MIT/LCS/TM-433

RANDOMNESS-EFFICIENT SAMPLING OF ARBITRARY FUNCTIONS

Mihir Bellare
John Rompel

July 1990

Randomness-Efficient Sampling of Arbitrary Functions

Mihir Bellare* John Rompel†

MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Abstract

We consider the problem of approximating the average value of an arbitrary function defined over a large space (say of size 2^l) in a randomness-efficient manner (i.e. using few coin tosses). The sampling method generates $poly(\epsilon^{-1}, \log \delta^{-1}, l)$ sample points using $O(l + \log \delta^{-1} \cdot \log l)$ coin tosses, with the guarantee that with probability $\geq 1 - \delta$, the average of the function values at the sample points differs from the average value of the function by at most ϵ.

As an application we show how to reduce the error probability of Arthur-Merlin games to an exponentially small amount in a randomness-efficient manner.

Keywords: randomness, pseudo-randomness, sampling, universal hash functions, interactive proofs, Arthur-Merlin games.

* Partially supported by ARO grant No. DAAL 03-86-K-0171.
† Supported by an NSF graduate fellowship.

1 Introduction

The problem of approximating the average value $\mathbf{E}[f] \stackrel{\text{def}}{=} 2^{-l} \sum_{x\in\{0,1\}^l} f(x)$ of an arbitrary function $f : \{0,1\}^l \to [0,1]$ arises in many applications. We will be interested in randomness-efficient constructions of a primitive (which we call (l, ϵ, δ)-sampling) for implementing such approximations. We begin by discussing this primitive and related work. We then describe the application which motivated its construction: randomness-efficient error-reduction for Arthur-Merlin games.

1.1 (l, ϵ, δ)-Sampling and Our Result

An (l, ϵ, δ)-sampler may be informally described as a randomized, polynomial time process which outputs a sequence of m *sample points* $x_1, \ldots, x_m \in \{0,1\}^l$ such that: for any function f : $\{0,1\}^l \to [0,1]$ it is the case that

$$\mathbf{P}\left[\left|\tfrac{1}{m}\sum_{i=1}^{m} f(x_i) - \mathbf{E}[f]\right| \le \epsilon\right] \ge 1-\delta\ .$$

Notice that we place no restriction (as of polynomial time computability, for example) on the function f – the sampler must be able to approximate *any* function with high probability. Moreover the sampler is independent of the function f it will approximate – in particular, it does not evaluate the function. Both these properties are important to our application.

We will be interested in designing samplers which use few random bits. (A related concern is the number m of sample points which must remain at least polynomial). Previous randomness-efficient sampling techniques have suffered from one of two drawbacks: either (1) they serve to approximate only a restricted class of functions (such as boolean functions [AKS],[CoWi],[IZ]), or (2) they generate a number of sample points proportional to δ^{-1} (rather than $\log\delta^{-1}$) and thus cannot be used when the desired error probability is exponentially small. We present a construction which suffers from neither of these drawbacks: our main result is the construction of a (l, ϵ, δ)-sampler which outputs *poly*$(\epsilon^{-1}, \log\delta^{-1}, l)$ sample points using $O(l + \log\delta^{-1} \cdot \log l)$ coin tosses.

1.2 Previous and Related Work

The straightforward procedure for sampling is of course to just select $m = O(\epsilon^{-2}\log\delta^{-1})$ independent and uniformly distributed sample points. This yields an (l, ϵ, δ)-sampler at the cost of $O(ml)$ coin tosses.

Dramatic savings in the number of coin tosses is possible by selecting $O(\epsilon^{-2}\delta^{-1})$ pairwise independent sample points (cf. [CG]). This yields a (l, ϵ, δ)-sampler at the cost of $2l$ coin tosses. However, the number of sample points here grows inversely proportional to the desired error probability δ, and thus this method cannot be applied when this desired error probability is exponentially small.

An alternative sampling method for the special case of boolean valued functions (i.e. f takes on only the values 0 and 1) is based on selecting a random walk on a 2^l node explicitly constructed expander graph (cf. [AKS],[CoWi],[IZ]). This method yields a $(l, \frac{1}{3}, \delta)$-oblivious sampler of boolean functions which outputs $O(\log\delta^{-1})$ sample points using $l + O(\log\delta^{-1})$ coin tosses[†].

A sampling primitive of a slightly different flavor was recently constructed by Goldreich [G]. He outputs a collection of $mk = O(\epsilon^{-2}\log\delta^{-1})$ sample points $x_1^1, \ldots, x_m^1, \cdots, x_1^k, \ldots, x_m^k$ grouped

[†] One can obtain an (l, ϵ, δ)-oblivious sampler of arbitrary functions by using the ideas of [IZ], but this will require $\Omega(\epsilon^{-2}\log\delta^{-1})$ sample points generated using $l + \Omega(\epsilon^{-2}\log\delta^{-1})$ coin tosses.

into $k = O(\log \delta^{-1})$ blocks of $m = O(\epsilon^{-2})$ points each with the property that, with probability $\geq 1 - \delta$, the average value of f on block j (i.e. $\sum_{i=1}^{m} f(x_i^j)$) differs from $\mathbf{E}[f]$ by at most ϵ for a majority of the blocks j. His method uses only $O(l + \log \delta^{-1})$ coin tosses.

Although potentially weaker than (l, ϵ, δ)-sampling, this primitive does suffice for the application to the randomness-efficient error reduction of Arthur-Merlin games which we discuss next.

1.3 Application: Randomness-Efficient Error-Reduction for Arthur-Merlin Games

An *Arthur-Merlin game* [B],[BM] is a two-party protocol played by an all-powerful "prover", called *Merlin*, and a polynomial-time "verifier", called *Arthur*. The game is played on a common input (and its purpose is to convince Arthur that the input belongs to some predetermined language). Arthur's role in the process is restricted to tossing coins, sending their outcome and finally evaluating a polynomial-time predicate applied to the common input and the full transcript of the interaction.

As such they are a special form of interactive proof systems [GMR], but their language recognition power has been shown to be equal to that of interactive proof systems [GS].

Usually we say the a language L possesses an Arthur-Merlin proof system if the error probability on any input w (the probability that Arthur accepts if $w \notin L$ or rejects if $w \in L$) is $\leq \frac{1}{3}$. The error probability can be decreased to 2^{-k} for any $k = k(n) \leq n^{O(1)}$. We are interested in implementations of this error-reduction process which have the additional property of preserving the number of rounds. Our concern will be Arthur's cost in randomness (per round).

In the standard implementation [B],[BM] one builds a new game in each round of which Arthur sends $O(lk)$ random bits (where l is the number of random bits Arthur uses per round in the original game). An alternative construction presented by [BG] reduces the error probability to 2^{-k} at the cost of $O(l + gk)$ random bits per round (where g is the number of rounds); this saves random bits compared to the $O(lk)$ per round of the standard method as long as g is small compared to l.

As an application of our (l, ϵ, δ)-sampling techniques we show how to improve this result to $O(l + k \log l)$ random bits per round. This saves random bits compared to the standard method for *all* values of the parameters l, k, g.

Goldreich [G] has recently improved this to only $O(l + k)$ random bits per round using his block sampling techniques mentioned above.

2 Sampling Using t-wise Independence

In this section we describe how to implement the sampling primitive using few random bits. We begin with a more precise specification of the primitive. Next we introduce the two major tools we will use: t-universal hash functions and the t-wise independence tail inequality.

We first construct, as an illustration of our methods, a simple oblivious sampler which nonetheless gives a non-trivial savings in coin tosses. We then present an iterated sampling technique which significantly reduces the number of random bits used. Finally we specify the sampler that results.

2.1 (l, ϵ, δ)-Sampling

Definition 2.1 Let $l : \mathbb{N} \to \mathbb{N}$ and $\epsilon, \delta : \mathbb{N} \to [0, 1]$. An (l, ϵ, δ)*-sampler* is a randomized, polynomial time algorithm which on input 1^n outputs a sequence of points $x_1, \ldots, x_m \in \{0,1\}^{l(n)}$ such that: for any collection of m functions $f_1, \ldots, f_m : \{0,1\}^{l(n)} \to [0,1]$ it is the case that

$$\mathbf{P}\left[\left| \tfrac{1}{m} \sum_{i=1}^{m} f_i(x_i) - \mathbf{E}[f_i] \right| \le \epsilon(n) \right] \ge 1 - \delta(n)$$

(where $\mathbf{E}[f_i] = 2^{-l(n)} \sum_{x \in \{0,1\}^{l(n)}} f_i(x)$).

Notice that this definition is slightly more general than what we discussed in §1.1 since we are talking about approximating a *collection* of functions rather than a single function. This will be important for our application.

The points $x_1, \ldots, x_m$ are called the *sample points*, and we refer to the sequence of coin tosses used by the sampler as the *seed*.

2.2 t-Universal Hash Functions

Definition 2.2 A collection H of functions mapping n bits to m bits is *t-universal* if for all distinct $x_1, \ldots, x_t \in \{0,1\}^n$ and all $y_1, \ldots, y_t \in \{0,1\}^m$, picking h at random from H implies that $(h(x_1), \ldots, h(x_t)) = (y_1, \ldots, y_t)$ with probability exactly 2^{-tm}.

For the rest of this section, $H_t(n,m)$ will denote a t-universal collection of hash functions mapping n bits to m bits in which the description of a function uses $t \cdot \max(n,m)$ bits (cf. [CaWe]).

2.3 The t-wise Independence Tail Inequality

Definition 2.3 A collection of random variables $\{X_i\}_{i=1}^{n}$ is *t-wise independent* if for any $a_1, \ldots, a_t$ and any t distinct indices $i_1, \ldots, i_t$ it is the case that $\mathbf{P}[X_{i_1} = a_1, \ldots, X_{i_t} = a_t] = \prod_{j=1}^{t} \mathbf{P}[X_{i_j} = a_j]$.

The t-wise independence tail inequality is a Chernoff-type bound for the sum of a collection of t-wise independent random variables.

Lemma 2.4 (The t-wise Independence Tail Inequality) *Let $t \ge 2$ be an even integer. Suppose $\{X_i\}_{i=1}^{n}$ is a collection of t-wise independent random variables in the range $[0,1]$. Let $X = X_1 + \cdots + X_n$ and $\mu = \mathbf{E}[X]$, and let $A > 0$. Then*

$$\mathbf{P}[|X - \mu| > A] \le \sqrt{4\pi t} \left(\frac{nt}{eA^2} \right)^{t/2} .$$

Remark: If $t \ge 4$ then

$$\sqrt{4\pi t} \left(\frac{nt}{eA^2} \right)^{t/2} \le \left(\frac{nt}{A^2} \right)^{t/2} ,$$

and it is this simpler bound that we will use.

Berger and Rompel [BR] prove the same bound for the restricted case of random variables which are $+1$ or -1 with probability $\frac{1}{2}$ each.

For a proof of Lemma 2.4 see Appendix A.

2.4 A Simple Sampler

Using t-wise independent hash functions we can construct a very simple (l, ϵ, δ)-sampler as follows. The sampler takes as input a randomly selected element h from $H_t(d, l)$, where $d \geq \lg m$, and outputs $h(1), \ldots, h(m)$ (identifying $\{0,1\}^d$ with $\{1, 2, \ldots, 2^d\}$). We use the t-wise independence tail inequality to specify m and t.

Let $Y_i = f_i(h(i))$ for $1 \leq i \leq m$. It follows from the definition of t-wise independent hash functions that $\{Y_i\}_{i=1}^m$ is a collection of t-wise independent random variables in the range $[0,1]$. Thus the t-wise independence tail inequality will hold to bound $Y = \sum_{i=1}^m Y_i$. In particular, assume t is an even integer ≥ 4. Then we have

$$\mathbf{P}\left[\,|Y - \mathbf{E}[Y]| \geq \epsilon m\,\right] \;\leq\; \left(\frac{mt}{(\epsilon m)^2}\right)^{t/2} \;=\; \left(\frac{t}{\epsilon^2 m}\right)^{t/2}.$$

But the left hand side is just

$$\mathbf{P}\left[\,\left|\sum_{i=1}^m f_i(x_i) - \sum_{i=1}^m \mathbf{E}[f_i]\right| \geq \epsilon m\,\right] \;=\; \mathbf{P}\left[\left|\tfrac{1}{m}\sum_{i=1}^m f_i(x_i) - \mathbf{E}[f_i]\right| \geq \epsilon\right].$$

Since we want this probability to be less than δ, it suffices to have

$$\delta \;\geq\; \left(\frac{t}{\epsilon^2 m}\right)^{t/2}$$

or equivalently,

$$m \;\geq\; \frac{t}{\epsilon^2 \delta^{2/t}}.$$

Restating the above, we have the following lemma.

Lemma 2.5 *Let t, m, d be integers such that $t \geq 4$ is even, $m \geq \frac{t}{\epsilon^2\delta^{2/t}}$, and $d \geq \lg m$. Then for any collection of m functions $f_1, \ldots, f_m : \{0,1\}^l \to [0,1]$, picking h at random from $H_t(d,l)$ implies that*

$$\mathbf{P}\left[\left|\tfrac{1}{m}\sum_{i=1}^m f_i(h(i)) - \mathbf{E}[f_i]\right| \leq \epsilon\right] \;\geq\; 1 - \delta.$$

Next, observe that this sampler uses $t \cdot \max(d, l)$ bits. This leads us to think that we just make t as small as possible and m as large as necessary to minimize the number of bits. However, we have another constraint: m must be bounded by a polynomial in the input size. Requiring that m be polynomial in n and optimizing, we get $t = \frac{\log \delta^{-1}}{\log n^{O(1)}}$ and thus use $O\left(\frac{l \log \delta^{-1}}{\log n} + \log \delta^{-1}\right)$ random bits.

2.5 Iterated Sampling

In the previous section, we showed how to sample a collection of functions using t-wise independent hash functions. The number of bits we used to (l, ϵ, δ)-sample, for fixed ϵ and δ, was proportional to the logarithm of the domain size of the functions and roughly inversely proportional to the logarithm of the size of the sample. Given a set of functions with a fixed domain size, this suggested that we simply make our sample as large as is tolerable (i.e. polynomial).

In this section we will improve our bounds by iterating the sampling primitive of the previous section in a novel manner. Roughly, the idea is to take a large sample and then take a smaller sample of the first sample. Each of these samples will require many fewer bits than our original method: the first because the sample is larger; the second because we are sampling a smaller space. Our sampler becomes the composition of two randomly chosen hash functions. Note that our first sample can be superpolynomial in size—we only sample a polynomial number of its points. This idea can then be improved by taking a sequence of smaller and smaller samples instead of just two.

First we need a variant of Lemma 2.5. The reason is that we actually think of sampling each function individually except for the final sample.

Lemma 2.6 *Let t, d be integers such that $t \geq 4$ is even and $2^d \geq \frac{t}{\epsilon^2 \delta^{2/t}}$. Then for any function $f : \{0,1\}^l \to [0,1]$, picking h at random from $H_t(d,l)$ implies that*

$$\mathbf{P}\left[\, |\mathbf{E}[f \circ h] - \mathbf{E}[f]| \leq \epsilon \,\right] \geq 1 - \delta .$$

Proof: Let $m = 2^d$. The probability of failure is

$$\mathbf{P}\left[\, |\mathbf{E}[f \circ h] - \mathbf{E}[f]| > \epsilon \,\right] = \mathbf{P}\left[\, |\textstyle\sum_{i=1}^m f(h(i)) - m\,\mathbf{E}[f]| > \epsilon m \,\right] .$$

By the t-wise independence tail inequality this is bounded by

$$\left(\frac{mt}{(\epsilon m)^2}\right)^{t/2} = \left(\frac{t}{\epsilon^2 m}\right)^{t/2} \leq (\delta^{2/t})^{t/2} = \delta. \quad \square$$

Now we can combine Lemmas 2.5 and 2.6 to obtain a new sampling lemma. Our sampler will be the composition of a sequence of length doubling hash functions. This represents a sequence of samples in which the size of each sample is the square root of the size of the preceding sample.

Lemma 2.7 *Let r, m, d be integers and $t_1, \ldots, t_r$ even integers ≥ 4. Suppose $d \geq \lg m$ and*

$$2^{2^{r-j}d} \geq \frac{t_j}{\epsilon^2 \delta^{2/t_j}} \quad (j = 1, \ldots, r-1) \quad \text{and} \quad m \geq \frac{t_r}{\epsilon^2 \delta^{2/t_r}} .$$

Then for any collection of m functions $f_1, \ldots, f_m : \{0,1\}^{2^r d} \to [0,1]$, picking h_j at random from $H_{t_j}(2^{r-j}d, 2^{r-j+1}d)$ implies that

$$\mathbf{P}\left[\, \left|\tfrac{1}{m}\textstyle\sum_{i=1}^m (f_i \circ h_1 \circ \cdots \circ h_r)(i) - \mathbf{E}[f_i]\right| \leq r\epsilon \,\right] \geq 1 - r\delta .$$

Proof: Let $f = \frac{1}{m}\sum_{i=1}^m f_i$. We first claim by induction that for $0 \leq j \leq r-1$

$$\mathbf{P}\left[\, |\mathbf{E}[(f \circ h_1 \circ \cdots \circ h_j)] - \mathbf{E}[f]| \leq j\epsilon \,\right] \geq 1 - j\delta .$$

The base case ($j = 0$) is immediate, and the induction step is just Lemma 2.6 (using $f \circ h_1 \circ \cdots \circ h_{j-1}$ as the function). The final step is to apply Lemma 2.5 (using $\{f_i \circ h_1 \circ \cdots \circ h_{r-1}\}_{i=1}^m$ as the collection of functions). $\square$

2.6 Our Sampler

We now optimize the parameters to get a particular sampler. In this optimization, there is a trade-off between the number of sample points the sampler outputs and the number of random bits it uses to do this. As we have seen, for polynomially bounded error δ we can use fewer random bits by allowing the number of sample points to grow proportional to δ^{-1} (rather than $\log \delta^{-1}$). For maximum generality we will thus consider an error of the form $\delta_1 \delta_2$ where we assume δ_1 is $\leq n^{O(1)}$.

Theorem 2.8 *Suppose $l : \mathbb{N} \to \mathbb{N}$ is $\leq n^{O(1)}$ with $\log n = o(l)$, and $\epsilon, \delta_1, \delta_2 : \mathbb{N} \to [0,1]$ are > 0 with $\epsilon^{-1}, \delta_1^{-1}, \log \delta_2^{-1} \leq n^{O(1)}$. Then we can construct an $(l, \epsilon, \delta_1\delta_2)$-sampler which outputs $m = O(\epsilon^{-6}\log^6 l + \delta_1^{-1}\log l + \log^3 \delta_2^{-1})$ sample points using $O(l + \log \delta_2^{-1} \cdot \log l)$ coin tosses.*

Proof: We apply Lemma 2.7. Let $m = \max(\epsilon^{-6}\log^6 l, \delta_1^{-1}\log l, [12(1+\log\delta_2^{-1})]^3)$ and $r = \log(l/\log m)$. Let the ϵ of Lemma 2.7 be ϵ/r and the δ be $\delta_1\delta_2/r$, and let

$$t_j = \frac{12}{2^{r-j+1}\log m}\left(2^{r-j+1}\log m + \log\delta_2^{-1}\right) \quad (j = 1, \ldots, r).$$

The conditions of Lemma 2.7 can now be verified. □

3 Randomness-Efficient Error-Reduction for Arthur-Merlin Games

In this section, we apply the results of §2 to derive a randomness-efficient method of reducing the error probability of Arthur-Merlin proof systems. We begin with a review of Arthur-Merlin games and proof systems and the standard method of error-reduction. We then discuss the ideas of our protocol and the particular (l, ϵ, δ)-sampler it requires, and conclude with a proof of our randomness-efficient error-reduction theorem.

3.1 Arthur-Merlin Games

An *Arthur-Merlin game* is a two-party protocol played by an all-powerful "prover", called *Merlin*, and a polynomial-time "verifier", called *Arthur*. The game is played on a common input (and its purpose is to convince Arthur that the input belongs to some predetermined language). Arthur's role in the process is restricted to tossing coins, sending their outcome and finally evaluating a polynomial-time predicate applied to the common input and the full transcript of the interaction.

Let w denote the common input to the (Arthur-Merlin) game, $n = |w|$ its length, $l(n)$ the length of Arthur's messages, $q(n)$ the length of Merlin's messages, and $g(n)$ the number of rounds. We denote by $\rho(w, C) \in \{0, 1\}$ Arthur's *decision* on input w and conversation C. The conversation C can be parsed uniquely into Arthur's and Merlin's messages: $C = r^1y^1 \cdots r^g y^g$, where r^j is Arthur's j-th message and y^j is Merlin's response (we assume that Arthur plays first and Merlin second in each round). A strategy for Arthur, $A = (\rho, g, l, q)$, consists of the decision predicate ρ, as well as (polynomially bounded) functions specifying the number of rounds and the length of messages sent in each round by each party. For sake of simplicity we assume that the length of the messages sent in each round is independent of the round.

Let M be a strategy for Merlin (i.e. M determines the next message of Merlin based on the common input and the messages received so far from Arthur). We denote by $\mathbf{P}[(A, M) \text{ accepts } w]$ the probability that $\rho(w, C) = 1$ when C is chosen at random (the probability space is that of all possible choices of $r^1, \ldots, r^{g(|w|)}$ taken with uniform distribution, and the y^j being set to $M(x, r^1r^2 \ldots r^j)$).

Definition 3.1 We say that the Arthur strategy A defines an Arthur-Merlin proof system for L if the following conditions hold:

(1) *Completeness:* There exists a Merlin strategy M such that $\mathbf{P}[\,(A, M) \text{ accepts } w\,] \geq \frac{2}{3}$ for every $w \in L$.

(2) *Soundness:* $\mathbf{P}[\,(A, \widehat{M}) \text{ accepts } w\,] \leq \frac{1}{3}$ for every Merlin strategy $\widehat{M}$ and every $w \notin L$.

The strategy $\widehat{M}$ in the soundness conditions is sometimes called a *cheating* Merlin, while the strategy M in the completeness condition is called the *honest* Merlin. In fact, it suffices to

$$\begin{array}{lcccc}
 & \underline{\text{subgame } 1} & \underline{\text{subgame } 2} & \dots & \underline{\text{subgame } m} \\
\left.\begin{array}{l}\text{Arthur's message:} \\ \text{Merlin's response:} \\ \vdots \\ \text{Arthur's message:} \\ \text{Merlin's response:}\end{array}\right. &
\begin{array}{c} r_1^1 \\ y_1^1 \\ \vdots \\ r_1^g \\ y_1^g \end{array} &
\begin{array}{c} r_2^1 \\ y_2^1 \\ \vdots \\ r_2^g \\ y_2^g \end{array} &
\begin{array}{c} \dots \\ \\ \\ \\ \dots \end{array} &
\left.\begin{array}{c} r_m^1 \\ y_m^1 \\ \vdots \\ r_m^g \\ y_m^g \end{array}\right\} g \text{ rounds}
\end{array}$$

Figure 1: Framework of the Standard Error-Reduction Protocol

consider (in both conditions) an "optimal Merlin", M_{opt_A}, that chooses all its messages in a way maximizing Arthur's accepting probability. Note that M_{opt_A} depends on A.

Based on this optimal Merlin strategy we extend the deciding predicate ρ to partial conversations as follows:

- $\rho(w, r^1y^1 \dots r^jy^jr^{j+1}) = \max_y \rho(w, r^1y^1 \dots r^jy^jr^{j+1}.y) = \rho(w, r^1y^1 \dots r^jy^jr^{j+1}.y^{j+1})$ where $y^{j+1} = M_{\text{opt}_A}(r^1 \dots r^{j+1})$, for $j = g(n) - 1, \dots, 0$
- $\rho(w, r^1y^1 \dots r^jy^j) = \mathbf{E}_r \rho(w, r^1y^1 \dots r^jy^j.r)$, for $j = g(n) - 1, \dots, 0$.

A's *accepting probability function* is then defined by $acc(w, r^1 \dots r^j) = \rho(w, r^1y^1 \dots r^jy^j)$ where $y^j = M_{\text{opt}_A}(w, r^1 \dots r^j)$ $(j = 0, \dots, g(n))$, and A's accepting probability on input w is $acc(w) \stackrel{\text{def}}{=} acc(w, \lambda)$. Observe that

Proposition 3.2 $\mathbf{E}_r acc(w, r^1 \dots r^{j-1}r) = acc(w, r^1 \dots r^{j-1})$.

(We will use this later). The *error probability* of A on input w (with respect to a language L) is defined as

$$err_L(w) = \begin{cases} 1 - acc(w) & \text{if } w \in L \\ acc(w) & \text{otherwise.} \end{cases}$$

The error probability of A (with respect to L) is $e_L : \mathbb{N} \to [0,1]$ defined by $e_L(n) = \sup_{|w|=n} err_L(w)$. Thus an Arthur strategy A defines a proof system for L if $e_L \leq \frac{1}{3}$.

3.2 Error-Reduction and its Standard Implementation

Error-reduction is the process of reducing the error probability of an Arthur-Merlin proof system from $\frac{1}{3}$ to 2^{-k} for a given $k = k(n) \leq n^{O(1)}$. We review the standard method of error-reduction [B],[BM].

Given $A = (\rho, g, l, q)$ defining an error $\leq \frac{1}{3}$ Arthur-Merlin proof system for L we want to design A^* defining an error $\leq 2^{-k}$ Arthur-Merlin proof system for L. The solution is to play in parallel $m = O(k)$ *independent* copies of the old game (the one defined by strategy A). The independence of Arthur's moves in the various "subgames" is used to prove that the error probability decreases exponentially with the number of subgames.

More concretely, A^* will, in round j, send ml random bits to Merlin. These bits are regarded as a sequence $r_1^j \dots r_m^j$ of m different round j messages of A. Merlin then responds with strings $y_1^j \dots y_m^j$, and y_i^j is regarded as the response of Merlin to r_i^j in the i-th subgame $(i = 1, \dots, m)$. This continues for g rounds (see Figure 1).

$$\left.\begin{array}{rcccc}
 & \underline{\text{subgame } 1} & \underline{\text{subgame } 2} & \cdots & \underline{\text{subgame } m} \\
\text{Arthur's message } s^1 \text{ specifies:} & r_1^1 & r_2^1 & \cdots & r_m^1 \\
\text{Merlin's response:} & y_1^1 & y_2^1 & & y_m^1 \\
\vdots & \vdots & \vdots & & \vdots \\
\text{Arthur's message } s^g \text{ specifies:} & r_1^g & r_2^g & & r_m^g \\
\text{Merlin's response:} & y_1^g & y_2^g & \cdots & y_m^g
\end{array}\right\} g \text{ rounds}$$

Figure 2: Framework of Our Error-Reduction Protocol

Finally, A^* will accept in the new game iff a majority of the subgames were accepting for the original A. That is, A^* accepts iff $|\{\, i : \rho(w, r_i^1 y_i^1 \ldots r_i^{g(n)} y_i^{g(n)}) = 1 \,\}| \geq \frac{m(n)}{2}$.

The bound on the error probability of the new game follows quite easily from the fact that the coin tosses used by Arthur in the different subgames are independent. However, the cost of this argument is in the large number of coin tosses used by A^*; namely $O(lk)$ coin tosses per round (to be contrasted with the l coin tosses used in each round of the original game).

3.3 Overview of Our Protocol

We will run m subgames in parallel (with m appropriately chosen). In each round j Arthur sends a random seed s^j of a sampler G (whose parameters we will specify later). This specifies a sequence $r_1^j \ldots r_m^j$ of messages that will play the role of A's j-th round messages. Although the same pseudo-random process is used at each round j, it will be with a completely new random seed s^j. At the end, A^* will as usual accept iff a majority of the subgames were accepting (see Figure 2).

We emphasize that Arthur sends a seed s^j and both parties then compute the sequence of messages by running the sampler on input s^j.

The difficulty now is that a cheating Merlin might be able to capitalize on the dependency between the subgames. That is, although an honest Merlin would compute y_i^j based only on $r_i^1 y_i^1 \ldots r_i^{j-1} y_i^{j-1}$ (using the honest Merlin for the original protocol) a cheating Merlin could compute the string $y_1^j \ldots y_m^j$ based on the entire submatrix above this string. Clearly we cannot prevent Merlin from following such a strategy. Using the properties of the sampler however, we can show that no such strategy would help.

We will guarantee that at each round the average accepting probability of the m subgames on the sequence specified by the seed approximates the average accepting probability of a sequence of independently chosen messages. That is, assuming $s^1, \ldots, s^{t-1}$ specifying $r_1^1 \ldots r_m^1, \cdots, r_1^{t-1} \ldots r_m^{t-1}$ have been chosen, we guarantee that with high probability we have

$$\tfrac{1}{m} \textstyle\sum_{i=1}^m acc(w, r_i^1 \ldots r_i^{t-1} r_i^t) \;\approx\; \tfrac{1}{m} \textstyle\sum_{i=1}^m \mathbf{E}_r\, acc(w, r_i^1 \ldots r_i^{t-1} r)$$

for the random choice of s^t (where $r_1^t \ldots r_m^t$ is the sequence specified by s^t). If all seeds selected provide good approximations in this sense then the rate of accepting subgames (in the new game) will approximate the accepting probability (in the original game). Hence, it all amounts to selecting a sampler which guarantees that all g approximations are "good" with very high probability.

3.4 The Sampler for Error-Reduction

The sampler which we will use to generate the messages at each round is specified by the following

Theorem 3.3 *Let $g, l, k : \mathbb{N} \to \mathbb{N}$ be $\leq n^{O(1)}$ with $\log n = o(l)$. Then we can construct a $\left(l, \frac{1}{6g}, \frac{2^{-k}}{g}\right)$-sampler which uses $O(l + k \log l)$ coin tosses.*

Proof: Apply Theorem 2.8 with $\epsilon = \frac{1}{6g}$, $\delta_1 = \frac{1}{g}$, and $\delta_2 = 2^{-k}$. $\square$

3.5 Randomness-Efficient Error-Reduction Theorem

Theorem 3.4 *Suppose $A = (\rho, g, l, q)$ is an Arthur strategy that has error probability $\leq \frac{1}{3}$ with respect to L. Then we can construct an Arthur strategy $A^* = (\rho^*, g, O(l + k \log l), q^*)$ which has error probability $\leq 2^{-k}$ with respect to L.*

We distinguish two cases. The first is when $l = O(\log n)$ for which we may prove the statement of Theorem 3.4 using just the simple sampler of §2.4. We omit that proof here, and proceed to the more interesting case of $\log n = o(l)$.

Let G be the $\left(l, \frac{1}{6g}, \frac{2^{-k}}{g}\right)$-sampler specified by Theorem 3.3. Let m be the number of sample points that it outputs and $s = O(l + k \log l)$ the number of random bits it uses. The new Arthur strategy is $A^* = (\rho^*, g, s, mq)$ where $\rho^*(w, s^1 y_1^1 \dots y_{m(n)}^1 \cdots\cdots s^{g(n)} y_1^{g(n)} \dots y_{m(n)}^{g(n)})$

$$= \begin{cases} 1 & \text{if } |\{\, i : \rho(w, G_i(s^1) y_i^1 \dots G_i(s^{g(n)}) y_i^{g(n)}) = 1 \,\}| \geq \frac{m(n)}{2} \\ 0 & \text{otherwise} \end{cases}$$

($n = |w|$ and $G_i(s^j)$ denotes the i-th coordinate of the output of G on input $s^j \in \{0,1\}^{s(n)}$).

The analysis will be round by round. For any fixed seeds $s^1, \dots, s^{t-1}$ the sampler guarantees that

$$\mathbf{P}\left[\, \left| \tfrac{1}{m} \textstyle\sum_{i=1}^m acc(w, r_i^1 \dots r_i^{t-1} r_i^t) - \mathbf{E}_r\, acc(w, r_i^1 \dots r_i^{t-1} r) \right| \leq \tfrac{1}{6g} \,\right] \geq 1 - \tfrac{2^{-k}}{g}$$

where $r_1^j \dots r_m^j$ is the sequence specified by s^j ($j = 1, \dots, t$). By the triangle inequality and Proposition 3.2 we then have that at the end of t rounds it is the case that

$$\mathbf{P}\left[\, \left| \tfrac{1}{m} \textstyle\sum_{i=1}^m acc(w, r_i^1 \dots r_i^t) - acc(w) \right| \leq \tfrac{t}{6g} \,\right] \geq 1 - \tfrac{t 2^{-k}}{g} .$$

Thus at the conclusion of the game ($t = g$) we are guaranteed that

$$\mathbf{P}\left[\, acc(w) - \tfrac{1}{6} \leq \tfrac{1}{m} \textstyle\sum_{i=1}^m acc(w, r_i^1 \dots r_i^g) \leq acc(w) + \tfrac{1}{6} \,\right] \geq 1 - 2^{-k} .$$

But $\frac{1}{m} \sum_{i=1}^m acc(w, r_i^1 \dots r_i^g)$ is just the fraction of accepting subgames. We conclude that the error probability of our game is $\leq 2^{-k}$ by using the fact that $err_L(w) \leq \frac{1}{3}$.

4 Concluding Remarks and Open Problems

Our main result is the construction of an (l, ϵ, δ)-sampler using $(l + \log \delta^{-1} \cdot \log l)$ coin tosses. Can this be improved to $O(l + \log \delta^{-1})$ coin tosses?

We point out that although our randomness-efficient error-reduction result has been improved to use only $O(l + k)$ random bits per round, it is still open whether the underlying (l, ϵ, δ)-sampling primitive can be implemented with $O(l + \log \delta^{-1})$ coin tosses.

Other tasks are to decrease the number of sample points while keeping the number of coin tosses the same, and to find more applications of the (l, ϵ, δ)-sampling primitive.

Acknowledgments

We are happy to thank Shafi Goldwasser and Oded Goldreich for many valuable comments on both the materiel and the exposition of this article.

References

[AKS] Ajtai, M., J. Komlos and E. Szemeredi, "Deterministic Simulation in Logspace," *Proceedings of the* 19th *ACM Symposium on the Theory of Computing* (May 1987).

[B] Babai, L., "Trading Group Theory for Randomness," *Proceedings of the* 17th *ACM Symposium on the Theory of Computing* (May 1985).

[BM] Babai, L. and S. Moran, "Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes," *J. Computer and System Sciences* **36**, 254–276 (1988).

[BG] Bellare, M. and S. Goldwasser, "Saving Randomness in Interactive Proofs," manuscript (November 1989).

[BR] Berger, B. and J. Rompel, "Simulating ($\log^c n$)-wise independence in NC," *Proceedings of the* 30th *IEEE Symposium on the Foundations of Computer Science* (October 1989).

[CaWe] Carter, L. and M. Wegman, "Universal Classes of Hash Functions," *J. Computer and System Sciences* **18**, 143–154 (1979).

[CG] B. Chor and O. Goldreich, "On the Power of Two–Point Based Sampling," *J. of Complexity* **5**, 96–106 (1989).

[CoWi] Cohen, A. and A. Wigderson, "Dispersers, Deterministic Amplification, and Weak Random Sources," *Proceedings of the* 30th *IEEE Symposium on the Foundations of Computer Science* (October 1989).

[G] Goldreich, O., private communication (June 1990).

[GMR] Goldwasser, S., S. Micali and C. Rackoff, "The Knowledge Complexity of Interactive Proofs," *SIAM J. Computing* **18**(1), 186–208 (1989).

[GS] Goldwasser, S. and M. Sipser, "Private Coins versus Public Coins in Interactive Proof Systems," *Proceedings of the* 18th *ACM Symposium on the Theory of Computing* (May 1986).

[IZ] Impagliazzo, R. and D. Zuckerman, "How to Recycle Random Bits," *Proceedings of the* 30th *IEEE Symposium on the Foundations of Computer Science* (October 1989).

A Appendix: Proof of the t-wise Independence Tail Inequality

The line of reasoning used by [BR] extends to this more general case. First, it suffices to appropriately bound the t-th central moment: one shows that

$$\mathbf{E}[(X-\mu)^t] \;\leq\; \sqrt{4\pi t}\left(\frac{nt}{e}\right)^{t/2}$$

and then obtains Lemma 2.4 by an application of Markov's inequality. Second, observe that by linearity of expectation, the t-th central moment of the sum of a collection of t-wise independent random variables is the same as the t-th central moment of this sum under the assumption that the random variables are *fully* independent. So our work reduces to showing

Lemma A.1 *Suppose $X_1,\ldots,X_n$ are independent random variables in the range $[0,1]$, $X = X_1+\cdots+X_n$, $\mu = \mathbf{E}[X]$, and $t \geq 2$ is an even integer. Then*

$$\mathbf{E}[(X-\mu)^t] \;\leq\; \sqrt{4\pi t}\left(\frac{nt}{e}\right)^{t/2} .$$

The proof Lemma A.1 is facilitated by the use of integrals to express the expectation and make the estimates. It also uses standard Chernoff bounds. Let us begin by recalling the relevant facts.

A.1 Facts

The following Chernoff-type bound is well known:

Lemma A.2 *Suppose $X_1,\ldots,X_n$ are independent random variables in the range $[0,1]$, $X = X_1+\cdots+X_n$, $\mu = \mathbf{E}[X]$, and $a > 0$. Then*

$$\mathbf{P}\left[\,|X-\mu| > a\,\right] \;<\; 2e^{-a^2/2n} .$$

A simple well known expression for the expectation is

Lemma A.3 *Let Z be a non-negative real valued random variable. Then $\mathbf{E}[Z] = \int_0^\infty \mathbf{P}[Z > x]dx$.*

Finally we need

Lemma A.4 *Let $\alpha > 0$ and $t \geq 2$ an even integer. Then*

$$\int_0^\infty e^{-\alpha x^{2/t}}dx \leq \sqrt{\pi t}\left(\frac{t}{2e\alpha}\right)^{t/2} .$$

Proof: With the change of variable $y = \alpha x^{2/t}$ the integral becomes

$$\left(\frac{1}{\alpha}\right)^{t/2}\frac{t}{2}\int_0^\infty y^{\frac{t}{2}-1}e^{-y}dy . \tag{1}$$

We claim that

$$\int_0^B y^c e^{-y}dy \;=\; c!\left(1-e^{-B}\textstyle\sum_{i=0}^{c}\frac{B^i}{i!}\right) \tag{2}$$

for $B \geq 0$ and integer $c \geq 0$, and thus the integral of equation (1) is bounded above by

$$\left(\frac{1}{\alpha}\right)^{t/2}\frac{t}{2}\left(\frac{t}{2}-1\right)! \;=\; \left(\frac{1}{\alpha}\right)^{t/2}\left(\frac{t}{2}\right)! \;\sim\; \sqrt{\pi t}\left(\frac{t}{2e\alpha}\right)^{t/2} .$$

To establish equation (2) let $f(c) = \int_0^B y^c e^{-y} dy$ and integrate by parts. We get

$$f(c) = [y^c e^{-y}]_B^0 + \int_0^B c y^{c-1} e^{-y} dy = [y^c e^{-y}]_B^0 + c f(c-1) .$$

Unraveling the recursion yields the desired expression. We omit the details. □

A.2 Proof of Lemma A.1

By Lemma A.3

$$\mathbf{E}[(X-\mu)^t] = \int_0^\infty \mathbf{P}\left[(X-\mu)^t > x \right] dx = \int_0^\infty \mathbf{P}\left[|X-\mu| > x^{1/t} \right] dx .$$

By Lemma A.2 this is bounded above by

$$2\int_0^\infty e^{-\frac{x^{2/t}}{2n}} dx ,$$

and by Lemma A.4 (set $\alpha = \frac{1}{2n}$) this is bounded by

$$2\sqrt{\pi t}\left(\frac{nt}{e}\right)^{t/2} = \sqrt{4\pi t}\left(\frac{nt}{e}\right)^{t/2} .$$