

3.6. Deterministic Amplification. We will use explicit constructions of dispersers to reduce the error probability of algorithms and generators.

DEFINITION 3.14 (Sipser 1988). *A function $Dis : \{0,1\}^{\widehat{m}} \times \{0,1\}^t \to \{0,1\}^m$ is a (u,η)-disperser if for every set $S \subseteq \{0,1\}^{\widehat{m}}$ of size 2^u, $\rho(Dis(S,\cdot)) \geq 1 - \eta$.*

The following technique is taken from Sipser (1988) (see also the survey papers Nisan (1996); Nisan & Ta-Shma (1999); Shaltiel (2002) for more details). Say we have a one-sided probabilistic algorithm A using m random coins and having success probability $\frac{1}{2}$. We design a new algorithm $\widehat{A}$ using a few more random bits and having a much larger success probability γ, as follows. We use a $(\widehat{m} - \log(\frac{1}{1-\gamma}), \frac{1}{2})$-disperser $Dis : \{0,1\}^{\widehat{m}} \times \{0,1\}^t \to \{0,1\}^m$. Algorithm $\widehat{A}$ picks $x \in \{0,1\}^{\widehat{m}}$ and accepts the input iff for some $r \in \{0,1\}^t$ A accepts with the random coins $Dis(x,r) \in \{0,1\}^m$. It is not difficult to verify (and we do that soon) that $\widehat{A}$ success probability is at least γ.

We need this amplification in two settings. One, is where we want to amplify the success probability of AM protocols. The other, is where we want to amplify the hitting properties of a generator. I.e., given a generator that is hitting very-large sets, we want to design a new generator that is hitting even smaller sets. Details follow.

3.6.1. Amplifying AM. Following Miltersen & Vinodchandran (1999) we need AM protocols to have extremely small error probability, not only small with respect to the input length but also small with respect to the number of random coins. Using dispersers we have:

LEMMA 3.15. *(Implicit in Miltersen & Vinodchandran (1999)) There exists some constant $\Delta > 1$, such that for every $0 < \delta < 1$, $AMTIME_{1/2}(n^c) \subseteq AM_{2^{-m+m^{\delta}}}(TIME = m^{2\Delta}, COINS = m = O(n^{c/\delta}))$.*

Similar amplifications of the success probability of single-valued Arthur-Merlin protocols are also true.

3.6.2. Amplifying Hitting-Set Generators. Given a generator that is hitting very large sets we want to design a new generator that is hitting even smaller sets. The penalty is that the new generator uses a (slightly) larger seed and outputs a (slightly) shorter sequence. The following lemma shows how to do that for the case of generators against co-nondeterministic circuits, which is the relevant class for this paper. However, the same arguments apply for other classes as well.

LEMMA 3.16. *Let $G_\ell : \{0,1\}^{k(\ell)} \to \{0,1\}^{m(\ell)}$ be an efficient generator with $k(\ell) \geq \log(m(\ell)))$. Then, there exists another efficient generator $\widehat{G}_\ell : \{0,1\}^{\widehat{k}(\ell)} \to \{0,1\}^{\widehat{m}(\ell)}$ with*

- $\widehat{k}(\ell) = k(\ell) + O(\log(m(\ell))) = O(k(\ell))$,
- $\widehat{m}(\ell) = m - \log(\frac{1}{1-\gamma})$, *and such that*
- *For every algorithm $\widehat{D}$ running in co-nondeterministic time $T \geq m$ there exists an algorithm D running in co-nondeterministic time poly(T) such that*
  - *If $\rho(\widehat{D}) \geq \frac{1}{2}$ then $\rho(D) \geq \gamma$.*
  - *If $\widehat{D}$ $\frac{1}{2}$-distinguishes $\widehat{G}_\ell$ then D γ-distinguishes G_ℓ.*

PROOF. We use an explicit construction of a $(\widehat{m} = m - \log(\frac{1}{1-\gamma}), \frac{1}{2})$-disperser

$$Dis : \{0,1\}^m \times \{0,1\}^{t=O(\log(m))} \to \{0,1\}^{\widehat{m}}$$

given by Saks *et al.* (1998) (see also Ta-Shma (1998); Ta-Shma *et al.* (2001)). We define

$$\widehat{G}(seed, r) = Dis(G(seed), r)$$

Define $D : \{0,1\}^m \to \{0,1\}$ by $D(x) = 1$ iff there exists some $r \in \{0,1\}^t$ such that $\widehat{D}(x,r) = 1$. Note that D can be implemented in co-nondeterministic time poly$(T, 2^t) =$ poly(T) . Now,

- Suppose $\rho(\widehat{D}) \geq \frac{1}{2}$. I.e., if we let $S \subseteq \{0,1\}^{\widehat{m}}$ be the set of $\widehat{x} \in \{0,1\}^{\widehat{m}}$ such that $\widehat{D}(\widehat{x}) = 0$, then $\rho(S) \leq \frac{1}{2}$. Let $X \subseteq \{0,1\}^m$ be the set of $x \in \{0,1\}^m$ such that for every $r \in \{0,1\}^t$, $Dis(x,r) \in S$. By the definition of dispersers we have that $|X| \leq 2^{\widehat{m}} = 2^{m-\log(\frac{1}{1-\gamma})}$. Thus, $1 - \rho(D) \leq 2^{-\log(\frac{1}{1-\gamma})} = 1 - \gamma$ and $\rho(D) \geq \gamma$.
- Suppose that in addition $\widehat{D} : \{0,1\}^{\widehat{m}} \to \{0,1\}$ is a $\frac{1}{2}$-distinguisher for $\widehat{G}$. I.e., for every $(seed, r)$ we have $\widehat{D}(\widehat{G}(seed, r)) = 0$. It follows that $D(G(seed)) = 0$ for every *seed*, and D is a γ-distinguisher for G_ℓ.

$\square$

4. The MV-generator and resilient reconstructions

4.1. The Miltersen-Vinodchandran Generator. Let $f : \{0,1\}^\ell \to \{0,1\}$. We look at f as a d-variate polynomial $f : H^d \to \{0,1\}$ where $|H| = h = 2^{\ell/d}$. For a field F with $q \geq 2h$ elements and $H \subseteq F$, let $\hat{f}$ be the low degree extension of f Babai *et al.* (1991a). That is, $\hat{f}$ is the unique multi-variate polynomial $\hat{f} : F^d \to F$ that extends f and has degree at most $h-1$ in each variable. We let $e_i = (0,\ldots,0,1,0,\ldots,0)$ be the i'th basis vector of F^d, with one in the i'th coordinate and zeros everywhere else. For $w \in F^d$, the points $\{w + ae_i \mid a \in F\}$ lie on an axis-parallel line that passes through w with direction i. The restriction of the multi-variate polynomial $\hat{f}$ to that line is a univariate polynomial of degree at most $h-1$, and we denote it by $\hat{f}|_{w+Fe_i}$. The generator $\mathrm{MV} : \{0,1\}^k \to \{0,1\}^m$,

$$\mathrm{MV} = \mathrm{MV}_{f,\ell,m,d,h,q} : [d] \times F^d \to F_{h-1}$$

is defined by:

$$\mathrm{MV}_{f,\ell,m,d,h,q}(i,w) = \hat{f}|_{w+Fe_i}$$

where F_{h-1} is the set of all degree $h-1$ univariate polynomials over F. Note that MV is a black-box generator. We often omit some (or all) of the subscripts. We now fix some of the parameters involved in the construction as a function of ℓ and some auxiliary parameter $0 < \delta < 1$. We choose: $q = 2h$ and $d = 1/\delta$. This makes $h = 2^{\delta\ell}$. We also require that $\ell \geq \Omega(1/\delta)$. When we analyze parameters we often look at the generator as a binary function $\mathrm{MV}_{f,\ell,\delta} : \{0,1\}^k \to \{0,1\}^m$ and we see that

- $k = \log(d) + \log(q^d) \leq 2\ell$.
- $m = h\log(q) \geq 2^{\delta\ell}$ (this is because $2^m = q^h$). We truncate the output of MV to be of length exactly $2^{\delta\ell}$.
- if $f \in \mathrm{DTIME}(2^{O(\ell)})$ then for every $0 < \delta < 1$, $\mathrm{MV}_{f,\ell,\delta} \in \mathrm{DTIME}(2^{O(\ell)})$.

The parameter $\delta > 0$ will be a constant, and under this choice the generator has "exponential stretch" and stretches 2ℓ bits into $2^{\delta\ell}$ bits.

Miltersen and Vinodchandran show that if f is sufficiently hard then this is a hitting-set generator for co-nondeterministic circuits, from which they derive a non-uniform hardness vs. randomness tradeoff for AM.

4.2. Resilient Reconstruction Algorithms. We define the notion of reconstruction algorithms for pseudo-random generator that are based on hard functions. The definition below is specialized to the case of generators which fool nondeterministic circuits. Let $G_\ell : \{0,1\}^{k(\ell)} \to \{0,1\}^{m(\ell)}$ be a black-box generator.

Definition 4.1 (Reconstruction algorithm). *A deterministic machine $R(\cdot,\cdot,\cdot)$ is a γ-reconstruction algorithm for G_ℓ with success probability p and complexity $T = T(\ell,t,\gamma)$, if*

- *For every $\ell \in N$ and every function $f : \{0,1\}^\ell \to \{0,1\}$, and,*

- *For every size t co-nondeterministic circuit D that is γ-distinguishing $G_f : \{0,1\}^{k(\ell)} \to \{0,1\}^{m(\ell)}$*

it holds that

$$\Pr_s \left[\ \exists_a \text{ s.t. } C = R(D,a,s) \text{ is a nondeterministic TSV circuit that computes } f\ \right] \quad \geq \quad p$$

and where the size of the circuit C is at most $T = T(t,\gamma,\ell)$. A reconstruction algorithm R is called efficient *if it runs in time polynomial in its output length T.*

We are interested in the behavior of the reconstruction algorithm R when given an "incorrect" advice, i.e, when D isn't a distinguisher or when a isn't the correct string. Clearly, we can not expect R to output the correct circuit given an incorrect advice. We can, however, hope that R outputs a PSV circuit even when given an incorrect (or malicious) advice. We call such a reconstruction algorithm *resilient*.

DEFINITION 4.2 (Resilient reconstruction algorithm). *A γ-reconstruction algorithm $R(D,a;r)$ is* resilient *against a co-nondeterministic circuit D with probability p, if*

$$\Pr_s \left[\forall a \ R(D,a,s) \text{ is PSV}\right] \quad \geq \quad p.$$

A reconstruction algorithm is γ-resilient if it is a γ-reconstruction and it is resilient against any circuit D with $\rho(D) > \gamma$.

4.3. A Resilient Reconstruction Algorithm For The MV-generator. Our main observation is that the reconstruction algorithm for MV_f that is given in Miltersen & Vinodchandran (1999) (with slightly different parameters) is γ–resilient, for some (large) $\gamma < 1$.

LEMMA 4.3. *Let $\delta > 0$ and $\ell \in N$ such that $\ell > \frac{1}{\delta^2}$. There exists an efficient $\gamma = 1 - 2^{m^\delta - m}$-resilient reconstruction algorithm R for $MV_{f,\ell,\delta}$, with success probability $p = 1 - 2^{-m^\delta}$ and complexity $T = O(2^{12\delta\ell} \cdot t^2)$.*

PROOF. We basically repeat the Miltersen-Vinodchandran proof that a reconstruction algorithm exists and we note that the reconstruction is resilient. Suppose D is a co-nondeterministic circuit that is γ–distinguishing $\mathrm{MV}_{f,\ell,\delta}$. That is, if we denote

$$\begin{aligned} I &= \mathrm{IMAGE}(\mathrm{MV}_f) = \{v \in F_{h-1} \mid \exists_{i,w}\ \mathrm{MV}_f(i,w) = v\} \\ Z &= \mathrm{ZEROS}(D) = \{v \in F_{h-1} \mid D(v) = 0\} \end{aligned}$$

then $I \subseteq Z$ because the generator does not hit any string v that D accepts, and $|Z| < (1-\gamma)2^m = 2^{m^\delta - m} \cdot 2^m = 2^{m^\delta}$ because $\rho(D) \geq \gamma$. Denote, $\mathrm{SIZE}(D) = t(m)$.

Every element $z \in Z$ is an element of F_{h-1} and is associated with some low-degree polynomial. For $q \in F_{h-1}$ and $S = \{x_1, \ldots, x_s\} \subseteq F$, let $q|_S$ be the restriction of q to the set S, i.e., the vector $(q(x_1), \ldots, q(x_s))$. We say that $S \subseteq F$ "splits" Z if for every $q_1 \neq q_2 \in Z$ it holds that $q_1|_S \neq q_2|_S$. The following claim says that a large enough randomly chosen S splits Z with high probability.

CLAIM 4.4. *For a uniformly chosen* $S \subseteq F$, $\Pr(S$ *does not split* $Z) < |Z|^2 2^{-|S|}$.

PROOF. Let $s = |S|$. Fix $q_1 \neq q_2 \in F_{h-1}$. As q_1 and q_2 are different univariate polynomials of degree at most $h-1$, the probability that q_1 and q_2 agree on s randomly chosen elements in the field is at most $(\frac{h-1}{q})^s \leq 2^{-s}$ since we chose q to be $2h$ (this probability is even smaller when S is chosen without repetitions). Taking the union bound over all pairs in Z, the probability that there exists such a bad pair q_1, q_2 is smaller than $\binom{|Z|}{2} 2^{-s} < |Z|^2 2^{-s}$. □

We are now ready to describe the reconstruction algorithm $R(\cdot,\cdot,\cdot)$. The inputs to R are:

- A co-nondeterministic circuit $D(F_{h-1}, \cdot)$ promised to be a γ-distinguisher for MV_f.
- The random string s is a uniformly chosen $S \subseteq F$, where $|S| = 3m^\delta \geq 3 \log |Z|$.
- The "correct" advice string a is $\hat{f}(S^d)$, i.e., the value $\hat{f}(v)$ for every element $v \in S^d$.

R outputs a nondeterministic circuit C which we describe now. The input to C is $y = (y_1, \ldots, y_d) \in F^d$, and its output is $\hat{f}(y_1, \ldots, y_d)$. C successively learns the values $\hat{f}(A_i)$ for $A_i = \{(y_1, \ldots, y_i, s_{i+1}, \ldots, s_d) \mid s_j \in S\}$. $A_0 = S^d$ and so we have $\hat{f}(A_0)$ as an input to R and we can hardwire it into C. $A_d = \{(y_1, \ldots, y_d)\}$, so after d iterations C can output $\hat{f}(A_d) = \hat{f}(y_1, \ldots, y_d)$.

Say we already have the values $\hat{f}(A_i)$, we show how C computes $\hat{f}(A_{i+1})$. For every $s_{i+2}, \ldots, s_d \in S$, C does the following guesses:

- C guesses $q \in F_{h-1}$,
- C guesses z.

C then checks that

- $D(q, z) = 0$, and
- for every $j \in S$, $q(j) = \hat{f}(y_1, \ldots, y_i, j, s_{i+2}, \ldots, s_d)$. This check is possible because for all $j \in S$ we already know $\hat{f}(y_1, \ldots, y_i, j, s_{i+2}, \ldots, s_d)$ (since the point $(y_1, \ldots, y_i, j, s_{i+2}, \ldots, s_d)$ is in A_i).

We will soon show that at this point the only non-rejecting paths are those who guessed the polynomial $q(j) = \hat{f}(y_1, \ldots, y_i, j, s_{i+2}, \ldots, s_d)$. In particular,

$$\hat{f}(y_1, \ldots, y_i, y_{i+1}, s_{i+2}, \ldots, s_d) \quad = \quad q(y_{i+1})$$

After doing that for every $s_{i+2}, \ldots, s_d \in S$ we know all the values in $\hat{f}(A_{i+1})$.

Claim 4.5. *The above algorithm is a resilient reconstruction algorithm for MV_f with parameters as stated in the lemma.*

Proof. **Correctness** : To see that R is a reconstruction algorithm we have to show that when D is a distinguisher, with probability p (over the choice of r) there exists a such that our conclusions are correct in every iteration. Since D is a γ–distinguisher, it must hold that $\rho(D) > \gamma$ and hence $|Z| = |ZEROS(D)| \leq (1-\gamma)2^m = 2^{m^\delta}$. Therefore, by Claim 4.4, with probability at least p, S splits Z. Suppose that S is indeed splitting.

If C guessed a polynomial q for which $D(q) = 1$, then for all z, $D(q;z) = 1$ and C rejects. If, on the other hand, C guessed a polynomial q for which $D(q) = 0$, then for some z, $D(q;z) = 0$. Thus, the surviving paths so far are exactly those who guessed $q \in Z$ and a witness z for that. Next, C checks that q and $q'(j) \stackrel{def}{=} \hat{f}(y_1, \ldots, y_i, j, s_{i+2}, \ldots, s_d)$ agree on S. Notice that $q' \in I \subseteq Z$. However, as both q and q' are in Z, and both agree on S, it must be that $q = q'$ (because S splits Z). We therefore conclude that the correct path guessing $q = q'$ survives, and furthermore, every surviving path guessed q'. It follows that every non-rejecting path computes the value $\hat{f}(A_{i+1})$ correctly. Hence, the algorithm is TSV.

Complexity : The algorithm of the circuit C makes d iterations. In each iteration, for every $s_{i+2}, \ldots, s_d$ the circuit C guesses a polynomial, i.e. C guesses at most $|S|^d$ polynomials (strings of length m) on which it evaluates the circuit D, and each evaluation takes at most $t(m) = \mathrm{SIZE}(D)$ time. Thus, the total running time is: $t(m) \cdot O(d \cdot |S|^d) = t(m) \cdot O(d(3m^\delta)^d) \leq t(m) \cdot O(d3^d m^{\delta d}) \leq t(m) \cdot O(2^{4d} m)$. However, $d = \frac{1}{\delta} \leq \sqrt{\ell} \leq \delta\ell$, and so $2^{4d} \leq 2^{4\delta\ell}$. We also have, $m \leq h^2 = 2^{2\ell/d}$. Altogether, the running time is at most $t(m) \cdot O(2^{6\delta\ell})$. The circuit size is at most the square of this.

Resiliency : Finally, we show that the reconstruction algorithm is γ-resilient with probability p. First note that Claim 4.4 is correct even if D is not a distinguisher, as long as $\rho(D) > \gamma$. This is because $|Z| = |ZEROS(D)| \leq (1-\gamma)2^m = 2^{m^\delta}$ and this is all that is needed in the proof of Claim 4.4. So S splits Z with probability at least p.

Now suppose for contradiction that $\rho(D) > \gamma$, the random set $S \subseteq F$ splits Z, and there is some (incorrect) advice a and some input $y \in F^d$ to $C = R(D, a, r)$, such that C has two different accepting paths on y that result in different values for $\hat{f}(y)$. It must hold then that at some iteration C chooses q_1 in the first path and q_2 in the other, and $q_1 \neq q_2$. Let us look at the first time this happens, and suppose it is during the computation of $val(A_i)$, and when the last values are fixed to some $s_{i+1}, \ldots, s_d \in F$. Since the two paths are accepting, it follows that for every $j \in S$, $val(y_1, \ldots, y_{i-1}, j, s_{i+1}, \ldots, s_d) = q_1(j) = q_2(j)$ (note that $val(y_1, \ldots, y_{i-1}, j, s_{i+1}, \ldots, s_d)$ does not necessarily equal $\hat{f}(y_1, \ldots, y_{i-1}, j, s_{i+1}, \ldots, s_d)$ because the advice $val(A_0)$ may be incorrect and is not necessarily the restriction of $\hat{f}$ to S^d). However, as before, since S splits Z, and $q_1, q_2 \in Z$ agree on S, it must hold that $q_1 = q_2$ contradicting our assumption. Thus, whenever S splits Z, $R(D, a, r)$ is PSV for every possible (correct or incorrect) advice a.

□

This completes the proof of Lemma 4.3. □

5. A generator against uniform co-nondeterministic machines

In this section we construct a generator that hits uniform co-nondeterministic machines under a uniform assumption and prove Theorem 1.1.

- Let $f : \{0,1\}^* \to \{0,1\}$ be the E–complete language from Lemma 3.13, and let $\mathrm{IC}(y, r)$ be the instance checker for it. In particular, on inputs $y \in \{0,1\}^\ell$, $\mathrm{IC}(y, r)$ makes queries of length exactly $\ell' = c\ell$, for some constant c.
- Let $G = G_{\ell'} : \{0,1\}^{k(\ell')} \to \{0,1\}^{m=m(\ell')}$ be a black box generator that has an efficient $\gamma = \gamma(m)$-resilient reconstruction algorithm $R(D, a, s)$ with probability $p = p(m) > 0.99$ and complexity $T = T(t, \ell, \gamma)$.

We use ℓ' as a subscript as we only intend to run G using input lengths of the form $\ell' = c\ell$ for the hard function f. We now describe the main protocol of this paper which will be used in all the proofs to follow. This protocol (which appears in Figure 5.1) is an Arthur-Merlin protocol where the two players are given as input a string $y \in \{0,1\}^\ell$ a bit b and a co-nondeterministic D_m which takes inputs of size m and has total size $t(m)$. Merlin's goal is to convince Arthur that $f(y) = b$. We show that if the circuit D_m fulfills certain requirements (specified below) then the protocol is complete and sound. We call the protocol $R\&IC_{D_m}(y)$, for Reconstruct-and-Instance-Check.

We claim that if $\rho(D_m)$ is large then for every (y, b) such that $f(y) \neq b$, no prover can convince the verifier to accept with probability larger than 0.1. Furthermore, if D_m is a distinguisher for G_f, then for every (y, b) such that $f(y) = b$, there exists a prover who can convince the verifier to accept (y, b) with probability at least 0.9.

Lemma 5.1. *For every co-nondeterministic circuit D_m and every $y \in \{0,1\}^\ell$,*

- *If $\rho(D_m) \geq \gamma(m)$, then for every prover P^* the verifier accepts $(y, 1 - f(y))$ with probability at most* 0.1.
- *If D_m γ–distinguishes G_f, then there exists a prover for which the verifier accepts $(y, f(y))$ with probability at least* 0.9.

Proof. Assume $\rho(D_m) \geq \gamma(m)$ and let the prover be arbitrary. By Definition 4.2, for almost all s (except for $1 - p$ fraction) for all possible values a, $C = R(D, a, s)$ is PSV. Thus, C defines a partial function $g : \{0,1\}^{\ell'} \to \{0,1\}$. We now run the instance checker for f over the input y with an oracle access to g (note that the input lengths are right, since on input length ℓ, the oracle calls are of length $\ell' = c\ell$). By Lemma 3.13 we get the correct answer $f(y)$ or "reject" with probability at least, say, 0.9.

Now, further assume that D_m γ–distinguishes G_f. By Definition 4.1, with probability at least p over s, there exists a witness a such that $C = R(D, a, s)$ defines a TSV circuit computing f. When interacting with the honest prover, Merlin sends this witness a and the right answers of the instance checker IC as specified by the protocol. Whenever C is indeed

Input :

- $y \in \{0,1\}^{\ell}, b \in \{0,1\}$. Merlin wants to convince Arthur that $f(y) = b$.
- A co-nondeterministic circuit D_m on m input bits.

Protocol :

1. Arthur: Sends a random s.
2. Merlin: Sends a.
3. Let C be the non-deterministic circuit $C = R(D, a, s)$ getting inputs of length ℓ'.
4. Arthur: Sends a randomly chosen string r to be used as the instance checker random coins.
5. Merlin:
   - Runs $\mathrm{IC}^C(y, r)$. I.e., it runs the instance checker (from Lemma 3.13) on the input y, with the random coins r and using the non-deterministic circuit C as an oracle.
   - Sends the queries and the answers of the oracle C during the execution of $\mathrm{IC}^C(y, r)$, and,
   - For each pair of query q and answer a, gives a witness w such that $C(q; w) = a$.
6. Arthur: Runs $\mathrm{IC}^C(y, r)$ verifying the queries and the answers (using the witnesses).

Output : $\mathrm{IC}^C(y, r)$.

Figure 5.1: Protocol $R\&IC_D(y)$ - Reconstruct and Instance Check

a TSV circuit for f, Lemma 3.13 guarantees that IC^C computes f correctly on inputs of length ℓ with probability 1. Thus, the protocol accepts with probability at least p on inputs of length ℓ.

□

We now check the running time of Protocol $R\&IC$:

Claim 5.2. *The interactive protocol of Figure 5.1 takes $poly(\ell) \cdot poly(T(t(m), \ell', \gamma))$ time.*

Proof. The size of the advice string a and the circuit $|C|$ are at most $T' = T(t(m), \ell', \gamma)$, the reconstruction algorithm R is efficient, hence the complexity of steps 1, 2 and 3 is $poly(T')$. Sending r takes $poly(\ell') = poly(\ell)$ bits. There are $\mathrm{poly}(\ell)$ steps of the instance checker IC, and each such step may involve a computation of the circuit C. Altogether, the running time of steps 4, 5 and 6 is $\mathrm{poly}(\ell) \cdot T'$.

□

We get the following corollary:

LEMMA 5.3.

- *If G_f is not [io]–γ–hitting against coNTIME$(t(m))$ then f has an SV–AMTIME(poly$(\ell)\cdot$ poly$(T(t(m),\ell',\gamma))$) protocol.*
- *If G_f is not γ–hitting against coNTIME$_\gamma(t(m))$ then f has an {SV–io–AMTIME} (poly$(\ell)\cdot$ poly$(T(t(m),\ell',\gamma))$) protocol.*

PROOF. If G_f is not γ–hitting against coNTIME$_\gamma(t(m))$ then there exists a uniform machine $D \in co$NTIME$_\gamma(t(m))$ that is a γ–distinguisher for G_f on infinitely many input lengths $m = m(\ell')$. If G_f is not [io]–γ–hitting then there exists such a machine M that is a γ–distinguisher for G_f for all input lengths $m = m(\ell')$, except for possibly finitely many. In both cases, by Lemma 5.1, for every such input length $m = m(\ell')$, and for every value $y \in \{0,1\}^\ell$, there exists a proof for which the protocol accepts the correct result with probability at least 0.9, and there is no proof that makes the verifier accept the wrong value with probability more than 0.1. This gives completeness for both cases, and soundness for the first.

In addition we know that for every input length $m = m(\ell')$ $\rho(D_m) \geq \gamma$ (by the definition of the class coNTIME$_\gamma(t(m))$). So by the first part of Lemma 5.1, for every input $y \in \{0,1\}^\ell$ and every prover, the probability the prover convinces the verifier to accept the wrong answer is at most 0.1, which gives soundness for the second case.

$\square$

5.1. Working With The MV Generator. We are now ready to prove Theorem 1.1. We do that by plugging the MV-generator and its resilient reconstruction algorithm into protocol $R\&IC$.

Let $\delta > 0$ be a constant. We will choose δ later, and for the time being we express other parameters in terms of δ. Let f be the E-complete language from Lemma 3.13 and IC be the instance checker for f, having queries of length exactly $\ell' = c\ell$ on inputs of length ℓ. We let $G_{f,\ell',\delta} = MV_{f,\ell',\delta}$. (recall that $d = 1/\delta$ and $m = 2^{\delta\ell'}$). We assume that ℓ is large enough so that $\ell > 1/\delta^2$. Recall that $G_{f,\ell',\delta}$ has an efficient $\gamma = 1 - 2^{m^\delta - m}$–resilient reconstruction algorithm with probability $p = 1 - 2^{-m^\delta}$ and complexity $T(t,\ell,\gamma) = 2^{O(\delta\ell)} \cdot t^2$ (Lemma 4.3). Let $\widehat{G}_{f,\ell',\delta}$ be the efficient generator defined in Lemma 3.16.

LEMMA 5.4.

- *If $\widehat{G}_{f,\ell',\delta}$ is not [io]–$\frac{1}{2}$–hitting against coNTIME$_{\frac{1}{2}}(n^{O(1)})$ then f has a SV–AMTIME$(2^{O(\delta\ell)})$ protocol.*
- *If $\widehat{G}_{f,\ell',\delta}$ is not $\frac{1}{2}$–hitting against coNTIME$_{\frac{1}{2}}(n^{O(1)})$ then f has a {SV–io–AMTIME} $(2^{O(\delta\ell)})$ protocol.*

In both cases the constant in the $O()$ notation is independent of δ.

PROOF. We do the second statement, the first is essentially similar (and simpler). If $\widehat{G}_{f,\delta}$ is not $\frac{1}{2}$–hitting against $co\mathrm{NTIME}_{\frac{1}{2}}(n^{O(1)})$, then there exists some $\widehat{D} \in co\mathrm{NTIME}_{\frac{1}{2}}(n^{O(1)})$ that for infinitely many input lengths n, $\frac{1}{2}$–distinguishes $\widehat{G}_{f,\delta}$. By Lemma 3.16 there exists D such that:

- For every input length n, $\rho(D_n) \geq \gamma$, and,
- For every input length where $\widehat{D}$ $\frac{1}{2}$–distinguishes $\widehat{G}_{f,\delta}$, D γ–distinguishes $G_{f,\delta}$.

I.e., $D \in co\mathrm{NTIME}_{\gamma}(n^{O(1)})$ and $G_{f,\ell',\delta}$ is not γ–hitting D. Recall that $t(m) = SIZE(D) = m^{O(1)} = 2^{O(\delta\ell)}$. By Lemma 5.3,

$$
\begin{aligned}
f \in \{\text{SV–io–AMTIME}\}\ (T(t(m), \ell', \gamma)) &= \{\text{SV–io–AMTIME}\}\ (2^{O(\delta\ell)} t^2(m)) \\
&= \{\text{SV–io–AMTIME}\}\ (2^{O(\delta\ell)})
\end{aligned}
$$

□

Combining Lemma 5.4 and Lemma 3.13 we get Theorem 1.1 as we rephrase it more formally:

THEOREM 5.5. *Let $c > 0$.*

- *If for every $\delta > 0$ $\widehat{G}_{f,\ell',\delta}$ is not [io]–$\frac{1}{2}$–hitting against coNTIME(n^c) then $E \subseteq \bigcap_{\beta>0}$ AMTIME($2^{O(\beta\ell)}$).*
- *If for every $\delta > 0$ $\widehat{G}_{f,\ell',\delta}$ is not $\frac{1}{2}$–hitting against coNTIME$_{\frac{1}{2}}$(n^c) then $E \subseteq \bigcap_{\beta>0}$ {io–AMTIME} ($2^{O(\beta\ell)}$).*

6. Explicit Constructions Under Uniform Assumptions

Klivans & van Melkebeek (1999) suggested a general framework for derandomization under hardness assumptions. They showed that non-uniform hardness vs. randomness tradeoffs can be used to conditionally derandomize a broader class of randomized processes beyond decision problems. They give some examples that demonstrate the usefulness of their approach. Some of these applications concern "explicit construction of combinatorial objects". We observe that in some cases *uniform* hardness vs. randomness tradeoffs suffice and we can use a weaker assumption than that of Klivans & van Melkebeek (1999); Miltersen & Vinodchandran (1999). We describe a general framework for derandomizing probabilistic constructions of combinatorial objects under uniform assumptions. Let us start by defining the notion of explicit and probabilistic constructions.

Definition 6.1. *Let $Q = \{Q_n\}_{n\geq 1}$, $Q_n \subseteq \{0,1\}^n$, be a property of strings. We say that a procedure A is an explicit construction for the property Q, if A runs in deterministic polynomial-time and on input 1^n outputs $x \in Q_{n'}$, for some $n' \geq n$. We say that A is a probabilistic construction for Q, if A runs in deterministic polynomial-time, and on input $(1^n, \rho)$, where $|\rho| = poly(n)$, it holds that $\Pr_\rho[A(1^n, \rho) \in Q_{n'}] > 1/2$. Infinitely often (i.o.) construction algorithms are similarly defined, where the algorithms are required to succeed only on infinitely many input lengths.*

Let Q be a property and A a probabilistic construction for Q. We need the following to hold in order to apply our approach.

1. $Q \in co\mathrm{NP}$.
2. There exists a deterministic polynomial-time procedure B that given a list, $x_1, \ldots, x_k$, of strings in $\{0,1\}^n$ such that for at least one $1 \leq i \leq k$, $x_i \in Q_n$, B outputs $x \in Q_{n'}$ for some $n' \geq n$.

Lemma 6.2. *Let Q be a property and A a probabilistic construction for Q such that conditions 1 and 2 hold, then Q has an explicit construction algorithm, unless $E \subseteq \bigcap_{\beta>0}\{io\text{–}AMTIME\}\ (2^{\beta n})$.*

Proof. For input length n, let $m = |\rho|$ be the length of the second part of A's input. Let f be the E-complete language from Lemma 3.13. Let $\delta > 0$, we define an explicit construction algorithm C for Q: on input 1^n, run the generator $\widehat{G}_{f,\ell',\delta}$ (i.e the generator obtained from applying Lemma 3.16 on the generator $G_{f,\ell',\delta}$ from Section 5.1) on all the possible seeds to obtain a list of strings $\rho_1, \ldots, \rho_k \in \{0,1\}^m$, where $k = \mathrm{poly}(n)$ (since the seed length is $O(\log m) = O(\log n)$). Then run A on $(1^n, \rho_i)$ for all $1 \leq i \leq k$, to obtain a list $x_1, \ldots, x_k \in \{0,1\}^{n'}$ ($n' = poly(n)$). Finally, run the procedure B from condition 2, on $x_1, \ldots, x_k$, to obtain $x \in \{0,1\}^{n''}$ ($n'' = poly(n') = poly(n)$). Output x.

Now suppose that for every $\delta > 0$, for infinitely many input lengths the algorithm C fails to produce elements in Q. We define a (deciding) co-nondeterministic algorithm A', taking inputs $\rho \in \{0,1\}^m$, as follows: on input ρ, run A on $(1^n, \rho)$ to obtain an instance x, accept if $x \in Q$. By condition 1, this is a co-nondeterministic algorithm. By the fact that A is a probabilistic construction algorithm for Q, we know that $\rho(A'_m) \geq \frac{1}{2}$ for every m (where A'_m is the restriction of A' to input length m). We conclude that $A' \in co\mathrm{NTIME}_{\frac{1}{2}}(n^c)$ (for some constant c). Furthermore, for infinitely many input lengths, A'_m does not accept any of the elements generated by $\widehat{G}_{f,\ell',\delta}$. In other words, for every $\delta > 0$ $\widehat{G}_{f,\ell',\delta}$ is not $\frac{1}{2}$–hitting against $co\mathrm{NTIME}_{\frac{1}{2}}(n^c)$. By Theorem 5.5 (second part) $E \subseteq \bigcap_{\beta>0}\{\text{io–AMTIME}\}\ (2^{\beta n})$. □

Remark 6.3. *By using the first part of Theorem 5.5, we can have a version of Lemma 6.2, in which the construction algorithm succeeds infinitely often, unless $E \subseteq \bigcap_{\beta>0} AMTIME(2^{\beta n})$.*

The matrix rigidity problem is a special case of Lemma 6.2. Valiant (1977) showed that a random matrix is rigid, property 1 is clear, and Klivans & van Melkebeek (1999) showed property 2. Together, this gives Theorem 1.5.

7. Gap Theorems

In this section we prove Theorem 1.2, Theorem 1.3 and Theorem 1.4. We start by setting up some parameters and notations, common to all the proofs in this section.

Let $\delta > 0$ be a constant. We will choose δ later, and for the time being express other parameters in terms of δ. Let f be the E-complete language from Lemma 3.13. Set $\ell' = c\ell$ where c is the constant from Lemma 3.13. Let $G_{f,\ell',\delta} = \mathrm{MV}_{f,\ell',\delta}$ as before we require that $\ell' > \frac{1}{\delta^2}$. Recall that $m = \Omega(2^{\delta\ell'})$. By Lemma 4.3, $G_{f,\ell',\delta}$ has an efficient $\gamma = 1 - 2^{m^\delta - m}$–resilient reconstruction algorithm with probability $p = 1 - 2^{-m^\delta}$ and complexity $T(t, \ell, \gamma) = 2^{O(\delta\ell)} \cdot t^2$.

For a language $L \in \mathrm{AMTIME}_\epsilon(\mathrm{TIME} = poly(n), \mathrm{COINS} = m = m(n))$, let M_L be the machine such that:

$$x \in L_n \implies \Pr_{z \in \{0,1\}^m} [\exists_w\ M_L(x; z, w) = 1\ ] = 1 \tag{7.1}$$

$$x \notin L_n \implies \Pr_{z \in \{0,1\}^m} [\exists_w\ M_L(x; z, w) = 1\ ] \leq \epsilon \tag{7.2}$$

The "derandomized" language L' is obtained by replacing the random coins of the AM protocol for L, with all the possible outputs of the generator. That is, let ℓ be the smallest integer such that $2^{\delta\ell'} \geq m$. We will use the generator G based on the function f with input length ℓ'.

$$L'(x) \quad = \quad \begin{cases} 1 & \forall_{z \in \mathrm{IMAGE}(G_{f,\ell',\delta})} \exists_w M_L(x; z, w) = 1 \\ 0 & \text{Otherwise.} \end{cases}$$

We show that $L' \in \mathrm{NP}$. Indeed, L' calls the non-deterministic machine $M_L \cdot |\mathrm{IMAGE}(G_{f,\ell',\delta})|$ times which is at most polynomial in m (and n). Also, since $G_{f,\ell',\delta}$ is efficient, and f is in E, each element in the set is generated in time polynomial in m (and n). Each execution of M_L takes $poly(m) = poly(n)$ non-deterministic time. Altogether, $L' \in \mathrm{NTIME}(n^{O(1)})$ and is in NP.

In Figure 7.1 we present a protocol that is used by all the proofs in this section. We assume that the prover is able to present us with a circuit D_m such that:

- For every $m = m(\ell)$, the prover can prove in $poly(m)$ time that $\rho(D_m) \geq \gamma$, and,
- For some (or all) input lengths, D_m is a distinguisher for $G_{f,\ell',\delta}$. The prover does not prove this part, though.

This assumption will be realized in different ways according to the different statements that we prove.

CLAIM 7.3. *If for every ℓ the prover can choose a circuit D_m that is γ–distinguishing for $G_{f,\ell',m,\delta}$ than $E \subseteq \mathrm{AMTIME}_{1/3}(2^{O(\delta\ell)})$. If the above only holds for infinitely many input lengths ℓ then $E \subseteq \{io\text{–}\mathrm{AMTIME}\}_{1/3}(2^{O(\delta\ell)})$. In both cases the constant in the O notation is independent of δ.*

Input : $y \in \{0,1\}^\ell$, $b \in \{0,1\}$. Merlin wants to prove that $f(y) = b$.

1. Merlin: Sends a co-nondeterministic circuit D_m.
2. Merlin proves to Arthur that $\rho(D_m) \geq \gamma$ (this is the place where the protocols for Theorem 1.2 and Theorem 1.3 differ, and we will later describe how this is done in each proof).
3. Arthur and Merlin play Protocol $R\&IC_{D_m}(y,b)$.

Figure 7.1: Common Protocol for Theorem 1.2 and Theorem 1.3.

PROOF. **Soundness** : For every input length ℓ the prover proves that $\rho(D_m) \geq \gamma$ with sufficient soundness (this is our assumption). The rest follows from the soundness of protocol $R\&IC$ (Lemma 5.1).

Completeness: Whenever D_m is γ–distinguishing for $G_{f,\ell',m,\delta}$, the completeness follows again from Lemma 5.1.

Running time: According to our assumption, the first part takes time $poly(m) = 2^{O(\delta\ell')} = 2^{O(\delta\ell)}$. By Claim 5.2 and Lemma 4.3, Protocol $R\&IC$ runs in time $2^{O(\delta\ell')} = 2^{O(\delta\ell)}$. Altogether, the total running time is $2^{O(\delta\ell)}$ with the constant in the O notation independent of δ.

It follows that in the first case, for every $\beta > 0$, $f \in \mathrm{AMTIME}(2^{\beta\ell})$, and in the second, for every $\beta > 0$, $f \in \{\text{io–AMTIME}\}\ (2^{\beta\ell})$. Lemma 3.13 shows that the above also holds for the whole class E as desired. $\square$

We now prove Theorem 1.2 and Theorem 1.3, which by the discussion above amounts to realizing our assumptions regarding D_m.

7.1. A gap theorem for AM.

PROOF. (of Theorem 1.2) Assume AM $\not\subseteq$ [io-pseudo(NTIME($n^{O(1)}$))]–NP. Then there exists a language $L \in \mathrm{AMTIME}_{1/3}(n^{O(1)})$ such that $L \notin$ [io-pseudo(NTIME($n^{O(1)}$))]–NP. By Lemma 3.15, $L \in \mathrm{AMTIME}_{2^{m^\delta}-m}(\mathrm{TIME} = m^{O(1)}, \mathrm{COINS} = m = O(n^{O(1/\delta)}))$. In Lemma 7.4 (see below) we prove that there exists a polynomial-time nondeterministic procedure A such that for all but finitely many ℓ', on input 1^n has at least one accepting computation path, and on every accepting path, A outputs a co-nondeterministic circuit D_m that is γ–distinguishing for $G_{f,\ell',m,\delta}$. The prover sends this D_m and a proof that $A(1^n) = D_m$, i.e., a computation path of $A(1^n)$ that outputs the circuit D_m. The rest follows now from Claim 7.3. $\square$

Informally, the following lemma says that if a black-box generator fails in the pseudo-setting then a distinguisher for the generator can be efficiently generated.

LEMMA 7.4. *Let $c > 0$ be an arbitrary constant. Let $f : \{0,1\}^{\ell} \to \{0,1\}$ be a language in E, and $G_\ell : \{0,1\}^{k(\ell)} \to \{0,1\}^{m(\ell)}$ an efficient black-box generator with $k(\ell) \leq O(\log(m))$. If*

$$AMTIME_{\epsilon=\epsilon(n)}(TIME = poly(n), COINS(n) = poly(n)) \not\subseteq [\text{io-pseudo}(NTIME(n^c))]\text{–}NP$$

then there exists a procedure $A \in NP$ such that for all but finitely many ℓ, on input $1^{n(\ell)}$, A has at least one accepting path, and on every accepting path A outputs a co-nondeterministic circuit D_m which is a $\gamma = 1 - \epsilon$–distinguisher for $G_{f,\ell,m}$, where $n = n(\ell)$ is the first integer such that $m(\ell) \geq COINS(n)$.

PROOF. Let L be a language in $\text{AMTIME}_\epsilon(\text{TIME} = poly(n), \text{COINS} = m = m(n))$, and yet L is not in $[\text{io-pseudo}(\text{NTIME}(n^c))]$–NP. Let M_L the machine that defines L, and L' the "derandomized" version of L with the generator $G_{f,\ell,m}$ (as defined in Section 7). Recall that $L' \in NP$.

As $L \notin [\text{io-pseudo}(\text{NTIME}(n^c))]$–NP, there exists a refuter $REF \in \text{NTIME}(n^c)$ such that for all large enough input lengths n, $REF(1^n) \in L \triangle L'$. Furthermore, $L \subseteq L'$ and so $REF(1^n) \in L' \setminus L$.

We can describe now the nondeterministic procedure A, that outputs distinguishers for the generator. On input 1^n, A runs the refuter $REF(1^n)$, to obtain an instance x_n (on accepting paths of REF). A outputs the co-nondeterministic circuit $D_m(z)$, with inputs from $\{0,1\}^m$, defined to be $D_m(z) = 1$ iff $\forall_w[\ M_L(x_n; z, w) = 0\ ]$. We see that:

- $\rho(D_m) \geq \gamma$ (because $REF(1^n) \notin L$), and,
- For every $z \in \text{IMAGE}(G_{f,\ell,m})$, $D_m(z) = 0$ (because $REF(1^n) \in L'$).
- $SIZE(D_m) \leq poly(m)$, (because $M_L \in \text{AM}_\epsilon(\text{TIME} = poly(n), COINS = m = poly(n))$, and the translation to a circuit is at most quadratic in size).

Therefore, by definition, D_m is a γ–distinguisher for $G_{f,\ell,m}$.

Clearly, A runs in $poly(n)$ non-deterministic time, which completes the proof. □

7.2. A gap theorem for AM $\cap$ *co*AM.

PROOF. (of Theorem 1.3(i)) Assume AM $\cap$ *co*AM $\not\subseteq$ [io]–NP $\cap$ [io]–*co*NP. It follows that AM $\cap$ *co*AM $\not\subseteq$ [io]–NP and therefore there is a language $L \in \text{AMTIME}_{1/3}(n^{O(1)}) \cap co\text{AMTIME}_{1/3}(n^{O(1)})$ such that $L \notin$ [io]–NP. By Lemma 3.15,[10]

$$L \in \text{AMTIME}_{2^{m^\delta - m}}(m^{O(1)}, m = n^{O(1/\delta)}) \cap co\text{AM}_{\frac{1}{10}}(n^{O(1)}, n^{O(1)}).$$

As before, the derandomized language L' is in NP and $L \notin$ [io]–NP, therefore it must be that for all input lengths, except finitely many, there exists an input $x_n \in L \triangle L' =$

[10]Note that we need the strong amplification of the success probability (i.e. Lemma 3.15) only for the protocol for L but not for $\bar{L}$. For the latter we can use the standard amplification technique of running many independent copies of the protocol in parallel and deciding by majority.

$L' \setminus L$. Merlin sends x_n, and proves that $x_n \notin L$ using the AM protocol for $\bar{L}$. Let D_m be the co-nondeterministic circuit taking inputs from $\{0,1\}^m$, defined to be $D_m(z) = 1$ iff $\forall_w[\ M_L(x_n; z, w) = 0\ ]$. That is, we hardwire x_n into the circuit obtained from M_L which now takes as inputs elements from $\{0,1\}^m$ (the random tape of M_L becomes the input). We see that:

- $\rho(D_m) \geq \gamma$ (because $x_n \notin L$), and,
- For every $z \in \mathrm{IMAGE}(G_{f,\ell',m,\delta})$, $D_m(z) = 0$ (because $x_n \in L'$).

Now the protocol continues as in Figure 7.1 with the circuit D_m.

Soundness : In the first step, the prover sends x_n. If $x_n \in L$, then the prover is caught cheating with probability at least 0.9 during the proof that $x_n \notin L$. Otherwise, $x_n \notin L$, which implies that D_m accepts almost all inputs, i.e., $\rho(D_m) \geq \gamma$. The soundness now follows from Claim 7.3.

Completeness and running time follows directly from Claim 7.3. □

7.3. An "almost everywhere" gap theorem for AM ∩ *co*AM.

Proof. (of Theorem 1.3(ii)) The proof of Theorem 1.3(ii) is slightly more complicated than that of Theorem 1.3(i) because when the derandomization fails it only fails only on infinitely many input lengths. We will have to allow Merlin to also choose a "correct" input length when sending the distinguisher.

We start with the assumption that $\mathrm{AM} \cap co\mathrm{AM} \not\subseteq \mathrm{NP}$, (whereas previously we had an [io]– in the statement) so there is a language $L \in \mathrm{AMTIME}_{1/3}(n^{O(1)}) \cap co\mathrm{AMTIME}_{1/3}(n^{O(1)})$ such that $L \notin \mathrm{NP}$. By Lemma 3.15,

$$L \ \in \ \mathrm{AM}_{2^{m^\delta - m}}(m^{O(1)}, m = O(n^{O(1/\delta)})) \cap co\mathrm{AM}_{\frac{1}{10}}(n^{O(1)}, n^{O(1)}).$$

As before, $L' \in \mathrm{NP}$, however, $L \notin \mathrm{NP}$. It follows that there is an infinite set I of input lengths, such that for every $n \in I$ there exists an input $x_n \in L \triangle L' = L' \setminus L$. We let D_m be the co-nondeterministic circuit taking inputs from $\{0,1\}^m$, defined to be $D_m(z) = 1$ iff $\forall_w[\ M_L(x_n; z, w) = 0\ ]$.

We modify the protocol as follows: Previously n was fixed by Arthur to be $n = m^{\Theta(\delta)} = 2^{c\delta^2\ell}$ for some constant c. We now allow Merlin to choose an input length n between $2^{c\delta^2\ell}$ and $2^{c\delta^2(\ell+1)}$. This is needed to allow Merlin to send an input length on which he can send a counterexample x_n. The rest of the protocol continues unchanged. Namely, Merlin sends x_n, and proves that $x_n \notin L$ using the AM protocol for $\bar{L}$, and the protocol in Figure 7.1 continues with the circuit D_m. □

Completeness : There are infinitely many n's such that there exist $x_n \in \{0,1\}^n$ for which $x_n \in L' \setminus L$. For such n, let $\ell = \lfloor \frac{\log(n)}{\delta^2 c} \rfloor$. So, $2^{\delta^2 c\ell} \leq n \leq 2^{\delta^2 c(\ell+1)}$. For this choice of ℓ the honest Merlin can choose a "good" n. As in the proof of Theorem 1.3(i), the circuit D_m, obtained from x_n, is a distinguisher for $G_{f,\ell',\delta}$ and the completeness follows from Claim 7.3. Thus, for infinitely many ℓ's, the protocol computes f correctly.

Soundness : For every input y, if $x_n \in L$, then the prover is caught cheating with probability at least 0.9 during the proof that $x_n \notin L$. Otherwise, $x_n \notin L$, hence $\rho(D_m) \geq \gamma$, and the soundness follows from Claim 7.3.

7.4. Replacing E by NE$\cap$*co*NE. We now prove Theorem 1.4. In fact we prove a stronger statement by replacing the class NE $\cap$ *co*NE with NEXP $\cap$ *co*NEXP.

THEOREM 7.5. *If NEXP $\cap$ coNEXP $\not\subseteq$ AMTIME$(2^{\beta n})$ for some $\beta > 0$, then:*

(i) AM $\subseteq$ [io-pseudo(NTIME(n^c))]–NTIME(2^{n^ϵ}) for every $c, \epsilon > 0$.

(ii) AM $\cap$ coAM $\subseteq$ [io]–NTIME(2^{n^ϵ}) $\cap$ [io]–coNTIME(2^{n^ϵ}) for every $\epsilon > 0$.

We will need the following from Impagliazzo *et al.* (2002):

THEOREM 7.6. *If NEXP $\cap$ coNEXP $\neq$ EXP then AM $\subseteq$ [io]–NTIME(2^{n^ϵ}) for every $\epsilon > 0$.*

We also need the following version of Theorem 1.2 and Theorem 1.3:

THEOREM 7.7. *If EXP $\not\subseteq$ AMTIME$(2^{\beta n})$ for some $\beta > 0$, then:*

(i) AM $\subseteq$ [io-pseudo(NTIME(n^c))]–NQP for every $c > 0$.

(ii) AM $\cap$ coAM $\subseteq$ [io]–NQP $\cap$ [io]–coNQP.

Where NQP = NTIME$(2^{polylog(n)})$ (nondeterministic quasi-polynomial time).

PROOF. (sketch) The proofs of Theorem 1.2 and Theorem 1.3 work just as well since complete functions in EXP have instance checkers Babai *et al.* (1991b). The loss in running time (from polynomial to quasi-polynomial) stems from the fact that now the generator computes a function in DTIME(2^{l^c}) (for some constant c) on inputs of size $l = O(\log n)$. □

We can now prove Theorem 7.5.

PROOF. We have two cases. Either NEXP $\cap$ *co*NEXP $\neq$ EXP and then by Theorem 7.6 the conclusions hold. Otherwise NEXP $\cap$ *co*NEXP $=$ EXP, and then by the hypothesis of the theorem, EXP $\not\subseteq$ AMTIME$(2^{\beta n})$ for some $\beta > 0$. Now applying Theorem 7.7 gives the desired conclusions. □

REMARK 7.8. *As we mentioned in the introduction, Theorem 7.5(ii) has a nice interpretation in terms of the relative power of randomness in the context of* single-valued *proof systems (note that both the assumption and the conclusion involves single-valued proofs only). We can weaken the assumption in Theorem 7.5 by replacing the class NEXP $\cap$ coNEXP with the class NEXP. The cost is that now the derandomizations are slightly non-uniform (taking advice of size at most n^ϵ for arbitrarily small ϵ). Thus we also get an interpretation regarding the power of randomness in the context of* general *proof systems (not necessarily single-valued), however it is not as clean (involving pseudo-classes and nonuniformity). We omit the details.*

8. Discussion And Open Problems

In this section we discuss directions for further research and present open problems.

8.1. Towards Removing The io–pseudo. In Lemma 4.3 we show that the Miltersen-Vinodchandran generator has a reconstruction algorithm that is resilient against a large family of co-nondeterministic circuits. An interesting open problem is to construct a generator that has a reconstruction algorithm which is resilient against *all* co-nondeterministic circuits. We call such a reconstruction algorithm *completely resilient*. Such a generator will immediately give a version of Theorem 1.2 without the [io-pseudo]– quantifier.

8.2. Towards A Low-End Gap Theorem. Our results are all in the "high-end" setting. That is we assume a hard function against AMTIME($2^{\Omega(n)}$) and conclude a "full derandomization" placing AM in NP. We do not know how to extend these results to the "low-end" setting. That is, assume a hard function against AM and conclude a "weak derandomization" (placing AM in NSUBEXP). This happens because the generator of Miltersen & Vinodchandran (1999) is only known to work in the "high-end" setting.

Recently, Shaltiel & Umans (2001) gave a generator construction that improves that of Miltersen & Vinodchandran (1999) and also works in the "low-end" setting. However, the known reconstruction algorithm for this construction does not seem to be resilient.

Another approach to try and get a "low-end" result is to use the technique of the gap theorem of Impagliazzo & Wigderson (1998). This paper uses a different property of reconstruction algorithms: They show that the reconstruction algorithm of Babai *et al.* (1993); Nisan & Wigderson (1994) is *learning*: The "correct" advice string a can be efficiently computed with oracle access to the hard function.

Definition 8.1 (Learning reconstruction algorithm). *A reconstruction algorithm R as in Definition 4.1 is p-learning if there exists a polynomial time oracle machine M^A such that for every function f, and a co-nondeterministic circuit D that is a distinguisher for G_f, with probability p over the choice of r, $R(D, M^f(r), r)$ outputs a nondeterministic TSV circuit C which computes f.*

In Impagliazzo & Wigderson (1998) this property is used to efficiently find the "correct" advice string a when the hard function is "downwards self reducible" (which they can assume in their setting) [11]. We'd like to point out that the reconstruction algorithm for the Miltersen-Vinodchandran generator is learning. We also want to point out that the reconstruction algorithm for Shaltiel & Umans (2001); Umans (2002) isn't learning [12].

8.3. Towards Placing AM In Σ_2^{P}. One of our motivations for studying this problem is the attempt to prove that AM $\subseteq \Sigma_2^{\mathrm{P}}$. We remark that AM is known to be in Π_2^{P} (and

[11]In Trevisan & Vadhan (2002) it was pointed out that this assumption is problematic when trying to extend their result to the "high-end" setting.

[12]The construction of Shaltiel & Umans (2001); Umans (2002) works for derandomizing both BPP and AM. The reconstruction algorithm given there for BPP is "learning". However, the one given for AM isn't and requires additional advice which we do not know how to compute even with oracle access to f

therefore in Σ_3^{P}), and that Santha (1989) constructs an oracle relative to which AM isn't in Σ_2^{P}.

Let us start with explaining why are "gap theorems" helpful to achieve this goal. Suppose that we could prove a "dream version" of the "gap theorem". That is, either EXP = AM or AM = NP. The goal follows as if EXP = AM then AM is closed under complement, and thus AM $= co$AM $\subseteq \Sigma_2^{\mathrm{P}}$.

As we only have weaker versions of the "gap theorem" we obtain much weaker results which have an [io-pseudo]– quantifier. We remark that even somewhat stronger results follow from the methods of Kabanets (2000) [13]. It seems that unlike Kabanets (2000), the use of [io-pseudo]– in our results may not be inherent to our technique - indeed, we do not need [io-pseudo]– when working with MA or AM $\cap$ coAM.

We now discuss the consequences of solving the open problems listed above on placing AM in "variants" of Σ_2^{P}.

- A "high-end gap theorem" (without [io-pseudo]–) will give that for every $\beta > 0$, AM $\subseteq [io]\Sigma_2^{Time(2^{\beta n})}$.
- A "low-end gap theorem" (without [io-pseudo]–) will give that for every $\beta > 0$, AM $\subseteq [io]\Sigma_2^{Time(2^{n^{\beta}})}$.
- It was pointed out to us by Umans (2003) that if there exists a "high-end" generator which has a completely resilient reconstruction that is learning for $p > 1/2$ then AM $\subseteq \Sigma_2^{\mathrm{P}}$. The argument is a modification of the proof of Arvind & Kobler (1997); Nisan & Wigderson (1994) that MA $\subseteq$ ZPP$^{\mathrm{NP}}$.

8.4. A high-end version based on a hard function in NE$\cap co$NE. As we mentioned in the introduction, we cannot prove a high-end version similar to Theorem 1.2 and Theorem 1.3 that is based on hard functions in NE$\cap co$NE (the conclusions in Theorem 1.4 are too weak). The reason is that it is not known that such functions have instance checkers. We leave this is an intriguing open question.

Acknowledgements

We thank Oded Goldreich, Russell Impagliazzo, Valentine Kabanets, Shien Jin Ong, Alon Rosen, Chris Umans, Salil Vadhan and Avi Wigderson for helpful discussions. We thank Oded Goldreich and Rahul Santhanam for very thorough and very helpful comments on the

[13] In Kabanets (2000) it was shown that for every $\beta > 0$, RP $\subseteq$ [io-pseudo($ZP-SUBEXP$)]–$ZPTIME(2^{n^{\beta}})$. The same technique gives that for every $\beta > 0$,

$$\mathrm{AM} \subseteq \text{[io-pseudo}(C)]\text{–}\Sigma_2^{Time(2^{n^{\beta}})}$$

where C allows refuters which run in $\Sigma_2^{Time(2^{n^{\beta}})}$. We remark that a slightly weaker result follows without delving into Kabanets (2000), by just observing that the argument of Kabanets (2000) relativizes, using the argument there with an NP oracle and observing that AM $\subseteq co$RP$^{\mathrm{NP}}$.

paper. The first author wishes to thank his supervisor, Michael Ben-Or, for his encouragement and support.

The first author is supported in part by the Leibniz Center, the Israel Foundation of Science, a US-Israel Binational research grant, and an EU Information Technologies grant (IST-FP5). The second author is supported by the Koshland Scholarship.

References

W. Aiello & J. Hastad (1991). Statistical Zero-Knowledge Languages can be Recognized in Two Rounds. *Journal of Computer and System Sciences* **42**(3), 327–345.

S. Arora & S. Safra (1998). Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM* **45**(1), 70–122.

V. Arvind & J. Kobler (1997). On resource-bounded measure and pseudorandomness. In *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science*, 235–249.

L. Babai, L. Fortnow, L. Levin & M. Szegedy (1991a). Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, 21–31.

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

L. Babai, L. Fortnow, N. Nisan & A. Wigderson (1993). BPP Has Subexponential Time Simulations Unless EXPTIME has Publishable Proofs. *Computational Complexity* **3**(4), 307–318.

L. Babai & S. Moran (1988). Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences* **36**, 254–276.

B. Barak, S.J. Ong & S. Vadhan (2003). Derandomization in Cryptography. In *Proceedings of the 23rd international Cryptography conference.*

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

M. Blum & S. Micali (1984). How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing* **13**(4), 850–864.

J. Friedman (1990). A note on matrix rigidity. Technical Report TR-308-91, Department of Computer Science, Princeton University.

M Furer, O. Goldriech, Y. Mansour, M. Sipser & S. Zachos (1989). On completeness and soundness in interactive proof systems. *Advances in Computing Research 5:Randomness and Computation* 429–442.

O. Goldreich & S. Goldwasser (1998). On the Limits of Non-Approximability of Lattice Problems. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1–9.

O. Goldreich, S. Micali & A. Wigderson (1991). Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM* **38**(1), 691–729.

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

S. Goldwasser & M. Sipser (1989). Private coins versus public coins in interactive proof systems. *Advances in Computing Research (5): Randomness and Computation* 73–90.

R. Impagliazzo, V. Kabanets & A. Wigderson (2002). In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences* **64**(4), 672–694.

R. Impagliazzo, R. Shaltiel & A. Wigderson (1999). Near-Optimal Conversion of Hardness into Pseudo-Randomness. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 181–190.

R. Impagliazzo, R. Shaltiel & A. Wigderson (2000). Extractors and pseudo-randomn generators with optimal seed-length. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 1–10.

R. Impagliazzo & A. Wigderson (1997). P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 220–229.

R. Impagliazzo & A. Wigderson (1998). Randomness vs. Time: De-randomization under a niform assumption. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, 734–743.

V. Kabanets (2000). Easiness Assumptions and Hardness Tests: Trading Time for Zero Error. In *Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, 150–157.

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

R. M. Karp & R. J. Lipton (1980). Some Connections Between Nonuniform and Uniform Complexity Classes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, 302–309.

A. R. Klivans & D. van Melkebeek (1999). Graph Nonisomorphism Has Subexponential Size Proofs Unless The Polynomial-Time Hierarchy Collapses. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 659–667.

C.J. Lu (2001). Derandomizing Arthur-Merlin Games under Uniform Assumptions. *Computational Complexity* **10**(3), 247–259.

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

N. Nisan (1996). Extracting Randomness: How and Why: A Survey. In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, 44–58.

N. Nisan & A. Ta-Shma (1999). Extracting Randomness: A Survey and New Constructions. *Journal of Computer and System Sciences* **58**, 148–173.

N. Nisan & A. Wigderson (1994). Hardness vs Randomness. *Journal of Computer and System Sciences* **49**(2), 149–167.

P. Pudlak & Z. Vavrin (1991). Computation of rigidity of order n^2/r for one simple matrix. *Comment. Math. University of Carolina* **32**, 213–218.

A. Razborov (1989). On rigid matrices. Manuscript (in Russian).

A. Sahai & S. Vadhan (1997). A Complete Promise Problem for Statistical Zero-Knowledge. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, 448–457.

M. Saks, A. Srinivasan & S. Zhou (1998). Explicit OR-Dispersers with Polylogarithmic Degree. *Journal of the ACM* **45**(1), 123–154.

M. Santha (1989). Relativized Arthur-Merlin versus Merlin-Arthur games. *Information and Computation* **80**(1), 44–49.

R. Shaltiel (2002). Recent Developments in Explicit Constructions of Extractors. *Bulletin of the European Association for Theoretical Computer Science* **77**, 67–.

R. Shaltiel & C. Umans (2001). Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, 648–657.

M. Sipser (1988). Expanders, Randomness, or Time Versus Space. *Journal of Computer and System Sciences* **36**(3), 379–383.

M. Sudan, L. Trevisan & S. Vadhan (1999). Pseudorandom generators without the XOR Lemma. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 537–546.

A. Ta-Shma (1998). Almost Optimal Dispersers. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 196–202.

A. Ta-Shma, C. Umans & D. Zuckerman (2001). Loss-less Condensers, Unbalanced Expanders, and Extractors. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 143–152.

L. Trevisan & S. Vadhan (2002). Pseudorandomness and Average-Case Complexity via Uniform Reductions. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, 129–138.

C. Umans (2002). Pseudo-Random Generators for all Hardnesses. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, 627–634.

C. Umans (2003). Personal communication.

L.G. Valiant (1977). Graph-theoretic arguments in low-level Complexity. In *Proceedings of the 6th Symposium on Mathematical Foundations of Computer Science, vol. 53 of LNCS, Springer-Verlag*, 162–176.

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

Manuscript received March 30, 2004.

Dan Gutfreund
School of Computer Science and Engineering,
The Hebrew University of Jerusalem,
Israel, 91904.
danig@cs.huji.ac.il

Ronen Shaltiel
Dept. of Applied Mathematics and
Computer Science,
Weizmann Institute of Science,
Rehovot, Israel, 76100.
ronens@wisdom.weizmann.ac.il

Amnon Ta-Shma
Computer Science Dept,
Tel-Aviv University, Israel, 69978
amnon@post.tau.ac.il