The Raymond and Beverly Sackler Faculty of Exact Sciences

The Blavatnik School of Computer Science

Applications of Pseudorandomness to Average-case Complexity

Thesis submitted for the degree of

Master of Science

by

Gonen Krak

This work was carried out under the supervision of

Professor Amnon Ta Shma

December 2019

Abstract

In this work we describe several important results in the field of average-case complexity and their relation to various well known pseudorandom objects. In addition, we give a more general and concise proof of Hirahara's worst-case to average-case reduction for the MinKT problem that that is based on a general reconstructive PRG. We then further analyze our proof and provide a new lossless-reduction between MinKT problems that reduces the hardness gap, a reduction that we believe that can be used to create better worst-case to average-case reduction for the MinKT problem.

Contents

1 Introduction

The study of the average-case complexity of intractable problems began in the 1970s motivated by two distinct applications: the developments of the foundations of cryptography and the search for methods to "cope" with the intractability of NP-hard problems.

All definitions of security for cryptographic problems require that any efficient algorithm that tries to "break" the protocol "succeeds" only with very small probability. The formalizations of breaking and succeeding depend on the specific application, but it has been known since the 1980s that there is a unifying concept: no cryptographic task (for example, electronic signature, or data encryption) is possible unless *one-way* functions exist. Informally, a one-way function is an efficiently computable function $f : \{0,1\}^* \to \{0,1\}^*$ that maps $\{0,1\}^n$ to $\{0,1\}^n$ such that, if we are given $f(x)$ for a random $x \in \{0,1\}^n$, it is intractable (in time polynomial in n) to find a pre-image x' such that $f(x) = f(x')$. In particular, the existence of one-way functions implies that there is a search problem in NP that is intractable to solve on random inputs sampled from a simple distribution. The fact that all of cryptography is predicated on the existence of average-case intractable problems in NP is a main motivation for the emergence of the theory of average-case complexity.

The second motivation for the study of the average-case complexity is reaching a better understanding of NP hard problems. Unless $\mathsf{P} = \mathsf{NP}$, we cannot hope for efficient algorithms that solve NP-complete problem exactly on all inputs .We may hope, however, for algorithms that are "typically efficient" on inputs sampled from distributions that occur in practice. In order to understand the limitations of such an approach, it would be desirable to have an "average-case analog" of the theory of NP-completeness. Such a theory would enable us to prove that for certain problems, with respect to certain distributions, it is impossible to have algorithms that perform well on "typical" inputs, unless an entire class of presumably intractable problems can be efficiently solved.

Not long after the basics of the average-case complexity theory were laid out, a third

important motivation for studying the average-case complexity of computational problems was found, and that is a deep connection between the existence of functions that are hard on average and the existence of efficient pseudorandom generators. In their seminal paper, Nisan and Wigderson [NW94] showed that the existence of a function in PSPACE that is hard on average, is equivalent to the existence of a pseudorandom generator that fools BPP algorithms(and more generally, polynomial-sized boolean circuits).

A very important and long-studied aspect of the average-case complexity theory is the field worst-case to average-case reductions, in which the relation between the worst-case hardness and the average-case hardness of computational problems is studied. The general question is whether the worst-case hardness of a problem is equivalent to the average-case hardness of the problem with respect to a "reasonable" distribution.

Worst-case to average-case reductions are known for complexity classes much higher than NP, or specific problems in NP∩coNP. For complexity classes above the polynomial-time hierarchy such as PSPACE and EXP, a general technique based on error-correcting codes provides a worst-case to average-case reduction [STV01]. Problems based on lattices admit worst-case to average-case reductions from some problems in $\mathsf{NP} \cap \mathsf{coNP}$ to distributional NP problems. In a seminal paper of Ajtai [Ajt96] it is shown that an approximation version of the shortest vector problem of a lattice in $\mathbb{R}^n$ admits a worst-case to average-case reduction.

There are significant obstacles to establishing worst-case to average-case connections for NP-complete problems. A standard technique to establish worst-case to average-case connections is by *black-box* reductions, meaning that a hypothetical heuristic algorithm is regarded as a (possibly inefficient) oracle. Building on Feigenbaum and Fortnow [FF93], Bogdanov and Trevisan [BT06] showed that if a language L reduces to a distributional NP problem via a black-box nonadaptive randomized polynomial-time reduction, then $L \in \mathsf{NP}/poly \cap \mathsf{coNP}/poly$ and thus unlikely to be NP-complete. In his seminal paper, Hirahara [Hir18] showed a non black-box worst-case to average-case reduction for the MinKT problem, a NP problem that is considered to be outside $\mathsf{NP} \cap \mathsf{coNP}$. Surprisingly, though not directly stated in his paper, Hirahara's technique is based on a pseudorandom object called a *reconstructive pesudorandom generator*, and

thus reveals another interesting connection between the theory of average-case complexity and the theory of pseudorandomness.

In this work we describe several important results in the field of average-case complexity and their relation to various well known pseudorandom objects. We begin with defining the basic notions of average complexity, and then we proceed to discuss various pseudorandom objects such as error correcting codes, PRGs, randomness extractors, while showing their importance in several important results in the field of average-case complexity. Finally, we give a more general and concise proof of Hirahara's worst-case to average-case reduction for the MinKT problem that that is based on a general reconstructive PRG. We then further analyze our proof and provide a new lossless-reduction between MinKT problems that reduces the hardness gap, a reduction that we believe that can be used to create better worst-case to average-case reduction for the MinKT problem.

2 Basics of Average-case Complexity

A *distributional decision problem* is a pair $(L, \mathcal{D})$ where L is a language and $\mathcal{D}$ describes how inputs are distributed. There are various possible formalizations of how $\mathcal{D}$ is specified, of what constitutes a "natural" subset of distribution of inputs to restrict to, and of what it means for a distributional problem to have a good-on-average algorithm. We will generally describe $\mathcal{D}$ as $\mathcal{D} = \{D_n\}_{n=1}^{\infty}$ as a family of distributions D_n that describe the distribution of the inputs of length n. We will call such $\mathcal{D}$ an *ensemble*.

The most standard notion of a "natural" ensemble is the notion of polynomial-time sampable ensemble, which corresponds to distributions that can be sampled by a polynomial-time algorithms

Definition 2.1 (Samplable Ensemble)**.** *An ensemble $\mathcal{D} = \{D_n\}_{n=1}^{\infty}$ is polynomial time samplable if there is a randomized algorithm A that, on input a number n, outputs a string in $\{0,1\}^*$ and the following holds:*

- *There is a polynomial p such that, on input n, A runs in time at most $p(n)$ regardless of its internal coin tosses*
- *For every n and every $x \in \{0,1\}^*$, $Pr[A(n) = x] = D_n(x)$.*

A *distributional complexity class* is a collection of distributional decision problems. For a class of languages $\mathcal{C}$ and a class of ensembles $\mathcal{D}$ we use $(\mathcal{C}, \mathcal{D})$ to denote the distributional complexity class consisting of all problems (L, D) where $L \in \mathcal{C}$ and $D \in \mathcal{D}$.

We next define the notion of average-case tractability. The are various natural ways to define when an algorithm runs fast on average with respect to a distributional problem. Those ways were first introduced by Levin [Lev86], which also proved that they are equivalent. We will present the one of those definitions, the definition of an *errorless-heuristic-algorithm*, which we will use throughout the paper.

One of the ways to think about a polynomial running time in average is that the solving algorithm A should run in a polynomial time on most inputs. On some inputs A might run very slowly, but we can then think of the inputs on which A takes super-polynomial time as inputs on which A "fails," because we have to stop the computation without being able to recover the result. The notion of an algorithm that fails on some inputs is captured by the following definition

Definition 2.2 (Errorless Heuristic algorithm)**.** *Let L be a language, D be an ensemble, and $\delta : \mathbb{N} \to \mathbb{R}^+$. We say that an algorithm A is an errorless heuristic algorithm for (L, D) with failure probability at most δ if*

- *For every n and every x in the support of D_n, $A(x)$ outputs either $L(x)$ or the special failure symbol $\perp$*
- *For every n, $Pr_{x \sim D_n}[A(x) = \perp] \leqslant \delta(n)$*

If an algorithm A is an errorless heuristic algorithm for (L, D) with failfure probability at most δ we say that $(L, D) \in Heur_\delta$.

We end this section with another important notion if the field of average-complexity which is the notion of *hardness* of a boolean function. Intuitively, a function should be hard on average if it is hard to compute correctly on randomly chosen inputs. This intuition is captured by the following definition

Definition 2.3 (Hardness of a Boolean Function)**.** *Let $f : \{0,1\}^* \to \{0,1\}$ be a boolean function and $\rho : \mathbb{N} \to [0,1]$ be a polynomial. We define $H^{\rho}_{avg}(f)$ to be the function from $\mathbb{N}$ to $\mathbb{N}$ that maps every number n to the largest number S such that*

$$Pr_{x\sim_R\{0,1\}^n}[C(x) = f(x)] < \rho(n)$$

for every boolean circuit C on n inputs of size at most S. We also define the average-case hardness of a function f to be

$$H_{avg}(f) = \max\left\{S \mid H^{\frac{1}{2}+\frac{1}{S}}_{avg}(f)(n) \geqslant S\right\}$$

and worst case hardness of a function to be

$$H_{wrs}(f) = H^1_{avg}(f)$$

In other words, a function's average-case hardness is the maximal size of circuits that calculate values of f better than random guessing (i.e. with probability higher than $\frac{1}{2}$) with advantage that is proportional to the circuit size.

3 Error Correcting Codes and Worst-case to Average-case Reductions of Boolean Functions in PSPACE

An important result in the field of average-case complexity is the worst-case case to average-case reduction for boolean functions that can be calculated in EXP (the result also holds for boolean functions in $PSPACE$ and $P^{\#P}$). In this section we will explain the main idea behind the result and the important role the first pseudorandom object that we will work with - *error correcting codes*.

Assume we have a boolean function f with worst-case hardness $H_{wrs}(f)$. We wish to create from f a new function f' that have a similar average-case hardness. To do so, we desire a way to transform any function f to another function g such that if there is a small circuit that computes g approximately (i.e., correctly outputs $g(x)$ for many x) then there is a small circuit that computes f at all points. Taking the contrapositive, we can conclude that if there is no small circuit that computes f then there is no small circuit that computes g approximately.

Let us reason abstractly about how to go about the above task. We will view a function $f : \{0,1\}^n \to \{0,1\}$ as its truth table, namely, as a string of length 2^n, and view any circuit C for computing this function as a device that, given any index $x \in [2^n]$ gives the x'th bit in this string. If the circuit only computes g on "average" then this device may be thought of as only partially correct; it gives the right bit only for many indices x's, but not all. Thus we need to show how to turn a partially correct string for g into a completely correct string for f. This is of course reminiscent of error correcting codes (ECC), but with a distinct twist involving computational efficiency of decoding, which is called *local decoding*.

Error correcting codes has a wide variety of usage in almost every field in computer science, and in particular they play a crucial building block in theory of pseudorandomness. We start with a definition of a binary error correcting code

Definition 3.1 (Binary Error Correcting Code)**.** *A function $C : \{0,1\}^n \to \{0,1\}^m$ is called a binary error correcting with relative distance δ if for every $x \neq y \in \{0,1\}^n$ we have*

$$\Delta(C(x), C(y)) \geqslant \delta$$

where Δ is the relative hamming distance defined by

$$\Delta(x, y) = \frac{1}{m}|\{i \mid x_i \neq y_i\}|$$

We note that the notion of error correcting has a more general definition but the above definition will be sufficient for our needs.

The mere existence of an error correcting code is not sufficient for most applications: we need to actually be able to compute them. For this we need to show an explicit function $C : \{0,1\}^n \to \{0,1\}^m$ that is an ECC satisfying the following properties:

Efficient encoding - There is a polynomial-time algorithm to compute $C(x)$ from x

Efficient decoding - There is a polynomial time algorithm to compute x from every y such that $\Delta(y, E(x)) < \rho$ for some ρ (For this to be possible, ρ must be less than $\frac{\delta}{2}$ where δ is the relative distance of C).

Our general strategy in creating an average-case hard function f' from a worst-case function f is to encode f using an error correcting code C, i.e. we wish to have $f' = C(f)$. To prove that it works, it suffices to show how to turn any circuit that correctly computes many bits of $C(f)$ into a circuit that correctly computes all bits of f. Note that our goal is to build a *circuit* that computes f. A circuit gets an index in the truth-table of f and returns the corresponding bit. Since we want to circuit computing f to be only polynomially-additive larger than the circuit that computes $C(f)$ on average, we require the decoding algorithm to run in polynomial time in the index length of the given corrupted word. We make this intuition formal using the definition of *local decoder*

Definition 3.2 (Local Decoder)**.** *Let $C : \{0,1\}^n \to \{0,1\}^m$ be an ECC and let ρ and q be some numbers. A local decoder for C handling ρ errors is an algorithm L that given a random access to a string y such that $\Delta(y, E(x)) < \rho$ for some (unknown) $x \in \{0,1\}^n$ and an index j, runs for $polylog(m)$ time and outputs x_j with probability at least $\frac{2}{3}$.*

Hence, if we have a circuit that $Circ$ that computes $C(f)$ on average, we can think of if as a random oracle access to a string y with small distance from $C(f)$, and using a local decoder algorithm that uses $Circ$ to implement the oracle calls, we get a randomized algorithm that computes f which in turn can be turned into a circuit of size $|Circ| + poly(m)$.

Thus, error correcting codes with local decoding give a worst-case to average-case reduction. However, the new function $C(f)$ has only "mild" average-case hardness. That is because, if f is is worst-case hard, then it seems hard to argue that the encoding of f, under any error correcting code is hard to compute with probability 0.6. The reason is that any error-correcting code has to have distance at most $1/2$, which implies that there is no decoding algorithm that can recover x from $C(x)$) if the latter was corrupted in more than a $\frac{1}{4}$ of its locations. Indeed, in this case there is not necessarily a unique codeword closest to the corrupted word. This seems like a real obstacle, and indeed was considered as such in many contexts where ECC's were used, until the realization of the importance of the following insight: "If y is obtained by corrupting $C(x)$ in, say, a 0.4 fraction of the coordinates (where E is some ECC with good enough distance) then, while there may be more than one codeword within distance 0.4 to y, there can not be *too many* such codewords.

This intuition is formalized by the following theorem:

Theorem 3.3 (Johnson Bound). *If $C : \{0,1\}^n \to \{0,1\}^m$ is an ECC with distance at least $\frac{1}{2} - \varepsilon$, then for every $x \in \{0,1\}^m$, and $\delta \geqslant \sqrt{\varepsilon}$, there exist at most $\frac{1}{2\delta^2}$ vectors $y_1, .., y_\ell$ such that $\Delta(x, y_i) \leqslant \frac{1}{2} - \delta$ for every $i \in [\ell]$.*

With the Johnson Bound in mind, we define the notion of list-decoding:

Definition 3.4 (List Decoder). *For every $n, m, L \in \mathbb{N}$ and $\varepsilon > 0$, a function $C : \{0,1\}^n \to \{0,1\}^m$ is called $(L, \frac{1}{2} - \varepsilon)$-list-decodable error-correcting code if there exists a function $Dec : \{0,1\}^m \to (\{0,1\}^n)^L$ such that, for every $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ with $\Delta(y, C(x)) \leqslant \frac{1}{2} - \varepsilon$ we have $x \in Dec(y)$. We call Dec a list-decoder of C.*

As in the previous case, since we need the decoding algorithm to run in polylogarithmic-time, we define the notion *local-list-decoding*

Definition 3.5 (Local-list Decoder). *Let $C : \{0,1\}^n \to \{0,1\}^m$ be an ECC and let ρ and q be some numbers. An algorithm L is called a local-list decoder for C handling ρ errors, if for every $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ with $\Delta(y, C(x)) \leqslant \rho$ there exists a number $i_0 \in [poly(\frac{n}{\varepsilon})]$ such that for every $j \in [m]$, on inputs i_0, j and with random access to y, L runs for $\frac{poly(\log m)}{\varepsilon}$ time and outputs x_j with probability at least $\frac{2}{3}$.*

To better understand this definition and why local-list decoding is complicated, note that what the algorithm L essentially does is to take the corresponding "index" of the correct original codeword in some list that describes the possible solutions, and using the index calculates the required bit of the corresponding codeword. That means that L, using a very few queries to y, should be able compute that list in a *consistent* way such that given an index to an element in that list, L should be able to output the required bit of the corresponding element, and that same L must work for every y.

After a long line of research, Sudan, Trevisan and Vadhan [STV01] managed to design an efficient local-list-decoding algorithm for the Reed-Muller code concatenated with the Hadamard code, and as a corollary proved the following worst-case to average-case theorem:

Theorem 3.6 (Worst-case to Average-case Reduction for boolean functions in $PSPACE$). *Let $S : \mathbb{N} \to \mathbb{N}$ and $f \in PSPACE$ such that $H_{wrs}(f)(n) \geqslant S(n)$ for every n. Then there exists a function $g \in PSPACE$ and constant $c > 0$ such that such $H_{avg}(g)(n) \geqslant S(n/c)^{1/c}$ for sufficiently large n.*

4 Pseudorandom Generators from Average-case Hardness

A long standing open problem in field of computational complexity is whether randomness gives us more power in devising efficient algorithms, i.e. whether $P = BPP$. One of the most basic notions in theory of pseudorandomness is the notion of a *pesudorandom generator*, or PRG in short. Roughly speaking, a PRG is an algorithm that takes as input a short seed of random bits and outputs a long string made of pseudorandom bits. We say that a PRG *fools* a class of circuits if for every circuit in that class, the output distribution of the PRG on a uniformly-distributed seed looks similar to uniform for all the circuits in the class. The most general definition of a PRG is the definition of a PRG against general boolean circuits

Definition 4.1 (Pseudorandom Generator)**.** *Let R be a distribution over $\{0,1\}^m$ and let $S \in \mathbb{N}$, $\varepsilon > 0$. We say that R is a (S, ε)-pseudorandom distribution if for every circuit C of size at most S*

$$|Pr[C(R) = 1] - Pr[C(U_m) = 1]| < \varepsilon$$

where U_m denotes the uniform distribution over $\{0,1\}^m$.
If $S : \mathbb{N} \to \mathbb{N}$ is a polynomial-time computable monotone function, then a function G is called a (S, ε)-PRG if:

- *For every $z \in \{0,1\}^\ell$, $|G(z)| = S(\ell)$ and $G(z)$ can be computed in time $2^{c\ell}$ for some constant c*
- *For every $\ell \in \mathbb{N}$, $G(U_\ell)$ is an $(S(\ell), \varepsilon)$-pseudorandom distribution.*

The function S is usually called the *stretch* of the PRG.

It is easy to see that if there exists a $(S(\ell), 1/10)$-PRG (where $1/10$ can be replaced by any small constant) then $BPTIME(S(\ell(n)) \subseteq DTIME(2^{c\ell(n)})$. This is true because one can deterministically enumerate on all possible seeds to the PRG, and for each seed calculate the corresponding PRG output and run the probabilistic algorithm on the original input and the PRG's output as random bits and see the results. Since randomized algorithms that run in $BPTIME$ make a correct answer with very high probability (close to 1) and the PRG's output distribution looks uniform to those algorithms, taking the majority of the answers will surely result in the correct answer.

In their important work titled "Hardness versus Randomness", Nisan and Wigderson [NW94] showed that the existence of a function $f \in DTIME(2^{cn})$ for some constant $c > 0$ with average case hardness $S(n)$ then there exists a PRG with roughly the same stretch. This implies that the existence of a function in EXP with average-case hardness of $2^{\varepsilon n}$ for some $\varepsilon > 0$ (i.e. the function cannot be approximated by circuits of that size) implies the existence of PRG's against polynomial-sized circuits and in turns implies that $P = BPP$.

The construction of Nisan and Wigderson is a black-box construction that can take any function and transform it into a PRG. The idea behind the construction is to output evaluations of the function on "pseudorandom" points that depend on the generator's seed, in a way that given a distinguisher circuit that knows to distinguish the PRG's output from the uniform distribution, it is possible to use it in order to construct a small circuit that approximates the original function. The contrapositive implies that if the original function is hard, there can't be a small circuit that distinguishes between the generator's output and the uniform distribution.

The "pseudorandom points" on which the generator evaluates the function are constructed using an object called *combinatorial design* that is defined as follows:

Definition 4.2 (Combinatorial Design)**.** *Let $\ell, n, d \in \mathbb{N}$ be numbers with $\ell > n > d$. A family $\mathcal{I} = \{I_1, ..., I_m\}$ of subsets of $[\ell]$ is called a (ℓ, n, d)-design if $|I_j| = n$ for every j and $|I_j \cap I_k| \leqslant d$ for every $j \neq k$.*

A design is basically a family of many sets of points in an interval of integers with a small number of intersections between them. Using the notion of a design, the NW PRG is defined as follows:

Definition 4.3 (NW PRG)**.** *If $\mathcal{I} = \{I_1, ..., I_m\}$ is a family of subsets of $[\ell]$ with $|I_j| = n$ for every j and $f : \{0,1\}^n \to \{0,1\}$ is any function, then the $(\mathcal{I}, f)$-NW PRG is the function $NW^f_{\mathcal{I}} : \{0,1\}^\ell \to \{0,1\}^m$ that maps any $z \in \{0,1\}^\ell$ to*

$$NW^f_{\mathcal{I}}(z) = f(z|_{I_1}) \circ f(z|_{I_2}) \circ ... \circ f(z|_{I_m})$$

where $z|_{I_j}$ are the bits of z indexed by I_j.

Nisan and Wigderson constructed a reconstruction algorithm R such that when given an oracle access to a circuit $Dist$ that distinguishes between U_m and $NW^f_{\mathcal{I}}(U_\ell)$ and a short non-uniform advice string(that depends on f), can approximately calculate f (i.e. can compute f on a significantly larger than $1/2$ fraction of the inputs).

In the next section we will describe more in detail the advice and the reconstruction algorithm. We end the section with the formal statement proved by Nisan and Wigderson:

Theorem 4.4 (Average-case Hardness implies PRG). *If $\mathcal{I}$ is an (ℓ, n, d)-design with $|\mathcal{I}| = 2^{d/10}$ and $f : \{0,1\}^n \to \{0,1\}$ is a function satisfying $2^d < \sqrt{H_{avg}(f)(n)}$, then $NW^f_{\mathcal{I}}$ is a $(H_{avg}(f)(n)/10, 1/10)$-PRG*

5 Randomness Extractors and Reconstructive PRGs

A randomness extractor is an algorithm that converts "weak source of randomness" into an almost uniform distribution by using a small number of additional truly random bits. Randomness extractors has a wide range of applications to many different areas of computer science, and is a central studied object in theory of pseudorandomness.

Definition 5.1 (Min-Entropy). *The min-entropy of a distribution X is defined as*

$$H_\infty(X) = \min_a\{-\log_2(Pr[X = a])\}$$

Definition 5.2 (Randomness Extractor). *A function $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ is (k, ε)-extractor if for every distribution X having k min-entropy we have*

$$|E(X, U_t) - U_m| < \varepsilon$$

where the distance is the statistical distance between distributions

Given an extractor, one can use a small random seed of uniform bits to change any "weak" source (on potentially very long strings) into an almost uniform source. This way, if a random source was compromised or weak to begin with, one can use an extractor to extract the real uniform bits from that source.

The problem of constructing efficient extractors with good parameters is a major researched problem in the field of pseudorandomness, and many different constructions are known.

In this section we describe Trevisan's extractor construction [Tre01] which relies on the notion of a reconstructive pseudorandom generator.

In his work, Trevisan noted that the NW PRG is actually a black-box transformation that can take any string x and transform it into a distribution $NW^x(U_t)$ in a way that given a distinguisher $Dist$ that distinguishes between the uniform distribution U_m and $NW^x(U_t)$, one can apply a reconstruction algorithm R^{Dist} on a small string of advice $a = A(x)$ to approximately-reconstruct x. Trevisan's idea is that if encode x using a list-decodable ECC before applying the NW PRG, we can use the list-decoding algorithm after the reconstruction procedure to re-construct x completely and not approximately. Such object is called a reconstructive PRG and is formally defined as follows:

Definition 5.3 (Reconstructive Pseudorandom Generator)**.** *A triplet of functions* (G, A, R) *where:*

- $G : \{0,1\}^n \times \{0,1\}^{r_G} \to \{0,1\}^m$ *is called the* generator *function,*
- $A : \{0,1\}^n \times \{0,1\}^{r_A} \to \{0,1\}^a$ *is called the* advice *function,*
- $R : \{0,1\}^a \times \{0,1\}^{r_A} \times \{0,1\}^{r_R} \to \{0,1\}^n$ *is called the* reconstruction *function,*

is called a (p, q) reconstructive pseudorandom generator *if for every* $x \in \{0,1\}^n$ *and every distinguisher* $T : \{0,1\}^m \to \{0,1\}$ *that knows to distinguish between the uniform distribution* U_m *and the generated output* $G(x, U_{r_G})$ *with advantage* p*, we have that*

$$\Pr_{y \sim U_{r_A}, z \sim U_{r_R}} [R^T(A(x,y), y, z) = x] \quad \geqslant \quad q.$$

Though Trevisan was the first to construct such an object, different constructions of reconstructive-PRGs are known [Tre01, TSZS06, TSUZ07, SU05, Uma02, TSUZ07].

To better understand the notion of reconstructive PRG, we briefly describe Trevisan's construction.

Definition 5.4 (Generator function of Trevisan's reconstructive PRG)**.** *Let* $C : \{0,1\}^n \to \{0,1\}^{\bar{n}}$ *be a* $[\bar{n}, n, \frac{1}{2} - \varepsilon]$ *be a binary list-decodable ECC and let* $Z_1, .., Z_m \subseteq [r_E]$ *be a* (ℓ, ρ)*-design with* $\ell = \log(\bar{n})$*. The Trevisan extractor* $Tre : \{0,1\}^n \times \{0,1\}^{r_E} \to \{0,1\}^m$ *is defined*

by

$$Tre(x,y) = C(x)_{y|_{Z_1}} \circ C(x)_{y|_{Z_2}} \circ ... \circ C(x)_{y|_{Z_m}}$$

where $y_{|Z_j}$ are the bits from y indexed by Z_j.

We know describe the advice function and the reconstruction function. This is only a brief description and we argue the read to read Trevisan's paper and the NW paper for a full description.

The advice function: Given $x \in \{0,1\}^n$ we first compute $\hat{x} = C(x)$ as with the extractor. We view y as being composed of two parts: $i \in [m]$ and $\beta \in \{0,1\}^{r_E - \ell}$ (so $r_A = \log m + r_E - \ell$). The output of A comprises the evaluations of $\hat{x}$ on a subset of inputs that is determined by i and β. We now describe what these inputs are.
Each input in the set is an ℓ bit string that is determined by β together with j and an additional string γ, where j ranges over $[i-1]$ and γ ranges over all strings of length $|S_i \cap S_j|$. Such a string is obtained by first writing down the r_E-bit string that has β in coordinates $[r_E]\backslash S_i$, γ in coordinates $S_i \cap S_j$ and zeroes elsewhere, and then projecting onto coordinates S_j. We denote this final string by $w(\beta, j, \gamma)$. Note that $w(\beta, j, \gamma)$ coincides with β on $S_j \backslash S_i$, and it coincides with γ on $S_i \cap S_j$. The advice function outputs the evaluations of $\hat{x}$ for all $j < i$ and all $\gamma \in \{0,1\}^{|S_i \cap S_j|}$. Formally:

$$A(x,i,\beta) = (\hat{x}(w(\beta,j,\gamma)))_{j<i, \gamma \in \{0,1\}^{|S_i \cap S_j|}}$$

The reconstruction function: The reconstruction function's inputs are $A(x,i,\beta)$ and the pair (i,β). The goal of the reconstruction function is to output x.
We mirror the process used by A to compute its output. For each $w \in \{0,1\}^\ell$, we write down the r_E-bit string y that has β in coordinates $[r_E]\backslash S_i$ and w in coordinates S_i.
Observe that for $j < i$, $b_j = \hat{x}(y_{|S_i})$ can be found in the advice $A(x,i,\beta)$. We thus feed the bits $b_1, .., b_{i-1}$ to the next-bit predictor T, i.e. we compute $z(w) = T(i, b_1, ..., b_{i-1})$, where T is a next-bit predictor that is obtained by the distinguisher $Dist$. This gives a guess for the value of $\hat{x}(y_{|S_i}) = \hat{x}(w)$. After obtaining guesses for all w, we output x for which $\hat{x}(\cdot)$ is closet (in Hamming distance) to $z(\cdot)$.

Though Trevisan didn't explicitly define reconstructive PRG in his paper, he noted that

the generating function of a reconstructive PRG can be used for extraction. Roughly, Trevisan's argument goes as follows:

Let $X \subseteq \{0,1\}^n$ be a large subset. If $G(X, U_{r_G})$ is not close to uniform, then there exists a function $Dist$ that ε-distinguishes $G(X, U_{r_G})$. By averaging, we can even say that $Dist$ $\varepsilon/2$-distinguishes $G(x, U_{r_G})$ for many $x \in X$. Therefore, for many $x \in X$ there exists a short advice string $z = A(x, \cdot)$ for which $R^{Dist}(z, \cdot)$ outputs x. The number of strings x with such short descriptions cannot exceed the number of possible advice strings. We conclude that if $G(X, U_{r_G})$ is not close to uniform, then X is small. The contrapositive says that if X is large, then $G(X, U_{r_G})$ is close to uniform; In other words, G is an extractor.

6 Worst-case to Average-case Reduction for the MinKT Problem using Reconstructive PRGs

In this section show a very interesting application of reconstructive PRGs first noted (though not explicitly stated) by Hirahara [Hir18]. It turns out that a reconstructive PRG naturally gives an elegant worst-case to average-case reduction for the MinKT problem, an important NP problem that is considered to be hard.

6.1 Background: Kolmogorov Complexity and the MinKT Problem

A program P is a tuple of the form $P = (M, A)$ where M is a fixed-sized description of a Turing machine, and A is an input to the Turing machine. The *length* (or size) of the program is defined as $|P| = |M| + |A|$. For a string $x \in \{0,1\}^*$, the *Kolmogorov Complexity* $K_t(x)$ of x with respect to time t is defined as the length of the shortest program P such that P outputs x within t steps.

Kolmogorov complexity enables us to define a notion of randomness, or more correctly, a notion of *compressibility* of a string. As the Kolmogorov complexity of a string is larger, it means that the string contains a large amount of non-uniformity, and thus is hard to compress. In order to have a better understanding of the notion of Kolmogorov Complexity, let's look at few simple facts:

- We always have $K_t(x) \leqslant n + O(1)$ for sufficiently large t for any string x. That is

because one can simply "compress" the string in an empty manner just by using the string itself, so a Turing machine that just contains x in its memory (or gets it as input) and just prints it is valid description of x.

- Despite having a clear upper bound, $K_t(x)$ does not have a "natural" lower bound when we allow t to grow as we wish. A simple way to see it - for every integer k define the logstar function $\log_k^*(n)$ defined as $\log_k^*(n) = \log(\log(...\log(n)...))$, i.e. the logarithm function composed to itself k times. Denote $Pow_k^*(x) = (\log_k^*)^{-1}(x)$ to be the function $2^{2^{\cdots 2^x}}$ where the 'power tower' is of size k. Then, even if we look for example at the n-bit all 1's string 1^n, we can easily see that one can describe it by the program $P_k = (M_k, A_k)$ where M_k is a Turing machine that gets as input a number x and prints $1^{Pow_k^*(x)}$ and A_k is its input of size $\log_k^*(n)$. We thus get that for every integer k we have $K_{t=Pow_k^*(n)}1^n \leqslant \log_k^*(n)$.

- A well known related problem in computer science is the Busy-Beaver problem, which asks how many 1's a k-state Turing machine can print while starting on a blank input. This number is denoted by $BB(k)$, and it is known that there is no computable upper bound on the BB function, which in turn gives that there is no computable lower bound even on the Kolmogorov Complexity of the all 1's string.

- Though we do not have a worst-case lower bound on the Kolmogorov Complexity of a string, we can easily show by a simple counting argument that there can't be many strings of small description. More specifically, if we fix $1 \leqslant k \leqslant n$ then the fraction of the n-bit strings of Kolmogorov Complexity less than k (with respect to any time t) is at most 2^{k-n}. That is true because the number of possible descriptions (strings) of length $< k$ is $1 + 2 + 3 + ... + 2^{k-1} = 2^k - 1 < 2^k$ and thus the fraction of strings of Kolmogorov Complexity less than k is at most 2^{k-n}.

A well known complexity problem that is studied for a long time is the MinKT problem:

Definition 6.1 (MinKT). *Given two functions $r, t : \mathbb{N} \to \mathbb{N}$, the language $\mathrm{MinKT}[r, t]$ is defined to be:*

$$\mathrm{MinKT}[r, t] = \{x \mid K_{t(|x|)}(x) < r(|x|)\}$$

We say x us non-r-random with respect to t if $x \in \mathrm{MinKT}[r,t]$.

Note that for efficiently commutable function r and polynomial t, $\mathrm{MinKT}[r,t]$ is in NP: to prove that an input string x is in the language, the prover provides a program P and the verifier validates P generates x in time $t(|x|)$. Indeed, this is exactly why we need the parameter t - it enables the verifier to verify the prover's claim about the Kolmogorov complexity of a string and thus puts the language in NP.

We now define a gap version of the problem:

Definition 6.2 (Promise version of MINKT)**.** *For any four functions $\alpha, \beta, t_1, t_2 : \mathbb{N} \to \mathbb{N}$ we define the promise problem $\mathrm{GapMinKT}_{(t_1,\alpha),(t_2,\beta)}$ to be:*

Yes instances: *x such that $K_{t_2} \geqslant \beta(|x|)$.*

No instances: *x such that $K_{t_1}(x) \leqslant \alpha(|x|)$*

Our goal is to show a probabilistic reduction from the worst-case version of the $\mathrm{GapMinKT}$ problem to its average-case version. That means, we wish to probabilistically solve $\mathrm{GapMinKT}$ on the worst-case, given an algorithm that solves $\mathrm{GapMinKT}$ only on a fraction of the inputs.

6.2 Worst-case to Average-case Reduction for GapMinKT using Reconstructive PRGs

There are many different formulations of the notion of "solving a problem on average". We will work with the notion of an errorless heuristic algorithm, defined as follows:

Definition 6.3 (Errorless Heuristic algorithm)**.** *Let L be a language, $D = \{D_n\}$ be a family of distributions, and $\delta : \mathbb{N} \to \mathbb{R}^+$. We say an algorithm A is an errorless heuristic algorithm for (L, D) with failure probability at most δ if*

- *For every n and every x in the support of D_n, $A(x)$ outputs either $L(x)$ or the special failure symbol $\perp$.*
- *For every n, $Pr_{x \sim D_n}[A(x) = \perp] \leqslant \delta(n)$.*

If an efficient algorithm A (i.e., an algorithm in P) is an errorless heuristic algorithm for (L, D) with failure probability at most δ we say that $(L, D) \in \mathrm{Heur}_\delta$.

We show a probabilistic reduction that reduces solving $\mathrm{GapMinKT}_{\alpha,\beta,t_1,t_2}$ on the worst case to solving $\mathrm{GapMinKT}_{\alpha',\beta',t_1',t_2'}$ in Heur_δ.

The result of Hirahara [Hir18] implemented with the language of re-constructive PRGs gives the following theorem:

Theorem 6.4 (Average-case to Worst-case Reduction)**.** *Suppose*

$$(\mathrm{GapMinKT}_{\alpha',\beta',t_1',t_2'}, U) \in \mathrm{Heur}_\delta$$

solvable by an efficient, heuristic algorithm M. Let (G, A, R) be a (p, q)-reconstructive PRG with parameters $a(n), m(n), r_G(n), r_A(n), r_R(a(n))$ as in definition 3.

Set $\alpha, t_1, \beta, t_2 : \mathbb{N} \to \mathbb{N}$ such that:

- $\alpha(n) \leq \alpha'(m(n)) - r_G(n) - O(1)$
- $\beta(n) > a(n) + r_A(n) + r_R(m(n)) + O(1)$
- $t_1(n) = t_1'(m(n))/Time(G)(n)$
- $t_2(n) = Time(R)(a(n)) \cdot Time(M)(m(n))$.

Then, there exists a polynomial-time, probabilistic oracle machine P^M solving $\mathrm{GapMinKT}_{(\alpha,t_1),(\beta,t_2)}$ *worst case with one sided error, i.e.,* for any *input x of length n:*

- *If $K_{t_1}(x) \leq \alpha(n)$ then $P^M(x) = 0$ with probability* 1.
- *If $K_{t_2}(x) > \beta(n)$ then $P^M(x) = 1$ with probability $\geq 1 - p - \delta(m(n)) - 2^{-(m(n)-\beta'(m(n)))}$.*

We write this in short as:

$$\mathrm{GapMinKT}_{\alpha,\beta,t_1,t_2} \sim_{BPP} \mathrm{Heur}_\delta \mathrm{GapMinKT}_{\alpha',\beta',t_1',t_2'}$$

This shows a reduction from the worst-case version of the gap problem to the average-case heuristic version of the gap problem with respect to the uniform distribution. In particular, in order to probabilistically solve $\mathrm{GapMinKT}_{(\alpha,t_1),(\beta,t_2)}$ on the worst case, it is sufficient to solve $\mathrm{GapMinKT}_{(\alpha',t_1'),(\beta',t_2')}$ heuristically on a small fraction of the inputs (we will be able to take δ which is close to 1). Equivalently, if the worst-case version of the gap problem is hard, then it is also hard to heuristically solve the problem. In what follows we first discuss the core idea of the reduction, then proceed

to give the formal proof, and finally discuss the parameters and black-box properties of the reduction.

The idea behind the reduction is as follows: Fix the efficient machine M solving heuristically $\mathrm{GapMinKT}_{(\alpha',t'_1),(\beta',t'_2)}$ on the uniform distribution.

1. On one hand, if x is easy (i.e., not hard) then $G(x,y)$ is always easy and has a short description given by the short description of x and the randomness $y \in \{0,1\}^{r_G}$. Inn that case M must output "easy" or "quit", because M is a heuristic algorithm and has zero-sided error.

2. On the other hand, most strings in $\{0,1\}^m$ are hard (because there are few strings of small length). As M solves MinKT correctly on most strings, it must be the case that M outputs "hard" with non-negligible probability on a random input.

3. Also, if we are given a hard input x, then M should not be able to distinguish between the uniform distribution U_m and the generated distribution $G(x,U_{r_G})$ with advantage higher than p, because otherwise, by the definition of the reconstruction-PRG, there will be some $y \in \{0,1\}^{r_A}, y' \in \{0,1\}^{r_R}$ for which $R^M(A(x,y),y,y') = x$, and thus x has a small description. This is the core of the argument, and we will shortly repeat it formally and in detail.

Items (2) and (3) together, imply that when x is hard $M(G(x,U_{r_G}))$ outputs "hard" with non-negligible probability over the internal random coins $y \sim U_{r_G}$. Item (1) implies that when x is easy $M(G(x,U_{r_G}))$ never outputs "hard". Thus $M(G(x,U))$ is a one-sided error probabilistic algorithm solving MinKT in the *worst-case*.

Next we give a formal proof:

Proof: (Of Theorem 6.4) Denote by M a one-sided error algorithm for $\mathrm{GapMinKT}_{\alpha',\beta',t'_1,t'_2}$ with error parameter δ on the uniform distribution. We first describe the probabilistic algorithm P: On input $x \in \{0,1\}^n$, P uses the random coins $s \sim U_{r_G}$ and outputs 1 if $M(G(x,s)) = 1$ and 0 otherwise (i.e., if $M(G(x,s)) \in \{0,\perp\}$).

Claim 6.5. *If $K_{t_1}(x) \leqslant \alpha(|x|)$ then for every $s \in \{0,1\}^{r_G}$ we have that $K_{t'_1}(G(x,s)) \leqslant \alpha'(m)$ and thus $Pr_s[P(x,s)=0] = 1$.*

Proof: Suppose $K_{t_1}(x) \leqslant \alpha(|x|)$ and $s \in \{0,1\}^{r_G}$. Denote $z = G(x,s)$. We can describe z by a machine M_z that runs G on the input (x,s), and whenever G wants to read a bit from x, it calculates that bit from the description of x (which is of size $K_{t_1}(x)$) using the universal Turing machine. Thus, z can be described by a description of size

$$|d_G| + |d_U| + K_{t_1}(x) + r_G = K_{t_1}(x) + r_G + O(1) \leq \alpha(n) + r_G + O(1) \leqslant \alpha'(|z|)$$

The required running time to generate z is $Run(G) \cdot t_1 = t_1'$. Thus, we have

$$K_{t_1'}(z) \leqslant \alpha'(m)$$

Since M is zero-sided error, we must have $M(G(x,s)) \in \{1, \perp\}$ and thus $P(x,s) = 0$. Since this is true for every possible s, we have $Pr_s[P(x,s) = 0] = 1$ as required. ■

Next we prove:

Claim 6.6. $\Pr_{u\sim U_m}[K_\infty(u) \geq \beta'] > 1 - 2^{\beta'-m}$ *and* $\Pr_{u\sim U_m}[M_0(u) = 1] > 1 - 2^{\beta'-m} - \delta$.

Proof: Notice that there are at most $\sum_{i=0}^{\beta'-1} 2^i \leqslant 2^{\beta'}$ strings with description size less than β'. Thus, for every time T, the probability that a uniform string has a description size at least β' with respect to T is at least $\frac{2^m - 2^{\beta'}}{2^m} = 1 - 2^{\beta'-m}$. As M solves $\mathrm{GapMinKT}_{\alpha',\beta',t_1',t_2'}$ on the uniform distribution with at most δ error we have

$$Pr_{u\sim U_m}[M(u) = 1] \geqslant Pr_{u\sim U_m}[K_{t_2'}(u) \geqslant \beta'(m)] - \delta$$
$$\geqslant Pr_{u\sim U_m}[K_\infty(u) \geqslant \beta'(m)] - \delta \geqslant 1 - 2^{\beta'(m)-m} - \delta$$

■

We also claim the following,

Claim 6.7. *Suppose* $K_{t_2}(x) \geqslant \beta(|x|)$. *Then*

$$\left| \Pr_{s\sim U_{r_G}}[M(G(x,s)) = 1] - \Pr_{u\sim U_m}[M(u) = 1] \right| < p$$

.

Proof: Assume the first item does not hold. Then M p-distinguishes the uniform and generated distributions. By the definition of the reconstruction-PRG, we have that

$$Pr_{y\sim U_{r_A},y'\sim U_{r_R}}[R^M(A(x,y),y,y')=x]\geqslant q>0$$

In particular, there exists y, y' for which $R^M(A(x,y),y,y') = x$. Thus, one can describe x by running the oracle machine R^M on input $(A(x,y),y,y')$, and therefore x has a description of size

$$|d_R|+|d_M|+|A(x,y)|+|y|+|y'| = a+r_A+r_R+O(1)<\beta(n)$$

in time $Time(R)(a)\cdot Time(M)(m) = t_2$. Hence, $K_{t_2}(x)\leq\beta(|x|)$ - a contradiction. The claim follows. ■

From Claim (6.6) and Claim (6.7) together we see that for x with $K_{t_2}(x)\geqslant\beta(|x|)$ we have:

$$Pr_s[P(x,s)=1] = Pr_{s\sim U_{r_G}}[M(G(x,s))=1]\geqslant 1-2^{\beta'(m)-m}-\delta-p$$

as desired. ■

6.3 Understanding the Parameters

In this subsection we discuss the different parameters in the reduction. In order to have a better understanding of the reduction we first define two important properties of $\mathrm{GapMinKT}$ problems

Definition 6.8 (entropy gap)**.** *Fix* $\alpha,\beta,t_1,t_2 :\to \mathbb{N}$. *The* entropy gap *of the problem is defined by* $n-\beta(n)$.

The proof of Theorem 6.4 shows a reduction from a $\mathrm{GapMinKT}$ problem with arbitrary entropy gap to a $\mathrm{GapMinKT}$ problem with entropy gap close to 0, i.e. $m-\beta'(m)$ is small. Any such reduction immediately implies a worst-case to average-case reduction, because an instance from the uniform distribution is with probability $2^{-\Theta(m-\beta'(m))}$ a "hard" instance, and thus the heuristic algorithm often answers "hard", and on an "easy" instance it never answers "hard". In order to make the output distribution close to uniform, Hirahara uses an extractor, a function that translates entropy, or hardness in our case, to almost uniform bits. However, using an extractor incurs an "entropy" (or hardness) loss, which motivates our next definition:

Definition 6.9 (hardness loss). *Fix $\alpha, \beta, t_1, t_2, \alpha', \beta', t_1', t_2'\mathbb{N} :\to \mathbb{N}$ and let R be a reduction (possibility probabilistic) from $\mathrm{GapMinKT}_{\alpha,\beta,t_1,t_2}$ to $\mathrm{GapMinKT}_{\alpha',\beta',t_1',t_2'}$. Denote by $m = m(n)$ the output of the reduction R on input of length n.*

- *The* hardness loss *of hard instances is defined by $\beta(n) - \beta'(m)$.*
- *The* hardness loss *of easy instances is defined by $\alpha'(m) - \alpha(n)$.*
- *The* hardness loss *of the reduction is defined by the sum of the hardness gap of hard and easy instances.*

The hardness loss of the reduction is in some sense a natural measure for the quality of the reduction. If R is a worst-case to average-case reduction such as the reduction described above, the hardness loss of the reduction must be non-negative, since if on input length m we can distinguish between hardness $\beta'(m)$ and $\alpha'(m)$ on average, the best we could hope for is to distinguish between the same gap in the worst-case. When the reduction has a positive hardness loss, this loss corresponds to the "price" we pay for getting worst-case hardness.

However, when we use an extractor to extract uniform bits from an entropy source, there is always unavoidable entropy loss. From Theorem 6.4 we see that that the bounds α, α' relate to each other in a natural way as we have $\alpha'(m) = \alpha(n) + r_G + O(1)$. When r_G is small, the difference is additive and small. We remark that potentially r_G can be as small as $O(\log n)$. The situation with the bounds β and β' is different, and we have $\beta(n) > a + r_A + r_R + O(1)$. It is not so clear how $\beta(n)$ relates to $\beta'(m)$ and weather we can make the loss in parameters here small.

We remark that in a typical setting of an extractor/PRG we generally have $a \geq (1 + c)m$ for a constant $c > 0$, which represents a huge additive loss. To solve this problem, Hirahara uses Trevisan's extractor in an unusual setting of parameters. We note that in Trevisan's extractor, when using the weak-design of RRV [RRV99], we have

$$a = m + \log^2 n \cdot m/r_G + r_G \sim \beta' + \log^2 n \cdot \beta'/r_G + r_G$$

Essentially, up to logarithmic factors, the loss is $\max\{r_G, \frac{\beta'}{r_G}\}$. To minimize losses, Hirahara chooses parameters so that, roughly, $r_G \approx \log n \cdot \sqrt{m} \approx \log n \cdot \sqrt{\beta'}$. This stands in sharp contrast to typical parameters in the extractor literature where r_G is usually

logarithmic or poly-logarithmic in m. Doing this Hirahara gets overall hardness loss and hence hardness gap of $O(\log n \cdot \sqrt{\beta'(m)})$ in the reduction.

6.4 Black-box versus Non-Black-box

An important observation to make regarding Hirahara's worst-case to average-case reduction is that the proof heavily relies on the fact that the algorithm that solves MinKT on average M is indeed a *uniform* algorithm, i.e. an algorithm of fixed-sized code. If for example M was an algorithm of large description (for example if we had $M \in \mathrm{P/Poly}$) then our security proof wouldn't work, because when claim the input x to be of small description (towards contradiction), the small description we provide is made of the code of the reconstruction procedure alongside oracle calls for M, and thus the description must also contain M's description in order to implement these oracle calls. Hence, if M is of high description, x will also be of high description and the proof won't work. Since the proof of the reduction depends on the size of the description of the algorithm solving MinKT on average, we conclude that the reduction is not *fully-black-box* (while it is *soft-black-box* for the uniform class of algorithms in the sense that if we are guaranteed that M is uniform, then the reduction proof works without considering any other properties of M).

Building on Feigenbaum and Fortnow [FF93], Bogdanov and Trevisan [BT06] showed that if a language L reduces to a distributional NP problem via a black-box nonadaptive randomized polynomial-time reduction, then $L \in \mathsf{NP}/\mathrm{poly} \cap \mathit{co}\mathsf{NP}/\mathrm{poly}$ where the advice "/poly" is mainly used to encode some information about the distributional problem and can be removed in some cases. Therefore, in order to reduce any problem outside $\mathsf{NP} \cap \mathit{co}\mathsf{NP}$ to a distributional NP problem, it is likely that a non-black-box reduction technique is needed. In his work, Hirahara conjectures that $\mathrm{GapMinKT} \notin \mathit{co}\mathsf{NP}$ and describes the evidence on which that conjecture is based on. Under Hirahara's conjecture and the result of Bogdanov and Trevisan, we get that we indeed can not hope to get a fully-black-box worst-case to average-case reduction for the $\mathrm{GapMinKT}$ problem.

6.5 Almost Lossless GapMinKT Reduction With Small Entropy Gap

As discussed in the previous section, Hirahara's worst-case to average case reduction is a reduction from a $\mathrm{GapMinKT}$ problem with arbitrary entropy gap to a $\mathrm{GapMinKT}$ problem with negligible entropy gap that has a small but non-negligible hardness loss. In this section we prove a new "dual" result and show a reduction of negligible hardness loss from a $\mathrm{GapMinKT}$ problem with arbitrary entropy gap to a $\mathrm{GapMinKT}$ problem with small (but non-negligible) entropy gap.

Our reduction is based on the observation made by [TSUZ07] that the advice function of a reconstructive extractor is actually a *lossless condenser*, a function that can non-optimally compress a string sampled from a weak entropy source without losing any entropy. To prove the correctness of our reduction, we will assume the existence of a heuristic algorithm solving $\mathrm{GapMinKT}$ on average with respect to the uniform distribution. We prove the following theorem:

Theorem 6.10 (Almost Lossless GapMinKT Reduction)**.** *Suppose*

$$(\mathrm{GapMinKT}_{\alpha'',\beta'',t_1'',t_2''}, U) \in \mathrm{Heur}_{\delta},$$

solvable by an efficient, heuristic algorithm M. Let (G, A, R) be a (p, q)-reconstructive PRG with parameters $a(n), m(n), r_G(n), r_A(n), r_R(a(n))$ as in definition 3. Define

$$pr = \min\{1 - p - \delta(m(n)) - 2^{-(m(n)-\beta''(m(n)))}, q \cdot 2^{-r_R}\}$$

Denote $o(n) = a(n) + m(n)$ and set $\beta', t_2' : \mathbb{N} \to \mathbb{N}$ such that:

- $\beta'(o(n)) \leqslant \alpha''(m(n))$
- $t_2'(o(n)) \leqslant t_1''(m(n))$

Also, set $\alpha, \beta, t_1, t_2, \alpha', t_1' : \mathbb{N} \to \mathbb{N}$ such that:

- $\alpha(n) \leqslant \alpha'(o(n)) - r_A - O(1)$
- $\beta(n) > \beta'(o(n)) + r_A + r_R + O(1)$
- $t_1(n) = t_1'(o(n))/(Time(G)(m(n)) + Time(A(n)))$
- $t_2(n) = t_2'(o(n)) \cdot Time(R)(a(n)) \cdot Time(M)(m(n))$

Then, there exists a polynomial-time probabilistic reduction P such that for any input x of length n:

- *If $K_{t_1}(x) \leq \alpha(n)$ then $K_{t'_1}(P(x)) \leqslant \alpha'(|P(x)|)$ with probability 1.*
- *If $K_{t_2}(x) > \beta(n)$ then $K_{t'_2}(P(x)) \geqslant \beta'(|P(x)|)$ with probability $\geqslant pr$.*

In other words, if $(\mathrm{GapMinKT}_{\alpha'',\beta'',t''_1,t''_2}, U) \in \mathrm{Heur}_\delta$ *then*

$$\mathrm{GapMinKT}_{\alpha,\beta,t_1,t_2} \sim_{BPP} \mathrm{GapMinKT}_{\alpha',\beta',t'_1,t'_2}$$

Comparing theorem 6.10 with theorem 6.4, we see that unlike theorem 6.4, in theorem 6.10 the hardness loss of both the hard and easy instances depends only on r_A and r_G, the seed lengths of the generator and advice functions, which can be made logarithmically small in n and not depend on the hardness of the original input. In contrast to Hirahara's reduction, this results in a reduction of negligible hardness loss.

Proof: The reduction P chooses uniform $s_1 \in \{0,1\}^{r_G}, s_2 \in \{0,1\}^{r_A}$ and outputs

$$z = G(x, s_1) \circ A(x, s_2)$$

.

Claim 6.11. *For any x such that $K_{t_1}(x) \leqslant \alpha(|x|)$ and for every $s_1 \in \{0,1\}^{r_G}, s_2 \in \{0,1\}^{r_A}$ we have that $K_{t'_1}(G(x, s_1) \circ A(x, s_2)) \leqslant \alpha'(o)$*

Proof: Suppose $K_{t_1}(x) \leqslant \alpha(|x|)$ and fix $s_1 \in \{0,1\}^{r_G}, s_2 \in \{0,1\}^{r_A}$. Denote

$$z = G(x, s_1) \circ A(x, s_2)$$

We can describe z by a machine M_z that runs G on the input (x, s_1) and then runs A on the input (x, s_2), and whenever G or A want to read a bit from x, it calculates that bit from the description of x (which is of size $K_{t_1}(x)$) using the universal Turing machine. Thus, z can be described by a description of size

$$|d_G|+|d_A|+|d_U|+K_{t_1}(x)+r_G+r_A = K_{t_1}(x)+r_G+r_A+O(1) \leq \alpha(n)+r_G+r_A+O(1) \leqslant \alpha'(o)$$

The required running time to generate z is $(Time(G) + Time(A)) \cdot t_1 = t'_1$. Thus, we have

$$K_{t'_1}(z) \leqslant \alpha'(o)$$

as required. ∎

We now prove the second direction.

Claim 6.12. *For any x such that $K_{t_2}(x) \geqslant \beta(|x|)$ we have that*

$$\Pr_{s_1,s_2}[K_{t_2'}(G(x,s_1) \circ A(x,s_2)) \geqslant \beta'(o)] \geq pr.$$

Proof: Before doing the argument formally, let us first consider the case where $K(x)$ is *exactly* $\beta(|x|)$. Then, we claim that M is a distinguisher between $G(x, U_{s_1})$ and U_m. This is because a ranndom string in U_m has almost full complexity m, whereas $G(x,y)$ has complexity at most $\beta + r_G + O(1) \leq \alpha''(m(n))$. However, without the assumption that $K(x)$ is *exactly* $\beta(|x|)$ we cannot argue that M is necessarily a distinguisher. For example, if x has more than $\beta''(m(n))$ entropy, it might be the case that $G(x,y)$ has full Kolmogorov complexity. Here, we use the fact that the output of the generator is also included in the output of P. We separate the proof into two cases: either $G(x,U)$ has typically Kolmogorov complexity smaller than $\alpha''(m(n))$ and then we have a distinguisher, or typically it has Kolmogorov complexity larger than $\alpha''(m(n))$ and then $G(x,U)$ is a hard component in the output of P.

We proceed to the formal argument. We look at the difference:

$$\text{Diff} = \left| \Pr_{s \sim U_{r_G}}[M(G(x,s)) = 1] - \Pr_{u \sim U_m}[M(u) = 1] \right|$$

- First case: assume $Diff < p$. Then from the counting argument provided in claim 6.6 of the previous theorem we have that

  $$Pr_{s \sim U_{r_G}}[M(G(x,s)) = 1] \geqslant 1 - p - \delta(m(n)) - 2^{-(m(n) - \beta''(m(n)))} \geqslant pr$$

  Since M is a heuristic algorithm, whenever it outputs 1 the input string cannot have low Kolmogorov complexity with respect to time t_1''. This means that

  $$Pr_{s \sim U_{r_G}}[K_{t_1''}(G(x,s)) > \alpha''(m(n))] \geqslant pr$$

  Since we chose the parameters so that $t_2'(o(n)) \leqslant t_1''(m(n))$ and since $\beta'(o) < \beta(n) < \alpha''(m(n))$ we get that

  $$Pr_{s_1 \sim U_{r_G}, s_2 \sim U_{r_A}}[K_{t_2'}(G(x,s_1) \circ A(x,s_2)) \geqslant \beta'(o(n))] \geqslant$$

  $$Pr_{s_1 \sim U_{r_G}, s_2 \sim U_{r_A}}[K_{t_1''(m(n))}(G(x,s_1) \circ A(x,s_2)) \geqslant \alpha''(m(n))] \geqslant$$

  $$Pr_{s_1 \sim U_{r_G}}[K_{t_1''}(G(x,s_1) \geqslant \alpha''(m(n))] \geqslant pr$$

  as required.

- Now assume $Diff > p$. This means M is a distinguisher for the reconstructive extractor, and thus we have

$$Pr_{y\sim U_{r_A},y'\sim U_{r_R}}[R^M(A(x,y),y,y')=x] \geqslant q > 0$$

Denote

$$\text{Hard} = \{y \in \{0,1\}^{r_A} \mid \exists y' \in \{0,1\}^{r_R} : R^M(A(x,y),y,y') = x\}$$

By averaging we get that $Pr_{y\sim U_{r_A}}[y \in \text{Hard}] \geqslant q \cdot 2^{-r_R}$.

Claim 6.13. *For every* $y \in \text{Hard}$ *we have* $K_{t_2'(o(n))}(A(x,y)) \geqslant \beta(o(n))$

Proof: Fix $y \in \text{Hard}$ and assume towards contradiction that $K_{t_2'(o(n))}(A(x,y)) < \beta(o(n))$. Since $y \in \text{Hard}$, there exists $y' \in \{0,1\}^{r_R}$ such that $R^M(A(x,y),y,y') = x$. This means that we can describe x using a description of size $|d_R| + |d_M| + \beta'(o(n)) + r_A + r_R$ as we can describe x by a Turing machine M_y that runs R on the input $(A(x,y),y,y')$ while simulating the oracle calls for M when needed and using the Turing machine describing $A(x,y)$ to generate its bits when needed. The running time of M_y is exactly $t_2'(o(n)) \cdot Time(R)(a(n)) \cdot Time(M)(m(n)) = t_2(n)$, and by the choice of β we get that $K_{t_2}(x) \leqslant \beta'(o(n)) + r_A + r_R + O(1) < \beta(n)$ which is a contradiction. ■

By claim 6.13 we get that

$$Pr_{s_1\sim U_{r_G},s_2\sim U_{r_A}}[K_{t_2'}(G(x,s_1)\circ A(x,s_2)) \geqslant \beta'(o(n))] \geqslant$$

$$Pr_{s_2\sim U_{r_A}}[K_{t_2'}(A(x,s_2) \geqslant \beta'(o(n))] \geqslant Pr_{s_2\sim U_{r_A}}[s_2 \in \text{Hard}] \geqslant q \cdot 2^{-r_R}$$

as required.

■

■

6.6 Comparison of Main Theorems and Conclusion

We now compare Hirahar's result(theorem 6.4) and theorem 6.10. Both results start with the assumption of the existence of a an errorless heuristic algorithm M solving a $\mathrm{GapMinKT}$ problem. In his result, Hirahara uses the heuristic algorithm M to show a worst-case to average-case reduction between two $\mathrm{GapMinKT}$ problems that their parameters relate as described in theorem 6.4. As discussed in a previous section, in order for Hirahara's reduction to be a worst-case to average-case reduction, Hirahara must reduce to a gap problem of negligible entropy-gap. The use of the extractor function in the reduction results in a non-negligible entropy loss of $\sqrt{\beta'}$.

Theorem 6.10 proves a similar reduction of negligible hardness loss that can be made as small as $O(\log n)$. By taking $m(n)$ to be equal $C \cdot \beta(n)$ for some constant C and taking the functions α'' and β'' to be of the form $O(1) \cdot m$ for sufficient small and large constants respectively, we get a reduction from a $\mathrm{GapMinKT}$ problem with arbitrary entropy $n - \beta(n)$ gap to a $\mathrm{GapMinKT}$ problem of entropy gap $O1() \cdot \beta(n)$, which is a much more condensed gap problem in the case of $\beta(n) << n$.

As the reduction of theorem 6.10 has a non-negligible entropy gap it does not show a worst-case to average-case reduction such as in Hirahara's result. However, theorem 6.10 might serve as a component in a worst-case to average-case reduction. For example, it shows that for a worst-case to average-case reduction, one can assume w.l.o.g. that the gap problem we start with is condensed, as given by the parameters of theorem 6.10. This is because, assuming a heuristics algorithm for the corresponding $\mathrm{GapMinKT}$ problems, one can first use theorem 6.10 to condense the distribution (with a negligible hardness loss) and then do on the output distribution the reduction for the condensed gap problem. This is similar to the way essentially all current small-loss extractors operate today: they first losslessly condense the input to a condensed source, and then they apply on top of it an extractor that works well on condensed sources.

7 Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Amnon Ta-Shma, whom with his patience, enthusiasm and immense insight made this work possible.

I attribute my emerging ability to think clearly and search for underlying ideas to his encouragement and effort.

References

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108. ACM, 1996.

[BT06] Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for np problems. *SIAM Journal on Computing*, 36(4):1119–1159, 2006.

[FF93] Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22(5):994–1005, 1993.

[Hir18] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within np. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 247–258. IEEE, 2018.

[Lev86] Leonid A Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

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

[RRV99] Ran Raz, Omer Reingold, and Salil Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 149–158. ACM, 1999.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM (JACM)*, 52(2):172–216, 2005.

[Tre01] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[TSUZ07] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Lossless condensers, unbalanced expanders, and extractors. *Combinatorica*, 27(2):213–240, 2007.

[TSZS06] Amnon Ta-Shmaa, David Zuckermanb, and Shmuel Safraa. Extractors from reed–muller codes. *Journal of Computer and System Sciences*, 72:786–812, 2006.

[Uma02] Christopher Umans. Pseudo-random generators for all hardnesses. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 627–634, New York, NY, USA, 2002. ACM.