Is Untrusted Randomness Helpful?

Uma Girish[*] Ran Raz[†] Wei Zhan[‡]

Abstract

Randomized algorithms and protocols assume the availability of a perfect source of randomness. In real life, however, perfect randomness is rare and is almost never guaranteed. The gap between these two facts motivated much of the work on randomness and derandomization in theoretical computer science.

In this work, we define a new type of randomized algorithms (and protocols), that we call robustly-randomized algorithms (protocols). Such algorithms have access to two separate (read-once) random strings. The first string is trusted to be perfectly random, but its length is bounded by some parameter $k = k(n)$ (where n is the length of the input). We think of k as relatively small, say sub-linear or poly-logarithmic in n. The second string is of unbounded length and is assumed to be random, but its randomness is not trusted.

The output of the algorithm is either an output in the set of possible outputs of the problem, or a special symbol, interpreted as *do not know* and denoted by $\perp$. On every input for the algorithm, the output of the algorithm must satisfy the following two requirements:

1. If the second random string is perfectly random then the algorithm must output the correct answer with high probability.
2. If the second random string is an arbitrary string, even adversarially chosen after seeing the input, the algorithm must output with high probability either the correct answer or the special symbol $\perp$.

We discuss relations of this new definition to several previously studied notions in randomness and derandomization. For example, when considering polynomial-time algorithms, if k is logarithmic we get the complexity class ZPP, while if k is unbounded we get the complexity class BPP, and for a general k, the algorithm can be viewed as an interactive proof with a probabilistic polynomial-time prover and a probabilistic polynomial-time verifier, where the prover is allowed an unlimited number of random bits and the verifier is limited to at most k random bits.

[*]Princeton University. E-mail: ugirish@cs.princeton.edu. Research supported by a Simons Investigator Award, by the National Science Foundation grants No. CCF-1714779, CCF-2007462 and by the IBM Phd Fellowship.

[†]Princeton University. E-mail: ranr@cs.princeton.edu. Research supported by a Simons Investigator Award and by the National Science Foundation grants No. CCF-1714779, CCF-2007462.

[‡]Princeton University. E-mail: weizhan@cs.princeton.edu. Research supported by a Simons Investigator Award and by the National Science Foundation grants No. CCF-1714779, CCF-2007462.

Every previously-studied class of randomized algorithms or protocols, and more generally, every previous use of randomness in theoretical computer science, can be revisited and redefined in light of our new definition, by replacing each random string with a pair of random strings, the first is trusted to be perfectly random but is relatively short and the second is of unlimited length but its randomness is not trusted. The main question that we ask is: In which settings and for which problems is the untrusted random string helpful?

Our main technical observation is that every problem in the class BPL (of problems solvable by bounded-error randomized logspace algorithms) can be solved by a robustly-randomized logspace algorithm with $k = O(\log n)$, that is with just a logarithmic number of trusted random bits. We also give query complexity separations that show cases where the untrusted random string is provenly helpful. Specifically, we show that there are promise problems that can be solved by robustly-randomized protocols with only one query and just a logarithmic number of trusted random bits, whereas any randomized protocol requires either a linear number of random bits or an exponential number of queries, and any zero-error randomized protocol requires a polynomial number of queries.

1 Introduction

The use of randomness is common in theoretical computer science and is essential for many applications. Randomized algorithms and protocols assume the existence of a source of randomness that is trusted to be perfectly random. In real life, however, sources of randomness are rarely trusted to be perfectly random. Many works on randomness and derandomization in theoretical computer science originated from the attempt to bridge that gap. In this work, we ask: Is untrusted randomness helpful?

We define robustly-randomized algorithms as algorithms that have access to two separate random strings. The first string is trusted to be perfectly random, but is very short. The second string is of unbounded length and is assumed to be random, but its randomness is not trusted. If the second random string is perfectly random then the algorithm must output the correct answer with high probability. If the second random string is an arbitrary string, even adversarially chosen after seeing the input, the algorithm must output with high probability either the correct answer or the special symbol $\perp$, interpreted as *do not know*.

For simplicity, we define here robustly-randomized algorithms for total functions $f : \{0,1\}^n \to \{0,1\}$. Similar definitions can be given for partial functions, search problems, etcetera. Similar definitions can also be given in essentially all other settings where random strings are used, for example, query complexity, interactive proofs, etcetera.

Definition 1.1. *Let $f = \{f_n : \{0,1\}^n \to \{0,1\}\}_{n \in \mathbb{N}}$ be a family of functions. Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone computable function. Let A be a randomized algorithm that uses two separate (read-once) random strings R_1, R_2. We say that A is a robustly-randomized algorithm for f, with $O(k)$ trusted random bits, if on every input x of length n, the algorithm A reads at most $O(k(n))$ bits from R_1 and the output $A(x)$ satisfies the following two requirements:*

1.

$$\Pr_{R_1,R_2}[A(x) = f_n(x)] \geq \tfrac{3}{4}$$

(where the probability is over the uniform distribution over R_1, R_2).

2. *For every r (even adversarially chosen after seeing the input x),*

$$\Pr_{R_1}[A(x) \in \{f_n(x), \perp\} | R_2 = r] \geq \frac{3}{4}$$

(where the probability is over the uniform distribution over R_1).

For a language $L \subseteq \mathbb{N}$, we say that A is a robustly-randomized algorithm for L if A is a robustly-randomized algorithm for the family f that corresponds to L.

Definition 1.1 is for the bounded-error case, where the algorithm is allowed to err with probability $\frac{1}{4}$, regardless of the value of $f_n(x)$. Similarly, we can define the one-sided-error case, where we require in addition that if $f_n(x) = 0$ then $A(x)$ is always in $\{0, \perp\}$ (regardless of the content of R_1, R_2).

Many questions could be asked about robustly-randomized algorithms. We focus here on the following motivating question: In which settings, are robustly-randomized algorithms that read $O(k)$ trusted random bits (and an unbounded number of untrusted random bits) stronger than randomized algorithms that read $O(k)$ (trusted) random bits? In other words, in which settings is the untrusted random string helpful?

1.1 Polynomial-Time Algorithms

We first consider polynomial-time algorithms.

Definition 1.2. *Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone computable function. The class $\mathsf{RPP}(k)$ is the class of all languages computable by a polynomial-time robustly-randomized algorithm with $O(k)$ trusted random bits.*

1.1.1 BPP and ZPP

We observe the following relations between the class $\mathsf{RPP}(k)$ and the classes ZPP, BPP and $\mathsf{BPP}(k)$ (where ZPP is the class of problems solvable by zero-error probabilistic polynomial-time algorithms, BPP is the class of problems solvable by bounded-error probabilistic polynomial-time algorithms, and $\mathsf{BPP}(k)$ is the class of problems solvable by bounded-error probabilistic polynomial-time algorithms that are limited to reading $O(k)$ random bits).

Proposition 1.3.

$$\mathsf{RPP}(\log n) = \mathsf{ZPP}.$$

Proposition 1.4. *For every k,*

$$\mathsf{BPP}(k) \subseteq \mathsf{RPP}(k).$$

Corollary 1.5. *For every k,*

$$\mathsf{BPP}(k)^{\mathsf{ZPP}} \subseteq \mathsf{RPP}(k).$$

Proposition 1.6.

$$\bigcup_c \mathsf{RPP}(n^c) = \mathsf{BPP}.$$

By the *hardness-vs-randomness* paradigm [Sha81, BM84, Yao82, NW94, IW97], all these complexity classes are conjectured to be identical, as they are all conjectured to be identical to P. From that perspective and in light of Proposition 1.6, it is an intriguing open problem to prove that $\mathsf{BPP} = \mathsf{RPP}(k)$ for small values of k, say, $k = o(n)$, and this can be viewed as an intermediate goal towards proving $\mathsf{BPP} = \mathsf{ZPP}$ or even $\mathsf{BPP} = \mathsf{P}$. In light of Proposition 1.4, it is an intriguing open problem to show examples of natural problems in $\mathsf{RPP}(k)$ that are currently not known to be in $\mathsf{BPP}(k) \cup \mathsf{ZPP}$. We note that there are artificial problems that give conditional separations:

Proposition 1.7. *For every k, if $\mathsf{BPP}(k)$ and ZPP do not contain one another, then*

$$\mathsf{BPP}(k) \cup \mathsf{ZPP} \neq \mathsf{RPP}(k).$$

1.1.2 Interactive Proofs for Verifying Randomness

We show that a polynomial-time robustly-randomized algorithm with $O(k)$ trusted random bits is equivalent to an interactive proof with a probabilistic polynomial-time prover and a probabilistic polynomial-time verifier, where the prover is allowed an unlimited number of random bits and the verifier is limited to at most $O(k)$ (public or private) random bits (that are guaranteed to be perfectly random). That is, unlike standard interactive proofs, the prover and the verifier here have the same computational power and only differ in the number of random bits that they are allowed to use. Such an interactive proof can be viewed as a proof of randomness: The prover sends a random string to the verifier and tries to convince the verifier that that random string is sufficiently random to perform the computation on a particular input x (that they both know).

Proposition 1.8. *A language L is in $\mathsf{RPP}(k)$ if and only if it is recognizable by an interactive proof (which outputs 1, 0 or $\perp$) with a probabilistic polynomial-time prover and a probabilistic polynomial-time verifier, where the verifier is limited to at most $O(k)$ random bits.*

1.2 Logarithmic-Space Algorithms

Next, we consider logarithmic-space algorithms.

Definition 1.9. *Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone computable function. The class $\mathsf{RPL}(k)$ is the class of all languages computable by a logarithmic-space and polynomial-time robustly-randomized algorithm with $O(k)$ trusted random bits.*

We prove that the entire class BPL (of problems solvable by bounded-error probabilistic logspace algorithms) collapses to the class $\mathsf{RPL}(\log n)$. That is, every problem in BPL is solvable by a robustly-randomized logspace algorithm with just a logarithmic number of trusted random bits. For comparison, the currently best known pseudorandom generators for BPL require $O(\log^2 n)$ truly random bits (and $O(\log^2 n)$ space to store them) [Nis92, INW94, GR14] and Saks' and Zhou's derandomization result for BPL shows that every problem in BPL is solvable by a deterministic algorithm with $O(\log^{3/2} n)$ space [SZ99].

Proposition 1.10.

$$\mathsf{RPL}(\log n) = \mathsf{BPL}.$$

Moreover, we show that every problem in BPL has a streaming proof between a probabilistic logspace prover and a probabilistic logspace verifier, where the verifier uses only $O(\log n)$ random bits and has a read-once one-way access to the proof that is streamed by the prover. In other words, the prover provides a polynomial-length proof that is streamed to the verifier and the verifier can check whether the computation was performed correctly using only $O(\log n)$ random bits.

We also characterize ZPL using robustly-randomized algorithms, thereby providing an alternate interpretation of the BPL versus ZPL problem.

Proposition 1.11.

$$\mathsf{RPL}(1) = \mathsf{ZPL}.$$

1.3 Query-Complexity Separations

Finally, we consider the query-complexity model, where one can fully prove that the untrusted random string is helpful.

Proposition 1.12. *For any $n, k \in \mathbb{N}$, there are two disjoint subsets $A, B \subset \{0,1,2\}^{2^{n+k}}$ (where $\{0,1,2\}^{2^{n+k}}$ is viewed as the set of all functions $f : \{0,1\}^{n+k} \to \{0,1,2\}$), such that, given a black box access to a function $f : \{0,1\}^{n+k} \to \{0,1,2\}$, the following three are satisfied:*

1. *There is a robustly-randomized protocol that uses only k trusted random bits and only one query to f and distinguishes between the cases $f \in A$ and $f \in B$ with probability at least $\frac{3}{4}$.*

2. *For any constant $\epsilon > 0$, any randomized protocol that uses at most $(1-\epsilon)n$ random bits and distinguishes between the cases $f \in A$ and $f \in B$ with probability at least $\frac{3}{4}$, requires at least $2^{\Omega(n)}$ queries to f.*

3. *Any zero-error randomized protocol that distinguishes between the cases $f \in A$ and $f \in B$ with probability at least $\frac{3}{4}$ (and outputs* do not know *with probability at most $\frac{1}{4}$) requires at least $2^{\Omega(k)}$ queries to f.*

Specifically, taking $k = \log n$ in proposition 1.12, we observe that there are promise problems that can be solved by robustly-randomized protocols with only one query and just a logarithmic number of trusted random bits, whereas any randomized protocol requires either a linear number of random bits or an exponential number of queries, and any zero-error randomized protocol requires a polynomial number of queries.

1.4 Relations to Other Work

Derandomization and Pseudorandomness: A large body of work on *derandomization* attempts to reduce the amount of randomness used by randomized algorithms. For example, a *pseudorandom generator* attempts to use a small seed of truly random bits to produce a larger number of *pseudorandom* bits, that look random to all randomized algorithms in a certain class of algorithms (see [Vad12] for a survey). The commonality between these works and our approach is the attempt to reduce the amount of true (trusted) randomness that is used. Here, we suggest to use, in addition to a small number of truly (trusted) random bits, an additional source of untrusted random bits. One could add such a source in essentially all attempts of derandomization. For example, one could consider pseudorandom generators that use in addition to a small seed of truly random bits, an unlimited number of untrusted random bits.

Seeded Extractors: *Seeded extractors* [NZ96] (and many follow-up works) attempt to use a small seed of truly random bits, together with a weakly-random source, to produce a larger number of almost-random bits (that in turn can be used as a random source for any randomized algorithm). The main difference between seeded extractors and our approach is that seeded extractors necessarily require some assumption on the weakly-random source used, usually requiring that it contains a sufficient amount of min-entropy, while here we don't make any assumption about the source. (An additional difference is that seeded extractors are required to produce an output that is close to be perfectly random, while here we only try to fool a particular algorithm or class of algorithms. We note though that there are many other works that consider only a particular algorithm or class of algorithms).

Certified Quantum Randomness: A large body of recent work suggests the use of a small number of truly random bits, together with quantum sources of randomness, to produce a large number of random bits that are guaranteed to be almost perfectly random, by the laws of physics. Such constructions were suggested based on violating Bell inequalities with multiple devices [PAM+10, VV12, MS16], based on post-quantum cryptography [BCMVV21] and based on advantage of quantum computation over classical computation [Aar18, BBFGT22]. While these works do consider a possible adversarial behavior of the source of randomness, as we do here, they all rely on the use of quantum devices (as well as on certain assumptions, such as, no communication between different devices, or hardness assumptions).

2 Preliminaries

Let $n \in \mathbb{N}$. We use $[n]$ to denote $\{1, 2, \ldots, n\}$. Let $v \in \mathbb{R}^n$. For $i \in [n]$ we use v_i to denote the i-th coordinate of v. Let $1 \leq k < \infty$. Let $\|v\|_k := \left(\sum_{i\in[n]} |v_i|^k\right)^{1/k}$ denote the ℓ_k-norm of v. This induces an operator norm on matrices $M \in \mathbb{R}^{n\times n}$ by $\|M\|_k := \max_{v\in\mathbb{R}^n\setminus\{\vec{0}\}} \frac{\|M(v)\|_k}{\|v\|_k}$. This norm is sub-multiplicative, i.e., $\|M \cdot N\|_k \leq \|M\|_k \cdot \|N\|_k$ for all $M, N \in \mathbb{R}^{n\times n}$. Let $\|v\|_\infty = \max_{i\in[n]} |v_i|$ denote the ℓ_∞-norm of v and let $\|M\|_\infty = \max_{i,j\in[n]} |M_{i,j}|$ (this is not an induced operator norm). We have the following inequalities for all $M \in \mathbb{R}^{n\times n}, v \in \mathbb{R}^n$ and $1 \leq k, k' < \infty$.

$$\|M\|_\infty \leq \|M\|_k \leq n \cdot \|M\|_\infty$$

$$k \geq k' \implies \|v\|_k \leq \|v\|_{k'} \leq n \cdot \|v\|_k$$

We say that a matrix is stochastic (resp. sub-stochastic) if each entry is in $[0, 1]$ and every column has entries which sum to 1 (resp. at most 1). A sub-stochastic matrix M satisfies

$\|M\|_1 \leq 1$. We use $M[i,j]$ to refer to the $(i,j)^{\text{th}}$ entry of the matrix M.

2.1 Our Model of Computation

In this work, a deterministic Turing machine consists of a read-only input tape, a work tape and a write-once output tape. A randomized Turing Machine has an additional infinite read-once randomness tape consisting of random bits. Let $S, T, R : \mathbb{N} \to \mathbb{N}$ be any monotone computable functions. We typically use S to denote the space complexity and T to denote the time complexity of a Turing Machine. When we say that an event occurs with high probability, we typically mean that it occurs with probability at least $2/3$. An algorithm is said to have bounded error if the probability of error is at most $1/3$. By standard error-reduction techniques, we could choose this number to be any constant in $(0, 1/2)$.

A deterministic (resp. bounded-error randomized) (S,T) algorithm refers to a deterministic (resp. randomized) Turing Machine such that for all $x \in \{0,1\}^*, |x| = n$, the machine with x on its input tape, uses at most $S(n)$ bits of space on its work tape and runs in at most $T(n)$ time. We say that an algorithm computes a family of functions $\{f_n : \{0,1\}^n \to \{0,1\}^*\}_{n\in\mathbb{N}}$ if for all $n \in \mathbb{N}, x \in \{0,1\}^n$, the output of the algorithm on input x is $f_n(x)$ (with high probability if the algorithm is bounded-error randomized). These functions may be partial, i.e., defined on a strict subset of $\{0,1\}^n$. The Turing machine is said to use R bits of randomness if on inputs of size $n \in \mathbb{N}$, the machine never reads more than $R(n)$ bits on the randomness tape.

Polynomial-Time Computation: A polynomial-time algorithm refers to a $(\text{poly}(n), \text{poly}(n))$ algorithm. The (promise) class BPP refers to all families of single-bit-output functions computable by randomized polynomial-time algorithms. The (promise) class ZPP refers to families computable by randomized algorithms whose *expected* runtime is $\text{poly}(n)$ and that have *zero error*. All these classes are promise classes for us, so for the rest of the paper, we omit this prefix.

Logspace Computation: A logspace algorithm refers to an $(O(\log(n)), \text{poly}(n))$ algorithm. The (promise) class L refers to all families of single-bit-output functions computable by deterministic logspace algorithms. The (promise) class BPL refers to all families of single-bit-output functions computable by randomized logspace algorithms with high probability. The (promise) class ZPL refers to families computable by randomized algorithms whose space is $O(\log(n))$ and whose *expected* runtime is $\text{poly}(n)$ and that have *zero error*. All these classes are inherently promise classes, so for the rest of the paper, we omit this prefix. We use the notation family of functions, languages and problems interchangeably.

We say that a family $\mathcal{F} = \{f_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ of functions is logspace-reducible to a family $\mathcal{G} = \{g_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ of functions if there is a polynomial function $M(n) = \text{poly}(n)$ and a deterministic logspace algorithm $\mathcal{A}$ which for all $n \in \mathbb{N}$, on input $x \in \text{supp}(f_n)$ outputs $\mathcal{A}(x) \in \text{supp}(g_m)$ where $m = M(n)$ such that $f_n(x) = 1 \iff g_m(\mathcal{A}(x)) = 1$. We say that a problem P is logspace-complete for a class $\mathcal{A}$ of algorithms if $P \in \mathcal{A}$ and for every $Q \in \mathcal{A}$, Q is logspace-reducible to P. We make use of the following logspace-complete problem for BPL.

Definition 2.1 (Stochastic Matrix Powering)**.** *In the Stochastic Matrix Powering Problem, the inputs are an $n \times n$ stochastic matrix M and a parameter T such that $T \leq \text{poly}(n)$. The promise on the input is that $M^T[n,1] \geq \frac{4}{5}$ or $M^T[n,1] \leq \frac{1}{5}$. The goal is to output 1 in the former case and 0 in the latter case.*

Proposition 2.2. *The Stochastic Matrix Powering Problem is logspace-complete for* BPL*.*

The proof of this fact is quite standard and is deferred to the appendix.

2.2 Robustly-Randomized Algorithms

We recall the definition of robustly-randomized algorithms.

Definition 1.1: *Let $f = \{f_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ be a family of functions. Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone computable function. Let A be a randomized algorithm that uses two separate (read-once) random strings R_1, R_2. We say that A is a robustly-randomized algorithm for f, with $O(k)$ trusted random bits, if on every input x of length n, the algorithm A reads at most $O(k(n))$ bits from R_1 and the output $A(x)$ satisfies the following two requirements:*

1.
$$\Pr_{R_1,R_2}[A(x) = f_n(x)] \geq \frac{3}{4}$$
(where the probability is over the uniform distribution over R_1, R_2).

2. *For every r (even adversarially chosen after seeing the input x),*
$$\Pr_{R_1}[A(x) \in \{f_n(x), \perp\} | R_2 = r] \geq \frac{3}{4}$$
(where the probability is over the uniform distribution over R_1).

In particular, we will be interested in the classes $\mathsf{RPP}(k)$ and $\mathsf{RPL}(k)$ defined as follows.

Definition 1.2: *Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone computable function. The class $\mathsf{RPP}(k)$ is the class of all languages computable by a polynomial-time robustly-randomized algorithm with $O(k)$ trusted random bits.*

Definition 1.9: *Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone computable function. The class $\mathsf{RPL}(k)$ is the class of all languages computable by a logspace polynomial-time robustly-randomized algorithm with $O(k)$ trusted random bits.*

Although the classes RPP and RPL are defined with error parameter $1/4$, we could choose it to be any constant in $(0, 1/2)$. This is because (by standard error-reduction techniques) we can reduce the error to any desired constant in $(0, 1/2)$ by repeating the algorithm $O(1)$ times in parallel and taking the majority of the outputs. This step only blows up the amount of randomness, space and time required by a constant multiplicative factor and doesn't change the definition of the classes.

2.3 Streaming Proofs

A streaming proof consists of a pair of randomized Turing machines (a randomized prover and a randomized verifier) which share a common stream tape. We assume that the verifier is a logspace machine. The prover doesn't have a separate output tape, instead, it has write-once access to the stream tape onto which it writes a proof Π. The verifier has read-once access to the stream tape from which it can read Π. Both the verifier and the prover have read-many access to the input $x \in \{0,1\}^*$. We allow the prover and verifier to output a special symbol $\perp$. Upon outputting this symbol, the algorithm stops all further processing and we say that the algorithm aborts.

Definition 2.3. *Let $\mathcal{F} = \{f_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ be a famility of functions. Let $P, S, T : \mathbb{N} \to \mathbb{N}$ be monotone computable functions. We say that $\mathcal{F}$ has an (S,T)-streaming proof of length P if there exists a randomized (S,T)-prover $\mathcal{P}$ using a random string R_2 and a randomized logspace verifier $\mathcal{V}$ using a random string R_1 such that on input $x \in \mathrm{supp}(f_n)$,*

1. *The honest prover $\mathcal{P}$, with at least $\frac{3}{4}$ probability over R_2, outputs a (randomized) proof $\Pi \in \{0,1\}^{P(n)}$ such that*
$$\Pr_{R_1}[\mathcal{V}(x,\Pi) = f_n(x)] \geq \tfrac{3}{4}$$
(where the probability is over the uniform distribution over R_1, R_2).

2. *For an arbitrary $\Pi \in \{0,1\}^{P(n)}$ (even adversarially chosen after seeing the input x),*
$$\Pr_{R_1}[\mathcal{V}(x,\Pi) \in \{f_n(x), \perp\}] \geq \tfrac{3}{4}$$
(where the probability is over the uniform distribution over R_1).

Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone computable function. If the verifier $\mathcal{V}$ never reads more than $O(k(n))$ random bits from R_1, we say that the verifier uses at most $O(k(n))$ random bits.

We sometimes omit the length of the proof and it is understood that $P \leq T$ always.

3 Streaming Proofs for BPL

In this section, we prove Proposition 1.10. Towards this, we show the following claims.

Claim 3.1. *A family of functions is in $\mathsf{RPL}(k)$ if it has a streaming proof between a logspace verifier and a logspace prover where the verifier uses $O(k)$ random bits.*

Proof of Claim 3.1. Given a streaming proof between a logspace verifier and a logspace prover where the verifier uses $O(k)$ random bits, consider a $\mathsf{RPL}(k)$ algorithm that simulates the verifier using trusted randomness. For each bit of the stream that the verifier wants to read, we have the algorithm simulate the honest prover using untrusted randomness. The completeness follows as both the verifier and the honest prover simultaneously succeed with probability at least $(3/4)^2 \geq 0.55$, furthemore, the probability that the verifier incorrectly answers given any proof is at most $1/4$. As mentioned before, by standard error-reduction techniques, we can amplify the completeness to be at least $3/4$. This completes the proof. $\square$

Claim 3.2. *A family of functions is in* BPL *if and only if it has a streaming proof between a logspace verifier and a logspace prover where the verifier uses* $O(\log(n))$ *random bits.*

Proof of Proposition 1.10 from Claim 3.2 and Claim 3.1. Firstly, it is easy to see that $\mathsf{RPL}(\log(n)) \subseteq \mathsf{BPL}$. We consider any $\mathsf{RPL}(\log(n))$ algorithm and modify it so that whenever it is supposed to output $\perp$, it outputs 0 instead. This can be viewed as a BPL algorithm with error at most $1/4$ and thus, $\mathsf{RPL}(\log(n)) \subseteq \mathsf{BPL}$.

The conclusion that $\mathsf{BPL} \subseteq \mathsf{RPL}(\log(n))$ follows immediately from Claim 3.1 and Claim 3.2. $\square$

We devote the rest of the section to the proof of Claim 3.2.

3.1 Proof of Claim 3.2

We begin by observing that one direction is easy, i.e., given a streaming proof between an honest randomized logspace prover and verifier, both these algorithms can be simulated by a randomized logspace algorithm. It suffices to show the other direction.

It suffices to develop a streaming proof for the Stochastic Matrix Powering problem (which is logspace-complete for BPL). Towards this, we define a notion of a δ-good sequence of vectors for a stochastic matrix M.

Definition 3.1. *Let* M *be any* $n \times n$ *stochastic matrix and* $T \leq \text{poly}(n)$ *be a natural number. Let* $v_i = M^i(e_1)$ *for all* $i \leq T$. *Let* $\delta \in [0,1]$. *A sequence of vectors* $v'_0, v'_1, \ldots, v'_T \in \mathbb{R}^n$ *is said to be* δ*-good for* M *if for all* $i \in [T]$, *we have* $\|v'_i - v_i\|_1 \leq \delta$ *and* $v'_0 = e_1$.

We make use of the following two claims.

Claim 3.3. *There is a randomized logspace prover which given an* $n \times n$ *stochastic matrix* M *and parameters* $T \leq \text{poly}(n), \delta \geq \frac{1}{\text{poly}(n)}$ *as input, outputs a* δ*-good sequence of vectors for* M *with probability at least* $\frac{3}{4}$.

Claim 3.4. *Let* $0 < \delta \leq \frac{1}{10^4 n T^3}$. *There is a randomized logspace verifier which given any* $n \times n$ *stochastic matrix* M *and parameters* $T \leq \text{poly}(n), \delta \geq \frac{1}{\text{poly}(n)}$ *as input and read-once access to a stream of vectors* $v'_0, \ldots, v'_T \in \mathbb{R}^n$ *(where each vector is specified up to* $\Theta(\log(n))$ *bits of precision), does the following.*

- *If the sequence is* δ*-good for* M, *then the probability that the algorithm aborts is at most* $1/4$.
- *If* $\|v'_T - v_T\|_1 \geq \frac{1}{4}$, *then the algorithm aborts with probability at least* $3/4$.

Furthermore, this algorithm only uses $O(\log(n))$ *bits of randomness.*

We now complete the proof of Claim 3.2 using Claim 3.3 and Claim 3.4. Consider the Stochastic Matrix Powering Problem. Given an $n \times n$ stochastic matrix M as input and a parameter $T \leq \text{poly}(n)$, set $\delta = \frac{1}{10^4 n T^3}$. Run the prover's algorithm from Claim 3.3 using this value of δ to produce a stream $v'_0, \ldots, v'_T$. Run the verifier's algorithm from Claim 3.4 on this stream to verify. If it doesn't abort, we have the verifier return 1 if $v'_T(n) \geq 2/3$, return 0 if $v'_T(n) \leq 1/3$ and return $\perp$ otherwise.

Completeness: Claim 3.3 implies that an honest prover outputs a δ-good sequence with probability at least $\frac{3}{4}$. Claim 3.4 implies that a verifier aborts an honest proof with very small probability. Since $\|v'_T - v_T\|_1 \leq \delta \ll 1/10$ by assumption, if $v_T(n) \geq 4/5$, then $v'_T(n) \geq 2/3$ and if $v_T(n) \leq 1/5$ then $v'_T(n) \leq 1/3$. Since $M^T[n,1] = v_T(n)$, the verifier will return the correct answer whenever the sub-routine doesn't abort.

Soundness: Consider the behavior of this verifier on an arbitrary proof. If the verifier makes a mistake and returns an incorrect answer, it must be the case that either $v_T(n) \geq 4/5$ and $v'_T(n) \leq 1/3$ or $v_T(n) \leq 1/5$ and $v'_T(n) \geq 2/3$. In either case, we must have $\|v'_T - v_T\|_k \geq |v'_T(n) - v_T(n)| \geq 1/4$. Claim 3.4 implies that such a proof is aborted with probability at least $\frac{3}{4}$.

This completes the proof of Claim 3.2. We now proceed to prove Claim 3.3 and Claim 3.4.

Algorithm 1 Algorithm for Prover in Claim 3.3.

```
Input  : An $n \times n$ stochastic matrix $M$ and parameters $T \leq \text{poly}(n), \delta \geq \frac{1}{\text{poly}(n)}$.
Output: A $\delta$-good sequence of vectors $v'_0, v'_1, \ldots, v'_T \in \mathbb{R}^n$ .
begin
    Output $v_0 = e_1$.
    Round down each entry of the input matrix $M$ to $\frac{\delta}{6nT}$ additive error to produce a sub-
      stochastic matrix $\widetilde{M}$ so that $\left\|M - \widetilde{M}\right\|_1 \leq \frac{\delta}{6T}$.
    Set $C = \Theta(n^2 \log(nT)/\delta^2)$.
    for $i = 1$ to $T$, $j = 1$ to $n$ do
        for count $= 1$ to $C$ do
            (*) Set estimate$_{i,j} = 0$.
            Create a register on $\lceil \log(n) \rceil$ bits initialized to the state 1.
            for $t = 1$ to $i$ do
                Suppose the current state is $k \in [n]$. Sample a new state $k' \in [n]$ with
                  probability exactly $\widetilde{M}[k,k']$. Suppose this sampling process fails (due to the
                  columns of $\widetilde{M}$ summing to less than one), increase count, then go to step (*)
                  if count $\leq C$ and go to (**) otherwise.
            end
            If the final state is $j$, update estimate$_{i,j} \leftarrow$ estimate$_{i,j} + 1$.
        end
        (**) Output $v'_i(j) = $ estimate$_{i,j}/C$.
    end
end
```

Proof of Claim 3.3. The prover's algorithm is formally described in **Algorithm 1**. The informal description is as follows. The prover first rounds down each entry of the input matrix M to $\frac{\delta}{6nT}$ additive error to produce a sub-stochastic matrix $\widetilde{M}$ that is close to M. The prover maintains a state in $[n]$ and evolves this state (approximately) according to the (nearly) stochastic matrix $\widetilde{M}$. The prover then uses multiple samples to get an estimate of the entries of $\widetilde{M}^i(e_1)$.

Time & Space Complexity of this Algorithm: It is clear that all these operations can be done by a randomized logspace algorithm.

Correctness of this Algorithm: Let $\widetilde{v}_i = \widetilde{M}^i(e_1)$ for $i \in \{0, \dots, T\}$. Since $\left\|\widetilde{M} - M\right\|_1 \leq \frac{\delta}{6T}$, we have

$$\text{for all } i \in [T], \left\|\widetilde{M}^i - M^i\right\|_1 \leq \left(1 + \frac{\delta}{6T}\right)^i - 1 \leq \frac{\delta}{2}. \tag{1}$$

Thus,

$$\text{for all } i \in [T], \|\widetilde{v}_i - v_i\|_1 \triangleq \left\|\widetilde{M}^i(e_1) - M^i(e_1)\right\|_1 \leq \left\|\widetilde{M}^i - M^i\right\|_1 \leq \frac{\delta}{2}. \tag{2}$$

It is not too hard to observe that by evolving the state 1 for i steps according to $\widetilde{M}$ (and aborting when the sampling process fails), the algorithm produces a (probabilistic) state which is j with probability exactly $\widetilde{v}_i(j)$ for each $j \in [n]$ (and $\perp$ with the remaining probability). By repeated sampling and taking the empirical average, the algorithm can estimate each $\widetilde{v}_i(j)$ up to $\frac{\delta}{2n}$ accuracy with probability at least $1 - O\left(\frac{1}{nT}\right)$ using $C = \Theta\left(\frac{n^2 \log(nT)}{\delta^2}\right)$ samples by the Chernoff bound. This, along with a union bound over $i \in [T], j \in [n]$ implies that the algorithm with probability at least $\frac{7}{8}$, estimates each $\widetilde{v}_i(j)$ up to $\frac{\delta}{2n}$ precision, and hence by Equation (2) produces v'_i such that $\|v'_i - v_i\|_1 \leq \delta$. □

Proof of Claim 3.4. The verifier's algorithm is formally described in **Algorithm 2**. The informal description is as follows. The verifier will try to check that $\widetilde{M}(v'_{i-1})$ is approximately equal to v'_i for all $i \in [T]$. However, to do this in a streaming fashion, the verifier will instead test that a random linear combination of these approximate equations holds. To reduce the randomness from T to $O(\log n)$, instead of using a truly random combination of the equations the verifier uses a pseudorandom combination drawn using a 4-wise independent collection of $\{-1, 1\}$-random variables. This is similar to the ℓ_2-frequency estimation algorithm in [AMS96].

Time & Space Complexity of this Algorithm: One can sample from a collection of 4-wise independent $\{-1, 1\}$-random variables of size $O(nT)$ in logspace using only $O(\log(nT))$ bits of randomness [AMS96]. Note that the quantity $\Delta \triangleq \sum_{\substack{i \in [T] \\ j \in [n]}} \alpha_{i,j} \cdot \left((\widetilde{M}(v'_{i-1}))(j) - v'_i(j)\right)$ can be expressed $\sum_{\substack{i \in \{0,\dots,T\} \\ j \in [n]}} \beta_{i,j} v'_i(j)$ where $\beta_{i,j}$ are coefficients that depend only on the entries of $\widetilde{M}$ and α, and can be computed in logspace. Thus, a logspace algorithm can read the stream of $v'_i(j)$ for $i = 0, \dots, T$ and $j \in [n]$ once from left to right and compute $\Delta \triangleq \sum_{i,j} \beta_{i,j} v'_i(j)$ in a streaming fashion. As the entries of the matrices and the vectors are $O(\log(n))$ bits long, the arithmetic can be done in logspace. The time complexity of this process is hence $\text{poly}(n)$ and the space complexity is $O(\log(n))$.

We now move on to the completeness and soundness. Let $w \in \mathbb{R}^{nT}$ be defined at $i \in [T], j \in [n]$ by $w_{i,j} \triangleq (\widetilde{M}(v'_{i-1}))(j) - v'_i(j)$ as before. Let $\widetilde{v}_0, \dots, \widetilde{v}_T$ be defined as before.

Algorithm 2 Algorithm for Verifier in Claim 3.4

```
Input  : An $n \times n$ stochastic matrix $M$, parameters $T \leq \mathrm{poly}(n)$, $\frac{1}{10^4 n T^3} \geq \delta \geq \frac{1}{\mathrm{poly}(n)}$ and
         read-once access to a stream of vectors $v'_0, \ldots, v'_T \in \mathbb{R}^n$.
Output: If the sequence is $\delta$-good for $M$, then return $\perp$ with probability at most $\frac{1}{4}$. If
         $\|v'_T - v_T\|_1 \geq \frac{1}{4}$, return $\perp$ with probability at least $\frac{3}{4}$.
begin
    Round down each entry of the input matrix $M$ to $\frac{\delta}{6nT}$ additive error to produce a sub-
      stochastic matrix $\widetilde{M}$ so that $\left\|M - \widetilde{M}\right\|_1 \leq \frac{\delta}{6T}$.
    Return $\perp$ if $v'_0 \neq e_1$.
    for $t = 1$ to 11 do
        Sample $\alpha_{i,j} \in \{-1, 1\}$ for $i \in [T], j \in [n]$ from a collection of 4-wise independent
          $\{-1,1\}$-random variables with mean 0.
        Compute $\Delta := \sum_{i \in [T], j \in [n]} \alpha_{i,j} \cdot w_{i,j}$ where for $i \in [T], j \in [n]$, we have $w_{i,j} :=$
          $(\widetilde{M}(v'_{i-1}))(j) - v'_i(j)$.
        Return $\perp$ if $|\Delta| > 30T\delta$.
    end
end
```

Completeness of the Algorithm: Suppose $v'_0, \ldots, v'_T$ is a δ-good sequence, then $\|v'_i - v_i\|_1 \leq \delta$ for all $i \in [T]$ and $v'_0 = e_1$. Since M is a contraction map with respect to $\|\cdot\|_1$, this along with Equation (1) implies that for all $i \in [T]$,

$$\begin{aligned}\left\|\widetilde{M}(v'_{i-1}) - v'_i\right\|_1 &\leq \left\|\widetilde{M}(v'_{i-1}) - M(v'_{i-1})\right\|_1 + \left\|M(v'_{i-1}) - M(v_{i-1})\right\|_1 \\ &\quad + \left\|M(v_{i-1}) - v_i\right\|_1 + \|v_i - v'_i\|_1 \\ &\leq \left\|\widetilde{M} - M\right\|_1 \cdot \|v'_{i-1}\|_1 + \|v_{i-1} - v'_{i-1}\|_1 + \|v_i - v'_i\|_1 \\ &\leq \tfrac{\delta}{6T} \cdot (1+\delta) + \delta + \delta \leq 3\delta.\end{aligned}$$

Thus, $\|w\|_2 \leq \|w\|_1 \leq 3T\delta$. Consider the quantity $\langle \alpha, w\rangle = \sum_{i,j} \alpha_{i,j} w_{i,j}$ that the algorithm estimates. Note that $\mathbb{E}\left[\langle \alpha, w\rangle\right] = 0$ and that $\mathbb{E}\left[\langle \alpha, w\rangle^2\right] = \sum_{i,j} w_{i,j}^2$. Chebyshev's Inequality implies that with probability at least 0.99, we have $|\langle \alpha, w\rangle| \leq 30T\delta$. This implies that with probability at least $(0.99)^{11} \geq 0.8$, every iteration of the inner loop in **Algorithm 2** does not reject.

Soundness of the Algorithm: Suppose a dishonest prover produces a stream $v'_0, \ldots, v'_T$ such that $\|v'_T(1) - v_T\|_1 \geq \frac{1}{4}$. The verifier always returns $\perp$ if $v'_0 \neq e_1$, so we may assume that $v'_0 = e_1$. Let $\varepsilon = \frac{1}{10T}$. We argue that for some $i \in [T]$, we must have $\|w_i\|_1 \geq \varepsilon$. Assume by contradiction that $\left\|\widetilde{M}(v'_{i-1}) - v'_i\right\|_1 \leq \varepsilon$ for all $i \in [T]$. Hence, by Triangle Inequality and

Equation (1), (and since $\widetilde{v}_0 = e_1$) we have

$$\begin{aligned}
\|\widetilde{v}_T - v'_T\|_1 = \left\|\widetilde{M}^T(v'_0) - v'_T\right\|_1 &\leq \sum_i \left\|\widetilde{M}^{T-(i-1)}(v'_{i-1}) - \widetilde{M}^{T-i}(v'_i)\right\|_1 \\
&\leq \sum_i \left\|\widetilde{M}^{T-i}\right\|_1 \cdot \left\|\widetilde{M}(v'_{i-1}) - v'_i\right\|_1 \\
&\leq \sum_i \varepsilon \\
&\leq T\varepsilon.
\end{aligned}$$

Equation (2) implies that $\|\widetilde{v}_T - v_T\|_1 \leq \frac{\delta}{2}$. This implies that $\|v'_T - v_T\|_1 \leq \frac{\delta}{2} + T\varepsilon$. We assumed that $\|v_T - v'_T\|_1 \geq \frac{1}{4}$. Hence, it follows that

$$\tfrac{1}{4} \leq \tfrac{\delta}{2} + T\varepsilon.$$

Since we chose $\varepsilon = \frac{1}{10T}$ and $\delta \leq 1/10$, this is a contradiction. Thus, we must have $\|w\|_2 \geq \frac{\|w\|_1}{nT} \geq \frac{\varepsilon}{nT}$. Note that $\mathbb{E}\left[\langle \alpha, w\rangle\right] = 0$ and $\mathbb{E}\left[\langle \alpha, w\rangle^2\right] = \|w\|_2^2$. Furthermore,

$$\mathbb{E}\left[\langle \alpha, w\rangle^4\right] = \mathbb{E}\left[\sum_{i,j,k,l} w_i w_j w_k w_l \alpha_i \alpha_j \alpha_k \alpha_l\right] \leq 6 \sum_{i,j} w_i^2 w_j^2 \leq 6\|w\|_2^4$$

Here, we used the fact that the random variables are 4-wise independent. The Payley-Zygmund Inequality implies that

$$\Pr\left[\langle \alpha, w\rangle^2 \geq \frac{1}{10} \cdot \|w\|_2^2\right] \geq \left(1 - \frac{1}{10}\right)^2 \cdot \frac{\left(\mathbb{E}\left[\langle \alpha, w\rangle^2\right]\right)^2}{\mathbb{E}\left[\langle \alpha, w\rangle^4\right]} \geq \frac{1}{8}.$$

This, along with the fact that $\|w\|_2 \geq \frac{\varepsilon}{nT}$ implies that $\Pr\left[|\langle \alpha, w\rangle| \geq \frac{\varepsilon}{10nT}\right] \geq \frac{1}{8}$. By repeating this experiment 11 times, we can ensure that with probability at least $1 - (1 - 1/8)^{11} \geq 3/4$, we find at least one instance so that $|\langle \alpha, w\rangle| \geq \frac{\varepsilon}{10nT}$. Since $\delta \leq \frac{1}{10^4 nT^3}$, we have

$$\tfrac{\varepsilon}{10nT} = \tfrac{1}{100nT^2} > 30T\delta$$

Thus, with probability at least 3/4, we have $|\langle \alpha, w\rangle| > 30T\delta$. This implies that the algorithm returns $\perp$ with probability at least 3/4. □

Remark: In our algorithm, the verifier essentially checks if a certain random linear combination of the $\widetilde{M}(v'_{i-1})$ is close to the same linear combination of the v'_i. The verifier could have instead checked if for every $i \in [T]$, $\widetilde{M}(v'_{i-1})$ was close to v'_i. (This could have been done by testing for each $i \in [T]$ whether $\left\langle \widetilde{M}(v'_{i-1}), w \right\rangle$ was close to $\langle v'_i, w\rangle$ for a certain random vector w.) While the latter test is conceptually simpler, the former test has the additional useful property that the verifier simply computes a linear function in the variables $\{v_i(j)\}$.

4 Relations with Other Complexity Classes

In this section we restate and prove the rest of the propositions in Section 1.1 and Section 1.2.

Proposition 1.3.

$$\mathsf{RPP}(\log n) = \mathsf{ZPP}.$$

Proof. Given a language L in ZPP, consider the zero-error randomized algorithm for L that outputs the correct answer with probability at least $\frac{3}{4}$. This algorithm can be directly viewed as a robustly-randomized algorithm where all the random bits are untrusted, and thus $\mathsf{ZPP} \subseteq \mathsf{RPP}(0)$.

On the other hand, given a language L in $\mathsf{RPP}(\log n)$, consider its robustly-randomized algorithm A with $O(\log n)$ trusted random bits R_1 and (polynomially many) untrusted random bits R_2. We design a zero-error randomized algorithm for L as follows. After sampling $r \sim R_2$ and storing r, it iterates through all $2^{O(\log n)}$ possible values of R_1. In each iteration it runs A based on the chosen value of R_1 and r, and takes the majority vote (breaking ties arbitrarily) over all $2^{O(\log n)}$ outputs as the final output.

To see the correctness of the algorithm, first notice that it is zero-error as for every r, any incorrect answer can be outputted by at most $\frac{1}{4}$ fraction of R_1 and thus cannot win the majority vote. Also notice that the correct answer wins the majority vote if it is outputted for more than half of R_1. Since the correct answer is outputted with probability at least $\frac{3}{4}$ over R_1, R_2, it must win the majority vote (and be the final output) with probability at least $\frac{1}{2}$ over R_2.

Therefore we showed $\mathsf{RPP}(\log n) \subseteq \mathsf{ZPP}$, and together it holds that $\mathsf{RPP}(0) = \mathsf{ZPP} = \mathsf{RPP}(\log n)$. □

Proposition 1.4. *For every k,*

$$\mathsf{BPP}(k) \subseteq \mathsf{RPP}(k).$$

Proof. Given a language L in $\mathsf{BPP}(k)$, consider the bounded-error randomized algorithm for L that outputs the correct answer with probability at least $\frac{3}{4}$. This algorithm can be directly viewed as a robustly-randomized algorithm where all the $O(k)$ random bits are trusted, and thus $\mathsf{BPP}(k) \subseteq \mathsf{RPP}(k)$. □

Corollary 1.5. *For every k,*

$$\mathsf{BPP}(k)^{\mathsf{ZPP}} \subseteq \mathsf{RPP}(k).$$

Proof. Consider an algorithm in $\mathsf{BPP}(k)^{\mathsf{ZPP}}$ with error probability $\frac{7}{8}$. By Proposition 1.3 and Proposition 1.4, we can simulate such an algorithm with a $\mathsf{RPP}(k)$ algorithm that uses only trusted random bits itself, but answers the oracle calls with $\mathsf{RPP}(0)$ subroutines which uses only untrusted random bits. The overall number of trusted random bits is $O(k)$.

Suppose there are m oracle calls. We can assume each $\mathsf{RPP}(0)$ subroutine outputs the correct answer to the oracle call with probability at least $1 - \frac{1}{8m}$ by repetition. The algorithm aborts whenever one of the subroutines outputs $\perp$, so the overall success probability is at least $\frac{3}{4}$ by union bound. □

Proposition 1.6.

$$\bigcup_c \mathsf{RPP}(n^c) = \mathsf{BPP}.$$

Proof. For every c, given a language L in $\mathsf{RPP}(n^c)$ and its robustly-randomized algorithm. We modify it so that whenever it is supposed to output $\perp$, it simply outputs 0 instead. Then the algorithm can be viewed as a randomized algorithm for L with error bounded by $\frac{1}{4}$, and thus $\mathsf{RPP}(n^c) \subseteq \mathsf{BPP}$. Combined with Proposition 1.4 we have

$$\mathsf{BPP} = \bigcup_c \mathsf{BPP}(n^c) = \bigcup_c \mathsf{RPP}(n^c). \qquad \square$$

Proposition 1.7. *For every k, if $\mathsf{BPP}(k)$ and ZPP do not contain one another, then*

$$\mathsf{BPP}(k) \cup \mathsf{ZPP} \neq \mathsf{RPP}(k).$$

Proof. Take $L_1 \in \mathsf{BPP}(k) \setminus \mathsf{ZPP}$, and $L_2 \in \mathsf{ZPP} \setminus \mathsf{BPP}(k)$. We claim that $L = L_1 \oplus L_2$ (the symmetric difference of L_1 and L_2) gives the desired separation.

Since $L \in \mathsf{BPP}(k)^{\mathsf{ZPP}}$, by Corollary 1.5 we have $L \in \mathsf{RPP}(k)$. On the other hand, $L \notin \mathsf{BPP}(k)$ since otherwise so does $L_2 = L \oplus L_1$. Similarly $L \notin \mathsf{ZPP}$. Notice that here we crucially use the fact that both classes are closed under symmetric difference. Therefore $\mathsf{BPP}(k) \cup \mathsf{ZPP} \neq \mathsf{RPP}(k)$. $\square$

Before we restate Proposition 1.8, we need to formally define the the language recognized by an interactive proof where the verifer is allowed to abort. This is slightly different from the usual definition, as it is symmetric in a language L and its complement L^c.

Definition 4.1. *A language L is recognized by an interactive proof protocol $(\mathcal{P}, \mathcal{V})$ which outputs 1, 0 or $\perp$, if the following requirements are satisfied:*

1. *With the honest prover $\mathcal{P}$, $(\mathcal{P}, \mathcal{V})$ outputs 1 if $x \in L$ and outputs 0 if $x \notin L$ with probability at least $\frac{3}{4}$.*

2. *With any (computationally unbounded) prover $\mathcal{P}'$, $(\mathcal{P}, \mathcal{V})$ outputs 1 or $\perp$ if $x \in L$ and outputs 0 or $\perp$ if $x \notin L$ with probability at least $\frac{3}{4}$.*

Proposition 1.8. *A language L is in $\mathsf{RPP}(k)$ if and only if it is recognizable by an interactive proof with a probabilistic polynomial-time prover and a probabilistic polynomial-time verifier, where the verifier is limited to at most $O(k)$ random bits.*

Proof. Given a robustly-randomized algorithm, we design an interactive proof where the verifier simulates the algorithm, but with the untrusted random bits provided by the prover as proof. The verifier also aborts when the length of the proof does not equal to the number of untrusted random bits it needs. On the other hand, given a interactive proof protocol, we can simulate the protocol with a robustly-randomized algorithm where the random bits used by the verifier are trusted and the random bits used by the prover are untrusted. The equivalence follows from Definition 1.2 and Definition 4.1. $\square$

Proposition 1.11.

$$\mathsf{RPL}(1) = \mathsf{ZPL}.$$

Proof. The proof is almost the same as the one for Proposition 1.3. The proof for $\mathsf{ZPL} \subseteq \mathsf{RPL}(0)$ goes in exactly the same way. However, in the proof for the other direction $\mathsf{RPL}(1) \subseteq \mathsf{ZPL}$, there is a caveat: a logarithmic-space algorithm cannot afford to store all the untrusted random bits. Therefore, instead of sequential iterations over all possible values of the trusted bits, we do it in parallel, so that the untrusted random bits are still read-once. As there are $O(1)$ trusted random bits, this only increases the space by an $O(1)$ factor. $\square$

5 Query-Complexity Separations

Proof for Proposition 1.12. We define the subsets $A, B \subset \{0,1,2\}^{2^{n+k}}$ as follows. Let R_1 and R_2 be uniformly distributed over $\{0,1\}^k$ and $\{0,1\}^n$ respectively. For every function $f : \{0,1\}^{n+k} \to \{0,1,2\}$, we say that

- $f \in A$ if and only if $\Pr[f(R_1, R_2) = 0] \geq \frac{3}{4}$, and for every $r \in \{0,1\}^n$,
$$\Pr[f(R_1, r) \in \{0,2\}] \geq \frac{3}{4}.$$
- $f \in B$ if and only if $\Pr[f(R_1, R_2) = 1] \geq \frac{3}{4}$, and for every $r \in \{0,1\}^n$,
$$\Pr[f(R_1, r) \in \{1,2\}] \geq \frac{3}{4}.$$

Clearly A and B are disjoint as $\Pr[f(R_1, R_2) = 0] + \Pr[f(R_1, R_2) = 1] \leq 1$.

1. A robustly-randomized protocol that distinguishes between $f \in A$ and $f \in B$ uses k trusted random bits R_1 and n untrusted random bits R_2. It makes a single query for $f(R_1, R_2)$ and based on the query result being $0, 1$ or 2, outputs A, B or $\perp$ accordingly. The correctness of the protocol is guaranteed by definition 1.1.

2. For any $\epsilon > 0$, consider a randomized protocol using at most $(1-\epsilon)n$ random bits and making at most q queries to f. Fix all the query answers to be 2, and let $Q \subseteq \{0,1\}^{n+k}$ be all the possible positions being queried over all choices of the random bits. Assuming $q \leq \frac{1}{4} \cdot 2^{\epsilon n}$, we have $|Q| \leq 2^{(1-\epsilon)n} q \leq \frac{1}{4} \cdot 2^n$.

   Now pick $f, g : \{0,1\}^{n+k} \to \{0,1,2\}$ to be:
$$f(r) = \begin{cases} 0 & \text{if } r \notin Q \\ 2 & \text{if } r \in Q \end{cases} \qquad g(r) = \begin{cases} 1 & \text{if } r \notin Q \\ 2 & \text{if } r \in Q \end{cases}$$
   It is straightforward to check that $f \in A$ and $g \in B$. The protocol behaves exactly the same on f and g, thus being incorrect with probability at least $\frac{1}{2}$ on either one of them. Therefore, a successful protocol must have $q \geq 2^{\Omega(n)}$.

3. Consider a zero-error randomized protocol making at most q queries to f. Fix f to be the constant function 0 and clearly $f \in A$. We arbitrarily fix a choice of the random bits where the protocol outputs $f \in A$, and let Q be the positions queried under the fixing. Assume that $|Q| \leq q \leq \frac{1}{4} \cdot 2^k$.

   Now pick $g : \{0,1\}^{n+k} \to \{0,1,2\}$ to be:
$$g(r) = \begin{cases} 1 & \text{if } r \notin Q \\ 0 & \text{if } r \in Q \end{cases}$$
   It is straightforward to check that $g \in B$. Under the fixed choice of random bits, the protocol behaves exactly the same on f and g , and thus incorrectly decides $g \in A$. Therefore, a successful protocol must have $q \geq 2^{\Omega(k)}$. □

References

[Aar18] Scott Aaronson: Certified Randomness from Quantum Supremacy. Talk at CRYPTO 2018, October 2018. 6

[AMS96] The Space Complexity of Approximating the Frequency Moments - Noga Alon, Yossi Matias, Mario Szegedy. STOC 1996. 12

[BBFGT22] Roozbeh Bassirian, Adam Bouland, Bill Fefferman, Sam Gun, Avishay Tal: On Certified Randomness from Quantum Advantage Experiments, 2022 6

[BCMVV21] Zvika Brakerski, Paul F. Christiano, Urmila Mahadev, Umesh V. Vazirani, Thomas Vidick: A Cryptographic Test of Quantumness and Certifiable Randomness from a Single Quantum Device. J. ACM 68(5): 31:1-31:47 (2021) 6

[BM84] Manuel Blum, Silvio Micali: How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. SIAM J. Comput. 13(4): 850-864 (1984) 4

[GR14] Anat Ganor, Ran Raz: Space Pseudorandom Generators by Communication Complexity Lower Bounds. APPROX-RANDOM 2014: 692-703 4

[INW94] Russell Impagliazzo, Noam Nisan, Avi Wigderson: Pseudorandomness for network algorithms. STOC 1994: 356-364 4

[IW97] Russell Impagliazzo, Avi Wigderson: P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma. STOC 1997: 220-229 4

[MS16] Carl A. Miller, Yaoyun Shi: Robust Protocols for Securely Expanding Randomness and Distributing Keys Using Untrusted Quantum Devices. J. ACM 63(4): 33:1-33:63 (2016) 6

[Nis92] Noam Nisan: Pseudorandom generators for space-bounded computation. Comb. 12(4): 449-461 (1992) 4

[NW94] Noam Nisan, Avi Wigderson: Hardness vs Randomness. J. Comput. Syst. Sci. 49(2): 149-167 (1994) 4

[NZ96] Noam Nisan, David Zuckerman: Randomness is Linear in Space. J. Comput. Syst. Sci. 52(1): 43-52 (1996) 6

[PAM+10] S. Pironio, A. Acin, S. Massar, A. Boyer de la Giroday, D. N. Matsukevich, P. Maunz, S. Olmschenk, D. Hayes, L. Luo, T. A. Manning, C. Monroe: Random Numbers Certified by Bell's Theorem. Nature volume 464, pages1021–1024 (2010) 6

[Sha81] Adi Shamir: On the Generation of Cryptographically Strong Pseudo-Random Sequences. ICALP 1981: 544-550 4

[SZ99] Michael E. Saks, Shiyu Zhou: $BP_H Space(S) \subseteq DSPACE(S^{3/2})$. J. Comput. Syst. Sci. 58(2): 376-403 (1999) 4

[Vad12] Salil Vadhan: Foundations and Trends in Theoretical Computer Science: Vol. 7: No. 1–3, pp 1-336, now publishers, 2012 5

[VV12] Umesh V. Vazirani, Thomas Vidick: Certifiable quantum dice: or, true random number generation secure against quantum adversaries. STOC 2012: 61-76 6

[Yao82] Andrew Chi-Chih Yao: Theory and Applications of Trapdoor Functions (Extended Abstract). FOCS 1982: 80-91 4

A Appendix

A.1 A BPL-complete Problem

We prove Proposition 2.2 which states that the Stochastic Matrix Powering Problem is complete for BPL. We already argued that this problem has a streaming proof between a randomized logspace prover and verifier and hence is in BPL. We now argue that any problem in BPL is logspace-reducible to this problem. The proof of this is similar to that of undirected reachability being complete for NL.

Consider any family of functions $\mathcal{F} = \{f_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ in BPL and consider a randomized logspace Turing Machine that decides $\mathcal{F}$. For any input $x \in \text{supp}(f_n)$, consider a graph G_x whose vertices consist of all possible configurations of the machine on input x. Since the machine is logspace, each configuration can be described by $O(\log n)$ bits and hence the number of vertices in this graph is $P(n) \le \text{poly}(n)$. Let s denote the initial state of the Turing Machine on input x and let us say it has the label 1. We add an additional vertex t with the label $P(n)+1$ and we draw two directed edges (with labels 1 and 0) from every final accepting configuration to the vertex t. There are two self-loops at t (with labels 1 and 0). Every vertex in this graph has two outgoing edges each corresponding to the outcome of a random coin toss. We denote the random walk matrix of this graph by M_x. Observe that M_x is a stochastic matrix whose entries are logspace-computable. Our reduction maps x to M_x.

Let $R(n)$ be the number of random bits read by the Turing Machine. We assume without loss of generality that the error of the Turing machine is at most 1/5 (by standard error reduction techniques). We are promised that $f_n(x) = 1$ if at least a 4/5 fraction of random walks starting at s of length $R+1$ reach t and $f_n(x) = 0$ if at most a 1/5 fraction of random walks starting at s of length $R+1$ reach t. Thus, we have a $(P(n)+1)\times(P(n)+1)$ matrix M_x such that $f_n(x) = 1$ if $M_x^{R(n)+1}[P(n)+1, 1] \ge 4/5$ and $f_n(x) = 0$ if $M_x^{R(n)+1}[P(n)+1, 1] \le 1/5$. This completes the reduction.