Electronic Colloquium on Computational Complexity, Report No. 166 (2021)

Average-case Hardness of NP and PH from Worst-case Fine-grained Assumptions*

Lijie Chen
MIT
lijieche@mit.edu

Shuichi Hirahara
National Institute of Informatics
s_hirahara@nii.ac.jp

Neekon Vafa
MIT
nvafa@mit.edu

November 21, 2021

Abstract

What is a minimal worst-case complexity assumption that implies non-trivial average-case hardness of NP or PH? This question is well motivated by the theory of fine-grained average-case complexity and fine-grained cryptography. In this paper, we show that several standard worst-case complexity assumptions are sufficient to imply non-trivial average-case hardness of NP or PH:

- $\mathsf{NTIME}[n]$ cannot be solved in quasi-linear time on average if $\mathsf{UP} \not\subseteq \mathsf{DTIME}\left[2^{\widetilde{O}(\sqrt{n})}\right]$.
- $\Sigma_2\mathsf{TIME}[n]$ cannot be solved in quasi-linear time on average if $\Sigma_k\mathsf{SAT}$ cannot be solved in time $2^{\widetilde{O}(\sqrt{n})}$ for some constant k. Previously, it was not known if even *average-case hardness* of $\Sigma_3\mathsf{SAT}$ implies the average-case hardness of $\Sigma_2\mathsf{TIME}[n]$.
- Under the Exponential-Time Hypothesis (ETH), there is no average-case $n^{1+\varepsilon}$-time algorithm for $\mathsf{NTIME}[n]$ whose running time can be estimated in time $n^{1+\varepsilon}$ for some constant $\varepsilon > 0$.

Our results are given by generalizing the non-black-box worst-case-to-average-case connections presented by Hirahara (STOC 2021) to the settings of fine-grained complexity. To do so, we construct quite efficient complexity-theoretic pseudorandom generators under the assumption that the nondeterministic linear time is easy on average, which may be of independent interest.

*Lijie Chen is supported by NSF CCF-2127597 and an IBM Fellowship. Shuichi Hirahara is supported by JST, PRESTO Grant Number JPMJPR2024, Japan. Neekon Vafa is supported by NSF fellowship DGE-1745302.

ISSN 1433-8092

Contents

1 Introduction

One of the central questions in theoretical computer science is to base the existence of one-way functions on the worst-case hardness of NP. Equivalently, this question is well known as the question of whether Heuristica and Pessiland can be excluded from Impagliazzo's five possible worlds [Imp95]. Heuristica is a hypothetical world in which NP is hard in the worst case but NP is easy on average; Pessiland is a hypothetical world in which NP is hard on average but one-way functions do not exist. The existence of a one-way function is indispensable for complexity-theory-based cryptography [IL89]; thus, excluding these hypothetical worlds (where one-way functions do not exist) from Impagliazzo's five possible worlds would make the security of cryptographic primitives more reliable.

There are a number of reasons why excluding Heuristica and Pessiland is difficult: Standard proof techniques, such as black-box reductions, hardness amplification procedures, and relativizing proof techniques, are known to be incapable of excluding Heuristica [FF93, BT06b, Vio05b, Vio05a, Imp11]. Similar impossibility results are known for Pessiland [AGGM06, BB15, Liv10, Wee06].

1.1 Fine-Grained Average-Case Complexity

To make progress on this challenging question, Ball, Rosen, Sabin, and Vasudevan [BRSV17] proposed to study a weak variant of the question: Can we construct a fine-grained one-way function under standard worst-case complexity assumptions? Informally, a *fine-grained one-way function* is a function $f\colon \{0,1\}^* \to \{0,1\}^*$ such that f can be computed in time $t(n)$ on inputs of length n but cannot be inverted in time $t(n)^{1+\varepsilon}$ on average for some time bound $t\colon \mathbb{N} \to \mathbb{N}$ and for some constant $\varepsilon > 0$. In contrast to the standard definition of a one-way function in which we require ε to be a super constant, a fine-grained one-way function is *slightly* hard to invert; thus, we expect that it is much easier to construct a fine-grained one-way function than a standard one-way function. LaVigne, Lincoln, and Vassilevska Williams [LLW19] showed that some fine-grained *average-case hardness* of Zero-k-Clique and k-SUM implies the existence of a fine-grained public-key cryptosystem and, in particular, a fine-grained one-way function. This result is relevant to the question of whether one can exclude a fine-grained version of Pessiland in which there is no fine-grained one-way function and the class $\mathsf{NTIME}[n]$ of problems solvable by nondeterministic linear-time algorithms (which contains Zero-k-Clique and k-SUM) is hard on average.[1]

The open question posed in [BRSV17] can be naturally decomposed into the following two open questions: (1) Can we exclude the fine-grained version of Pessiland? In other words, can we construct a fine-grained one-way function from super-linear-time average-case hardness of $\mathsf{NTIME}[n]$? (2) Can we prove super-linear-time average-case hardness of $\mathsf{NTIME}[n]$ under standard worst-case complexity assumptions? Resolving this second question is (almost) necessary to resolve the open question of [BRSV17] because the existence of a fine-grained one-way function implies that (the search version of[2]) $\mathsf{NTIME}[n]$ cannot be solved in time $n^{1+\varepsilon}$ on average with respect to some $O(n)$-time samplable distribution for some constant $\varepsilon > 0$. In this paper, we focus on the second question:

[1] Note that [LLW19] does not exclude the fine-grained version of Pessiland because the average-case hardness of Zero-k-Clique and k-SUM is not implied by the average-case hardness of $\mathsf{NTIME}[n]$, which only means *some* problem in $\mathsf{NTIME}[n]$ is average-case hard.

[2] A standard search-to-decision reduction [BCGL92] incurs a multiplicative overhead of $O(n)$, which is prohibitively large in the fine-grained setting; thus, it is unclear to us whether the existence of a fine-grained one-way function implies an average-case hard *decision* problem in $\mathsf{NTIME}[n]$.

Question 1.1. *What is a minimal worst-case complexity assumption that implies "non-trivial"[3] average-case hardness of linear-time versions of* NP *or* PH*? Can we exclude a fine-grained version of Heuristica?*[4]

Regarding this question, the original work of [BRSV17] implicitly showed that $\mathsf{MATIME}[n]$ (which is a class sandwiched between $\mathsf{NTIME}[n]$ and $\Sigma_2\mathsf{TIME}[n]$) cannot be solved in time $n^{2-o(1)}$ on average under the Strong Exponential-Time Hypothesis (SETH [IP01]). Specifically, Ball et al. [BRSV17] showed that worst-case hardness assumptions of popular fine-grained complexity problems, such as OV, 3SUM, and Zero-Weight-Triangle, imply the existence of average-case hard problems. For example, they introduced a problem $\mathcal{F}\mathsf{OV}$, which "encodes" OV as a low degree polynomial over a finite field $\mathcal{F}$, and showed that $\mathcal{F}\mathsf{OV}$ cannot be solved in sub-quadratic time on average unless OV can be solved in sub-quadratic time. They also observed that $\mathcal{F}\mathsf{OV}$ is a problem in $\mathsf{MATIME}[\widetilde{O}(n)]$[5] using Williams' MA protocol that refuted an MA variant of SETH [Wil16]. As a consequence, the average-case hardness of (a padded version of) $\mathcal{F}\mathsf{OV} \in \mathsf{MATIME}[\widetilde{O}(n)]$ follows from the worst-case hardness of OV, which in particular follows from SETH [Wil05]. Their subsequent work [BRSV18] demonstrated the usefulness of average-case hard problems by constructing a cryptographic system called Proofs of Work. A subsequent line of research [GR18, BBB19, DLW20, HS21] showed that worst-case hardness assumptions imply average-case hardness of natural problems, such as counting the number of k-Cliques in a random graph, which is in the linear-time variant of #P but is unlikely to be in $\mathsf{NTIME}[n]$. Brakerski, Stephens-Davidowitz, and Vaikuntanathan [BSV21] showed that a nearly optimal average-case lower bound for the k-SUM problem follows from the worst-case assumption that the Short Independent Vector problem (SIVP) over an n-dimensional lattice cannot be approximated to within an $n^{1+\varepsilon}$ factor in time $2^{o(n)}$. We mention that the approximation of SIVP is unlikely to be NP-complete because the problem is known to be in $\mathsf{NP} \cap \mathsf{coNP}$ [GMR05].

Currently, it is an open question to prove that $\mathsf{NTIME}[n]$ is super-linearly average-case hard under SETH. In fact, the proof techniques developed in the above-mentioned literature on fine-grained average-case complexity are unlikely to resolve this question without refuting a slightly nonuniform AM variant of SETH. The fundamental impossibility result of Feigenbaum and Fortnow [FF93] shows that if there exists a randomized k-time k-query nonadaptive "locally random"[6] reduction from a problem L to some average-case problem in $\mathsf{NTIME}[n]$, then L can be solved by an $\mathsf{AM} \cap \mathsf{coAM}$ protocol in time $n \cdot k^{O(1)}$ with $O(\log n)$ bits of advice on inputs of length n. For example, the reduction of [BRSV17] is a k-query nonadaptive reduction from OV to an average-case version of $\mathcal{F}\mathsf{OV}$ for $k = \log^{O(1)} n$. If $\mathcal{F}\mathsf{OV}$ were in $\mathsf{NTIME}[n]$, then by combining [FF93, BRSV17] we would obtain that OV is in $\mathsf{coAMTIME}[\widetilde{O}(n)]/\log n$, which would refute an $\mathsf{AM}/\log n$ variant of SETH. Since the $\mathsf{AM}/\log n$ variant of SETH is not yet refuted, this connection explains the difficulty of resolving the open question by using the current proof techniques.

[3] It is evident that $\mathsf{NTIME}[n]$ cannot be computed in sub-linear time. Here, we aim at proving average-case hardness of NP or PH that does not follow from such an unconditional result.

[4] One possible definition of a fine-grained version of Heuristica is a world in which $\mathsf{NTIME}[n] \not\subseteq \mathsf{DTIME}[n^{1+\varepsilon}]$ for some constant $\varepsilon > 0$ but (the search version of) $\mathsf{NTIME}[n]$ can be solved in time $n^{1+\varepsilon}$ on average with respect to every linear-time samplable distribution for every constant $\varepsilon > 0$.

[5] Throughout this paper, we always use $\widetilde{O}(f(n))$ to denote $f(n) \cdot \mathrm{polylog}(f(n))$.

[6] A worst-case-to-average-case nonadaptive reduction R is said to be *locally random* [BFKR97] if the query distribution of R on input x depends only on the length $|x|$ of x. The reductions presented in [BRSV17] satisfy this property. Bogdanov and Trevisan [BT06b] improved [FF93] and presented a similar impossibility result for reductions that are not locally random.

1.2 Non-Black-Box Worst-Case-to-Average-Case Connections

Recently, Hirahara [Hir18, Hir21] developed a new type of proof technique that uses non-black-box reductions and is not subject to the impossibility results of [FF93, BT06b]. We say that a worst-case-to-average-case reduction is *non-black-box* if the reduction exploits the efficiency of a hypothetical average-case solver. Using non-black-box reductions, the following connections from worst-case hardness to average-case hardness were established.

Theorem 1.2 ([Hir21])**.** *The following hold:*

1. $\mathsf{UP} \not\subseteq \mathsf{DTIME}(2^{O(n/\log n)})$ *implies* $\mathsf{NP} \times \{\mathcal{U}, \mathcal{T}\} \not\subseteq \mathsf{AvgP}$.
2. $\mathsf{PH} \not\subseteq \mathsf{DTIME}(2^{O(n/\log n)})$ *implies* $\mathsf{PH} \times \{\mathcal{U}, \mathcal{T}\} \not\subseteq \mathsf{AvgP}$.
3. $\mathsf{NP} \not\subseteq \mathsf{DTIME}(2^{O(n/\log n)})$ *implies* $\mathsf{NP} \times \{\mathcal{U}, \mathcal{T}\} \not\subseteq \mathsf{Avg_PP}$.

Here, $\mathcal{U}$ denotes the uniform distribution and $\mathcal{T}$ denotes the tally distribution, i.e., the family $\{\mathcal{T}_n\}_{n\in\mathbb{N}}$ of distributions such that $\mathcal{T}_n$ is the singleton distribution on $\{1^n\}$. AvgP denotes the class of distributional problems solvable by average-case polynomial-time algorithms or, equivalently, errorless heuristic schemes. $\mathsf{Avg_PP}$ denotes the class of distributional problems solvable by average-case polynomial-time algorithms whose running time can be estimated in polynomial time. We refer the reader to the survey of Bogdanov and Trevisan [BT06a] for more background on average-case complexity.

It would be very interesting to establish average-case lower bounds from weaker worst-case lower bounds, especially due to the fact that the Exponential-Time hypothesis (ETH; [IPZ01]) only implies that $\mathsf{NP} \not\subseteq \mathsf{DTIME}(2^{o(n/\log n)})$, which just falls short of satisfying the hypothesis of Item (3) of Theorem 1.2.[7] Moreover, building on the work of Impagliazzo [Imp11], Hirahara and Nanashima [HN21] constructed an oracle $\mathcal{O}$ such that $\mathsf{PH}^{\mathcal{O}} \not\subseteq \mathsf{DTIME}(2^{o(n/\log n)})^{\mathcal{O}}$ and yet $\mathsf{DistPH}^{\mathcal{O}} \subseteq \mathsf{AvgP}^{\mathcal{O}}$, meaning that no relativizing proof techniques can show strong average-case lower bounds such as $\mathsf{DistPH} \not\subseteq \mathsf{AvgP}$ from worst-case hardness assumptions such as $\mathsf{PH} \not\subseteq \mathsf{DTIME}(2^{o(n/\log n)})$. We remark that the proof for Item (2) in Theorem 1.2 in [Hir21] does relativize, while the proofs for Item (1) and (3) "almost relativize" in the sense that the only non-relativizing part of the proofs is the theorem of Buhrman, Fortnow and Pavan [BFP05] showing the existence of a complexity-theoretic pseudorandom generator in Heuristica.

1.3 Our Results

In this paper, we generalize the proof techniques from [Hir21] to show that worst-case lower bounds much weaker than $2^{O(n/\log n)}$ already imply super-linear-time average-case lower bounds. Formally, we have the following theorem.

Theorem 1.3. *The following hold:*

1. $\mathsf{UP} \not\subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n\log n}\right)}\right]$ *implies* $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$.
2. $\Sigma_k\mathsf{TIME}[n] \not\subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n\log n}\right)}\right]$ *implies* $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$ *for every constant* k.

[7] ETH implies that 3SAT cannot be solved in time $2^{o(m)}$ on 3CNF formulas with m variables and $O(m)$ clauses, which implies that $\mathsf{3SAT} \notin \mathsf{DTIME}(2^{o(n/\log n)})$ because 3CNF formulas can be encoded in $n := O(m \log m)$ bits as a binary string.

3. ETH *implies* $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}_{\mathsf{DTIME}[n^{1+\varepsilon}]}\mathsf{TIME}[n^{1+\varepsilon}]$ *for some constant* ε.

Here, $\mathcal{U}^{\mathsf{para}}$ denotes a "parameterized uniform distribution", which is a slight generalization of the uniform distribution (see Definition 3.7). $\mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$ denotes the class of distributional problems that can be solved by quasi-linear-time errorless heuristics with failure probability at most $1/2$.[8]

We present the significance of the three results of Theorem 1.3 below. The second item of Theorem 1.3 shows that the second level $\Sigma_2\mathsf{TIME}[n]$ of the linear-time version of PH is super-linearly average-case hard under the worst-case assumption that $\Sigma_k\mathsf{SAT}$ cannot be solved in time $2^{\widetilde{O}(\sqrt{n})}$ on inputs of length n. Here, $\Sigma_k\mathsf{SAT}$ is the problem of deciding, given a Boolean circuit C on mk inputs, whether

$$\exists y_1 \in \{0,1\}^m, \forall y_2 \in \{0,1\}^m, \ldots, \mathcal{Q}_k y_k \in \{0,1\}^m,\ C(y_1, \ldots, y_k) = 1$$

is true or not, where $\mathcal{Q}_k := \exists$ if k is odd and $\mathcal{Q}_k := \forall$ if k is even. This problem is a canonical complete problem for the k-th level $\Sigma_k\mathsf{P}$ of PH under quasi-linear time reductions (see, e.g., [JMV15]). In particular, $\Sigma_1\mathsf{SAT}$ is equivalent to the Circuit Satisfiability problem and ETH implies that $\Sigma_1\mathsf{SAT} \notin \mathsf{DTIME}(2^{o(n/\log n)})$. Solving $\Sigma_k\mathsf{SAT}$ is known to be notoriously hard: The first algorithm faster than the trivial brute-force algorithm for $k \geq 2$ was given in [SW15] and runs in time $2^{n-n^{1/(k+1)}}$ on *CNF formulas* with n variables. No non-trivial algorithm for general Boolean circuits is known.

The second item of Theorem 1.3 makes an important progress on a central and long-standing open question in the theory of structural average-case complexity. The influential paper of Impagliazzo [Imp95] mentioned

> "a central problem in the structure of average-case complexity is: if all problems in NP are easy on average, can the same be said of all problems in the polynomial hierarchy?"

Impagliazzo [Imp11] constructed an oracle under which $\mathsf{DistNP} \subseteq \mathsf{AvgP}$ and $\mathsf{Dist}\Sigma_2\mathsf{P} \not\subseteq \mathsf{HeurSIZE}(2^{n^\alpha})$ for some constant $\alpha > 0$, thereby explaining the difficulty of resolving this open problem. Previously, it was unknown whether *average-case easiness* of $\Sigma_{k+1}\mathsf{TIME}[n]$ follows from average-case easiness of $\Sigma_k\mathsf{TIME}[n]$ for any constant k. The contrapositive of our second result shows that even *worst-case easiness* of $\Sigma_k\mathsf{TIME}[n]$ follows from average-case easiness of $\Sigma_2\mathsf{TIME}[n]$ for all constants k.

The third item of Theorem 1.3 shows that some super-linear-time average-case hardness of $\mathsf{NTIME}[n]$ follows from ETH, which is one of the popular worst-case assumptions. $\mathsf{Avg}_{\mathsf{DTIME}[n^{1+\varepsilon}]}\mathsf{TIME}[n^{1+\varepsilon}]$ is the class of distributional problems $(L, \mathcal{D})$ such that there exists an errorless heuristic scheme running in time $n^{1+\varepsilon}/\delta$ that solves L with failure probability δ for any given parameter δ, and moreover whether the heuristic algorithm fails or not can be computed in time $n^{1+\varepsilon}$; see Definition 9.1 for a precise definition. We mention that the Hamiltonian path problem can be solved on average with respect to the Erdős–Rényi random graph in this average-case sense [GS87]. We note that, as in our second result, $\Sigma_k\mathsf{TIME}[n] \not\subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$ also implies $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}_{\mathsf{DTIME}[\widetilde{O}(n)]}\mathsf{TIME}[\widetilde{O}(n)]$ for every constant k; see Corollary 9.6.

The first item of Theorem 1.3 shows that $\mathsf{NTIME}[n]$ is super-linearly hard on average under the worst-case assumption that UP cannot be solved in time $2^{\widetilde{O}(\sqrt{n})}$ on inputs of length n. UP is the complexity class of problems that can be solved by nondeterministic polynomial-time algorithms

[8] We mention that the constant $1/2$ can be actually improved to any constant smaller than 1.

whose accepting path is always at most 1 and is known to characterize the complexity of worst-case injective one-way functions [Ko85, GS88]. The trivial deterministic upper bound on UP is $\mathsf{DTIME}(2^{n^{O(1)}})$ and thus the hypothesis that $\mathsf{UP} \not\subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$ is quite plausible.

In addition to Theorem 1.3, we suggest a new approach to bypass the impossibility results of [FF93, BT06b]. We observe that a reduction that uses a limited amount of nondeterministic bits is not subject to these impossibility results. Let $\mathsf{NTIMEGUESS}[t(n), g(n)]$ denote the complexity class of problems solvable by $t(n)$-time nondeterministic algorithms that use at most $g(n)$ nondeterministic bits on inputs of length n. We show that the average-case hardness of NP follows from the worst-case assumption that certificates for UP cannot be "compressed" into $o(n)$ bits:

Theorem 1.4. *For every constant $\varepsilon > 0$, if $\mathsf{UP} \not\subseteq \mathsf{NTIMEGUESS}[\mathrm{poly}(n), \varepsilon n]$, then $\mathsf{NP} \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}_{1/2}\mathsf{P}$.*

Interestingly, for $\varepsilon = 1$, Theorem 1.4 can be proved by a black-box reduction that uses n nondeterministic bits. Similarly, we also prove the following theorem.

Theorem 1.5. *For every constant $\varepsilon > 0$, if $\mathsf{NP} \not\subseteq \mathsf{NTIMEGUESS}[\mathrm{poly}(n), \varepsilon n]$, then $\Sigma_2\mathsf{P} \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}_{1/2}\mathsf{P}$.*

We remark that for Theorem 1.4 and Theorem 1.5, we indeed show a certain "easy-witness lemma" [IKW02, MW20]. Take Theorem 1.5 for example, we indeed prove that assuming $\Sigma_2\mathsf{P} \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{P}$, for every $L \in \mathsf{NP}$ and every verifier V for L[9], $x \in L$ implies that there exists a y such that $\mathsf{K}^{\mathrm{poly}(n)}(y) \leq \varepsilon n$ and $V(x, y) = 1$. In other words, every yes instance x of L admits an "easy witness" y that can be compressed into εn bits. This is similar to the easy-witness lemmas proved in [IKW02, MW20]. In particular, [MW20] proved that if NP admits fixed-polynomial size circuits, then for every verifier V for some $L \in \mathsf{NP}$, $x \in L$ implies that there exists a y such that y is the truth-table of a small circuit and $V(x, y) = 1$.

Finally, we mention that the most technical ingredient of our result is a construction of extremely efficient hitting set generators (HSGs) and pseudorandom generators (PRGs)[10] under the assumption that $\mathsf{NTIME}[n]$ is easy on average. More specifically, we prove the following.

Lemma 1.6. *Assuming that $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, for every large enough t and m such that $\log t \leq m \leq t$, there exist an HSG $H_{t,m}$ and a PRG $G_{t,m}$ satisfying the following:*

1. *$H_{t,m}$ 0.1-hits t-time deterministic algorithms with m bits of advice on m-bit inputs.*
2. *$H_{t,m}$ has $O(\log(t))$ seed length and is computable in $\widetilde{O}(t) \cdot \mathrm{poly}(m)$ time.*
3. *$G_{t,m}$ 0.1-fools t-time deterministic algorithms with m bits of advice on m-bit inputs.*
4. *$G_{t,m}$ has $O(\log(m))$ seed length and is computable in $\widetilde{O}(t) \cdot \mathrm{poly}(m)$ time with $O(\log t)$ bits of advice.*

We note that our construction of the PRG $G_{t,m}$ has a shorter seed length comparing to that of the HSG $H_{t,m}$, at the expense of requiring $O(\log t)$ bits of advice to compute. In [BFP05], $O(\log t)$-seed length PRG fooling t-time computation that is computable in $\mathrm{poly}(t)$ time is constructed, under the assumption that $\mathsf{DistNP} \subseteq \mathsf{AvgP}$. Lemma 1.6 is a fine-grained version of the construction in [BFP05]: we start from a much stronger assumption to get a much more efficient PRG/HSG

[9] V is a verifier for L if (1) $V(x, y)$ runs in $\mathrm{poly}(x)$ time and (2) $x \in L$ if and only if there exists y such that $V(x, y) = 1$.
[10] See Section 3.5 for formal definitions of HSGs and PRGs.

construction. See Section 2 for an overview of how Lemma 1.6 is proved and why it is needed, and Section 4 for formal proofs.[11]

2 Techniques Overview

Now we discuss the intuitions behind our results. For concreteness we will first focus on proving the following theorem, and then discuss how to adapt the techniques to prove our other results.

Theorem 2.1. $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$ *implies that* $\mathsf{NP} \subseteq \mathsf{TIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$.

Note that the contrapositive of Theorem 2.1 says that $\mathsf{NP} \not\subseteq \mathsf{TIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$ implies $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$.

2.1 Review of the Framework in [Hir21]

Since our results crucially build on the framework introduced by Hirahara [Hir21], it would be instructive to first review his approach for worst-case to average-case reduction based on *meta-complexity*, and examine why it requires a $2^{O(n/\log n)}$ worst-case lower bound.

2.1.1 Key insight: Computational shallowness implies efficient algorithms

Computational depth and worst-case to average-case reduction. One crucial concept introduced in [Hir21] is the (s,t)*-time-bounded computational depth*, defined as $\mathsf{cd}^{s,t}(x) := \mathsf{K}^s(x) - \mathsf{K}^t(x)$, where $\mathsf{K}^t(x)$ is the time-bounded Kolmogorov complexity (see Section 3.4 for the formal definition). This is a generalization of the computation depth, $\mathsf{cd}^t(x) := \mathsf{K}^t(x) - \mathsf{K}(x)$, defined by [AFvMV06].

Computational depth provides a fundamental link between worst-case complexity and average-case complexity of NP. In particular, [AF09] showed that, if NP is easy on average ($\mathsf{DistNP} \subseteq \mathsf{AvgP}$), then an input x to a language $L \in \mathsf{NP}$ can be solved in $2^{O(\mathsf{cd}^{\mathrm{poly}(n)}(x)+\log|x|)}$ time; that is, one can solve all *shallow* inputs (inputs with small $\mathsf{cd}^{\mathrm{poly}(n)}$) very efficiently.

Time-bounded computational depth suffices. The key insight of [Hir21] is that under certain assumptions, one can generalize the above results to hold for time-bounded computational depth as well. For instance, Hirahara [Hir21] proved:

Lemma 2.2 (Informal). *If* $\Sigma_2\mathsf{P} \times \{\mathcal{U}, \mathcal{T}\} \subseteq \mathsf{AvgP}$, *then for every* $L \in \mathsf{NP}$ *there is a polynomial* p *such that for every input* x *and* $t \geq \mathrm{poly}(n)$, *one can solve* L *on input* x *in* $2^{\mathsf{cd}^{t,p(t)}(x)} \cdot \mathrm{poly}(t)$ *time.*

Lemma 2.2 is a significant conceptual improvement: now we can consider multiple values of t, and L is easy to solve on input x if $\mathsf{cd}^{t,p(t)}(x)$ is small for *any* of the considered t. In more details, let $\tau \in \mathbb{N}$ be a parameter, $t_0 = \mathrm{poly}(n)$, and $t_i = p(t_{i-1})$ for $i \in [\tau]$. We have the following telescoping sum

$$\sum_{i\in[\tau]} \mathsf{cd}^{t_{i-1},t_i}(x) = \sum_{i\in[\tau]} \left[\mathsf{K}^{t_{i-1}}(x) - \mathsf{K}^{t_i}(x)\right] = \mathsf{K}^{t_0}(x) - \mathsf{K}^{t_\tau}(x) \leq \mathsf{K}^{t_0}(x) \leq n + O(1).$$

[11] Indeed, in Section 4 we obtain a general trade-off between the average-case easiness of $\mathsf{NTIME}[n]$ and the efficiency of the constructed PRGs/HSGs, see Theorem 4.1 and Lemma 4.6 for details.

It then follows that there exists an $i \in [\tau]$ such that $\mathsf{cd}^{t_{i-1},t_i}(x) \leq n/\tau + O(1)$, meaning that we can solve $L(x)$ in $2^{n/\tau} \cdot \mathrm{poly}(t_i) \leq 2^{n/\tau} \cdot \mathrm{poly}(t_\tau)$ time. So our goal now is to carefully choose τ to minimize the running time. Since $t_\tau = 2^{\log n \cdot O(1)^\tau}$, setting $\tau = \varepsilon \log n$ for a small enough constant $\varepsilon > 0$ leads to $t_\tau \leq 2^{n^{0.99}}$ and the final running time $2^{O(n/\tau)} = 2^{O(n/\log n)}$ for $L \in \mathsf{NP}$. To summarize, from Lemma 2.2 we can show that $\Sigma_2\mathsf{P} \times \{\mathcal{U}, \mathcal{T}\} \subseteq \mathsf{AvgP}$ implies $\mathsf{NP} \subseteq \mathsf{DTIME}[2^{O(n/\log n)}]$.

Bottleneck for improvement: the blow-up function $p(t)$. The argument above can only achieve running time $2^{O(n/\log n)}$ since $t_\tau = 2^{\log n \cdot O(1)^\tau}$ grows *too fast*: to make t_τ smaller than 2^n, we have to take $\tau \leq O(\log n)$, meaning that the running time is at least $2^{\Omega(n/\log n)}$.

Assume for now that $p(t)$ is *linear* in t. It follows that $t_\tau = 2^{\log n + O(\tau)}$, and then setting $\tau = \sqrt{n}$ leads to a much faster running time of $2^{O(\sqrt{n})}$. Similarly, by a slightly more involved calculation, if we have $p(t) = \widetilde{O}(t) \cdot \mathrm{poly}(n)$, we can also obtain a running time of $2^{O(\sqrt{n \log n})}$. In this case, since $p(t)$ also depends on n, we will write it as $p_n(t)$ to avoid confusion.

2.1.2 Why $p(t)$ has to be a large polynomial in [Hir21]

We now know that the key to improving the result in [Hir21] is the blow-up function $p(t)$. Therefore, it is crucial to understand why the blow-up function $p(t)$ has to be a polynomial in Lemma 2.2.

The proof of Lemma 2.2 in [Hir21] consists of many components, and $p(t)$ is indeed a composition of several polynomial blow-up functions, each for one component.[12]

We will focus on one important component in the proof of Lemma 2.2 and understand why the blow-up function of the component, which we denote by $\tau(t)$, has to be a big polynomial. Since the overall blow-up $p(t)$ has to be at least $\tau(t)$, improving this $\tau(t)$ to be quasi-linear in t is *necessary* for improving $p(t)$. Furthermore, it turns out that the ideas behind improving this $\tau(t)$ are already enough to *simultaneously* improve the blow-up functions for all other components.

Fast algorithm for $\mathsf{Gap}(\mathsf{K}^A$ vs $\mathsf{K})$. One important component used by [Hir21] is the following algorithm for the $\mathsf{Gap}(\mathsf{K}^A$ vs $\mathsf{K})$ problem. Roughly speaking, if $\Sigma_2\mathsf{P}$ is easy on average, then one can distinguish between strings with small $\mathsf{K}^{t,\mathsf{SAT}}$ complexity and large $\mathsf{K}^{\tau(t)}$ complexity. (See Section 3.4 for a formal definition of $\mathsf{K}^{t,A}(x)$.)

Lemma 2.3 ([Hir18, Hir20b, Hir20a]). *If $\Sigma_2\mathsf{P} \times \{\mathcal{U}, \mathcal{T}\} \subseteq \mathsf{AvgP}$, then there is an algorithm $A^{\mathsf{Gap\text{-}K^t}}$ and a polynomial τ such that*

1. *$A^{\mathsf{Gap\text{-}K^t}}$ takes $x \in \{0,1\}^n$ and $s, t \in \mathbb{N}$ with $t \geq \max(n, s)$ as inputs, and runs in $\mathrm{poly}(t)$ time.*
2. *If $\mathsf{K}^{t,\mathsf{SAT}}(x) \leq s$, then $A^{\mathsf{Gap\text{-}K^t}}(x, s, t) = 1$.*
3. *If $\mathsf{K}^{\tau(t)}(x) \geq s + c \log t$, then $A^{\mathsf{Gap\text{-}K^t}}(x, s, t) = 0$, where $c \geq 1$ is a constant.*

Most importantly, the blow-up function τ is one crucial component in the overall blow-up function p, meaning that $p(t)$ must be at least $\tau(t)$.

2.1.3 Analyzing the blow-up function τ in $A^{\mathsf{Gap\text{-}K^t}}$: Derandomization is the bottleneck

Now we wish to analyze why the blow-up function $\tau(t)$ in $A^{\mathsf{Gap\text{-}K^t}}$ has to be a polynomial and see if we can improve it to a quasi-linear function in t. We need the following two crucial technical components to discuss the proof of Lemma 2.3.

[12] We will not review all the components in this technical overview; the interested reader can refer to [Hir21, Section 2] for an exposition of the ideas behind Lemma 2.2.

Direct product generators and derandomization from $\mathsf{DistNP} \subseteq \mathsf{AvgP}$**.** The first ingredient is the direct product generator $\mathsf{DP}_{n,k}\colon \{0,1\}^n \times \{0,1\}^{nk} \to \{0,1\}^{nk+k}$, defined as

$$\mathsf{DP}_{n,k}(y; z_1, z_2, \ldots, z_k) = (z_1, z_2, \ldots, z_k, \langle y, z_1\rangle, \langle y, z_2\rangle, \ldots, \langle y, z_k\rangle),$$

where $\langle y, z_i\rangle$ denotes the inner product between the two vectors y and z_i over $\mathbb{F}_2$. We use $d = nk$ to denote the seed length of the $\mathsf{DP}_{n,k}$ and write $\mathsf{DP}_{n,k}$ as DP_k when n is clear from the context.

Theorem 2.4. *(Reconstruction property of* DP_k*; Informal) There exists a randomized polynomial-time oracle algorithm* $R^{(\cdot)}$ *satisfying the following. Let* $D\colon \{0,1\}^{d+k} \to \{0,1\}$ *be an oracle such that* D *distinguishes the output distribution* $\mathsf{DP}_{n,k}(y; U_d)$ *from* U_{d+k}*; that is*

$$\left| \Pr_{z\sim U_d}[D(\mathsf{DP}_k(y;z)) = 1] - \Pr_{w\sim U_{d+k}}[D(w) = 1] \right| \geq 1/4.$$

Then with high probability over an internal coin flip of $R^{(\cdot)}$*, there exists an advice string* $\alpha \in \{0,1\}^{k+O(\log n)}$ *such that* $R^D(\alpha)$ *outputs* y*.*

Note that the reconstruction algorithm $R^{(\cdot)}$ of Theorem 2.4 is randomized. One can derandomize that reconstruction algorithm using a PRG (pseudorandom generators), whose existence is implied by $\mathsf{DistNP} \subseteq \mathsf{AvgP}$.

Theorem 2.5 ([BFP05])**.** $\mathsf{DistNP} \subseteq \mathsf{AvgP}$ *implies that there is an* $O(\log n)$*-seed* $\mathrm{poly}(n)$*-time computable PRG fooling circuits of size* n*.*

Proof sketch of Lemma 2.3. We will solve the following task instead:

- Given $x \in \{0,1\}^n$ and $s, t \in \mathbb{N}$ as input with the promise that $\mathsf{K}^{t,\mathsf{SAT}}(x) \leq s$, find a witness to $\mathsf{K}^{\tau(t)}(x) \leq s + O(\log t)$.[13]

Note that an algorithm B for the task above immediately gives an algorithm for $A^{\mathsf{Gap\text{-}K^t}}$: run $B(x, s, t)$ to obtain a program Π, and accept iff Π outputs x in $\tau(t)$ time and $|\Pi| \leq s + c \log t$.

Let k be a parameter to be chosen later. We consider the output distribution $\mathsf{DP}_k(x; U_d)$ (recall that here $d = nk$). For all $z \in \{0,1\}^d$ and a large enough universal constant $c_1 \geq 1$, we have

$$\mathsf{K}^{2t,\mathsf{SAT}}(\mathsf{DP}_k(x;z)) \leq \mathsf{K}^{t,\mathsf{SAT}}(x) + d + c_1 \leq s + d + c_1, \tag{1}$$

since one can first compute x and then compute $\mathsf{DP}_k(x;z)$, and our promise ensures $\mathsf{K}^{t,\mathsf{SAT}}(x) \leq s$.

We then set $k = s + 2c_1$ so that $s + d + c_1 = d + k - c_1$. Now, consider the following function $\bar{D}\colon \{0,1\}^{d+k} \to \{0,1\}$, defined as $\bar{D}(w) := [\mathsf{K}^{2t,\mathsf{SAT}}(w) \leq s + d + c_1]$. Note that $\bar{D}(w)$ can be computed in $O(t)$ Σ_2-time. Hence, from our assumption that $\mathsf{Dist}\Sigma_2\mathsf{P} \subseteq \mathsf{AvgP}$, for some $T_D(t) = \mathrm{poly}(t)$, we have a $T_D(t)$-time heuristic $D(w)$ such that $D(w) \in \{\bar{D}(w), \perp\}$ for every $w \in \{0,1\}^{d+k}$, and $D(w) = \bar{D}(w)$ for at least half of $w \in \{0,1\}^{d+k}$.

In particular, from the above conditions on D, together with (1) and the fact that $s + d + c_1 = |w| - c_1$ for a large enough constant c_1, we have:

1. $D(\mathsf{DP}(x;z)) \neq 0$ for all $z \in \{0,1\}^d$. (*i.e.*, $\Pr_{z\sim U_d}[D(\mathsf{DP}_k(x;z)) = 0] = 0$.)
2. $\Pr_{w\sim U_{d+k}}[D(w) = 0] \geq 1/4$.

That is, D is a distinguisher between $\mathsf{DP}(x; U_d)$ and U_{d+k} with a constant advantage. By Theorem 2.4, there is a $\mathrm{poly}(n)$-time randomized oracle algorithm $R^{(\cdot)}$ that takes $k + O(\log n)$ bits of advice and D as oracle and outputs x with high probability. The composed algorithm R^D runs in $\mathrm{poly}(n) \cdot T_D(t)$ randomized time, and takes $k + O(\log n)$ bits of advice.

[13] *i.e.*, an $(s + c \log t)$-bit program outputting x in $\tau(t)$ time.

Derandomize R^D. Still, to find a witness to $\mathsf{K}^{\tau(t)}(x) \leq k + O(\log t)$, we have to *derandomize* R^D into a deterministic algorithm. We can achieve this by replacing the randomness of R^D with the outputs of the PRG from Theorem 2.5. Now R^D is derandomized with a polynomial-overhead into a $p^{\mathsf{dr}}(T_D(t) \cdot \mathrm{poly}(n))$-time deterministic algorithm with $k + O(\log t)$ bits of advice that outputs x. Here, $p^{\mathsf{dr}}(\text{-})$ (dr stands for derandomization) is the derandomization overhead incurred by applying Theorem 2.5.

To summarize, we have $\mathsf{K}^{p^{\mathsf{dr}}(T_D(t)\cdot\mathrm{poly}(n))}(x) \leq k + O(\log t)$, meaning that we need to set

$$\tau(t) = p^{\mathsf{dr}}(T_D(t) \cdot \mathrm{poly}(n)). \tag{2}$$

Improving $\tau(t)$. As discussed above, we hope to get $\tau(t) = \tau_n(t) = \widetilde{O}(t) \cdot \mathrm{poly}(n)$. Examining (2), we have to ensure two conditions:

$$(1)\ T_D(t) = \widetilde{O}(t) \qquad \text{and} \qquad (2)\ p^{\mathsf{dr}}(t) = p^{\mathsf{dr}}_n(t) \leq \widetilde{O}(t) \cdot \mathrm{poly}(n).$$

Note that $T_D(t) = \widetilde{O}(t)$ can be achieved if we are willing to assume $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, as we already did in Theorem 2.1. Achieving $p^{\mathsf{dr}}(t) \leq \widetilde{O}(t) \cdot \mathrm{poly}(n)$ is much more involved, as we have to get a near-optimal derandomization of the randomized reconstruction algorithm R^D from the assumption that $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$.

2.2 Extremely Efficient HSGs and PRGs from Average-case Easiness of NP

One of our main technical contributions is the construction of *extremely* efficient HSGs (hitting set generators)[14] and PRGs that suffice to derandomize R^D with minor overhead, from the assumption that $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{AvgTIME}_{1/2}[\widetilde{O}(n)]$.

Requirements on the extremely efficient HSGs for the derandomization of R^D. Recall that we are given a randomized algorithm R^D that runs in $t_1 = \widetilde{O}(t) \cdot \mathrm{poly}(n)$ time and takes $n_{\mathsf{adv}} = k + O(\log n) \leq O(n)$ bits as advice. We further observe that R^D only takes $n_{\mathsf{r}} = \mathrm{poly}(n)$ random bits from Theorem 2.4, since the oracle D is a uniform deterministic algorithm. Let $R^D(r)$ be the output of R^D given randomness $r \in \{0,1\}^{n_{\mathsf{r}}}$; then, we have

$$\Pr_{r \sim U_{n_{\mathsf{r}}}}[R^D(r) = x] \geq 2/3.$$

We wish to replace the randomness r of $R^D(r)$ by an output of an HSG. To achieve this, we want an HSG $H\colon \{0,1\}^{O(\log t)} \to \{0,1\}^{n_{\mathsf{r}}}$ that 0.1-hits t_1-time randomized algorithms that take n_{adv}-bits as advice on n_{r}-bit inputs. Given such an HSG H, it follows that there is $u \in \{0,1\}^{O(\log t)}$ such that

$$R^D(H(u)) = x.$$

In other words, given an additional advice $u \in \{0,1\}^{O(\log t)}$, we have a deterministic algorithm $R^D(H(u))$ that outputs x. Here, $R^D(H(u))$ uses $k + O(\log t)$ bits of advice in total, and runs in $t_1 + T_G$ time, where T_G is the running time of computing $H(u)$ given u. Therefore, to get $\mathsf{K}^{\widetilde{O}(t)\cdot\mathrm{poly}(n)}(x) \leq k + O(\log t)$ from the algorithm $R^D(H(u))$, we need $T_G = \widetilde{O}(t) \cdot \mathrm{poly}(n)$.

[14] See Section 3.5 for a formal definition of HSGs.

Our HSG construction. As the technical centerpiece of this paper, we construct the required HSG from the assumption that $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{AvgTIME}[\widetilde{O}(n)]$.

Lemma 2.6 (Special case of Lemma 4.6)**.** *Assuming that* $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$*, for every large enough* t *and* m *such that* $\log t \leq m \leq t$*, there exists an HSG* $H_{t,m}$ *satisfies the following:*

1. $H_{t,m}$ *0.1-hits* t*-time deterministic algorithms with* m *bits of advice on* m*-bit inputs.*
2. $H_{t,m}$ *has* $O(\log(t))$ *seed length and is computable in* $\widetilde{O}(t) \cdot \mathrm{poly}(m)$ *time.*

We mention that combing Lemma 2.6 with a careful "bootstrapping" argument, we are able to construct a $\widetilde{O}(t) \cdot \mathrm{poly}(m)$-time-computable PRG $G_{t,m}$ fooling similar algorithms, with a near-optimal seed length $O(\log m)$. However, the cost is that now our PRG $G_{t,m}$ requires $O(\log t)$ bits of advice to compute. We also remark that we have a trade-off between the $\mathsf{AvgTIME}_{1/2}$ upper bound and the running time of the PRG (HSG). See Theorem 4.1 for the details.

2.2.1 HSGs for "weakly non-uniform" algorithms using the HSG for "low-end" derandomization and strings with high time-bounded Kolmogorov complexity

Note that the most interesting setting for Lemma 2.6 is when $t \gg m$, since if $m \geq t^{\Omega(1)}$, the running time requirement becomes $\mathrm{poly}(t)$, and we can resort to Theorem 2.5. Hence, unlike the usual goal in derandomization that one wishes to hit (or fool for PRGs) maximumly non-uniform algorithms (a.k.a. circuits), our goal here is only to hit "weakly non-uniform" algorithms, whose non-uniformity is much less than its running time. But on the other hand, we wish the running time of the HSG H is quasi-linear in the running time t and polynomial in the non-uniformity m.[15]

Perhaps surprisingly, we managed to prove Lemma 2.6 using the HSG construction from [SU05] intended for "low-end" derandomization (*i.e.*, it was intended for the trade-off between weaker lower bounds and slower derandomization). In particular, [SU05] gives the following construction of HSG.

Theorem 2.7 ([SU05], Informal version of Theorem 3.11)**.** *For every* $t, m \in \mathbb{N}$ *such that* $t \geq m \geq \log t$*, there is a* $t \cdot \mathrm{poly}(m)$*-time algorithm* $\mathsf{SU}_{t,m} \colon \{0,1\}^t \times \{0,1\}^{O(\log t)} \to \{0,1\}^m$ *and a* $\mathrm{poly}(m)$*-time deterministic oracle algorithm* $\mathsf{Rcon}^{(\cdot)}$ *that takes* $\mathrm{poly}(m)$ *bits of advice, such that for every* $x \in \{0,1\}^t$ *and for every* D *that* 0.1*-avoids* $\mathsf{SU}_{t,m}(x, \cdot)$*, there exists an advice* α *so that for every* $i \in [t]$*,* $\mathsf{Rcon}^D(i, \alpha) = x_i$*.*

Our crucial observation here is that Theorem 2.7 enables us to construct HSGs fooling weakly non-uniform algorithms with assumptions *much weaker than circuit lower bounds*. This observation is crucial for our proof of Lemma 2.6.

Formally, we recall a "local" version of t-time bounded Kolmogorov complexity, denoted by $\mathsf{K}^t_{\mathsf{loc}}(x)$, such that $\mathsf{K}^t_{\mathsf{loc}}(x)$ is the minimum size of a program Π such that $\Pi(i)$ outputs x_i in t time for every $i \in [|x|]$. It is not hard to see that $\mathsf{K}^t_{\mathsf{loc}}(x) \geq t$ is essentially equivalent to saying that the circuit complexity of x is at least t (up to a $\mathrm{polylog}(t)$ factor).

We claim that for some large enough constant $c \geq 1$, $t_1 = t \cdot m^c$ and any $x \in \{0,1\}^*$, if $\mathsf{K}^{t_1}_{\mathsf{loc}}(x) \geq m^c$, then $\mathsf{SU}_{|x|,m}(x, \cdot)$ 0.1-hits all t-time algorithms on m-bit inputs and with m-bit advice. Since if $\mathsf{SU}_{|x|,m}(x, \cdot)$ fails to 0.1-hit some t-time algorithm D on m-bit inputs with m-bit advice, by Theorem 2.7, there is an advice $\alpha \in \{0,1\}^{\mathrm{poly}(m)}$ such that for every $i \in [|x|]$, $\mathsf{Rcon}^D(i, \alpha) = x_i$. This procedure $\mathsf{Rcon}^D(\cdot, \alpha)$ implies that $\mathsf{K}^{t_1}_{\mathsf{loc}}(x) < m^c$, a contradiction to our assumption.

Therefore, to prove Lemma 2.6, it suffices to resolve the following construction problem.

[15] Our task is somewhat similar to the question of optimal derandomization for $\mathsf{BPTIME}[n^k]$ studied in [CT21], which aims to derandomize n^k-time randomized algorithms with n-bit non-uniform advice. The difference is that [CT21] also requires the seed length of the PRG to be $(1 + o(1)) \log n$ to keep the derandomization overhead at most $n^{1+o(1)}$. But here, we are fine with an $O(\log t)$-length seed.

Problem 2.8. *Assuming that* $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$. *Given* $t, m \in \mathbb{N}$ *with* $m \ll t$, *in* $t \cdot \mathrm{poly}(m)$ *time construct a string* $x \in \{0,1\}^*$ *satisfying* $\mathsf{K}^t_{\mathsf{loc}}(x) \geq m$.

Note that $|x|$ is clearly bounded by the construction time $t \cdot \mathrm{poly}(m)$, so $\mathsf{SU}_{|x|,m}$ is computable in $|x| \cdot \mathrm{poly}(m) = t \cdot \mathrm{poly}(m)$ time, as desired.

2.2.2 A refined version of [BFP05]

Our solution to Problem 2.8 is inspired by the proof of Theorem 2.5 and follows a similar outline.[16] However, since here we need a $t \cdot \mathrm{poly}(m)$ running time, our algorithm has to be very refined.

Our starting point is the time hierarchy theorem [HS65]. In particular, there is a language $L \in \mathsf{TIME}[2^n] \setminus \mathsf{TIME}[2^n/n^2]$. Moreover, this language is in fact equipped with a uniform "refuter" $\mathcal{R}$, such that given a code of an algorithm B with running time at most $2^n/n^2$, for every large enough input length n, $\mathcal{R}(B, n)$ outputs an n-bit input x_n satisfying $L(x_n) \neq B(x_n)$. (See the proof of Theorem 4.2 for details.)

Next, we apply an efficient PCP to L (*e.g.*, [BGH+05, BV14]), with proof length $\ell = \ell(n) = n + O(\log n)$ and verifier V running in $\mathrm{poly}(n)$ time with ℓ bits randomness.

1. For an input $x \in \{0,1\}^n$, V_x takes an oracle $O\colon \{0,1\}^\ell \to \{0,1\}$ and also ℓ random bits.
2. If $x \in L$, there exists an oracle $O_x\colon \{0,1\}^\ell \to \{0,1\}$ such that $\Pr_{r \sim U_\ell}\left[V_x^{O_x}(r) = 1\right] = 1$. And there is a uniform algorithm computing the truth table of O_x from x in $2^n \cdot \mathrm{poly}(n)$ time.
3. If $x \notin L$, for all oracle $O\colon \{0,1\}^\ell \to \{0,1\}$, we have $\Pr_{r \sim U_\ell}\left[V_x^{O}(r) = 1\right] \leq 1/2$.

Our algorithm solving Problem 2.8. Our algorithm works as follows[17]:

1. Given $t, m \in \mathbb{N}$, we set $n = \log t + c_1 \log m$ for a large enough constant c_1.
2. We construct the code for a "cheating" algorithm A_{cheat} with running time $2^n/n^2$ from our assumption that $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$. We then apply the refuter $\mathcal{R}$ to find an input $x_n \in \{0,1\}^n$ such that $A_{\mathsf{cheat}}(x_n) \neq L(x_n)$.
3. We output the truth table of O_{x_n} (our proof will guarantee that $L(x_n) = 1$), which has length $2^n \cdot \mathrm{poly}(n) \leq t \cdot \mathrm{poly}(m)$.

It remains to specify the algorithm A_{cheat}, which is constructed in the following three steps:

1. We first design a Merlin–Arthur algorithm A_1 that attempts to solve L. On an input $x \in \{0,1\}^n$, it guesses an m-bit program Π with a running time bound t, draws $r \sim U_\ell$, and accepts iff $V_x^{\Pi}(r) = 1$. To summarize, A_1 is a Merlin–Arthur algorithm with running time $t \cdot \mathrm{poly}(m)$, proof complexity m, and randomness complexity ℓ.
2. Under the assumption that $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, A_1 can be simulated by a nondeterministic algorithm A_2 with running time $t \cdot \mathrm{poly}(m)$. (See Lemma 4.5.)
3. Again, under the assumption that $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, A_2 can be simulated by a deterministic algorithm A_3 with running time $t \cdot \mathrm{poly}(m)$. (See Proposition 4.3.)

[16] See [Hir21, Lemma 3.4] for a quick proof sketch of Theorem 2.5.

[17] In fact, the proof strategy below is also inspired by the refuter-based proof of [CLW20] for proving almost-everywhere circuit lower bounds via the algorithmic method.

4. We set $A_{\mathsf{cheat}} = A_3$. Since c_1 is a large enough constant, it follows that A_{cheat} runs in $2^n/n^2$ deterministic time.

From time-hierarchy theorem and the refuter $\mathcal{R}$, it follows that $A_{\mathsf{cheat}}(x_n) \neq L(x_n)$. We claim that this means $L(x_n) = 1$ and $A_{\mathsf{cheat}}(x_n) = A_1(x_n) = 0$. This is because if $L(x_n) = 0$, then on any guessed program Π, A_1 rejects with probability at least $1/2$, and hence $A_{\mathsf{cheat}}(x_n) = A_1(x_n) = 0 = L(x_n)$.

Finally, we claim that when $A_{\mathsf{cheat}}(x_n) = 0 \neq 1 = L(x_n)$, it follows that $\mathsf{K}^t(O_{x_n}) > m$. This is because, if $\mathsf{K}^t(O_{x_n}) \leq m$, then on some guessed m-bit program Π, we would have Π computes O_{x_n} in time t, and therefore A_1 would accept that proof. Consequently $A_{\mathsf{cheat}}(x_n) = A_1(x_n) = 1$, a contradiction to our assumption. To summarize, our algorithm solves Problem 2.8 in $t \cdot \mathrm{poly}(m)$ time, as desired. (See Section 4 for more details.)

2.2.3 Proof for Theorem 2.1

Finally, combining the algorithm above for Problem 2.8 with Theorem 2.7 proves Lemma 2.6, which gives us the desired derandomization to prove the following variant of Lemma 2.2:

Lemma 2.9 (Special case of Lemma 7.2, informal). *If $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, then for every $L \in \mathsf{NP}$ there is $p_n(t) = \widetilde{O}(t) \cdot \mathrm{poly}(n)$ such that for every input x and $t \geq \mathrm{poly}(n)$, one can solve L on input x in $2^{\mathsf{cd}^{t,p(t)}(x)} \cdot \mathrm{poly}(t)$ time.*

We refer the readers to Section 7 for a formal proof of Lemma 7.2. The proof uses important technical ingredients that are proved in Section 5 and Section 6.

As already discussed in Section 2.1.1, Lemma 2.9 implies a $2^{O\left(\sqrt{n \log n}\right)}$-time algorithm for NP from the assumption that $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, thereby proving Theorem 2.1.

2.3 Average-case Hardness of $\Sigma_2\mathsf{TIME}[n]$ from Worst-case Hardness of $\Sigma_k\mathsf{TIME}[n]$

Finally, we discuss how to prove Item (2) of Theorem 1.3. We state its contrapositive below.

Theorem 2.10. $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$ *implies* $\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$ *for any constant $k \in \mathbb{N}$.*

To prove Theorem 2.10, we utilize the advice-efficient HSG construction of [Hir20a] (see Theorem 7.1 for details) to prove the following fine-grained version of Lemma 2.9, which gives algorithms for any $\mathsf{NTIME}[T(n)]$ language.

Lemma 2.11 (Informal version of Lemma 7.2). *If $\Sigma_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, then for every $L \in \mathsf{NTIME}[T(n)]$ there is $p_n(t) = \widetilde{O}(t) \cdot \mathrm{poly}(n)$ such that for every input x and $t \geq \mathrm{poly}(T(n))$, one can solve $L(x)$ in $2^{\mathsf{cd}^{t,p(t)}(x)} \cdot \mathrm{poly}(t)$ time.*

Lemma 2.11 is very powerful. In particular, set $T(n) = 2^{O\left(\sqrt{n \log n}\right)}$, $t_0 = \mathrm{poly}(T(n))$, and $t_i = p_n(t_{i-1})$ for $i \in [\tau]$, where τ is a parameter. One can calculate that for $\tau \leq n$, it holds that $t_\tau \leq 2^{O\left(\sqrt{n \log n} + \tau \log n\right)}$, meaning that we can set $\tau = \sqrt{n/\log n}$ to get a $2^{O\left(\sqrt{n \log n}\right)}$ time algorithm for $\mathsf{NTIME}[T(n)]$. That is:

Corollary 2.12 (Simplified version of Corollary 7.5). *If* $\Sigma_2\mathsf{TIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, *then*

$$\mathsf{NTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right] \subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]. \tag{3}$$

We claim that (3) implies that $\Sigma_k\mathsf{TIME}[n] \subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$ for every $k \in \mathbb{N}$. This can be shown by a simple induction. First note that the base case for $k = 1$ follows directly from (3).

Now, for $k \geq 2$, assume that $\Sigma_{k-1}\mathsf{TIME}[n] \subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$, we will establish the same containment for $\Sigma_k\mathsf{TIME}[n]$. For a language $L \in \Sigma_k\mathsf{TIME}[n]$, by definition (see Definition 3.2), there is a verifier $V(x, y)$ computable in $\Pi_{k-1}\mathsf{TIME}[n]$, such that $L(x) = 1$ iff there exists $y \in \{0,1\}^{O(n)}$ with $V(x, y) = 1$. From our induction hypothesis, we also have that $\Pi_{k-1}\mathsf{TIME}[n] \subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$, meaning that $V(x, y)$ can be computed in $2^{O\left(\sqrt{n \log n}\right)}$ time (note that $|x| + |y| \leq O(n)$).

Hence, it follows that $L \in \mathsf{NTIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$. Then from (3), we have $L \in \mathsf{TIME}\left[2^{O\left(\sqrt{n \log n}\right)}\right]$ as well, which completes the proof.

Organization

In Section 3, we introduce the necessary technical ingredients for this paper. In Section 4, we construct extremely efficient PRGs/HSGs from the average-case easiness of $\mathsf{NTIME}[n]$. In Section 5, we prove a fine-grained version of the algorithmic compression theorem in [Hir21] (see Theorem 5.5). In Section 6, we prove a fine-grained version of the weak symmetry of information in [Hir21] (see Theorem 6.1). In Section 7, we give a worst-case to average-reduction from $\Sigma_k\mathsf{TIME}[n]$ to $\Sigma_2\mathsf{TIME}[n]$, for every constant k, and prove Item (2) of Theorem 1.3. In Section 8, we give a worst-case to average-case reduction from UP to $\mathsf{NTIME}[n]$, and then prove Item (1) of Theorem 1.3 and Theorem 1.4. In Section 9, we consider error-less heuristic schemes whose running time can be efficiently estimated (*i.e.*, $\mathsf{AvgTIME}_{\mathsf{DTIME}[\beta(n)]}[\beta(n)]$), and prove Item (3) of Theorem 1.3. Finally, in Section 10, we use our new techniques to obtain interesting witness compression for languages in NP and prove Theorem 1.5.

3 Preliminaries

3.1 Notation

We use $\mathbb{N}$ to denote the set of all non-negative integers and $\mathbb{N}_{\geq 1} = \mathbb{N} \setminus \{0\}$. For any $k \in \mathbb{N}$, we let $[k]$ denote the set $\{1, 2, \ldots, k\}$. For an integer $a \in \mathbb{N}$, we use $\mathsf{bin}(a)$ to denote the Boolean string representing a in binary (from the most significant bit to the least significant bit). For a Boolean string $x \in \{0,1\}^*$, we slightly abuse the notation and also use $\mathsf{bin}(x)$ to denote the integer represented by x in binary (*i.e.*, $\mathsf{bin}(x) = \sum_{i=1}^{|x|} 2^{|x|-i} \cdot x_i$).

We will use U_n to denote the uniform distribution over $\{0,1\}^n$. For a function $f\colon \mathbb{N} \to \mathbb{N}$, we use $f^{(k)}$ to denote the following $k \in \mathbb{N}$ times composition of f:

$$f^{(k)}(x) := \begin{cases} x & k = 0, \\ f(f^{(k-1)}(x)) & k \geq 1. \end{cases}$$

Let Σ be an alphabet set. For two strings $\alpha, \beta \in \Sigma^*$, we use $\alpha \circ \beta$ the concatenation between α and β. Sometimes we also simply write $\alpha\beta$ to denote concatenation when its meaning is clear from the context. We also use ε to denote the empty string and let $\Sigma^0 = \{\varepsilon\}$ for convenience.

We say a function $\beta\colon \mathbb{N} \to \mathbb{N}$ is a good resource function if it is time-constructible, increasing, satisfies the property that $\beta(a \cdot b) \geq a \cdot \beta(b)$, and is dominated by a polynomial (*i.e.*, $\beta(n) \leq O(n^k)$ for some $k \in \mathbb{N}$).

We use the shorthand $(x, 1^t)$ to denote the string $x \circ 01^t \in \{0,1\}^{|x|+t+1}$, as both x and t can be recovered in time $O(|x| + t)$. In particular, $(\varepsilon, 1^t)$ is encoded by 01^t. We also need the following encoding of a tuple of integers into one single integer.

Proposition 3.1. *There is a pair of encoding/decoding algorithm* $\mathsf{Enc} : \mathbb{N}^* \to \mathbb{N}$ *and* $\mathsf{Dec} : \mathbb{N} \to \mathbb{N}^* \cup \{\perp\}$ *such that:*

1. *Both* Enc *and* Dec *runs in polynomial with respect to their inputs length.*
2. *For* $k \in \mathbb{N}$ *and* $\vec{a} = (a_i)_{i\in[k]} \in \mathbb{N}^k$*, we also use* $\langle \vec{a} \rangle = \langle a_1, \ldots, a_k \rangle$ *to denote* $\mathsf{Enc}(\vec{a})$ *for notation convenience, and we have* $\max_{i\in[k]} a_i \leq \langle \vec{a} \rangle \leq O_k(1) \cdot a_1 \cdot (\prod_{i=2}^k a_i)^2$ *and* $\mathsf{Dec}(\langle \vec{a} \rangle) = \vec{a}$*.*
3. $\mathsf{Dec}(u) = \perp$ *if* $u \neq \mathsf{Enc}(\vec{a})$ *for every* $\vec{a} \in \mathbb{N}^*$*.*

Proof. Given a vector $\vec{a} = (a_1, \ldots, a_k)$, we define $\mathsf{Enc}(\vec{a})$ as follows. For each $i \in \{2, \ldots, k\}$, we let $\ell_i = |\mathsf{bin}(a_i)|$, we duplicate each bit in $\mathsf{bin}(\ell_i)$ to get a string z_i of length $2 \cdot \mathsf{bin}(\ell_i)$ (for example, if $\mathsf{bin}(\ell) = 101$, we get 110011).

Then we set

$$z = 1 \circ \mathsf{bin}(a_1) \circ \mathsf{bin}(a_2) \circ \cdots \mathsf{bin}(a_k) \circ 01 \circ z_2 \circ 01 \circ z_3 \circ \cdots 01 \circ z_\ell,$$

where $\circ$ means concatenation, and define $\mathsf{Enc}(\vec{a})$ as the integer with binary representation z (*i.e.*, $\mathsf{bin}(z)$).

By simple calculation, one can verify that $\mathsf{Enc}(\vec{a}) \leq O_k(1) \cdot a_1 \cdot (\prod_{i=2}^k a_i)^2$. Also, given the integer $\mathsf{Enc}(\vec{a})$, one can easily decode the tuple by first recovering $\ell_2, \ldots, \ell_k$ and then all of the a_i's, thereby constructing the algorithm Dec. We further let Dec output $\perp$ if the decoding fails. □

For a non-uniform randomized algorithm A on input $x \in \{0,1\}^n$ using advice string $\alpha_n \in \{0,1\}^*$ and randomness $z \in \{0,1\}^*$, we let $A(x; z)_{/\alpha_n}$ denote the output of algorithm A on input x with randomness z and advice α_n. (If the algorithm is deterministic, we omit z, and if the algorithm is uniform, we omit α_n.)

3.2 Complexity Classes

We will always consider the RAM model for computation in this paper.

Definition 3.2 (Polynomial hierarchy with bounded running time)**.** *For functions* $T\colon \mathbb{N} \to \mathbb{N}$ *and a constant* $k \in \mathbb{N}$*, let* $\Sigma_k\mathsf{TIME}[T]$ *denote the class of languages* L *such that there is an* $O(T(n))$*-time algorithm* V *such that, for every input* x*, it holds that* $x \in L$ *if and only if*

$$\exists y_1 \in \{0,1\}^{O(T(|x|))}, \forall y_2 \in \{0,1\}^{O(T(|x|))}, \ldots, \mathcal{Q}_k y_k \in \{0,1\}^{O(T(|x|))},\ V(x, y_1, y_2, \ldots, y_k) = 1,$$

where $\mathcal{Q}_k := \exists$ *if* k *is odd and* $\mathcal{Q}_k := \forall$ *otherwise.*

We also define $\Pi_k\mathsf{TIME}[T]$ *be the class of languages whose complement is in* $\Sigma_k\mathsf{TIME}[T]$*. For the special case of* $k = 1$*, we let* $\mathsf{NTIME}[T]$ *denote* $\Sigma_1\mathsf{TIME}[T]$ *and* $\mathsf{coNTIME}[T]$ *denote* $\Pi_1\mathsf{TIME}[T]$*.*

Definition 3.3 (Polynomial hierarchy with limited nondeterminism)**.** *For functions $T, G\colon \mathbb{N} \to \mathbb{N}$ and a constant $k \in \mathbb{N}$, let $\Sigma_k\mathsf{TIMEGUESS}[T, G]$ denote the class of languages L such that there is a $O(T(n))$-time algorithm V such that, for every input x, it holds that $x \in L$ if and only if*

$$\exists y_1 \in \{0,1\}^{G(|x|)}, \forall y_2 \in \{0,1\}^{G(|x|)}, \ldots, \mathcal{Q}_k y_k \in \{0,1\}^{G(|x|)},\ V(x, y_1, y_2, \ldots, y_k) = 1,$$

where $\mathcal{Q}_k := \exists$ if k is odd and $\mathcal{Q}_k := \forall$ otherwise. Let $\mathsf{PHTIMEGUESS}[T, G]$ denote $\bigcup_{k\in\mathbb{N}} \Sigma_k\mathsf{TIMEGUESS}[T, G]$. We also use the shorthand $\mathsf{NTIMEGUESS}[T, G]$ to denote $\Sigma_1\mathsf{TIMEGUESS}[T, G]$.

Note that $\Sigma_k\mathsf{TIME}[T] = \Sigma_k\mathsf{TIMEGUESS}[T, O(T)]$.

Definition 3.4 (Unambiguous time)**.** *For a function $T : \mathbb{N} \to \mathbb{N}$, we let $\mathsf{UTIME}[T]$ denote the class of languages L such that there is a $O(T(n))$-time algorithm V such that for every input x,*

1. *If $x \in L$, then there is a unique $y \in \{0,1\}^{O(T(|x|)}$ such that $V(x, y) = 1$.*
2. *If $x \notin L$, there does not exist any $y \in \{0,1\}^{O(T(|x|))}$ such that $V(x, y) = 1$.*

We let UP denote $\bigcup_{c\in\mathbb{N}} \mathsf{UTIME}[n^c]$.

3.3 Average-case Complexity

We recall some definitions in average-case complexity (see [BT06a] for an exposition on this topic).

Definition 3.5. *Let β be a good resource function and $\delta : \mathbb{N} \to (0, 1)$. For a language L and a distribution family $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$, we say that $(L, \mathcal{D}) \in \mathsf{Avg}_\delta\mathsf{TIME}[\beta(n)]$ if there is an algorithm A such that, for every $n \in \mathbb{N}$ the following hold:*

1. *For every $x \in \mathcal{D}_n$, $A(x, n) \in \{L(x), \perp\}$, and $\Pr_{x\sim\mathcal{D}_n}[A(x, n) = L(x)] \geq 1 - \delta(n)$.*
2. *$A(x, n)$ runs in at most $\beta(|x|)$ time.*

Similarly, we say that $(L, \mathcal{D}) \in \mathsf{Avg}^1_\delta\mathsf{TIME}[\beta(n)]$, if there is an algorithm A such that for every $n \in \mathbb{N}$,

1. *$L(x) = 0$ implies $A(x, n) = 0$ for every $x \in \mathcal{D}_n$, and $\Pr_{x\sim\mathcal{D}_n}[A(x, n) = L(x)] \geq 1 - \delta(n)$.*
2. *$A(x, n)$ runs in at most $\beta(|x|)$ time.*

We say $(L, \mathcal{D}) \in \mathsf{Avg}_\delta\mathsf{P}$ if $(L, \mathcal{D}) \in \mathsf{Avg}_\delta\mathsf{TIME}[p(n)]$ for some polynomial p. Similarly, $(L, \mathcal{D}) \in \mathsf{Avg}^1_\delta\mathsf{P}$ if $(L, \mathcal{D}) \in \mathsf{Avg}^1_\delta\mathsf{TIME}[p(n)]$ for some polynomial p.

Definition 3.6. *Let β be a good resource function. For a language L and a distribution family $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$, we say that $(L, \mathcal{D}) \in \mathsf{AvgTIME}[\beta(n)]$ if there is an algorithm A such that, for every n and $k \in \mathbb{N}$ the following hold:*

1. *For every $x \in \mathcal{D}_n$, $A(x, n, k) \in \{L(x), \perp\}$.*
2. *$\Pr_{x\sim\mathcal{D}_n}[A(x, n, k) = L(x)] \geq 1 - 2^{-k}$.*
3. *$A(x, n, k)$ runs in at most $2^k \cdot \beta(|x|)$ time.*

We say $(L, \mathcal{D}) \in \mathsf{AvgP}$ if $(L, \mathcal{D}) \in \mathsf{AvgTIME}[p(n)]$ for some polynomial p.

We define the parameterized uniform distribution $\mathcal{U}^{\mathsf{para}}$ and the tally distribution $\mathcal{T}$ as below:

Definition 3.7. *$\mathcal{U}^{\mathsf{para}} = \{\mathcal{U}^{\mathsf{para}}_n\}_{n\in\mathbb{N}}$ is the family of distributions such that for every $n \in \mathbb{N}$, $\mathcal{U}^{\mathsf{para}}_n$ is defined as follows: If $\mathsf{Dec}(n) \neq (t,m)$ for any $(t,m) \in \mathbb{N}^2$, then $\mathcal{U}^{\mathsf{para}}_n$ is the singleton distribution on $\{0^n\}$. Otherwise, $\mathcal{U}^{\mathsf{para}}_n$ is the uniform distribution over the set $\{(z,1^t) : z \in \{0,1\}^m\}$ of $(t+m+1)$-bit strings.*

$\mathcal{T} = \{\mathcal{T}_n\}_{n\in\mathbb{N}}$ is the family of distributions that for every $n \in \mathbb{N}$, $\mathcal{T}_n$ is the singleton distribution on $\{1^n\}$.

We note that in some sense $\mathcal{T}$ can be seen as a special case of $\mathcal{U}^{\mathsf{para}}$: For all $n \in \mathbb{N}$, $\mathcal{U}^{\mathsf{para}}_{\langle t,0\rangle}$ is the singleton distribution on the input $(\varepsilon, 1^t) = 01^t$.

3.4 Kolmogorov Complexity and Its Variants

We use $\mathsf{K}(x)$ and $\mathsf{K}^t(x)$ to denote the Kolmogorov complexity and the t-time bounded Kolmogorov complexity of the string x, which we will formally define shortly below. We also refer the readers to [All20] for a recent survey on variants of Kolmogorov complexity.

We also use $\mathsf{Univ}^t(d)$ to denote the output of running a universal RAM machine Univ for t time with input d; if it does not terminate within time t, we write $\mathsf{Univ}^t(d) = \perp$. We also use $\mathsf{Univ}(d)$ to denote its output and we write $\mathsf{Univ}(d) = \perp$ if Univ does not terminate on input d. We will also assume the input programs to Univ are *paddable*, in the sense that Π and $\Pi \circ 0^t$ for every t denote the same program.

In particular, K and K^t are defined with respect to this universal RAM machine Univ. Formally, we have

$$\mathsf{K}(x) := \min_{\Pi\in\{0,1\}^*} \{|\Pi| : \mathsf{Univ}(\Pi) = x\},$$

and

$$\mathsf{K}^t(x) := \min_{\Pi\in\{0,1\}^*} \{|\Pi| : \mathsf{Univ}^t(\Pi) = x\}.$$

We will also use the following "local" version of t-time bounded Kolmogorov complexity:

$$\mathsf{K}^t_{\mathsf{loc}}(x) := \min_{\Pi\in\{0,1\}^*} \{|\Pi| : \mathsf{Univ}^t(\Pi(i)) = x_i \ \ \forall i \in [|x|]\}.$$

Note that the input program Π to Univ may take additional inputs. In the definition of K and K^t we can assume Π set its input to be all zero; in the definition of $\mathsf{K}_{\mathsf{loc}}$, Π takes an integer i as the input, and we use $\Pi(i)$ to denote the program obtained by fixing the input of Π to be i.

For an oracle $A : \{0,1\}^* \to \{0,1\}$, we similarly define the oracle version of the time-bounded Kolmogorov complexity $\mathsf{K}^{t,A}(x)$ as the minimum size of a program that outputs x in time t, given oracle access to A.[18]

For time bounds $s, t \in \mathbb{N}$ where $s \leq t$ and $x \in \{0,1\}^*$, we define the (s,t)-time-bounded computational depth of x as

$$\mathsf{cd}^{s,t}(x) := \mathsf{K}^s(x) - \mathsf{K}^t(x).$$

We also write $\mathsf{cd}^s(x)$ to denote $\mathsf{K}^s(t) - \mathsf{K}(t)$.

[18] As a standard assumption, querying A on an m-bit input takes m time.

3.5 Pseudorandomness

We say a PRG $G\colon \{0,1\}^s \to \{0,1\}^m$ ε-fools a class of functions $\mathcal{F}$ with m-bit inputs, if for every $f \in \mathcal{F}$, it holds that

$$\left| \Pr_{x\sim\{0,1\}^m}[f(x)=1] - \Pr_{z\sim\{0,1\}^s}[f(G(z))=1] \right| \leq \varepsilon.$$

If G fails to ε-fool a particular function f (*i.e.*, the above inequality does not hold for f), then we call f an ε-distinguisher for G.

Similarly, we say an HSG $H\colon \{0,1\}^s \to \{0,1\}^m$ ε-hits a class of functions $\mathcal{F}$ with m-bit inputs, if for every $f \in \mathcal{F}$ with

$$\Pr_{x\sim\{0,1\}^m}[f(x)=1] \geq \varepsilon,$$

there is $u \in \{0,1\}^s$ such that $f(H(u)) = 1$. If H fails to ε-hit a particular function (*i.e.*, f accepts an ε fraction of strings and rejects all outputs of H), then we say that f ε-avoids H.

We need the following construction of extractors.

Theorem 3.8 ([RRV02, Theorem 1]). *For every $m \leq n$ such that $m \geq n^{\Omega(1)}$, there is a poly-time function $\mathsf{RRV}_{n,m}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ for $d = d(n) = O(\log n)$ such that for every $x \in \{0,1\}^n$ and for every 0.1-distinguisher D between $\mathsf{RRV}_{n,m}(x, U_d)$ and U_m, there is a polynomial-time deterministic oracle algorithm Rcon taking $O(m^{1.5})$ bits of advice, such that there is an advice string α so that $\mathsf{Rcon}^D(\alpha) = x$.*

Remark 3.9. *The entropy parameter k in [RRV02, Theorem 1] is essentially the advice complexity of reconstruction mentioned above. Hence, we can set $k = m^{1.5}$ and $m \geq n^{\Omega(1)}$ in Item (1) of [RRV02, Theorem 1] to get seed length $O(\log n)$. Also note that the theorem becomes trivial if $k = m^{1.5} \geq n$ as one can simply store the whole n-bit string as advice. Therefore we can assume the constraint $m \leq k \leq n$ of [RRV02, Theorem 1] is satisfied.*

Theorem 3.10. *There is a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that, for every large enough integer n, m, t such that $t \geq n \geq m$ and $m \geq n^{0.1}$, given a string $x \in \{0,1\}^n$ satisfying $\mathsf{K}^{t\cdot p(n)}(x) \geq m^2$, it follows that $\mathsf{RRV}_{n,m}(x,\cdot)$ is a PRG with seed length $d = O(\log n)$ that 0.1-fools all t-time algorithms on m-bit inputs with m bits of advice.*

Proof. Suppose for the sake of contradiction that there is a t-time algorithm A on m-bit inputs with m-bit of advice such that A is an 0.1-distinguisher between $\mathsf{RRV}_{n,m}(x, U_d)$ and U_m. From Theorem 3.8, there is an advice α of length $O(m^{1.5})$ such that $\mathsf{Rcon}^A(\alpha) = x$, this in particular shows that $\mathsf{K}^{\bar{t}}(x) \leq O(m^{1.5})$ for $\bar{t} = t \cdot \mathrm{poly}(n)$, which is a contradiction to the assumption that $\mathsf{K}^{t\cdot p(n)}(x) \geq m^2$ for a large enough polynomial p. □

We also need the following construction of HSGs from [SU05].

Theorem 3.11 ([SU05]). *For every n and m such that $n \geq m \geq \log n$, there is an $n \cdot \mathrm{poly}(m)$-time algorithm $\mathsf{SU}_{n,m}\colon \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^m$ for $s = s(n) = O(\log n)$, such that for every $x \in \{0,1\}^n$ and for every D which 0.1-avoids $\mathsf{SU}_{n,m}(x, U_s)$, there is a $\mathrm{poly}(m)$-time deterministic oracle algorithm Rcon taking $\mathrm{poly}(m)$ bits of advice, such that there is an advice string α so that for every $i \in [n]$, $\mathsf{Rcon}^D(i, \alpha) = x_i$.*

Proof Sketch. The HSG described in [SU05, Section 5.4] is a reconstructive HSG, with reconstruction algorithm described in [SU05, Section 5.5]. The running time and advice complexity of the

reconstruction procedure is implicit in the proof of [SU05, Theorem 5.5 and Lemma 5.6].[19]

It remains to verify that $\mathsf{SU}_{n,m}$ can be computed in $n \cdot \mathrm{poly}(m)$ time ([SU05] only proved a $\mathrm{poly}(n)$ running time, which is not affordable for us). Without loss of generality, we assume that m is a power of 2 (otherwise, we can always round m up to the nearest power of 2, and use the corresponding construction). Let $h = m$ and $q = m^{20}$, $d = \log(2n)/\log m$.[20] Let $F = \mathbb{F}_q$ and H be a subfield of F with h elements (the choice of H will be clarified later when we introduce two matrices A and B).

Overview of the HSG construction from [SU05]. We first give an overview of the construction of the HSG in [SU05]. Since our main purpose is to bound its running time by $n \cdot \mathrm{poly}(m)$, we omit some minor details that do not affect the analysis.

1. One first constructs d *candidate q-ary PRGs* $G^{(0)}_{\mathsf{q\text{-}ary}}, G^{(1)}_{\mathsf{q\text{-}ary}}, \ldots, G^{(d-1)}_{\mathsf{q\text{-}ary}} \colon \mathbb{F}_q^d \to \mathbb{F}_q^m$. By q-ary PRG we mean that the output string has alphabet $\mathbb{F}_q$ with size q (see [SU05, Definition 5.3] for details).

2. Next, for each $i \in [d]$, one converts the candidate q-ary PRG $G^{(i)}_{\mathsf{q\text{-}ary}}$ to a candidate (binary) PRG $G^{(i)}_{\mathsf{bin}} \colon \{0,1\}^{O(\log n)} \to \{0,1\}$ at the expense of additional $O(\log n)$ seed length and minor running time overhead (see [SU05, Lemma 5.6] for details).

3. Finally, let $s_0(n) = O(\log n)$ be seed length of the $G^{(i)}_{\mathsf{bin}}$'s (without loss of generality, we may assume that these d candidate PRGs have the same seed length). Our final HSG $\mathsf{SU}_{n,m}(x, \cdot)$ takes an $s_0(n) + \lceil \log d \rceil$-length seed, treats the seed as a pair (α, i) from $\{0,1\}^{s_0(n)} \times \{0, 1, \ldots, d-1\}$, and outputs $G^{(i)}_{\mathsf{bin}}(\alpha)$.[21]

Since the transformation from $G^{(i)}_{\mathsf{q\text{-}ary}}$ to $G^{(i)}_{\mathsf{bin}}$ only adds minor overhead (at most $n \cdot \mathrm{poly}(m)$, see [SU05, Lemma 5.6]), to bound the running time of $\mathsf{SU}_{n,m}$, it suffices to bound the running time of the $G^{(i)}_{\mathsf{q\text{-}ary}}$'s.

Bounding the running time of $G^{(i)}_{\mathsf{q\text{-}ary}}$'s. In the following, we first describe the construction $G^{(i)}_{\mathsf{q\text{-}ary}}$ and then analyze the running time. To compute the $G^{(i)}_{\mathsf{q\text{-}ary}}$'s, we first constructs two matrices $A, B \in \mathbb{F}_q^{d \times d}$ (we will get to the complexity of constructing A and B at the end of the proof) such that

1. $A^{q^d - 1}$ and $B^{h^d - 1}$ are the identity matrix.

2. For any non-zero vector $\vec{v} \in F^d$: $\{A^i \vec{v}\}_{1 \le i < q^d} = F^d \setminus \{0\}$.

3. For any non-zero vector $\vec{v} \in H^d$: $\{B^i \vec{v}\}_{1 \le i < h^d} = H^d \setminus \{0\}$.

[19] In particular, it is summarized at the end of proof of [SU05, Theorem 5.5]. We also remark that the reconstruction algorithm described in [SU05] is indeed randomized and uses at most $\mathrm{poly}(m)$ random bits. We can nonetheless first amplify its correctness probability on each input to at least $1 - 1/2n$ by repeating the reconstruction for $O(\log n)$ times, and then specify a string of good random bits that makes the reconstruction algorithm computing x_i correctly for every $i \in [n]$ as part of the advice. Note that such a string of good random bits exists by a union bound. Since $m \ge \log n$, the resulting reconstruction algorithm is deterministic, and still runs in $\mathrm{poly}(m)$ time and uses $\mathrm{poly}(m)$ bits of advice.

[20] d is set to be $\log n / \log m$ in [SU05, Section 5.3]. We set it to be slightly larger so that $h^d > n$.

[21] We only get an HSG instead of the PRG because [SU05] proved that for any given small circuit C, *at least one* of the candidate PRGs fools this circuit. Since we simply put all outputs of these d candidate PRGs together, their combined output may no longer fools the given circuit C. But nonetheless, their combined output would still be a valid HSG.

4. $B = A^{(q^d-1)/(h^d-1)}$.

We also note that from the properties above, the matrix B already determines the subfield H. Let $\vec{1}$ be the all-one vector in $\mathbb{F}^d$. We then compute a degree-hd polynomial $\hat{x} : \mathbb{F}^d \to \mathbb{F}$ such that $\hat{x}(B^j\vec{1}) = x_j$ for all $j \in [n]$. Then $G^{(i)}_{\mathsf{q\text{-}ary}}$ is specified as follows:

$$G^{(i)}_{\mathsf{q\text{-}ary}}(\vec{v}) = \hat{x}(A^{q^i\cdot 1}\vec{v}) \circ \hat{x}(A^{q^i\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q^i\cdot m}\vec{v}).$$

Note that we cannot afford to compute a full description of $\hat{x}$, as that takes $\mathrm{poly}(n)$ bits to store (recall that we aim to compute $G^{(i)}_{\mathsf{q\text{-}ary}}(\vec{v})$ in $n \cdot \mathrm{poly}(m)$ time). Therefore, we aim to compute $\hat{x}(\vec{u})$ directly from x, A and B, without constructing a full description of $\hat{x}$.

Let $\vec{a} \in H^d$. We define the following polynomial

$$e_{\vec{a}}(\vec{v}) := \prod_{i\in[d]} \prod_{h\in H\setminus\{a_i\}} \frac{h - v_i}{h - a_i}.$$

Observe that for every $\vec{v} \in H^d$, $e_{\vec{a}}(\vec{v})$ equals 1 if $\vec{a} = \vec{v}$, and it equals 0 otherwise. The polynomial $\hat{x}$ can then be described as

$$\hat{x}(\vec{v}) = \sum_{j\in[n]} x_i \cdot e_{B^j\vec{1}}(\vec{v}).$$

From the above description, one can see that for any $\vec{v} \in \mathbb{F}^d$, $\hat{x}(\vec{v})$ can be computed in $n \cdot \mathrm{poly}(q, m, \log n) = n \cdot \mathrm{poly}(m)$ time (recall that $m \geq \log n$), since B^j can be computed by repeated squaring in $\mathrm{poly}(q) \cdot O(\log n) = \mathrm{poly}(m)$ time, and $e_{B^j\vec{1}}(\vec{v})$ can be computed in $\mathrm{poly}(q)$ time.

Next, note that for each $i \in \{0, 1, \ldots, d-1\}$ and $j \in [m]$, we can also compute $A^{q^i\cdot j}$ by repeated squaring in $\mathrm{poly}(q) \cdot O(\log n) = \mathrm{poly}(m)$ time. Therefore, $G^{(i)}_{\mathsf{q\text{-}ary}}(\vec{v})$ can be computed in $n \cdot \mathrm{poly}(m)$ time as well, as desired.

Specifying the matrix A. Finally, we need to account for the running time of finding A (B can be determined from A). In the proof of [SU05, Lemma 5.4], it is shown that we only need to find a generator of the multiplicative group of $\mathsf{GF}(F^d) = \mathsf{GF}(\mathbb{F}_{q^d})$, and such a generator can be found in $\mathrm{poly}(q^d)$ time by simply enumerating all elements in $\mathsf{GF}(\mathbb{F}_{q^d})$. Unfortunately, we cannot afford to perform such an exhaustive search as we aim for $n \cdot \mathrm{poly}(m)$ time.

Fortunately, we observe that a generator of $\mathsf{GF}(\mathbb{F}_{q^d})$ can be described by $d \cdot \log q = O(\log n)$ bits. Therefore, we can specify such a generator as $O(\log n)$ bits of advice, and this implies that we can compute $\mathsf{SU}_{n,m}(x, \cdot)$ in $n \cdot \mathrm{poly}(m)$ time with $O(\log n)$ bits of advice.

Finally, note that for an HSG computable with advice, one can simply append the advice bits to the end of the seed to obtain another HSG without advice that fools the same set of algorithms. Therefore, we can make $\mathsf{SU}_{n,m}(x, \cdot)$ computable in $n \cdot \mathrm{poly}(m)$ time without advice. One can also see this transformation does not affect the reconstruction algorithm, as the reconstruction algorithm can also take $O(\log n) \leq \mathrm{poly}(m)$ bits of additional advice to specify the matrix A. □

Remark 3.12. *It is worth mentioning why we cannot use simpler PRG constructions such as the Nisan–Wigderson PRG [NW94, IW97, STV01]. The issue is that we require the PRG to have a seed length of $O(\log n)$ while keeping the running time and advice complexity to be $\mathrm{poly}(m)$. When $m \ll n$, the Nisan–Widgerson PRG requires a seed length of $O(\log^2 n / \log m)$, which is too much for our applications.*

Theorem 3.13. *There is a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that, for every large enough integer n, t, m such that $t \geq m$, given a string $x \in \{0,1\}^n$ satisfying $\mathsf{K}^t_{\mathsf{loc}}(x) \geq p(m)$, it follows that $\mathsf{SU}_{n,m}(x,\cdot)$ is an HSG with seed length $d = O(\log n)$ that 0.1-hits all $t/p(m)$-time algorithm on m-bit inputs with m bits of advice.*

Proof. Suppose for the sake of contradiction that there is a $t/p(m)$-time algorithm A on m-bit inputs with m bits of advice such that A 0.1-avoid $\mathsf{SU}_{n,m}(x,\cdot)$. From Theorem 3.11, it follows that there is a $\mathrm{poly}(m)$-bit length advice α such that $\mathsf{Rcon}^A(i,\alpha) = x_i$ for each $i \in [n]$. This in particular implies that $\mathsf{K}^{t/p(m)\cdot\mathrm{poly}(m)}_{\mathsf{loc}}(x) \leq \mathrm{poly}(m)$, which is a contradiction to the assumption that $\mathsf{K}^t_{\mathsf{loc}}(x) \geq p(m)$ for a large enough polynomial p. □

4 Extremely Efficient PRGs from Average-case Easiness of $\mathsf{NTIME}[n]$

In this section, we prove the following theorem, which constructs a super-efficient PRG from (one-sided) average-case easiness of $\mathsf{coNTIME}[n]$.

Theorem 4.1. *Assuming that $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ for a good resource function $\beta\colon \mathbb{N} \to \mathbb{N}$, for every large enough $t, m \in \mathbb{N}$ such that $\log t \leq m \leq t$, there is a PRG $G_{t,m}$ satisfying the following:*

1. *$G_{t,m}$ 0.1-fools t-time deterministic algorithms with m bits of advice on m-bit inputs.*
2. *$G_{t,m}$ has $O(\log(m))$ seed length and is computable in $\beta^{(3)}(t \cdot \mathrm{poly}(m))$ time with $O(\log t)$ bits of advice.*

4.1 Technical Ingredients

We need the following standard time hierarchy theorem.

Theorem 4.2 ([HS65])**.** *For a time-constructible function $T\colon \mathbb{N} \to \mathbb{N}$, there is another function $\bar{T} = T \cdot \mathrm{polylog}(T)$ such that there is a language $L \in \mathsf{TIME}[\bar{T}]$ such that for every large enough integer n the following holds:*

- *For every $\Pi \in \{0,1\}^n$, we have $L(\Pi) \neq \mathsf{Univ}^{T(n)}(\Pi(\Pi))$.*

Proof. For every $\Pi \in \{0,1\}^*$, we define L as follows:

- Let $z = \mathsf{Univ}^{T(n)}(\Pi(\Pi))$. If z is not a single bit, we set $L(\Pi) = 0$.
- Otherwise, $z \in \{0,1\}$ and we set $L(\Pi) = 1 - z$.

It is straightforward to verify that the language L above satisfies our requirements. (In the RAM model, the simulation $\mathsf{Univ}^{T(n)}(\Pi(\Pi))$ takes $T(n) \cdot \mathrm{polylog}(T(n))$ time.) □

Note that since we assumed our encoding of programs is paddable, given a program B with running time at most $T(n)$, for a large enough input length n, we have $L(B \circ 0^{n-|B|}) \neq B(B \circ 0^{n-|B|})$. Thus, a refuter $\mathcal{R}$ for L against all $T(n)$-time algorithms can be defined by $\mathcal{R}(B,n) = B \circ 0^{n-|B|}$.

We also need the following standard proposition.

Proposition 4.3. *Let $\beta\colon \mathbb{N} \to \mathbb{N}$ be a good resource function. Assuming $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$, it follows that $\mathsf{NTIME}[2^n] \subseteq \mathsf{TIME}[\beta(O(2^n))]$. Furthermore, for large enough input length n, given a nondeterministic program Π_0 of length $2\log n$ and running time $t \geq 2^n$, one can construct in $\mathrm{polylog}(n)$ time a deterministic program Π_1 of length $2\log n + O(1)$ such that*

- *Π_1 and Π_0 agree on all n-bit inputs.*
- *Π_1 runs in $\beta(\widetilde{O}(t))$ time.*

Proof. Let $L \in \mathsf{NTIME}[2^n]$. We now consider the following mapping τ from $\{0,1\}^*$ and $\mathbb{N}_{\geq 1}$: For every $x \in \{0,1\}^*$, we set $\tau(x) = \mathsf{bin}(1 \circ x)$. In particular we have $2^{|x|} \leq \tau(x) \leq 2^{|x|+1}$. We also note that τ is bijective.

Next we define another language L' such that

1. $L'(x) = 0$ if x is not of the form 01^t for some $t \geq 1$ and
2. $L'(x) = L(\tau^{-1}(t))$ otherwise.

We can see that $L' \in \mathsf{NTIME}[n]$. From $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ and the fact that L' is only supported on $\mathcal{U}^{\mathsf{para}}_{\langle t,0\rangle}$ for some $t \in \mathbb{N}_{\geq 1}$, it follows that $L' \in \mathsf{TIME}[\beta(n)]$. Now, applying the deterministic algorithm for L' to solve L, we have that $L \in \mathsf{TIME}[\beta(O(2^n))]$. Now we prove the "furthermore" part of the proposition. Consider a language $H \in \mathsf{NTIME}[2^n]$ defined by the following algorithm:

- Given an n-bit input x, we treat the first $2\log n$ bits as a nondeterministic program Π. Next, we let y be the remaining $n - 2\log n$ bits, and let i be the smallest integer such that $y_i = 1$. if no such i exists then we immediately reject.
- We then set $z = y_{>i}$, and simulate Π on input z for 2^n steps and output its answer.

From above discussions we have that $H \in \mathsf{TIME}[\beta(O(2^n))]$ as well; we also let A be the corresponding deterministic algorithm for H.

Now, suppose we are given a $2\log n$-bit nondeterministic program Π_0 with running time $t \geq 2^n$. Let ℓ be the smallest integer such that $2^\ell \geq t$ and $\ell - 2\log\ell - 1 \geq n$ and A_ℓ be the restriction of A to ℓ-bit inputs. Fixing the first $2\log\ell$ bits of the input to A_ℓ to be Π_0 and the next $\ell_0 = \ell - 2\log\ell - n$ bits to be $0^{\ell_0-1}1$ (note that we assumed our encoding of algorithms are paddable, we can pad Π_0 to be length exactly $2\log\ell$), we obtain a $\beta(O(2^\ell)) = \beta(\widetilde{O}(t))$ time deterministic algorithm Π_1 satisfying our requirements. □

We also define the following language.

Definition 4.4. *A language $\mathsf{MINKT}_{\mathsf{pad}}$ is defined as follows. On input $z \in \{0,1\}^n$:*

1. *If $\mathsf{Dec}(n) \neq (t,m)$ for any $(t,m) \in \mathbb{N}^2$ then $\mathsf{MINKT}_{\mathsf{pad}}(z) = 0$.*
2. *Otherwise, $\mathsf{MINKT}_{\mathsf{pad}}(z) = 1$ if and only if $\mathsf{K}^t(z_{\leq m}) \geq \sqrt{m}$.*

From the definition one can see that $\mathsf{MINKT}_{\mathsf{pad}} \in \mathsf{coNTIME}[n]$.

We also need the following lemma.

Lemma 4.5. *Let $\beta\colon \mathbb{N} \to \mathbb{N}$ be a good resource function. Assuming $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$, there is a nondeterministic algorithm $\mathsf{A}_{\mathsf{Nderand}}$ satisfying the following:*

1. $\mathsf{A}_{\mathsf{Nderand}}$ *takes* $t, m \in \mathbb{N}$ *and a program* $\Pi \in \{0,1\}^m$ *as input, and runs in* $\beta(t \cdot \mathrm{poly}(m))$ *time.*

2. *For every computational path,* $\mathsf{A}_{\mathsf{Nderand}}$ *either outputs* $\perp$ *or an estimate* p_{est} *such that*

$$\left| p_{\mathsf{est}} - \Pr_{x \sim \{0,1\}^m}[\mathsf{Univ}^t(\Pi(x)) = 1] \right| \leq 0.1.$$

   Note that on different computational paths, it may output different estimates.

3. *On at least one computational path,* $\mathsf{A}_{\mathsf{Nderand}}$ *outputs an estimate.*

Proof. Recall that $\mathsf{MINKT}_{\mathsf{pad}} \in \mathsf{coNTIME}[n]$. By our assumption that $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$, it follows that there is a $\beta(n)$-time algorithm A such that for every $(t, m) \in \mathbb{N}^2$ and $n = \langle t, m \rangle$, the following hold:

1. $\mathsf{MINKT}_{\mathsf{pad}}(z) = 0$ implies $A(z) = 0$ for all $z \in \mathcal{U}^{\mathsf{para}}_{\langle n-m-1,m \rangle}$.

2. $\Pr_{z \sim U_m}[A(z \circ 01^{n-m-1}) = \mathsf{MINKT}_{\mathsf{pad}}(z \circ 01^{n-m-1})] \geq 1/2$. Note that $z \circ 01^{n-m-1} = (z, 1^{n-m-1}) \in \mathcal{U}^{\mathsf{para}}_{\langle n-m-1,m \rangle}$.

Now, suppose we are given inputs t, m and a program $\Pi \in \{0,1\}^m$. Let $\bar{t} = t \cdot p(m^4)$ where p is the polynomial defined in Theorem 3.10, and $u = \langle \bar{t}, m^4 \rangle$. Our nondeterministic algorithm guesses an m^4-bit string z, and rejects immediately if $A(z \circ 01^{u-m^4-1}) \neq 1$.

Otherwise, it holds that $A(z \circ 01^{u-m^4-1}) = 1$. From the first property of A, it follows that $\mathsf{MINKT}_{\mathsf{pad}}(z \circ 01^{u-m^4-1}) = 1$. This further implies that $\mathsf{K}^{\bar{t}}(z) \geq m^2$. Applying Theorem 3.10 with z, we obtain an $O(\log m)$ seed-length PRG fooling t-time algorithms on m-bit inputs with m bits of advice. We then use this PRG to obtain an estimate of $\Pr_{x \in \{0,1\}^n}[\mathsf{Univ}^t(\Pi(x)) = 1]$ ($\mathsf{Univ}^t(\Pi(x))$ is an algorithm with running time at most t).

Finally, one can verify that the whole algorithm runs in $\beta(u) = \beta(t \cdot \mathrm{poly}(m))$ nondeterministic time, and there must exist a guess z making our algorithm accept, as more than half of strings z in $\{0,1\}^{m^4}$ satisfy $\mathsf{K}^{\bar{t}}(z) \geq m^2$. □

4.2 Proof of Theorem 4.1

To prove Theorem 4.1, we first construct $O(\log t)$ seed length HSGs from $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$, as captured by the following lemma.

Lemma 4.6. *Assuming that* $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ *for a good resource function* $\beta \colon \mathbb{N} \to \mathbb{N}$*, for every large enough* $t, m \in \mathbb{N}$ *such that* $\log t \leq m \leq t$*, there is an HSG* $H_{t,m}$ *satisfying the following:*

1. $H_{t,m}$ *0.1-hits t-time deterministic algorithms with m bits of advice on m-bit inputs.*

2. $H_{t,m}$ *has* $O(\log(t))$ *seed length and is computable in* $\beta^{(2)}(t \cdot \mathrm{poly}(m))$ *time.*

We first show that Lemma 4.6 implies Theorem 4.1.

Proof of Theorem 4.1. $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ implies that $(\mathsf{MINKT}_{\mathsf{pad}}, \mathcal{U}^{\mathsf{para}}) \in \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$. It follows that there is a $\beta(n)$-time algorithm A such that for every $(t, m) \in \mathbb{N}^2$ and $n = \langle t, m \rangle$, the following hold:

1. $\mathsf{MINKT}_{\mathsf{pad}}(z) = 0$ implies $A(z) = 0$ for all $z \in \mathcal{U}^{\mathsf{para}}_{\langle n-m-1,m \rangle}$.

2. $\Pr_{z\sim U_m}[A(z\circ 01^{n-m-1}) = \mathsf{MINKT}_{\mathsf{pad}}(z\circ 01^{n-m-1})] \geq 1/2$.

Let $\bar{t} = t\cdot p(m^4)$ where p is the polynomial defined in Theorem 3.10, and $u = \langle \bar{t}, m^4\rangle$. Let $\tilde{t} = \beta(u)$ and $H_{\tilde{t},m^4}$ be the $O(\log \tilde{t})$-seed length HSG from Lemma 4.6 that 0.1-hits $\tilde{t}$-time algorithm with m^4-bit inputs and m^4-bit advice.

When running A over $u = \langle \bar{t}, m^4\rangle$ bit inputs, we have that

$$\Pr_{z\sim U_{m^4}}[A(z\circ 01^{u-m^4-1}) = \mathsf{MINKT}_{\mathsf{pad}}(z\circ 01^{u-m^4-1})] \geq 1/2.$$

By the definition of $\mathsf{MINKT}_{\mathsf{pad}}$, we also have

$$\Pr_{z\sim U_{m^4}}[\mathsf{MINKT}_{\mathsf{pad}}(z\circ 01^{u-m^4-1}) = 1] \geq 0.99.$$

Putting the above two inequalities together, we have

$$\Pr_{z\sim U_{m^4}}[A(z\circ 01^{u-m^4-1}) = 1] \geq 1/3.$$

Since $A(z\circ 01^{u-m^4-1})$ is an algorithm with m^4-bit inputs, $O(\log u)$ bits of advice (specifying the length of zeros appending to the end of the input), and $\tilde{t} = \beta(u)$ running time, we know that there must exist an $O(\log \tilde{t})$ length seed s such that

$$A(H_{\tilde{t},m^4}(s)\circ 01^{u-m^4-1}) = 1.$$

From the first property of A, this also implies that $\mathsf{MINKT}_{\mathsf{pad}}(H_{\tilde{t},m^4}(s)\circ 01^{u-m^4-1}) = 1$. Hence, for $x = H_{\tilde{t},m^4}(s)$, we have $\mathsf{K}^{\tilde{t}}(x) \geq m^2$. Our final PRG $G_{t,m}(\text{-})$ is then defined to be $\mathsf{RRV}_{m^4,m}(x,\text{-})$. Now we verify it satisfies all the required properties.

- By Theorem 3.10, $\mathsf{RRV}_{m^4,m}(x,\text{-})$ has seed length $O(\log m)$ and 0.1-fools all t-time algorithms with m-bit inputs and m-bit advice.
- The running time of $G_{t,m}$ is dominated by the running time of $H_{\tilde{t},m^4}$, which is

  $$\beta^{(2)}(\tilde{t}\cdot \mathrm{poly}(m)) \leq \beta^{(2)}(\beta(t\cdot \mathrm{poly}(m))\cdot \mathrm{poly}(m)) \leq \beta^{(3)}(t\cdot \mathrm{poly}(m)).$$

  The last inequality above uses the fact that β is a good resource function and $\beta(a\cdot b) \geq a\cdot\beta(b)$. And we only need to specify $O(\log \tilde{t}) \leq O(\log t)$ bits of advice s to compute $x = H_{\tilde{t},m^4}(s)$, since $\tilde{t} \leq \mathrm{poly}(t)$ as $t \geq m$ and β is at most polynomial.

□

Finally, we prove Lemma 4.6.

Proof of Lemma 4.6. Letting $T(n) = 2^n$, and applying Theorem 4.2 with $T(n)$, it follows that there is another time function $\bar{T}(n) = T(n)\cdot \mathrm{polylog}(T(n))$ such that there is a language $L \in \mathsf{TIME}[\bar{T}]$ satisfying the conditions in Theorem 4.2.

Without loss of generality, we assume that m is a power of 2. We also use $\mathsf{pw}(x)$ to denote the smallest power of 2 which is at least x (*i.e.*, $\mathsf{pw}(x) = 2^{\lceil \log x\rceil}$).

Applying PCP. Now we take a very efficient PCP algorithm for L (*e.g.*, [BV14, BGH+05]) such that for $\ell = \log \bar{T}(n) + O(\log\log \bar{T}(n))$, there is a verifier $V^{(\cdot)}$ satisfying the following:

1. For an input $x \in \{0,1\}^n$, $V_x^{(-)}$ (V_x denotes V with the input set to x) takes an oracle $O\colon \{0,1\}^\ell \to \{0,1\}$, ℓ random bits, and runs in $\mathrm{poly}(\ell)$ time.
2. If $x \in L$, there exists an oracle $O_x\colon \{0,1\}^\ell \to \{0,1\}$ such that

$$\Pr_{r\in\{0,1\}^\ell}\left[V_x^{O_x}(r) = 1\right] = 1.$$

   Moreover, there is a uniform algorithm computing the truth-table of O_x from x in $\widetilde{O}(\bar{T}(n))$ time.
3. If $x \notin L$, for all oracles $O\colon \{0,1\}^\ell \to \{0,1\}$, we have

$$\Pr_{r\in\{0,1\}^\ell}\left[V_x^{O}(r) = 1\right] \le 1/2.$$

Set up. Let $t_0 = \mathsf{pw}(t \cdot m^{\alpha_1})$ and $\bar{t} = \mathsf{pw}(\beta^{(2)}(t_0 \cdot m^{\alpha_2}))$, where $\alpha_1 > 1$ and $\alpha_2 > 1$ are two constants to be specified later.

In the following we fix $n = \log \bar{t}$. Note that since t is large enough, we can assume that n is also large enough and the condition in Theorem 4.2 holds.

Now we consider the following algorithm that aims to speed up L over n-bit inputs. Let $\tau \ge 1$ be a parameter to be set later.

Constructing a faster algorithm. We first consider a nondeterministic algorithm Π_0 as follows. On input $x \in \{0,1\}^n$:

1. Guess an m^τ-bit program Π and construct an oracle $O_\Pi\colon \{0,1\}^\ell \to \{0,1\}$ defined by letting $O_\Pi(r) = \mathsf{Univ}^{t_0}(\Pi(r))$ for every $r \in \{0,1\}^\ell$.
2. Observe that now $V_x^{O_\Pi}(r)$ is a $t_0 \cdot \mathrm{poly}(n)$-time randomized algorithm with ℓ bit inputs and $n_{\mathsf{adv}} = m^\tau + O(\log n)$ bits of advice. We apply Lemma 4.5 and run $\mathsf{A}_{\mathsf{Nderand}}$ with relevant parameters to derandomize $V^{O_\Pi}(r)$; and we accept if $\mathsf{A}_{\mathsf{Nderand}}$ outputs an estimate p_{est} such that $p_{\mathsf{est}} \ge 3/4$.

Note that Π_0 can be specified by the integers m, t_0 and the constant τ (we do not have to encode integer n in Π_0 as it is the input length). Hence it can be described by $\log\log m + O(1) + \log\log t_0 \le 2\log n$ bits (recall that m is assumed to be a power of 2, and t_0 is a also power of 2 by the definition of pw). Also, on n-bit inputs, from Lemma 4.5, Π_0 runs in

$$\beta(t_0 \cdot m^{O(\tau)})$$

nondeterministic time.

Now, from Proposition 4.3, we can further obtain a $2\log n + O(1)$-length deterministic program Π_1 agreeing with Π_0 on all n-bit inputs, with running time

$$\beta(\beta(t_0 \cdot m^{O(\tau)})).$$

Obtaining the PRG. Setting τ and α_2 properly, the above algorithm runs faster than $\bar{t}$. Padding Π_1 to be length n to obtain a program Π_2, by Theorem 4.2, we have

$$L(\Pi_2) \neq \Pi_2(\Pi_2).$$

Note that $\Pi_0(\Pi_2) = \Pi_1(\Pi_2) = \Pi_2(\Pi_2)$ by the definitions of Π_1 and Π_2. Hence, we have that $L(\Pi_2) \neq \Pi_0(\Pi_2)$. This is only possible when $L(\Pi_2) = 1$ and $\Pi_0(\Pi_2) = 0$, which implies that the accepting truth-table O_{Π_2} for L on the input Π_2 satisfies $\mathsf{K}^{t_0}_{\mathsf{loc}}(\mathrm{tt}(O_{\Pi_2})) > m^\tau$ (otherwise, it would be guessed by Π_0 and we would have $\Pi_0(\Pi_2) = 1$ as well, a contradiction).

Now we are ready to specify our HSG construction. We first construct the n-bit program Π_2 in $\mathrm{poly}(n) = \mathrm{polylog}(t)$ time and then compute the truth table of O_{Π_2}. Let $x = \mathrm{tt}(O_{\Pi_2})$. Our HSG is then obtained by applying Theorem 3.13 and fixing the first parameter of $\mathsf{SU}_{|x|,m}(\text{-},\text{-})$ to be x. Finally, we verify that $\mathsf{SU}_{|x|,m}(x,\text{-})$ satisfies our requirement:

- The seed length of $\mathsf{SU}_{|x|,m}(x,\text{-})$ is $O(\log|x|) \leq O(\ell) \leq O(\log \bar{T}(n)) \leq O(\log \bar{t}) \leq O(\log t)$. The last inequality holds since β is at most polynomial and $m \leq t$.
- The running time of $\mathsf{SU}_{|x|,m}(x,\text{-})$ is $\widetilde{O}(\bar{t}) \cdot \mathrm{poly}(m) \leq \beta^{(2)}(t \cdot \mathrm{poly}(m))$.
- Let $p_{\mathsf{SU}}(m)$ be the universal polynomial p in Theorem 3.13. We set the constants τ and α_1 so that $t_0/p_{\mathsf{SU}}(m) \geq t \cdot m^{\alpha_1}/p_{\mathsf{SU}}(m) \geq t$ and $m^\tau \geq p_{\mathsf{SU}}(m)$. From Theorem 3.13 and the condition that $\mathsf{K}^{t_0}_{\mathsf{loc}}(\mathrm{tt}(O_{\Pi_2})) > m^\tau$, it follows that $\mathsf{SU}_{|x|,m}(x,\text{-})$ 0.1-hits all t-time algorithms on m-bit inputs with m bits of advice.

□

5 Fine-grained Algorithmic Compression

We begin by introducing the definitions of the ensemble of languages and algorithms. Recall that we use $(x, 1^t)$ to denote the string $x \circ 01^t$.

Definition 5.1 (Ensemble of languages (Definition 4.1 of [Hir21]))**.** *For every language $L \subseteq \{0,1\}^*$ and every $t \in \mathbb{N}$, let L_t denote $\{x \in \{0,1\}^* \mid (x,1^t) \in L\}$. We say L is an* ensemble of languages *if there exists a constant c_L such that $|x| \leq c_L \cdot t$ for all $t \in \mathbb{N}$ and $x \in L_t$. We identify an ensemble $L \subseteq \{0,1\}^*$ of languages with a family $\{L_t\}_{t\in\mathbb{N}}$.*

For a promise problem $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$, let $\Pi_{\mathsf{YES},t}$ denote $\{x \in \{0,1\}^* \mid (x,1^t) \in \Pi_{\mathsf{YES}}\}$ and $\Pi_{\mathsf{NO},t}$ denote $\{x \in \{0,1\}^* \mid (x,1^t) \in \Pi_{\mathsf{NO}}\}$. We say Π is an ensemble of promise problems *if there exists a constant c_Π such that $|x| \leq c_\Pi \cdot t$ for all $t \in \mathbb{N}$ and $x \in \Pi_{\mathsf{YES},t} \cup \Pi_{\mathsf{NO},t}$.*

Remark 5.2. *We slightly change the definition as given in [Hir21] so that $(x,1^t) \in L$ implies $|x| \leq O(t)$, as opposed to implying $|x| \leq \mathrm{poly}(t)$. This is so that the fine-grained analysis goes through.*

Definition 5.3 (Ensemble of algorithms)**.** *For every ensemble of languages $L \subseteq \{0,1\}^*$, we say that $\{A_t\}_{t\in\mathbb{N}}$ is an* ensemble of algorithms for L *if for all $t \in \mathbb{N}$, A_t is a program such that for all $x \in \{0,1\}^{\leq c_L \cdot t}$, we have*[22]

$$(x,1^t) \in L \iff A_t(x) = 1.$$

Moreover, we say that this ensemble of algorithms has size $s : \mathbb{N} \to \mathbb{N}$ *if $|A_t| \leq s(t)$ for all $t \in \mathbb{N}$, where $|A_t|$ denotes the size of the program A_t.*

[22] We use $\{0,1\}^{\leq c_L \cdot t}$ to denote the set of Boolean strings with length at most $c_L \cdot t$ for convenience.

Similarly, for an ensemble of promise problems $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$*, we say that* Π has an ensemble of algorithms *if for all* $t \in \mathbb{N}$*,* A_t *is a program such that for all* $x \in \{0,1\}^{\leq c_\Pi \cdot t}$*, we have*

$$(x, 1^t) \in \Pi_{\mathsf{YES}} \implies A_t(x) = 1,$$
$$(x, 1^t) \in \Pi_{\mathsf{NO}} \implies A_t(x) = 0.$$

We define size analogously as above.

Remark 5.4. *The reason we do not use the standard notion of advice is that we will need the "advice" to be indexed by the parameter* $t \in \mathbb{N}$ *and not by the input length, since the input length could correspond to multiple values of* $t \in \mathbb{N}$*.*

The main result of this section is the following.

Theorem 5.5 (Algorithmic language compression)**.** *Let* A *be an oracle and* $L = \{L_t\}_{t \in \mathbb{N}} \in \mathsf{NTIME}[n]^A$ *be an ensemble of languages. Assuming that* $\mathsf{coNTIME}[n]^A \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ *for a good resource function* $\beta \colon \mathbb{N} \to \mathbb{N}$*, for all constant* $\varepsilon > 0$*, there is a function* $\tau_n(t) = \beta^{(4)}(t \cdot \mathrm{poly}(n))$ *and a constant* c *depending only on* β *such that the parameterized promise problem* $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$ *defined as*

$$\Pi_{\mathsf{YES}} \coloneqq \{(x, 1^t) \mid x \in L_t,\ \varepsilon \log t \leq |x| \leq t\},$$
$$\Pi_{\mathsf{NO}} \coloneqq \{(x, 1^t) \mid \mathsf{K}^{\tau_n(t)}(x) > \log |L_t| + c \log t,\ \varepsilon \log t \leq |x| \leq t\},$$

has an ensemble of (deterministic) algorithms of size $O(\log t)$ *running in time* $\tau_n(t)$*, where* $n = |x|$*.*

Before proving Theorem 5.5, we first show we can use it to give a fast algorithm for $\mathsf{GapK}^{t,A}$ under the same assumptions.

Theorem 5.6 (Fast Algorithm for $\mathsf{GapK}^{t,A}$)**.** *Assuming that* $\mathsf{coNTIME}[n]^A \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ *for a good resource function* $\beta \colon \mathbb{N} \to \mathbb{N}$*, there is a function* $\tau_n(t) = \beta^{(4)}(t \cdot \mathrm{poly}(n))$ *and a constant* c *depending only on* β *such that the parameterized promise problem* $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$ *defined as*

$$\Pi_{\mathsf{YES}} = \{(x, 1^{\langle t,s \rangle}) \mid \mathsf{K}^{t,A}(x) \leq s,\ \log t \leq |x| \leq t\},$$
$$\Pi_{\mathsf{NO}} = \{(x, 1^{\langle t,s \rangle}) \mid \mathsf{K}^{\tau_n(t)}(x) > s + c \log t,\ \log t \leq |x| \leq t\},$$

has an ensemble of (deterministic) algorithms of size $O(\log t)$ *running in* $\tau_n(t)$ *time, where* $n = |x|$ *and* $s \leq O(n)$*.*

Proof. Consider the ensemble $L = \{L_{\langle t,s \rangle}\}_{t,s \in \mathbb{N}}$ given by

$$L_{\langle t,s \rangle} = \{x \in \{0,1\}^* \mid \mathsf{K}^{t,A}(x) \leq s\}.$$

From basic properties of Kolmogorov complexity, we know $|L_{\langle t,s \rangle}| \leq 2^{s+1}$. We can now apply Theorem 5.5 (with running-time bound labeled as $\tau'_n(\langle t,s \rangle)$, constant $\varepsilon > 0$ chosen later, and constant labeled as c') to get an ensemble of algorithms of size $O(\log t)$ for $\Pi' = (\Pi'_{\mathsf{YES}}, \Pi'_{\mathsf{NO}})$ given by

$$\Pi'_{\mathsf{YES}} = \{(x, 1^{\langle t,s \rangle}) \mid \mathsf{K}^{t,A}(x) \leq s,\ \varepsilon \log(\langle t,s \rangle) \leq |x| \leq \langle t,s \rangle\},$$
$$\Pi'_{\mathsf{NO}} = \{(x, 1^{\langle t,s \rangle}) \mid \mathsf{K}^{\tau'_n(\langle t,s \rangle)}(x) > s + 1 + c' \log t,\ \varepsilon \log(\langle t,s \rangle) \leq |x| \leq \langle t,s \rangle\}.$$

Since $s \leq O(n)$, the running time is at most

$$\tau'_n(\langle t,s \rangle) = \beta^{(4)}(\langle t,s \rangle \cdot \mathrm{poly}(n)) \leq \beta^{(4)}(t \cdot \mathrm{poly}(n)) =: \tau_n(t).$$

Now, we claim that for ε sufficiently small, this ensemble of algorithms also solves the following promise problem:

$$\Pi_{\mathsf{YES}} = \{(x, 1^{\langle t,s\rangle}) \mid \mathsf{K}^{t,A}(x) \le s,\ \log t \le |x| \le t\},$$
$$\Pi_{\mathsf{NO}} = \{(x, 1^{\langle t,s\rangle}) \mid \mathsf{K}^{\tau_n(t)}(x) > s + 1 + c' \log t,\ \log t \le |x| \le t\}.$$

Since $s \le O(n)$, we know

$$\varepsilon \log(\langle t,s\rangle) \le \varepsilon\,(\log t + 2\log s + O(1)) \le \varepsilon(\log t + 2\log n + O(1)),$$

so if $n \le t$, choosing ε to be a sufficiently small universal constant implies

$$\varepsilon \log(\langle t,s\rangle) \le \log t$$

for sufficiently large t. Lastly, we know $t \le \langle t,s\rangle$ by the properties of our natural number encoding scheme.

Putting together the facts that $\tau_n(t) \ge \tau'_n(\langle t,s\rangle)$, $\varepsilon \log(\langle t,s\rangle) \le \log t$, and $t \le \langle t,s\rangle$, we see that $\Pi_{\mathsf{YES}} \subseteq \Pi'_{\mathsf{YES}}$ and $\Pi_{\mathsf{NO}} \subseteq \Pi'_{\mathsf{NO}}$. Therefore, the ensemble of algorithms from Theorem 5.5 with some $\varepsilon < 1$ now solves the promise problem Π as given in the theorem statement, where the constant c is a slight tweak to c'. □

We will use $\mathsf{DP}_{n,k}\colon \{0,1\}^n \times \{0,1\}^{nk} \to \{0,1\}^{nk+k}$ to denote the direct product generator as in [Hir21]. Explicitly, it is given by

$$\mathsf{DP}_{n,k}(x; z^1, z^2, \dots, z^k) = (z^1, \dots, z^k, \mathsf{Had}(x)(z^1), \dots, \mathsf{Had}(x)(z^k)),$$

where $z^i \in \{0,1\}^n$ for each $i \in [k]$, and $\mathsf{Had}(x,z) = (\sum_{j\in[n]} x_j z_j) \mod 2$.

We first need the following improved reconstruction for $\mathsf{DP}_{n,k}$.

Theorem 5.7. *Assuming that* $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ *for a good resource function* $\beta\colon \mathbb{N} \to \mathbb{N}$*, given a* t*-time algorithm* D *with* m*-bit randomness and* ℓ*-bit advice such that* D ε*-distinguishes between* U_{nk+k} *and* $\mathsf{DP}_k(x; U_{nk})$*, and assuming* $\ell \le \mathrm{poly}(m,n,1/\varepsilon)$*,* $\log t \le \mathrm{poly}(n,m,1/\varepsilon)$*,* $n + m + 1/\varepsilon \le t$*, and* $k \le O(n)$*, there is an algorithm running in time* $\beta^{(3)}(t \cdot \mathrm{poly}(n,m,1/\varepsilon))$ *with* $O(\log t)$ *bits of advice that takes as input* $(x, 1^{\langle m,k,1/\varepsilon\rangle})$ *and* D*, along with the advice* $\alpha_D \in \{0,1\}^\ell$ *for* D*, and outputs a program of size at most* $k + \ell + O(\log t)$ *that prints* x *in time* $\beta^{(3)}(t \cdot \mathrm{poly}(n,m,1/\varepsilon))$*. In particular,*

$$\mathsf{K}^{\beta^{(3)}(t\cdot\mathrm{poly}(n,m,1/\varepsilon))}(x) \le k + \ell + O(\log t),$$

where the constants hidden in the $\mathrm{poly}(\cdot)$ *and* $O(\cdot)$ *depend only on* β*.*

To prove Theorem 5.7, we first recall the following randomized reconstruction algorithm for DP generator as in [Hir21].

Lemma 5.8 (A slight variant of [Hir21, Lemma 3.14])**.** *For any parameters* $n, k \in \mathbb{N}$*, and* $\varepsilon \in (0,1)$ *with* $k \le O(n)$*, there exists a pair of algorithms* A *and* Rcon*, such that*

- Rcon^D *and* A^D *take oracle access to a function* $D\colon \{0,1\}^{nk+k} \to \{0,1\}$*.*
- $A^D\colon \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^{n_{\mathsf{adv}}}$ *is called an advice function and is computable in time* $\mathrm{poly}(n/\varepsilon)$*, where* $n_{\mathsf{adv}} := k + O(\log(n/\varepsilon))$*.*
- $\mathsf{Rcon}^D\colon \{0,1\}^{n_{\mathsf{adv}}} \times \{0,1\}^r \to \{0,1\}^n$ *is called a reconstruction procedure and is computable in time* $\mathrm{poly}(n/\varepsilon)$*.*

- *The randomness complexity r is at most* $\mathrm{poly}(n/\varepsilon)$.
- *For any string $x \in \{0,1\}^n$ and any function D that ε-distinguishes the output distribution of $\mathsf{DP}_k(x; U_{nk})$ from U_{nk+k}, it holds that*

$$\Pr_{w \sim U_r}\left[\mathsf{Rcon}^D(A^D(x,w),w) = x\right] \geq 1 - 1/\mathrm{poly}(n/\varepsilon).$$

Remark 5.9. *The main differences between Lemma 5.8 and Lemma 3.14 of [Hir21] are as follows. First, we repeat the local list-decoding algorithm with oracle access to D as described in Lemma 3.14 of [Hir21] many times, so that with probability at least $1 - 1/\mathrm{poly}(n/\varepsilon)$, a list of length $\mathrm{poly}(n/\varepsilon)$ containing x can be reconstructed in $\mathrm{poly}(n/\varepsilon)$ time. Second, for a random seed $s \sim U_{\mathrm{poly}(n/\varepsilon)}$ (included as part of the randomness $w \in \{0,1\}^r$), we store $H(x,s)$ as part of the advice $A(x,w)$ for a polynomial-time computable pairwise-independent hash function $H : \{0,1\}^n \times \{0,1\}^{\mathrm{poly}(n/\varepsilon)} \to \{0,1\}^{O(\log(n/\varepsilon))}$, as this allows us to uniquely recover x from this list with probability at least $1 - 1/\mathrm{poly}(n/\varepsilon)$.*

Proof of Theorem 5.7. For randomness $r_D \in \{0,1\}^m$ for the distinguisher D, we let $D(\cdot; r_D)$ denote the deterministic function when fixing the randomness of D to r_D. Since D ε-distinguishes $\mathsf{DP}_k(x; U_{nk})$ from U_{nk+k}, we know that

$$\left| \mathop{\mathbb{E}}_{\substack{r_D \sim U_m \\ z \sim U_{nk}}} [D(\mathsf{DP}_k(x;z); r_D)] - \mathop{\mathbb{E}}_{\substack{r_D \sim U_m \\ b \sim U_{nk+k}}} [D(b; r_D)] \right| \geq \varepsilon.$$

By an averaging argument,

$$\Pr_{r_D \sim U_m}\left[\left| \mathop{\mathbb{E}}_{z \sim U_{nk}} [D(\mathsf{DP}_k(x;z); r_D)] - \mathop{\mathbb{E}}_{b \sim U_{nk+k}} [D(b; r_D)] \right| \geq \frac{\varepsilon}{2}\right] \geq \frac{\varepsilon}{2}.$$

Therefore, with probability at least $\varepsilon/2$ over $r_D \sim U_m$, the deterministic function $D(\cdot; r_D)$ is an $\varepsilon/2$ distinguisher for $\mathsf{DP}_k(x; U_{nk})$ and U_{nk+k}. Therefore, when taking $u = O(1/\varepsilon)$ independent samples $r_D^{(j)} \sim U_m$ for $j \in [u]$, with probability $\geq 2/3$, there exists some $j^* \in [u]$ such that $D(\cdot; r_D^{(j^*)})$ is an $\varepsilon/2$-distinguisher.

We let $r = \mathrm{poly}(n/\varepsilon)$ be the randomness complexity of the reconstruction procedure (with distinguishing advantage $\varepsilon/2$), and we let parameter $\bar{r} = \mathrm{poly}(n, m, 1/\varepsilon)$ be a sufficiently large parameter chosen later, in particular so that $\bar{r} \geq r + um$. For $w \in \{0,1\}^{\bar{r}}$, we can consider the substring $w_{>r} \in \{0,1\}^{\bar{r}-r}$ and divide its prefix of length um into u groups of m to naturally give $r_D^{(1)} \circ \cdots \circ r_D^{(u)} \in \{0,1\}^{um}$. Now, by applying Lemma 5.8, we have

$$\Pr_{w \sim U_{\bar{r}}}\left[\exists j \in [u] \text{ s.t. } \mathsf{Rcon}^{D\left(\cdot; r_D^{(j)}\right)}\left(A^{D\left(\cdot; r_D^{(j)}\right)}(x, w_{\leq r}), w_{\leq r}\right) = x\right] \geq 1/2.$$

We can define a function $f(w) := \left[\exists j \in [u] \text{ s.t. } \mathsf{Rcon}^{D\left(\cdot; r_D^{(j)}\right)}\left(A^{D\left(\cdot; r_D^{(j)}\right)}(x, w_{\leq r}), w_{\leq r}\right) = x\right]$. One can see that f is computable in $t' = \mathrm{poly}(n, m, 1/\varepsilon) \cdot t$ time given x and advice $\alpha_D \in \{0,1\}^\ell$ for the distinguisher D.

We now invoke Theorem 4.1 with time parameter t' and input size $\bar{r}$, chosen such that $\bar{r} \geq \ell + n$, as the function f can be computed on input $w \in \{0,1\}^{\bar{r}}$ with advice $\alpha_D \in \{0,1\}^\ell$ and $x \in \{0,1\}^n$. To satisfy the preconditions of Theorem 4.1, we need $\log(t') \leq \bar{r} \leq t'$. First, notice that $\log(t') = \log t + \log(\mathrm{poly}(n, m, 1/\varepsilon))$, and since $t \geq n + m + 1/\varepsilon$, it follows that $\log(\mathrm{poly}(n, m, 1/\varepsilon)) \leq$

$O(\log t)$. Therefore, $\log(t') \le O(\log t)$, and so $\log(t') \le O(\log t) \le \mathrm{poly}(n, m, 1/\varepsilon) \le \bar{r}$ for sufficiently large $\bar{r}$. Second, since we can choose the bound t' to have a sufficiently large $\mathrm{poly}(n, m, 1/\varepsilon)$ factor, we can make $\bar{r} \le t'$ to satisfy the preconditions of Theorem 4.1.

Now applying Theorem 4.1, we can apply $G_{t',\bar{r}}$ with advice $\alpha \in \{0,1\}^{O(\log t')}$ to derandomize the function $f(w)$. That is, we have

$$\Pr_{s \sim U_{O(\log \bar{r})};\ w = G_{t',\bar{r}}(s)_{/\alpha}} \left[\exists j \in [u] \text{ s.t. } \mathsf{Rcon}^{D\left(\cdot; r_D^{(j)}\right)} \left(A^{D\left(\cdot; r_D^{(j)}\right)}(x, w_{\le r}), w_{\le r} \right) = x \right] \ge 1/3.$$

Therefore, our algorithm can take $\alpha \in \{0,1\}^{O(\log t')}$ as advice (so $O(\log t') = O(\log t)$ bits of advice, as desired) and brute force search over all seeds $s \in \{0,1\}^{O(\log \bar{r})}$ until it finds one such that $f(G_{t',\bar{r}}(s)_{/\alpha}) = 1$, using x, D, and α_D. (If no such seed is found, we reject and output 0, which will never happen for correct advice α_D.) Since $f(G_{t',\bar{r}}(s)_{/\alpha}) = 1$, we know there exists $j^* \in [u]$ such that

$$\mathsf{Rcon}^{D\left(\cdot; r_D^{(j^*)}\right)} \left(A^{D\left(\cdot; r_D^{(j^*)}\right)}(x, w_{\le r}), w_{\le r} \right) = x.$$

All that is needed to specify the program that outputs x is now $\mathsf{bin}(j^*) \in \{0,1\}^{\log(1/\varepsilon)+O(1)}$, α_D, α, s, and $A(x, w_{\le r}) \in \{0,1\}^{k+O(\log(n/\varepsilon))}$, and an $O(1)$-size program, as f can be deterministically computed given these strings via the reconstruction algorithm Rcon with $w = G_{t',\bar{r}}(s)_{/\alpha}$ and randomness $r_D^{(j^*)}$ for the distinguisher D. The total length of this program is

$$|\mathsf{bin}(j^*)| + |\alpha_D| + |\alpha| + |s| + |A(x, w_{\le r})| + O(1) \le k + \ell + O(\log t).$$

Moreover, the time needed to output this program is

$$2^{O(\log \bar{r})} \cdot (\beta^{(3)}(t' \cdot \mathrm{poly}(\bar{r})) + t') \le \beta^{(3)}(t \cdot \mathrm{poly}(n, m, 1/\varepsilon)),$$

since we enumerate over all seeds $s \in \{0,1\}^{O(\log \bar{r})}$, and for each seed s we compute $w = G_{t',\bar{r}}(s)_{/\alpha}$ and then compute $f(w)$ by brute forcing over all $j \in [u]$ to ensure we have found a valid index j^* and a valid seed s and with corresponding advice $A(x, w_{\le r})$.

The program itself first computes $w = G_{t',\bar{r}}(s)_{/\alpha}$ in $\beta^{(3)}(t' \cdot \mathrm{poly}(\bar{r}))$ time and then computes x in $t' = \mathrm{poly}(n, m, 1/\varepsilon) \cdot t$ time given w, j^*, α_D, and $A(x, w_{\le r})$ by the algorithms Rcon and D. As the running time of computing w dominates the computation of x given w and the full advice, it follows that the running time of the program itself is at most $\beta^{(3)}(t \cdot \mathrm{poly}(n, m, 1/\varepsilon))$, as desired. □

We are now ready to present a proof of Theorem 5.5.

Proof of Theorem 5.5. Let

$$L' = \{(\mathsf{DP}_{n,k}(x; z), 1^{\langle t,n \rangle}) \mid x \in L_t \cap \{0,1\}^n, z \in \{0,1\}^{nk},\ \varepsilon \log t \le |x| \le t\}$$

for a parameter $k = k(t, n) = \min(n, \lfloor \log |L_t| \rfloor) + d \log t \le n + d \log t$, where d is a constant to be chosen later. Note that by construction, we also have

$$\log(|L_t \cap \{0,1\}^n|) \le \min(n, \log |L_t|) \le k + 1 - d \log t. \tag{4}$$

Since $L \in \mathsf{NTIME}[n]^A$, it follows that we can nondeterministically compute L' as follows. Let V^A be a verifier for L, so that

$$(x, 1^t) \in L \iff \exists y_{x,t} \in \{0,1\}^{O(n+t)} \text{ such that } V^A(x, 1^t, y_{x,t}) = 1,$$

and V^A runs in linear time in its input, $O(n+t)$. We can construct a verifier W^A for L' as follows. On a witness $y' = (x, y_{x,t}, z)$, where $z \in \{0,1\}^{nk}$, we define

$$W^A(w, 1^{\langle t,n\rangle}, y') := V^A(x, 1^t, y_{x,t}) \wedge (\mathsf{DP}_{n,k}(x;z) = w) \wedge (\varepsilon \log t \le |x| \le t).$$

It is clear by the construction of L' and W^A that W^A is a verifier for L'. The total witness length is $|y'| = n + O(n+t) + nk \le t \cdot \mathrm{poly}(n)$. Since V^A has running time $O(n+t)$, $\mathsf{DP}_{n,k}$ has running time $\mathrm{poly}(n,k)$, and decoding $\langle t,n\rangle$ takes time $\mathrm{poly}(\log t, \log n)$, the running time for W^A is bounded by $O(n+t) + \mathrm{poly}(n,k) + \mathrm{poly}(\log t, \log n) \le t \cdot \mathrm{poly}(n)$. Therefore, L' admits a nondeterministic algorithm with running time (and witness length) $t \cdot \mathrm{poly}(n)$ with oracle access to A.

Applying our hypothesis, we know $(\neg L', \mathcal{U}^{\mathsf{para}}) \in \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$. In particular, negating the resulting algorithm gives us a one-sided-error heuristic algorithm B for L' running in time $\beta(nk+k+1+\langle t,n\rangle) \le \beta(t \cdot \mathrm{poly}(n))$ with the following properties:

1. For all $(w, 1^{\langle t,n\rangle}) \in L'$, it holds that $B(w, 1^{\langle t,n\rangle}) = 1$.
2. $\Pr_{w \sim U_{nk+k}}[B(w, 1^{\langle t,n\rangle}) \neq L'(w, 1^{\langle t,n\rangle})] \le 1/2$.

Moreover, the number of YES instances of L' is small. By a union bound and (4), we have

$$\begin{aligned}
\Pr_{w \sim U_{nk+k}}[(w, 1^{\langle t,n\rangle}) \in L'] &= \Pr_{w \sim U_{nk+k}}[\exists x \in L_t \cap \{0,1\}^n, \exists z \in \{0,1\}^{nk} : w = \mathsf{DP}_{n,k}(x;z)] \\
&\le |L_t \cap \{0,1\}^n| \cdot 2^{nk} \cdot 2^{-(nk+k)} \\
&\le 2^{k+1-d\log t} \cdot 2^{-k} \\
&= 2t^{-d}.
\end{aligned}$$

We can now give an ensemble of randomized algorithms $\{E_t\}_{t \in \mathbb{N}}$ of size $\log t + O(1)$ that solves the desired promise problem $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$. Since $L = \{L_t\}_{t \in \mathbb{N}}$ is an ensemble of languages, we know $|L_t| \le 2^{O(t)}$, and thus $\log |L_t| \le O(t)$, making $\alpha_t := \mathsf{bin}(\lfloor \log |L_t| \rfloor)$ have length at most $\log t + O(1)$. We hardcode α_t into each algorithm E_t. On input $x \in \{0,1\}^n$, we define $E_t(x)$ as follows. First, $E_t(x)$ checks whether $\varepsilon \log t \le |x| \le t$ (and rejects if not). Then, the program computes $k(t,n) = \min(n, \alpha_t) + d \log t = O(n)$, samples $z \sim U_{nk}$ uniformly, and outputs $B(\mathsf{DP}_{n,k}(x;z), 1^{\langle t,n\rangle})$. That is, we define E_t such that $E_t(x;z) = B(\mathsf{DP}_{n,k}(x;z), 1^{\langle t,n\rangle})$ as long as $\varepsilon \log t \le |x| \le t$, where $k = \min(n,\alpha) + d \log t$. The running time of E_t will be dominated by the running time of B, so we can bound the running time of E_t by $t' = \beta(t \cdot \mathrm{poly}(n))$.

We now prove correctness:

Claim 5.10. *For all large $t \in \mathbb{N}$, the following hold for all $x \in \{0,1\}^n$:*

1. *If $(x, 1^t) \in \Pi_{\mathsf{YES}}$, then $\Pr_{z \sim U_{nk}}[E_t(x;z) = 1] = 1$.*
2. *If $(x, 1^t) \in \Pi_{\mathsf{NO}}$, then $\Pr_{z \sim U_{nk}}[E_t(x;z) = 1] < 3/4$.*

Proof. For Item 1, if $(x, 1^t) \in \Pi_{\mathsf{YES}}$, then $x \in L_t$ and $\varepsilon \log t \le |x| \le t$. Since B does not err on YES instances, for all $z \in \{0,1\}^{nk+k}$, we know $E_t(x;z) = B(\mathsf{DP}_{n,k}(x;z), 1^{\langle t,n\rangle}) = 1$, as desired.

For Item 2, we will show the contrapositive. Suppose $\Pr_z[E_t(x;z) = 1] \ge 3/4$. In other words,

$$\Pr_{z \sim U_{nk+k}}[B(\mathsf{DP}_{n,k}(x;z), 1^{\langle t,n\rangle}) = 1] \ge 3/4. \tag{5}$$

Since L' does not have many YES instances, we also know that

$$\begin{aligned}\Pr_w[B(w,1^{\langle t,n\rangle})=1] &= \Pr_w[L'(w,1^{\langle t,n\rangle})=1] + \Pr_w[B(w,1^{\langle t,n\rangle}) \neq L'(w,1^{\langle t,n\rangle})] \\ &\leq 2t^{-d} + \frac{1}{2} \\ &\leq \frac{2}{3}, \end{aligned} \tag{6}$$

for sufficiently large t, d. Combining (5) and (6), we get a deterministic $1/12$-distinguisher using $\log t + O(1)$ bits of advice between $\mathsf{DP}_{n,k}(x; U_{nk})$ and U_{nk+k}, running in time at most t'.

We now apply Theorem 5.7. To do so, we need to verify the hypotheses of Theorem 5.7 for the case when with $1/12$ distinguishing advantage and $m = 0$, as our distinguisher is deterministic. First, note that $\ell = \log t + O(1)$. Since $(x, 1^t) \in \Pi_{\mathsf{NO}}$, we have $\varepsilon \log t \leq |x| \leq t$, so $\ell = \log t + O(1) \leq O(n) \leq \mathrm{poly}(n)$, as needed. Next, we have $\log(t') = O(\log t + \log n) = O(n) = \mathrm{poly}(n)$, as needed. Next, we have $n + 1/(1/12) = n + 12 \leq \mathrm{poly}(n) \leq \beta(t \cdot \mathrm{poly}(n)) = t'$, as we can choose the polynomial $\mathrm{poly}(n)$ in t' to be sufficiently large. Lastly, $k \leq n + O(\log t) \leq O(n)$, as desired.

Thus, we can apply Theorem 5.7 to get that

$$\mathsf{K}^{\beta^{(3)}(t' \cdot \mathrm{poly}(n))}(x) \leq k + \log t + O(1) + O(\log t') \leq \log |L_t| + O(\log t).$$

This implies that $(x, 1^t) \notin \Pi_{\mathsf{NO}}$ for an appropriately chosen constant c (depending only on β), completing the contraposition, as $\beta^{(3)}(t' \cdot \mathrm{poly}(n)) \leq \beta^{(4)}(t \cdot \mathrm{poly}(n))$. □

The above claim shows that Π has an ensemble of one-sided error algorithms of size $\log t + O(1)$ running in time t'. We can then use the PRG from Theorem 4.1, setting $m = nk$ and the time parameter to t'. Observe that m is also a bound on the advice, as for sufficiently large t, d and n, we have $\log t + O(1) \leq d \log t \leq k < nk = m$. We also have $\log(t') \leq O(\log t + \log n) \leq O(\log t) \leq n \log t \leq n \cdot k = m$, as needed to apply Theorem 4.1. Lastly, we need $m \leq t'$, which holds for sufficiently large $t' = \beta(t \cdot \mathrm{poly}(n))$ (that is, the $\mathrm{poly}(n)$ can be chosen to be large enough to make it hold).

Thus, by applying the PRG in Theorem 4.1, we can derandomize E_t by enumerating over all seeds in time

$$2^{O(\log(nk))} \cdot \beta^{(3)}(t' \cdot \mathrm{poly}(nk)) = \mathrm{poly}(nk) \cdot \beta^{(3)}(t' \cdot \mathrm{poly}(n)) \leq \beta^{(4)}(t \cdot \mathrm{poly}(n))$$

using $O(\log t') = O(\log t)$ bits of advice, which can be further hardcoded into each program E_t. Thus, Π has an ensemble of deterministic algorithms of size $O(\log t)$ running in time $\beta^{(4)}(t \cdot \mathrm{poly}(n))$. Setting $\tau_n(t) = \beta^{(4)}(t \cdot \mathrm{poly}(n))$ for a sufficiently large polynomial $\mathrm{poly}(n)$ (constants depending only on β), we get the desired result. □

6 Fine-grained Weak Symmetry of Information

In this section, we prove the following theorem.

Theorem 6.1. *If* $\mathsf{coNTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ *for a good resource function* $\beta\colon \mathbb{N} \to \mathbb{N}$, *there exists a constant* c *depending only on* β *such that for every sufficiently large* $n, m, t \in \mathbb{N}$, $\varepsilon \in (0,1)$ *with* $\log t \leq n/2$, *for every* $x \in \{0,1\}^n$, *it holds that*

$$\Pr_{w \sim U_m}\left[\mathsf{K}^t(xw) \geq \mathsf{K}^{\beta^{(7)}(t \cdot \mathrm{poly}(nm/\varepsilon))}(x) + m - c\log(t/\varepsilon)\right] \geq 1 - \varepsilon.$$

Proof. Let c' be the constant in Theorem 5.6 and let k be some parameter to be chosen later which will satisfy $k \leq n$. We define the parameters $s = nk + k + m - c' \log t - 2\log(1/\varepsilon)$ and $t' = t + p(n,m)$ for some $p(n,m) = \mathrm{poly}(n,m)$ specified later. Let $\{A_r\}_{r\in\mathbb{N}}$ be the ensemble of algorithms in Theorem 5.6. We fix our attention on the algorithm $A = A_{\langle t',s\rangle}$ and consider inputs of size $n' = nk + k + m$. Note that $s \leq n'$ as needed for Theorem 5.6. We also have

$$\begin{aligned}\log(\langle t',s\rangle) \leq O(1) + \log(t') + 2\log s &\leq O(1) + \log t + \log(p(n,m)) + 2\log(n') \\ &\leq O(1) + n/2 + O(\log n) + O(\log m) + 2\log(n') \\ &\leq 2n/3 + n'/3 \\ &\leq n',\end{aligned}$$

for sufficiently large n and m, as well as $n' \leq t' \leq \langle t',s\rangle$ for sufficiently large $p(n,m)$, so the condition $\log(\langle t',s\rangle) \leq n \leq \langle t',s\rangle$ in the definition of $\Pi = (\Pi_{\mathsf{YES}}, \Pi_{\mathsf{NO}})$ in Theorem 5.6 is satisfied. Letting $\tau_{n'}(t')$ be the runtime bound in Theorem 5.6, A has running time $\tau_{nk+k+m}(t') = \beta^{(4)}(t' \cdot \mathrm{poly}(nk+k+m)) = \beta^{(4)}(t \cdot \mathrm{poly}(n,m))$ and has size $O(\log t') = O(\log t)$.

Note that with probability at least $1 - \varepsilon/10$, a random string x from $\{0,1\}^{nk+k+m}$ satisfies $\mathsf{K}(x) \geq nk + k + m - 2\log(1/\varepsilon) = s + c' \log t$. Hence, A rejects U_{nk+k+m} with probability at least $1 - \varepsilon/10$.

On the other hand, consider running A on $\mathsf{DP}_k(x; U_{nk}) \circ U_m$. If it rejects with probability less than $1 - \varepsilon/2$, this gives an $\varepsilon/3$-distinguisher between $\mathsf{DP}_k(x; U_{nk})$ and U_{nk+k}, computable by a randomized $\tilde{t} = \beta^{(4)}(t \cdot \mathrm{poly}(n,m))$-time algorithm with m bits of randomness and $O(\log t)$ bits of advice. We now apply Theorem 5.7, but we first verify that all hypotheses of the theorem are met. First, observe $\ell = O(\log t) \leq O(n) = \mathrm{poly}(n, m, 1/\varepsilon)$, as needed. Next, observe that $\log(\tilde{t}) \leq O(\log t + \log(nm)) \leq \mathrm{poly}(n, m, 1/\varepsilon)$. By choosing $\tilde{t}$ to be $\beta^{(4)}(t \cdot \mathrm{poly}(n, m, 1/\varepsilon))$ for a sufficiently large $\mathrm{poly}(n, m, 1/\varepsilon)$ without loss of generality, we have $\tilde{t} \geq n + m + 1/\varepsilon$. Lastly, $k \leq n = O(n)$, as desired.

Now applying Theorem 5.7, by setting

$$\bar{t} = \beta^{(3)}(\tilde{t} \cdot \mathrm{poly}(nm/\varepsilon)) \leq \beta^{(7)}(t \cdot \mathrm{poly}(nm/\varepsilon)),$$

we have

$$\mathsf{K}^{\bar{t}}(x) \leq k + O(\log(\beta^{(4)}(t \cdot \mathrm{poly}(n,m)))) + O(\log t) \leq k + c'' \log t,$$

for some constant c'' depending only on β (since we will set $k \leq n$).

Now we further set k so that $k + c'' \log t = \mathsf{K}^{\bar{t}}(x) - 1$. That is, we set $k = \mathsf{K}^{\bar{t}}(x) - 1 - c'' \log t \leq n$. The above is now a contradiction by construction, so A rejects $(\mathsf{DP}_k(x; U_{nk}), U_m)$ with probability at least $1 - \varepsilon/2$. This means that

$$\Pr_{z\sim U_{nk},\, w\sim U_m}[\mathsf{K}^{t'}(\mathsf{DP}_k(x;z), w) > s] \geq 1 - \varepsilon/2.$$

By an averaging argument, there exists a $z \in \{0,1\}^{nk}$ such that

$$\Pr_{w\sim U_m}[\mathsf{K}^{t'}(\mathsf{DP}_k(x;z), w) > s] \geq 1 - \varepsilon/2. \tag{7}$$

Note that we also have

$$\mathsf{K}^{t'}(\mathsf{DP}_k(x;z), w) \leq \mathsf{K}^{t}(xw) + nk + O(1), \tag{8}$$

as given x, w in time t, if given advice $z \in \{0,1\}^{nk}$, one can compute $(\mathsf{DP}_k(x;z), w)$ with an $O(1)$-sized program in additional time $t' - t = p(n,m)$. Combining inequalities (7) and (8), with probability at least $1 - \varepsilon/2$ over w, we have

$$\begin{aligned}\mathsf{K}^t(xw) > s - nk - O(1) &= k + m - c' \log t - 2\log(1/\varepsilon) - O(1) \\ &= \mathsf{K}^{\bar{t}}(x) - c'' \log t + m - c' \log t - 2\log(1/\varepsilon) - O(1) \\ &= \mathsf{K}^{\bar{t}}(x) + m - c \log(t/\varepsilon),\end{aligned}$$

for some appropriately chosen constant c depending only on β. □

7 New Worst-case to Average-case Reduction for PH

In this section, we study worst-case to average-case reductions for various subclasses of PH. In particular, we prove Item (2) of Theorem 1.3.

We need the following HSG construction in [Hir20a].

Theorem 7.1 ([Hir20a, Theorem 4.3]). *For any sufficiently large $n, m \in \mathbb{N}$ such that $m \leq 2n$, there exists a function $H_{n,m} \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ and a deterministic reconstruction procedure $R^{(-)} \colon \{0,1\}^a \to \{0,1\}^n$ where $d = O(\log n + \log^3 m)$ and $a = 2m + O(\log n + \log^3 m)$, such that, for any $x \in \{0,1\}^n$ and any function $D \colon \{0,1\}^m \to \{0,1\}$ that 0.1-avoids $H_{n,m}(x,-)$, there exists an advice string $\alpha \in \{0,1\}^a$ such that $R^D(\alpha) = x$. Moreover, H can be computed in time $\mathrm{poly}(n)$ and R^D can be computed in time $\mathrm{poly}(n)$ with oracle access to D.*

The following lemma is crucial for the results of this section.

Lemma 7.2. *Let $T \colon \mathbb{N} \to \mathbb{N}$ be such that $T(n) \leq 2^{o(n)}$ and $L \in \mathsf{NTIME}[T(n)]$. Assume that $\Pi_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ for a good resource function $\beta \colon \mathbb{N} \to \mathbb{N}$. There exists a function $\tau_n(t) = \beta^{(O(1))}(t \cdot \mathrm{poly}(n))$ and a nondeterministic algorithm S_L such that the followings hold.*

1. *S_L takes an input $x \in \{0,1\}^n$ to L, an integer $t \in [T(n)^{c_{\mathrm{t}}}, 2^{\varepsilon_{\mathrm{t}} n}]$ and an integer $w \leq n$, where $c_{\mathrm{t}} \geq 1$ is universal constant and $\varepsilon_{\mathrm{t}} \in (0,1)$ is a constant that only depends on β.*
2. *S_L then guesses at most $2w + O(\log^3 n + \log t)$ bits of witness and runs in $\mathrm{poly}(T(n)) \cdot \tau_n(t)$ time.*
3. *If $L(x) = 0$, S_L rejects on all possible witnesses.*
4. *If $L(x) = 1$ and $w \geq \mathsf{cd}^{t,\tau_n(t)}(x)$, S_L accepts on at least one witness.*

Proof.

Set up. Let $L \in \mathsf{NTIME}[T(n)]$. We fix an input $x \in \{0,1\}^n$ to L. From the definition of L, there is a linear-time algorithm V taking two inputs $x \in \{0,1\}^n$ and $y \in \{0,1\}^{T(n)}$, such that $L(x) = 1$ if and only if there exists $y \in \{0,1\}^{T(n)}$ satisfying $V(x,y) = 1$.

If $L(x) = 1$, we let $y_x \in \{0,1\}^{T(n)}$ be the lexicographically first string y satisfying $V(x,y) = 1$. Otherwise $L(x) = 0$, and we simply let $y_x = 0^{T(n)}$. We also set c_{t} to be a large enough universal constant.

Algorithm $A^{\mathsf{Gap\text{-}K^t}}$. Let L_{comp} be a complete language for $\mathsf{NTIME}[n]$. From Theorem 5.6, there is a function $q_n(t) = \beta^{(4)}(t \cdot \mathrm{poly}(n))$ such that for every $t, s \in \mathbb{N}$ with $s \leq O(n)$, there is an $O(\log t)$-size program $A^{\mathsf{Gap\text{-}K^t}}_{\langle t,s\rangle}$ that satisfies the following:

- $A^{\mathsf{Gap\text{-}K^t}}_{\langle t,s\rangle}$ takes a string $x \in \{0,1\}^n$ with $\log t \leq n \leq t$, and runs in $q_n(t)$ time.
- (Yes case) If $\mathsf{K}^{t,L_{\mathsf{comp}}}(x) \leq s$, then $A^{\mathsf{Gap\text{-}K^t}}_{\langle t,s\rangle}(x) = 1$.
- (No case) If $\mathsf{K}^{q_n(t)}(x) \geq s + c_1 \log t$, then $A^{\mathsf{Gap\text{-}K^t}}_{\langle t,s\rangle}(x) = 0$, where c_1 is a constant that only depends on β.

Constructing the distinguisher from $A^{\mathsf{Gap\text{-}K^t}}$. Let $m \leq O(n)$ be a parameter to be specified later, and $d = O(\log T(n) + \log^3 m)$ be the seed length of $H_{T(n),m}$ from Theorem 7.1. From now on, we will always use H to denote $H_{T(n),m}$ for simplicity.

Our goal now is to use the $A^{\mathsf{Gap\text{-}K^t}}$ algorithm to construct an algorithm that 0.1-avoids the output of $H(y_x, \cdot)$, as by Theorem 7.1, this would give us a way to reconstruct the desired witness y_x.

We begin by analyzing the time-bounded Kolmogorov complexity of $(x, H(y_x, z))$. For all $z \in \{0,1\}^d$, we have

$$\mathsf{K}^{2t,L_{\mathsf{comp}}}(x, H(y_x; z)) \leq \mathsf{K}^t(x) + d + O(1). \tag{9}$$

We let $s = \mathsf{K}^t(x) + d + O(1)$ be the right side of (9).

The above holds since we can first compute x in t time with $\mathsf{K}^t(x)$ bits of advice, then compute y_x by running the search-to-decision reduction, taking $\mathrm{poly}(T(n)) \leq o(t)$ time using the oracle to L_{comp}, and finally compute $H(y_x; z)$ in $\mathrm{poly}(T(n)) = o(t)$ time by specifying z as d bits of advice.

On the other hand, we set ε_{t} so that $q_n(2^{\varepsilon_{\mathrm{t}} n+1}) \leq 2^{n/2}$. Since $q_n(t) = \beta^{(4)}(t \cdot \mathrm{poly}(n))$, ε_{t} only depends on β. Now applying Theorem 6.1 with $\varepsilon = 0.01$, we have that

$$\Pr_{w \sim U_m}\left[\mathsf{K}^{t_1}(xw) \geq \mathsf{K}^{t_2}(x) + m - c_2 \log(t)\right] \geq 0.99. \tag{10}$$

where $t_1 = q_n(2t)$ and $t_2 = \beta^{(7)}(t_1 \cdot \mathrm{poly}(n, m)) = \beta^{(7)}(t_1 \cdot \mathrm{poly}(n))$, and c_2 is a constant that only depends on β. Note that $\log(t_1) \leq n/2$ from our choice of ε_{t}, which satisfies the requirement of Theorem 6.1. We set m so that

$$\mathsf{K}^{t_2}(x) + m - c_2 \log t \geq s + c_1 \log t = \mathsf{K}^t(x) + d + O(1) + c_1 \log t. \tag{11}$$

That is, we set m so that

$$\begin{aligned} m - d &= m - O(\log^3 m + \log T(n)) \\ &\geq \mathsf{K}^t(x) + O(1) + (c_1 + c_2)\log t - \mathsf{K}^{t_2}(x) = \mathsf{cd}^{t,t_2}(x) + O(\log t). \end{aligned}$$

One can see that choosing

$$m = \mathsf{cd}^{t,t_2}(x) + c_3(\log t + \log^3 n) \tag{12}$$

for some big constant $c_3 \geq 1$ would be enough. Note that c_3 is a constant that only depends on β. Hence we can choose ε_{t} to be small enough so that we can set $m \leq 2n$.

Now we claim that for m satisfying (12), $D(w) := \neg A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s\rangle}(xw)$ on m-bit inputs[23] 0.1-avoids $H(y_x; \cdot)$. Note that $\log(2t) \leq |xw| = n + m \leq 2t$. We argue as follows:

[23] D gets an input $w \in \{0,1\}^m$, simulates $A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s\rangle}(xw)$, and negates its output.

1. From (9), for every $z \in \{0,1\}^d$, it holds that $D(H(y_x;z)) = \neg A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s\rangle}(xH(y_x;z)) = 0$.
2. Next, by (10) and (11), together with our choice $t_1 = q_n(2t)$, it holds that for 0.99 fraction of strings $w \in \{0,1\}^m$, we have $D(w) = \neg A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s\rangle}(xw) = 1$.

Now, by Theorem 7.1, for $a = 2m + O(\log T(n) + \log^3 m)$, there exists an advice $\alpha \in \{0,1\}^a$ such that $R^D(\alpha) = y_x$.

The final algorithm S_L. Finally, we are ready to describe our algorithm S_L.

- Given parameter t and w, it sets $m = w + c_3 \cdot (\log t + \log^3 n)$.
- Then it guesses an integer $s \le O(n)$ and an $O(\log t)$-bit program Π as $A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s\rangle}$. Let $D(w) = \neg\Pi(xw)$ be the candidate distinguisher on m-bit inputs, where the running time of Π is truncated at $q_n(t)$.
- It further guesses an advice $\alpha \in \{0,1\}^a$ with $a = 2m + O(\log T(n) + \log^3 m)$, and accepts if and only if $V(x, R^D(\alpha)) = 1$.

Running time and witness complexity. Note that the constructed candidate distinguisher $D(w)$ runs in $q_n(2t)$ time, and hence reconstruction algorithm $R^D(\alpha)$ runs in $\mathrm{poly}(T(n)) \cdot q_n(2t)$, which dominates the running time of the whole algorithm S_L. Hence the running time is $\mathrm{poly}(T(n)) \cdot q_n(2t) = \mathrm{poly}(T(n)) \cdot \beta^{(O(1))}(t \cdot \mathrm{poly}(n))$ as desired.

Regarding the witness complexity, it guesses a bits for reconstruction advice, $O(\log t)$ bits for the program of $A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s\rangle}$, and $O(\log n)$ bits for guessing s. Hence the total witness complexity is bounded by $2m + O(\log T(n) + \log t + \log^3 m) = 2w + O(\log t + \log^3 m)$.

Correctness. Finally, we argue that S_L satisfies our correctness conditions (Items (3) and (4) in the theorem).

- When $L(x) = 0$, since there is no string y making $V(x,y) = 1$ in this case, S_L rejects on all guesses.
- When $L(x) = 1$ and $w \ge \mathsf{cd}^{t,t_2}(x)$, our choice of m now satisfies (12). Hence on our correct guess of s and the program for $A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s\rangle}$, there exists $\alpha \in \{0,1\}^a$ such that $R^D(\alpha) = y_x$ for the corresponding candidate distinguisher D. So on this witness S_L accepts.

Finally, we set

$$\tau_n(t) = t_2 = \beta^{(7)}(t_1 \cdot \mathrm{poly}(n)) = \beta^{(7)}(\beta^{(4)}(2t \cdot \mathrm{poly}(n)) \cdot \mathrm{poly}(n)) = \beta^{(11)}(t_1 \cdot \mathrm{poly}(n)).$$

The last inequality above is due to the fact that β is a good resource function. This completes the proof. □

Remark 7.3. *The reason why we cannot use the direct product generator DP_k in the proof of Lemma 7.2 is that $\mathsf{DP}_k(y_x;z)$ has output length at least $|y_x| = T(n)$. This in particular means in (10), t_2 would be $\beta^{(5)}(t_1 \cdot \mathrm{poly}(T(n)))$ instead of $\beta^{(5)}(t_1 \cdot \mathrm{poly}(n,m))$. Such a t_2 is too large for our theorem.*

Next, we are ready to prove our main result, a parametrized worst-case to average-case reduction from $\mathsf{NTIME}[T]$ to $\Sigma_2\mathsf{TIME}[n]$.

Theorem 7.4. *Let $T\colon \mathbb{N} \to \mathbb{N}$ be such that $T(n) \leq 2^{o(n)}$ and assume that $\Pi_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ for a good resource function $\beta\colon \mathbb{N} \to \mathbb{N}$. Let $\tau_n(t) = \beta^{(O(1))}(t \cdot \mathrm{poly}(n))$ be the corresponding function in Lemma 7.2. Then for every $k\colon \mathbb{N} \to \mathbb{N}$, letting $T_{k(n)}(n) = \tau_n^{(k(n))}(\mathrm{poly}(T(n)))$ and assuming $T_{k(n)}(n) \leq 2^{o(n)}$, it holds that*

$$\begin{aligned}\mathsf{NTIME}[T] &\subseteq \mathsf{NTIMEGUESS}[T_{k(n)}(n), 2n/k(n) + O(\log T_{k(n)}(n)) + \log^3 n)] \\ &\subseteq \mathsf{TIME}[\mathrm{poly}(T_{k(n)}(n)) \cdot 2^{2n/k(n)} \cdot 2^{O(\log^3 n)}]\end{aligned}$$

Proof. Let $p_i(n) = \tau_n^{(i)}(T(n)^{c_{\mathsf{t}}})$, where c_{t} is the universal constant in Lemma 7.2. Note that we have

$$\sum_{i \in [k(n)]} \mathsf{cd}^{p_{i-1}(n), p_i(n)}(x) = \mathsf{K}^{p_0(n)}(x) - \mathsf{K}^{p_i(n)} \leq n + O(1).$$

Hence, it follows that there exists $i \in [k(n)]$ such that $\mathsf{cd}^{p_{i-1}(n), p_i(n)}(x) \leq n/k(n) + O(1)$. Our nondeterministic algorithm then first guesses an integer $i \in [k(n)]$, and then runs S_L with parameter $t = p_i(n)$ and $w = n/k(n) + O(1)$. From our assumption, we have $t \in [T(n)^{c_{\mathsf{t}}}, 2^{\varepsilon_{\mathsf{t}} n}]$, where ε_{t} is the constant in Lemma 7.2. From Lemma 7.2, this algorithm runs in $\mathrm{poly}(T(n)) \cdot p_i(n)$ time and guesses at most $2w + O(\log t + \log^3 n)$ bits of witness.

Furthermore, by Lemma 7.2, our algorithm always rejects when $L(x) = 0$. On the other hand, if $L(x) = 1$, then on the guess i such that $\mathsf{cd}^{p_{i-1}(n), p_i(n)}(x) \leq n/k(n) + O(1)$, our algorithm accepts on at least one witness. This proves the first part of the theorem.

For the second part, we can simply set $k(n) := \arg\min_{u \in [n]} \left[\tau_n^{(u)}(T(n)^{c_{\mathsf{t}}}) \cdot 2^{2n/u}\right]$. □

The following corollary follows from Theorem 7.4 directly.

Corollary 7.5. *If $\Pi_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, then*

$$\mathsf{NTIME}[2^{O(\sqrt{n \log n})}] \subseteq \mathsf{DTIME}[2^{O(\sqrt{n \log n})}].$$

Proof. Let $c > 0$ be an arbitrary constant. Applying Theorem 7.4 for $\beta(n) = \widetilde{O}(n)$ and $T(n) = 2^{c\sqrt{n \log n}}$, we obtain $\tau_n(t) = \beta^{O(1)}(t \cdot \mathrm{poly}(n))$. We note that when $t \leq 2^n$, it holds that that $\tau_n(t) \leq t \cdot \mathrm{poly}(n)$, where the $\mathrm{poly}(n)$ only depends on β but not t.

Letting $k := \sqrt{n/\log n}$, we have that

$$T_{\mathsf{g}}(n) \leq \mathrm{poly}(\tau_n^{(k)}(\mathrm{poly}(T(n)))) \cdot 2^{2n/k + \log^3 n} \leq 2^{O(\sqrt{n \log n})}$$

as desired. □

We now extend the above to include the full polynomial hierarchy version of limited nondeterminism.

Theorem 7.6. *If $\Pi_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\widetilde{O}(n)]$, then*

$$\mathsf{PHTIMEGUESS}[2^{O(\sqrt{n \log n})}, n] \subseteq \mathsf{DTIME}[2^{O\left(\sqrt{n \log n}\right)}].$$

Proof. For every constant $k \geq 1$, we prove

$$\Sigma_k\mathsf{TIMEGUESS}[2^{O(\sqrt{n \log n})}, n] \subseteq \Sigma_{k-1}\mathsf{TIMEGUESS}[2^{O(\sqrt{n \log n})}, n]. \tag{13}$$

The theorem immediately follows from Eq. (13) because

$$\begin{aligned}
&\Sigma_k\mathsf{TIMEGUESS}[2^{O(\sqrt{n\log n})}, n] \\
\subseteq\ &\Sigma_{k-1}\mathsf{TIMEGUESS}[2^{O(\sqrt{n\log n})}, n]. \\
\subseteq\ &\cdots \\
\subseteq\ &\Sigma_0\mathsf{TIMEGUESS}[2^{O(\sqrt{n\log n})}, n] = \mathsf{DTIME}[2^{O(\sqrt{n\log n})}].
\end{aligned}$$

It remains to prove Eq. (13). Let $t(n) := 2^{O(\sqrt{n\log n})}$ and let $L \in \Sigma_k\mathsf{TIMEGUESS}[t, n]$. By definition, there exists a $t(n)$-time algorithm V such that $x \in L$ if and only if

$$\exists y_1 \in \{0,1\}^n, \ldots, \mathcal{Q}_{k-1}y_{k-1} \in \{0,1\}^n, \mathcal{Q}_k y_k \in \{0,1\}^n,\ V(x, y_1, \ldots, y_k) = 1$$

for every input x of length n. Define a language L' so that

$$L' := \{(x, y_1, \ldots, y_{k-1}) \mid \mathcal{Q}_k y_k, V(x, y_1, \ldots, y_k) = 1, |x| = |y_1| = \cdots = |y_k|\}.$$

Clearly, this language is in either $\mathsf{NTIME}[t]$ or $\mathsf{coNTIME}[t]$. By Corollary 7.5, we obtain

$$L' \in \mathsf{NTIME}[t] \cup \mathsf{coNTIME}[t] \subseteq \mathsf{DTIME}[t^{O(1)}].$$

By the property of V, for every input x of length n, we have

$$\begin{aligned}
&x \in L \\
\iff &\exists y_1 \in \{0,1\}^n, \ldots, \mathcal{Q}_{k-1}y_{k-1} \in \{0,1\}^n, \mathcal{Q}_k y_k \in \{0,1\}^n,\ V(x, y_1, y_2, \ldots, y_k) = 1 \\
\iff &\exists y_1 \in \{0,1\}^n, \ldots, \mathcal{Q}_{k-1}y_{k-1} \in \{0,1\}^n, (x, y_1, \ldots, y_{k-1}) \in L'
\end{aligned}$$

Since L' can be accepted by some $t(nk)^{O(1)}$-time algorithm on input $(x, y_1, \ldots, y_{k-1}) \in (\{0,1\}^n)^k$, we conclude that

$$L \in \Sigma_{k-1}\mathsf{TIMEGUESS}[t^{O(1)}, n].$$

□

Finally, Item (2) of Theorem 1.3 follows as an easy corollary of Theorem 7.6.

Corollary 7.7 (Strengthening of Item (2) of Theorem 1.3)**.** *If* $\Sigma_k\mathsf{TIME}[n] \not\subseteq \mathsf{DTIME}[2^{O\left(\sqrt{n\log n}\right)}]$ *for some constant* k*, then*

$$\Pi_2\mathsf{TIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\widetilde{O}(n)].$$

Proof. By Theorem 7.6, it suffices to prove that $\Sigma_k\mathsf{TIME}[n] \not\subseteq \mathsf{DTIME}[2^{O\left(\sqrt{n\log n}\right)}]$ implies that $\mathsf{PHTIMEGUESS}[2^{O(\sqrt{n\log n})}, n] \not\subseteq \mathsf{DTIME}[2^{O\left(\sqrt{n\log n}\right)}]$. We will prove the contrapositive.

Assume that $\mathsf{PHTIMEGUESS}[2^{O(\sqrt{n\log n})}, n] \subseteq \mathsf{DTIME}[2^{O\left(\sqrt{n\log n}\right)}]$. Let $L \in \Sigma_k\mathsf{TIME}[n]$ and $c \geq 1$ be a constant so that $L \in \Sigma_k\mathsf{TIMEGUESS}[cn, cn]$. Now, we construct a language L' such that $x \in L \Leftrightarrow x \circ 0^{(c-1)|x|} \in L'$. Note that $L' \in \mathsf{PHTIMEGUESS}[2^{O(\sqrt{n\log n})}, n]$. Hence we also have $L' \in \mathsf{DTIME}[2^{O\left(\sqrt{n\log n}\right)}]$, which in turn implies that $L \in \mathsf{DTIME}[2^{O\left(\sqrt{n\log n}\right)}]$. □

8 New Worst-case to Average-case Reduction for UP

In this section, we study the worst-case to average-reduction from UP to $\mathsf{NTIME}[n]$.

The following lemma is crucial for the results of this section.

Lemma 8.1. *Let $T\colon \mathbb{N} \to \mathbb{N}$ be such that $T(n) \leq 2^{o(n)}$ and $L \in \mathsf{UTIME}[T(n)]$. Assume that $\mathsf{coNTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ for a good resource function $\beta\colon \mathbb{N} \to \mathbb{N}$. There is a function $\tau_n(t) = \beta^{(O(1))}(t \cdot \mathrm{poly}(n))$ such that, there is a nondeterministic algorithm S_L that satisfies the following:*

- *S_L takes an input $x \in \{0,1\}^n$ to L, an integer $t \in [T(n)^{c_{\mathsf{t}}}, 2^{\varepsilon_{\mathsf{t}} n}]$ and an integer $w \leq n$, where $c_{\mathsf{t}} \geq 1$ is universal constant and $\varepsilon_{\mathsf{t}} \in (0,1)$ is a constant that only depends on β.*
- *S_L then guesses at most $2w + O(\log T(n) + \log^3 n)$ bits of witness and runs in $\mathrm{poly}(T(n)) \cdot \tau_n(t)$ time.*
- *If $L(x) = 0$, S_L rejects on all possible witnesses.*
- *If $L(x) = 1$ and $w \geq \mathsf{cd}^{t,\tau_n(t)}(x)$, S_L accepts on at least one witness.*

Proof. The proof structure of the lemma is very similar to that of Lemma 7.2. The only difference is that we now try to construct a distinguisher for $H_{T(n),m}(y_x, \cdot)$ using the algorithmic compression from Theorem 5.5 instead of the fast $\mathsf{GapK}^{t,A}$ algorithm from Theorem 5.6.

Set up. Let $L \in \mathsf{UTIME}[T(n)]$. We fix an input $x \in \{0,1\}^n$ to L. From the definition of L, there is a linear-time algorithm V taking two inputs $x \in \{0,1\}^n$ and $y \in \{0,1\}^{T(n)}$, such that $L(x) = 1$ if and only if there exists $y \in \{0,1\}^{T(n)}$ satisfying $V(x,y) = 1$. Moreover, for every $x \in \{0,1\}^n$, there is at most one y satisfying $V(x,y) = 1$. We also set c_{t} to be a large enough constant.

If $L(x) = 1$, we let $y_x \in \{0,1\}^{T(n)}$ be the unique string satisfying $V(x,y) = 1$. Otherwise $L(x) = 0$ and we simply let $y_x = 0^{T(n)}$. Next we define a language ensemble L' as follows:

- For every tuple $(t,n,s,m) \in \mathbb{N}^4$, $(x,w) \in L'_{\langle t,n,s,m\rangle}$ if and only if there exists $y \in \{0,1\}^{T(n)}$ and $z \in \{0,1\}^{d_{T(n),m}}$ for $d_{T(n),m} = O(\log T(n) + \log^3 m)$ (of Theorem 7.1) such that
- $|x| = n$, $\mathsf{K}^t(x) \leq s$, $V(x,y) = 1$, and $w = H_{T(n),m}(y,z)$.

Note that we can guess an s-bit program Π, a $T(n)$-bit witness y and a d-bit seed to verify whether $(x,w) \in L'_{\langle t,n,s,m\rangle}$. This takes nondeterministic $O(t + \mathrm{poly}(T(n))) = O(t) \leq O(\langle t,n,s,m\rangle)$ time (since c_{t} is large enough) and hence $L' \in \mathsf{NTIME}[n]$.

Apply algorithm compression to get an algorithm A^{Comp}. By Theorem 5.5, there is a function $q_n(t) = \beta^{(4)}(t \cdot \mathrm{poly}(n))$ such that for every tuple $(t,n,s,m) \in \mathbb{N}^4$ and $\gamma = \langle t,n,s,m\rangle$ such that $\log\gamma \leq n + m \leq \gamma$, there is an $O(\log\gamma)$-size program A^{Comp}_γ satisfying the following:

1. $A^{\mathsf{Comp}}_\gamma(z)$ runs in $q_{|z|}(\gamma)$ time.
2. (Yes case) for every $(x,w) \in L'_\gamma$, $A^{\mathsf{Comp}}_\gamma((x,w)) = 1$.
3. (No case) for every $(x,w) \in \{0,1\}^n \times \{0,1\}^m$, if $\mathsf{K}^{q_{n+m}(\gamma)}(x,w) \geq s + d_{T(n),m} + c_1 \log\gamma$, then $A^{\mathsf{Comp}}_\gamma((x,w)) = 1$, where $c_1 \geq 1$ is a constant that only depends on β. (Note that $\log|L'_\gamma| \leq s + d_{T(n),m}$, as V only accepts at most one witness y for each input x.)

Let $m = O(n)$ be a parameter to be specified later, and $d = O(\log T(n) + \log^3 m)$ be the seed length of $H_{T(n),m}$ from Theorem 7.1 and $s = \mathsf{K}^t(x)$. From now on, we will always use H to denote $H_{T(n),m}$ for simplicity.

Applying Theorem 6.1 with $\varepsilon = 0.01$, we have that

$$\Pr_{w \sim U_m}\left[\mathsf{K}^{t_1}(xw) \geq \mathsf{K}^{t_2}(x) + m - c_2 \log(t)\right] \geq 0.99. \tag{14}$$

where $t_1 = q_{n+m}(\langle t, n, s, m\rangle)$ and $t_2 = \beta^{(7)}(t_1 \cdot \mathrm{poly}(n, m)) = \beta^{(7)}(t_1 \cdot \mathrm{poly}(n))$, and $c_2 \geq 1$ is a constant that only depends on β (note that $O(\log t_1) = O(\log t)$ because $n, s, m \leq O(t)$ and q_{n+m} is at most a polynomial). We set ε_{t} to be small enough so that $\log t_1 \leq n/2$ when $t \leq 2^{\varepsilon_{\mathsf{t}} n}$.

We set m so that

$$\mathsf{K}^{t_2}(x) + m - c_2 \log t \geq s + d + c_1 \log t = \mathsf{K}^t(x) + d + c_1 \log t. \tag{15}$$

That is, we set m so that

$$\begin{aligned} m - d &= m - O(\log^3 m + \log T(n)) \\ &\geq \mathsf{K}^t(x) + (c_1 + c_2)\log t - \mathsf{K}^{t_2}(x) = \mathsf{cd}^{t,t_2}(x) + O(\log t). \end{aligned}$$

One can see that choosing

$$m = \mathsf{cd}^{t,t_2}(x) + c_3(\log t + \log^3 n) \tag{16}$$

for some big constant $c_3 \geq 1$ would be enough. Since c_3 only depends on β, we can set ε_{t} to be small enough so that $m \leq 2n$.

From the definition of $L'_{\langle t,n,s,m\rangle}$ together with (15) and (14), one can see that $D(w) := \neg A^{\mathsf{Comp}}_{\langle t,n,s,m\rangle}(xw)$ on m-bit inputs 0.1-avoids $H(y_x, \cdot)$. Note that we have $\log\langle t, n, s, m\rangle \leq n + m \leq \langle t, n, s, m\rangle$, which satisfies the condition of A^{Comp}.

Now, by Theorem 7.1, for $a = 2m + O(\log T(n) + \log^3 m)$, there exists an advice $\alpha \in \{0,1\}^a$ such that $R^D(\alpha) = y_x$.

The final algorithm S_L. Finally, we are ready to describe our algorithm S_L.

- Given parameter t and w, it sets $m = w + c_3 \cdot (\log t + \log^3 n)$.
- Then it guesses an integer $s \leq O(n)$ and an $O(\log t)$-bit program Π as $A^{\mathsf{Comp}}_{\langle t,n,s,m\rangle}$. Let $D(w) = \neg D(xw)$ be the candidate distinguisher on m-bit inputs, where the running time of Π is truncated at $q_{n+m}(t)$.
- It further guesses an advice $\alpha \in \{0,1\}^a$ with $a = 2m + O(\log T(n) + \log^3 m)$, and accepts if and only if $V(x, R^D(\alpha)) = 1$.

With an identical argument of the last part of proof for Lemma 7.2, we can verify the running time, witness complexity and the correctness of the above algorithm by setting $\tau_n(t) = t_2 = \beta^{O(1)}(t \cdot \mathrm{poly}(n))$ (note that by Proposition 3.1, $\langle t, n, s, m\rangle \leq t \cdot \mathrm{poly}(n)$), which completes the proof. □

The following Theorem 8.2 and Corollary 8.3 follows from Lemma 8.1 in exactly the same way that Theorem 7.4 and Corollary 7.5 follows from Lemma 7.2. We omit their proof.

Theorem 8.2. *Let $T\colon \mathbb{N} \to \mathbb{N}$ be such that $T(n) \leq 2^{o(n)}$ and assume that $\mathsf{coNTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ for a good resource function $\beta\colon \mathbb{N} \to \mathbb{N}$. Let $\tau_n(t) = \beta^{(O(1))}(t \cdot \mathrm{poly}(n))$ be the corresponding function in Lemma 7.2. Then for every $k\colon \mathbb{N} \to \mathbb{N}$, letting $T_{k(n)}(n) = \tau_n^{(k(n))}(\mathrm{poly}(T(n)))$ and assuming $T_{k(n)}(n) \leq 2^{o(n)}$, it holds that*

$$\begin{aligned}\mathsf{UTIME}[T] &\subseteq \mathsf{NTIMEGUESS}[T_{k(n)}(n), 2n/k(n) + O(\log T_{k(n)}(n)) + \log^3 n)] \\ &\subseteq \mathsf{TIME}[\mathrm{poly}(T_{k(n)}(n)) \cdot 2^{2n/k(n)} \cdot 2^{O(\log^3 n)}].\end{aligned}$$

Now Item (1) of Theorem 1.3 and Theorem 1.4 follows easily from Theorem 8.2.

Corollary 8.3 (Strengthening of Item (1) of Theorem 1.3)**.** *If $\mathsf{coNTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$, then*

$$\mathsf{UTIME}[2^{O(\sqrt{n \log n})}] \subseteq \mathsf{DTIME}[2^{O(\sqrt{n \log n})}].$$

Corollary 8.4 (Strengthening of Theorem 1.4)**.** *For every $\varepsilon > 0$, if $\mathsf{UP} \not\subseteq \mathsf{NTIMEGUESS}[\mathrm{poly}(n), \varepsilon n]$, then $\mathsf{coNP} \times \mathcal{U}^{\mathsf{para}} \not\subseteq \mathsf{Avg}^1_{1/2}\mathsf{P}$.*

Proof. We will prove the contrapositive of the theorem. Suppose $\mathsf{coNP} \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{P}$, we know that for some polynomial β, it holds that $\mathsf{coNTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$.

Let $k = 3/\varepsilon$. By Theorem 8.2, for every polynomial T we have

$$\mathsf{UTIME}[T] \subseteq \mathsf{NTIMEGUESS}[\mathrm{poly}(n), 2n/k + o(n)] \subseteq \mathsf{NTIMEGUESS}[\mathrm{poly}(n), \varepsilon n].$$

Therefore $\mathsf{UP} \subseteq \mathsf{NTIMEGUESS}[\mathrm{poly}(n), 2n/k + o(n)] \subseteq \mathsf{NTIMEGUESS}[\mathrm{poly}(n), \varepsilon n]$ as well, this completes the proof. □

9 New Worst-case to Average-case Reduction for Computable Heuristic Schemes

We consider error-less heuristic schemes whose running time can be efficiently estimated. We first present the formal definition of $\mathsf{AvgTIME}_{\mathsf{DTIME}[\beta(n)]}[\beta(n)]$.

Definition 9.1. *Let β be a good resource function, for a language L and a distribution family $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$, we say that $(L, \mathcal{D}) \in \mathsf{AvgTIME}_{\mathsf{DTIME}[\beta(n)]}[\beta(n)]$ if there exists a pair of algorithms (S, C) such that, for every $n, k \in \mathbb{N}$ the following hold:*

1. *For every $x \in \mathcal{D}_n$, if $C(x, n, k) = 1$, then $S(x, n, k) = L(x)$.*
2. *$\Pr_{x\sim\mathcal{D}_n}[C(x, n, k) = 1] \geq 1 - 2^{-k}$.*
3. *$C(x, n, k)$ runs in at most $\beta(|x|)$ time, and $S(x, n, k)$ runs in at most $2^k \cdot \beta(|x|)$ time.*

We remark that from Definition 9.1 and Definition 3.6, it is clear that $(L, \mathcal{D}) \in \mathsf{AvgTIME}_{\mathsf{DTIME}[\beta(n)]}[\beta(n)]$ implies that $(L, \mathcal{D}) \in \mathsf{AvgTIME}[\beta(n)]$.

Following the proof of [Hir21, Theorem 10.2], we show that the assumption $\mathsf{NTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{AvgTIME}[\beta(n)]$ implies the following near-optimal compression and decompression scheme.

Theorem 9.2. *Assuming that $\mathsf{NTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{AvgTIME}[\beta(n)]$ for a good resource function $\beta\colon \mathbb{N} \to \mathbb{N}$. Then, for $\tau_n(t) = \beta^{O(1)}(t \cdot \mathrm{poly}(n))$, there are two algorithms E and D such that*

1. *Fix $n, t \in \mathbb{N}$ such that $t \in [n^{c_t}, 2^n]$, where $c_t \geq 1$ is universal constant. For every $x \in \{0,1\}^n$, it holds that $D(E(x,t), n, t) = x$.*

2. *$|E(x,t)| \leq \mathsf{K}^t(x) + O(\log t)$.*

3. *$D(z, n, t)$ runs in $\tau_n(t)$ time and $E(x,t)$ runs in $\mathrm{poly}(t)$ time.*

Proof. We set $c_t \geq 1$ to be a large enough universal constant. From Theorem 5.6, there is a function $q_n(t) = \beta^{(4)}(t \cdot \mathrm{poly}(n))$ such that for every $t, s \in \mathbb{N}$ with $s \leq O(n)$, there is an $O(\log t)$-size program $A^{\mathsf{Gap\text{-}K^t}}_{\langle t,s \rangle}$ satisfies the following:

- $A^{\mathsf{Gap\text{-}K^t}}_{\langle t,s \rangle}$ takes a string $x \in \{0,1\}^n$ with $\log t \leq n \leq t$, and runs in $q_n(t)$ time.
- (Yes case) If $\mathsf{K}^t(x) \leq s$, then $A^{\mathsf{Gap\text{-}K^t}}_{\langle t,s \rangle}(x) = 1$.
- (No case) If $\mathsf{K}^{q_n(t)}(x) \geq s + c_1 \log t$, then $A^{\mathsf{Gap\text{-}K^t}}_{\langle t,s \rangle}(x) = 0$, for some constant c_1 that only depends on β.

We will first use $A^{\mathsf{Gap\text{-}K^t}}$ together the DP generator to obtain a compression scheme, and then define our algorithms E and D. For a parameter $k \leq O(n)$, letting $d = nk$, for all $z \in \{0,1\}^d$ we have

$$\mathsf{K}^{2t}(\mathsf{DP}_k(x; z)) \leq \mathsf{K}^t(x) + d + c_2 \log t, \tag{17}$$

where $c_2 \geq 1$ is a universal constant. This holds because one can compute $\mathsf{DP}_k(x; z)$ by first computing x in t time and then compute $\mathsf{DP}_k(x; z)$ in $d + k = o(t)$ time. (We also need to specify t and k, which takes $O(\log t)$ bits.)

By a simple counting argument, we have that

$$\Pr_{w \sim U_{d+k}}[\mathsf{K}(w) \geq d + k - c_3] \geq 0.99, \tag{18}$$

where $c_3 \geq 1$ is a universal constant.

Let $s = \mathsf{K}^t(x) + d + c_2 \log t$ in (17). We will set k so that

$$d + k - c_3 \geq s + c_1 \log t = \mathsf{K}^t(x) + d + (c_2 + c_1) \log t,$$

which can be satisfied by picking k so that

$$k = \mathsf{K}^t(x) + c_3 \log t \tag{19}$$

for a large enough constant $c_3 \geq 1$ only depending on β.

For k that satisfies (19), we have $\log(2t) \leq d + k \leq 2t$ and it follows from (17) and (18) that $D(w) \coloneqq A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s \rangle}(w)$ 0.1-distinguishes between $\mathsf{DP}_k(x; U_d)$ and U_{d+k}. Note that D takes $O(\log t)$ bits of advice to specify the code for $A^{\mathsf{Gap\text{-}K^t}}_{\langle 2t,s \rangle}(w)$, and runs in $t_1 = q_{d+k}(2t)$ time on $(d+k)$-bit inputs. Hence, by Theorem 5.7, there is a function $\eta_n(t) = \beta^{(3)}(t \cdot \mathrm{poly}(n))$ such that there is an algorithm R running in $\eta_n(t_1)$ time that takes x and D (along with the advice for D) as input and $O(\log t_1) = O(\log t)$ bits as advice, and outputs a program of size $k + O(\log t)$ that prints x in $\eta_n(t_1)$ time.

Now we are ready to specify our encoding algorithm $E(x,t)$:

1. We first enumerate all possible values $u \leq n + O(1)$ for $\mathsf{K}^t(x)$, from the smallest to the largest. Then we set s and k according to u as the guess of $\mathsf{K}^t(x)$ (*i.e.*, $s = u + d + c_2 \log t$ and $k = u + c_3 \log t$).

2. Next, we enumerate all possible advice strings $\alpha_D \in \{0,1\}^{O(\log t)}$ for the (candidate) distinguisher D, and all possible advice strings $\alpha_R \in \{0,1\}^{O(\log t)}$ for the reconstruction algorithm R, and run $R_{/\alpha_R}$ with input x and D_{α_D}. Then we check whether $R_{/\alpha_R}$ outputs a desired program of size $k + O(\log t)$ that prints x in $\eta_n(t_1)$ time. We output the program and terminate the algorithm if the outputted program passes the check.

We now define $D(z, n, t)$ as the algorithm that simply simulates the program z for $\eta_n(t_1)$ steps and then outputs its output, and we set $\tau_n(t) = \beta^{O(1)}(t \cdot \mathrm{poly}(n))$ to be large enough so that $\tau_n(t) \geq \eta_n(t_1)$. This ensures the running time of $D(z, n, t)$ satisfies our requirement.

For $E(x, t)$, note that the running time is at most $\mathrm{poly}(t) \cdot \mathrm{poly}(n) \leq \mathrm{poly}(t)$ as required. Now, let $\bar{u}$ be the smallest guess for $\mathsf{K}^t(x)$ on which the algorithm terminates. Note that by our previous discussions, the algorithm must terminate on the correct guess of $\mathsf{K}^t(x)$, so we have $\bar{u} \leq \mathsf{K}^t(x)$. Finally, on this guess $\bar{u}$, our algorithm outputs a program of size $\bar{u} + O(\log t) \leq \mathsf{K}^t(x) + O(\log t)$ which outputs x in $\eta_n(t_1) \leq \tau_n(t)$ time. This completes the proof. □

The following lemma is crucial for our results in this section.

Lemma 9.3. *Let $T\colon \mathbb{N} \to \mathbb{N}$ be such that $T(n) \leq 2^{o(n)}$, $L \in \mathsf{NTIME}[T(n)]$ and $\beta\colon \mathbb{N} \to \mathbb{N}$ be a good resource function. Assume that $\mathsf{NTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{AvgTIME}_{\mathsf{DTIME}[\beta(n)]}[\beta(n)]$. There is a function $\tau_n(t) = \beta^{(O(1))}(t \cdot \mathrm{poly}(n))$ and an algorithm S_L such that the following hold:*

1. *S_L takes an input $x \in \{0,1\}^n$ to L, an integer $t \in [\max(T(n), n^{c_{\mathsf{t}}}), 2^n]$ and an integer $w \leq n$, where c_{t} is the universal constant in Theorem 9.2.*

2. *S_L runs in $2^w \cdot \mathrm{poly}(t)$ time and $S_L(x, t, w) \in \{L(x), \perp\}$.*

3. *If $w \geq \mathsf{cd}^{t,\tau_n(t)}(x)$, then $S_L(x) = L(x)$.*

Proof. Our proof follows the same argument of the proof of [Hir21, Theorem 10.1].

Compressed Language L^{zip} and its heuristic. We first define a language L^{zip} (*i.e.*, a compressed version of L) such that for every $(t, n, s) \in \mathbb{N}^3$, $u \in L^{\mathsf{zip}}_{\langle t,n,s\rangle}$ if and only if the following holds:

1. $|u| = s$ and $t \geq T(n)$.

2. For $x = \mathsf{Univ}^t(u)$, it holds that $x \in L$ and $|x| = n$.

From its definition we can see that $L^{\mathsf{zip}} \in \mathsf{NTIME}[n]$, since given an input $(u, 1^{\langle t,n,s\rangle})$, one can first compute $x = \mathsf{Univ}^t(u)$ in $O(t)$ time, and then check whether $x \in L$ in nondeterministic $T(n) = O(t)$ time.

From our assumption that $\mathsf{NTIME} \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{AvgTIME}_{\mathsf{DTIME}[\beta(n)]}[\beta(n)]$ and the Definition 9.1, there are two algorithms S and C such that for every $(t, n, s) \in \mathbb{N}^3$ and every $k \in \mathbb{N}$, letting $\gamma = \langle t, n, s\rangle$, the followings hold:

1. For every $u \in \{0,1\}^s$, if $C(u, \langle\gamma, s\rangle, k) = 1$, then $S(u, \langle\gamma, s\rangle, k) = L^{\mathsf{zip}}_\gamma(u)$. Note that here we are applying S and C to solve L^{zip} over the distribution $\mathcal{U}^{\mathsf{para}}_{\langle\gamma,s\rangle}$.

2. $\Pr_{u\sim U_s}[C(u, \langle\gamma, s\rangle, k) = 1] \geq 1 - 2^{-k}$.

3. $C(u, \langle\gamma, s\rangle, k)$ runs in at most $\beta(\gamma + s)$ time, and $S(u, \langle\gamma, s\rangle)$ runs in at most $\beta(\gamma + s) \cdot 2^k$ time.

Applying algorithmic compression to the language L^{fail} consisting of failed inputs. Now, we consider another language L^{fail}, such that for every $(t,n,s,k) \in \mathbb{N}^4$, $u \in L^{\mathsf{fail}}_{\langle t,n,s,k\rangle}$ if and only if (1) $u \in L^{\mathsf{zip}}_{\langle t,n,s\rangle}$ and (2) $C(u,\langle \gamma, s\rangle, k) = 0$ for $\gamma = \langle t,n,s\rangle$. That is, $L^{\mathsf{fail}}_{\langle t,n,s,k\rangle}$ contains all inputs u to $L^{\mathsf{zip}}_{\langle t,n,s\rangle}$ on which our heuristic algorithm with parameter k fails.

Recall that we have $\Pr_{u\sim U_s}[C(u,\langle \gamma, s\rangle, k) = 0] \le 2^{-k}$, which implies that $|L^{\mathsf{fail}}_{\langle t,n,s,k\rangle}| \le 2^{s-k}$. Also, note that a straightforward algorithm solves $L^{\mathsf{fail}}_{\langle t,n,s,k\rangle}(u)$ in $\beta(\langle t,n,s,k\rangle)$ nondeterministic time by computing $L^{\mathsf{zip}}_{\langle t,n,s\rangle}(u) \wedge [C(u,\langle \gamma, s\rangle, k) = 0]$, meaning that $L^{\mathsf{fail}} \in \mathsf{NTIME}[\beta(n)]$.

Applying Theorem 5.5 together with a simple padding argument[24], it follows that for some $q_n(t) = \beta^{O(1)}(t \cdot \mathrm{poly}(n))$ and for every $u \in L^{\mathsf{fail}}_{\langle t,n,s,k\rangle}$, we have

$$\mathsf{K}^{q_{|u|}(\langle t,n,s,k\rangle)}(u) \le s - k + O(\log t). \tag{20}$$

Our algorithm S_L. Now our algorithm S_L works as follows:

1. Let $k = w + c_1 \log t$ for a large constant $c_1 \ge 1$ to be specified later. Compute $z = E(x,t)$ and let u be the code of the algorithm outputting $D(z,n,t)$. Let $s = |u|$. Applying Theorem 9.2, and note that u can be specified by the integer t, the string z and the code of D, we have

$$s = |u| = |z| + O(\log t) \le \mathsf{K}^t(x) + O(\log t). \tag{21}$$

   Moreover, for some $t_1 = \beta^{(O(1))}(t \cdot \mathrm{poly}(n))$, we have $\mathsf{Univ}^{t_1}(u) = x$.

2. Let $\gamma = \langle t_1, n, s\rangle$. If $C(u,\langle \gamma, s\rangle, k) = 0$, output $\perp$.

3. Otherwise, output $S(u,\langle \gamma, s\rangle, k)$.

Running time and the correctness of S_L. Now we argue that S_L satisfies our requirements. The running time of S_L is dominated by the running time of $S(u,\langle \gamma, s\rangle, k)$, which is $\beta(\gamma + s) \cdot 2^k \le 2^k \cdot \mathrm{poly}(t)$. Hence, the overall running time of S_L is $2^k \cdot \mathrm{poly}(t)$ as well.

For the correctness, we first note that whenever S_L does not output $\perp$, it means $C(u,\langle \gamma, s\rangle, k) = 1$, and in this case $S_L(x) = L^{\mathsf{zip}}_{\gamma}(u) = L^{\mathsf{zip}}_{\langle t_1,n,s\rangle}(u)$. Since $|u| = s$ and $\mathsf{Univ}^{t_1}(u) = x$, we have $L^{\mathsf{zip}}_{\langle t_1,n,s\rangle}(u) = L(x)$, and therefore $S_L(x) = L(x)$.

Now, assume that $w \ge \mathsf{cd}^{t,\tau_n(t)}(x)$ for some $\tau_n(t) = \beta^{O(1)}(t \cdot \mathrm{poly}(n))$ to be specified later. We will prove that in this case we must have $C(u,\langle \gamma, s\rangle, k) = 1$ and thus $S_L(x) = L(x)$.

Assume that $C(u,\langle \gamma, s\rangle, k) = 0$ for the sake of contradiction. By definition it means that $u \in L^{\mathsf{fail}}_{\langle t_1,n,s,k\rangle}$ and hence

$$\mathsf{K}^{q_s(\langle t_1,n,s,k\rangle)}(u) \le s - k + O(\log t).$$

Recalling that $\mathsf{Univ}^{t_1}(u) = x$, we further have

$$\mathsf{K}^{2q_s(\langle t_1,n,s,k\rangle)}(x) \le s - k + O(\log t). \tag{22}$$

[24] Theorem 5.5 applies to languages in $\mathsf{NTIME}[n]$ but our L^{fail} is in $\mathsf{NTIME}[\beta(n)]$. Still, we can define a padded version of L^{fail} such that it is in $\mathsf{NTIME}[n]$, and apply Theorem 5.5 to the padded version. The resulting algorithm can solve the algorithmic compression task for L^{fail} as well. Note that we will later set s so that the constraint $\log t \le |x| \le t$ in Theorem 5.5 is satisfied.

Combining (21) and (22), we have that

$$\mathsf{cd}^{t,2q_s(\langle t_1,n,s,k\rangle)}(x) \geq k - O(\log t). \tag{23}$$

Combining (23) and the definition of k, we further have

$$w \leq \mathsf{cd}^{t,2q_s(\langle t_1,n,s,k\rangle)}(x) + O(\log t) - c_1 \cdot \log t.$$

Setting c_1 to be large enough, the above translates to

$$w < \mathsf{cd}^{t,2q_s(\langle t_1,n,s,k\rangle)}(x). \tag{24}$$

Finally, we set $\tau_n(t) = \beta^{O(1)}(t \cdot \mathrm{poly}(n))$ so that $\tau_n(t) \geq 2q_s(\langle t_1, n, s, k\rangle)$. In this case, (24) contradicts our promise that $w \geq \mathsf{cd}^{t,\tau_n(t)}(x)$, and therefore we have $C(u, \langle \gamma, s\rangle, k) = 1$ when $w \geq \mathsf{cd}^{t,\tau_n(t)}(x)$. This completes the proof. □

Theorem 9.4. *Let $T\colon \mathbb{N} \to \mathbb{N}$ be such that $T(n) \leq 2^{o(n)}$ and assume that $\mathsf{NTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{AvgTIME}_{\mathsf{DTIME}[\beta(n)]}[\beta(n)]$ for a good resource function $\beta\colon \mathbb{N} \to \mathbb{N}$. Let $\tau_n(t) = \beta^{(O(1))}(t \cdot \mathrm{poly}(n))$ be the corresponding function in Lemma 9.3. Then for every $k\colon \mathbb{N} \to \mathbb{N}$, letting $T_{k(n)}(n) = \tau_n^{(k(n))}(T(n) \cdot \mathrm{poly}(n))$ and assuming $T_{k(n)}(n) = 2^{o(n)}$, it holds that*

$$\mathsf{NTIME}[T] \subseteq \mathsf{TIME}[\mathrm{poly}(T_{k(n)}(n)) \cdot 2^{n/k(n)}].$$

Proof. Let $L \in \mathsf{NTIME}[T]$ and let $p_i(n) = \tau_n^{(i)}(T(n) \cdot n^{c_{\mathsf{t}}})$, where c_{t} is universal constant in Theorem 9.2. Note that we have

$$\sum_{i \in [k(n)]} \mathsf{cd}^{p_{i-1}(n),p_i(n)}(x) = \mathsf{K}^{p_0(n)}(x) - \mathsf{K}^{p_i(n)} \leq n + O(1).$$

Hence, it follows that there exists $i \in [k(n)]$ such that $\mathsf{cd}^{p_{i-1}(n),p_i(n)}(x) \leq n/k(n) + O(1)$. Our algorithm for L then runs S_L from Lemma 9.3 with parameter $t = p_i(n)$ and $w = n/k(n) + O(1)$, for every $i \in [k(n)]$. From Lemma 9.3, this algorithm runs in $k(n) \cdot \mathrm{poly}(t) \cdot 2^w = \mathrm{poly}(T_{k(n)}(n)) \cdot 2^{n/k(n)}$ time. Our algorithm outputs the output of $S_L(p_i(n), w)$ if it is not $\perp$ (if many $S_L(p_i(n), w)$'s are not $\perp$, it simply outputs the one with the smallest i.)

By Lemma 9.3, if there is at least one $S_L(p_i(n), w) \neq \perp$, then our algorithm is correct on x. Also, since there must be an i such that $\mathsf{cd}^{p_{i-1}(n),p_i(n)}(x) \leq n/k(n) + O(1)$, we know that our algorithm correctly solves $L(x)$. □

From Theorem 9.4, we immediately have the following corollaries.

Corollary 9.5. *The following holds:*

1. *If $\mathsf{NTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{AvgTIME}_{\mathsf{DTIME}[\widetilde{O}(n)]}[\widetilde{O}(n)]$, then $\mathsf{NTIME}[2^{O(\sqrt{n \log n})}] \subseteq \mathsf{DTIME}[2^{O(\sqrt{n \log n})}]$, and consequently*

$$\Sigma_k\mathsf{TIMEGUESS}[2^{O(\sqrt{n \log n})}, n] \subseteq \mathsf{TIME}[2^{O(\sqrt{n \log n})}].$$

2. *If $\mathsf{NTIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{AvgTIME}_{\mathsf{DTIME}[n^{1+\varepsilon}]}[n^{1+\varepsilon}]$ for every $\varepsilon > 0$, then for every $\delta > 0$ and $\kappa > 0$, it holds that $\mathsf{NTIME}[2^{n^{1-\delta}}] \subseteq \mathsf{DTIME}[2^{\kappa n/\log n}]$.*

Proof. The first item follows from Theorem 9.4 in exactly the same way that Corollary 7.5 follows from Theorem 7.4. The inclusion for $\Sigma_k\mathsf{TIMEGUESS}[2^{O(\sqrt{n\log n})}, n]$ follows from the proof of Theorem 7.6. So we omit the proof of the first item and focus on the second item.

Fix $\delta > 0$ and $\kappa > 0$, we set $\varepsilon = \varepsilon(\delta, \kappa) > 0$ be a constant to be fixed later and set $\beta(n) = n^{1+\varepsilon}$. We also set $T(n) = 2^{n^{1-\delta}}$. Recall that $T_k(n) = \tau_n^{(k)}(T(n) \cdot \mathrm{poly}(n))$ in Theorem 9.4. For a universal constant $c_1 \geq 1$, it holds that

$$T_k(n) \leq \beta^{(c_1 k)}(2^{n^{1-\delta}} \cdot n^{c_1 k}).$$

We will only consider $k \leq O(\log n)$. The above can be further bounded by

$$\beta^{(c_1 k)}(2^{n^{1-\delta/2}}) = 2^{n^{1-\delta/2}\cdot(1+\varepsilon)^{c_1 k}} \leq 2^{n^{1-\delta/2}\cdot e^{\varepsilon c_1 k}},$$

the last inequality follows from the fact that $(1+\varepsilon)^t \leq e^{\varepsilon t}$ for all $t \geq 0$.

Now, we set $k = 2\log n/\kappa$ and $\varepsilon = \ln(n^\delta)/(3c_1 k)$ so that $e^{\varepsilon c_1 k} = n^{\delta/3}$. Note that $\varepsilon > 0$ is a constant that only depends on δ and κ, as desired.

Finally, applying Theorem 9.4, we have $\mathsf{NTIME}[2^{n^{1-\delta}}] \subseteq \mathsf{DTIME}[\mathrm{poly}(2^{n^{1-\delta/6}}) \cdot 2^{n/k}] \leq 2^{\kappa n/\log n}$, which completes the proof. □

Taking contrapositive of Corollary 9.5, the following corollary follows immediately.

Corollary 9.6. *The following hold:*

1. *(Strengthening of Item (3) of Theorem 1.3) For every $\delta > 0$ and $\kappa > 0$, $\mathsf{NTIME}[2^{n^{1-\delta}}] \not\subseteq \mathsf{DTIME}[2^{\kappa n/\log n}]$ implies that $\mathsf{NTIME}[n] \times \mathcal{U}^{\mathsf{para}} \not\subseteq \mathsf{AvgTIME}_{\mathsf{DTIME}[n^{1+\varepsilon}]}[n^{1+\varepsilon}]$ for some $\varepsilon > 0$. In particular, ETH implies this conclusion.*
2. *$\mathsf{NTIME}[2^{O(\sqrt{n\log n})}] \not\subseteq \mathsf{DTIME}[2^{O(\sqrt{n\log n})}]$ implies $\mathsf{NTIME}[n] \times \mathcal{U}^{\mathsf{para}} \not\subseteq \mathsf{AvgTIME}_{\mathsf{DTIME}[\widetilde{O}(n)]}[\widetilde{O}(n)]$.*
3. *$\Sigma_k\mathsf{TIME}[n] \not\subseteq \mathsf{DTIME}\left[2^{O\left(\sqrt{n\log n}\right)}\right]$ implies $\mathsf{NTIME}[n] \times \{\mathcal{U}^{\mathsf{para}}\} \not\subseteq \mathsf{Avg}_{\mathsf{DTIME}[\widetilde{O}(n)]}\mathsf{TIME}[\widetilde{O}(n)]$ for every constant k.*

10 Applications to NP Witness Compression

In addition to giving a fast deterministic algorithm, Theorem 7.4 implies NP has compressible witness. In fact, the proof of Theorem 7.4 allows for compression in a somewhat strong sense, as we define below.

Definition 10.1. *Let $L \in \mathsf{NTIME}[T(n)]$. We say that L has ℓ-*compressible witnesses *if for all verifiers V for L, there exists an algorithm A_V running in time $\mathrm{poly}(T(n))$ such that on input $x \in \{0,1\}^n$ and $y \in \{0,1\}^\ell$, $A_V(x,y)$ outputs a string z such that for all $x \in L \cap \{0,1\}^n$, there exists $y \in \{0,1\}^\ell$ such that $V(x, A_V(x,y)) = 1$. That is, there is some "witness" $y \in \{0,1\}^\ell$ such that $A_V(x,y)$ outputs a witness z for $x \in L$ under V.*

Note that this definition is interesting only if ℓ is less than the amount of nondeterminism used in the algorithm that shows $L \in \mathsf{NTIME}[T(n)]$.

Remark 10.2. *Here and throughout, definitions and results can also be phrased in terms of NP relations instead of verifiers to avoid focusing on a particular verifier V, but these formulations are essentially equivalent.*

We now observe that the proof of Theorem 7.4 gives us compressible witnesses for all of NP.

Corollary 10.3 (Formal version of Theorem 1.5)**.** *Let $T\colon \mathbb{N} \to \mathbb{N}$, $\varepsilon > 0$ be a constant, and assume that $\Pi_2\mathsf{TIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{TIME}[\beta(n)]$ for a good resource function $\beta\colon \mathbb{N} \to \mathbb{N}$. For $\ell = \ell(n) = \varepsilon n + O(\log T(n) + \log^3 n)$, all languages $L \in \mathsf{NTIME}[T]$ have ℓ-compressible witnesses.*

In particular, if $\Pi_2\mathsf{TIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{P}$, then for all constant $\varepsilon > 0$ and for sufficiently large n,

$$\mathsf{NP} \subseteq \mathsf{NTIMEGUESS}[\mathrm{poly}(n), \varepsilon n].$$

Moreover, for all $L \in \mathsf{NP}$ and any corresponding verifier V, there is a $\mathrm{poly}(n)$-time algorithm using εn bits of nondeterminism outputting witnesses for L under V (when they exist).

Proof. We use the proof of Theorem 7.4 with $k(n) = 3/\varepsilon$. Let $L \in \mathsf{NTIME}[T]$ and V be a verifier for L. Following the notation of Lemma 7.2, the algorithm S_L uses $2\varepsilon n/3 + O(\log T(n) + \log^3 n)$ bits of nondeterminism to output some witness $y = R^D(\alpha)$ such that $V(x, y) = 1$, where the nondeterminism is used to guess α and advice for the distinguisher D. Thus, we can set $A_V(x, y) = R^D(\alpha)$ to show the first part of the statement.

For the second part, if $T(n) = \mathrm{poly}(n)$, this becomes $2\varepsilon n/3 + O(\log n + \log^3 n) \leq \varepsilon n$ bits of nondeterminism for sufficiently large n, as desired. □

We give a natural class of problems for which this gives a useful notion of witness compressible.

Definition 10.4. *Let $p(n)$ be an arbitrary polynomial. For a language $L \in \mathsf{NP}$, let V be a polynomial-time verifier for L with witnesses of length $p_1(n)$. We define*

$$L_{V,p(n)} \coloneqq \{x \in \{0,1\}^* : \exists y_1, \ldots, y_{p(|x|)} \in \{0,1\}^{p_1(|x|)} \text{ such that } \forall i, V(x, y_i) = 1 \text{ and } \forall i < j, y_i \neq y_j\}.$$

That is, $L_{V,p(n)}$ is the set of all $x \in \{0,1\}^*$ that have at least $p(|x|)$ distinct witnesses under V.

Note that $L_{V,p(n)} \in \mathsf{NP}$ for any polynomial time verifier V, as a set of distinct witnesses $\{y_i\}_{i \in [p_1(|x|)]}$ for L form a witness for $L_{V,p(n)}$. These witnesses for $L_{V,p(n)}$ have length $p(|x|) \cdot p_1(|x|)$. However, Corollary 10.3 tells us the following:

Corollary 10.5. *If $\Pi_2\mathsf{TIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{P}$, then for all constant $\varepsilon > 0$ and sufficiently large n, $L_{V,p(n)}$ has εn-compressible witnesses.*

As an example, we can apply this corollary to $\mathsf{SAT} \in \mathsf{NP}$.

Corollary 10.6. *If $\Pi_2\mathsf{TIME}[n] \times \mathcal{U}^{\mathsf{para}} \subseteq \mathsf{Avg}^1_{1/2}\mathsf{P}$, then for all constant $\varepsilon > 0$, sufficiently large n, and polynomials $p(n)$, there is a polynomial time algorithm $A_{p,\varepsilon}$ with the following property: if $\varphi \in \{0,1\}^n$ is a SAT instance with at least $p(n)$ satisfying assignments, then there exists $y \in \{0,1\}^{\varepsilon n}$ such that $A_{p,\varepsilon}(\varphi, y)$ outputs $p(n)$ distinct satisfying assignments of φ.*

Acknowledgements

We are grateful to Rahul Ilango and Ryan Williams for helpful discussions. In particular, we want to thank Ryan Williams for the observation that an HSG with seed length $O(\log t)$ already suffices for our proof.

References

[AF09] Luis Filipe Coelho Antunes and Lance Fortnow. Worst-case running times for average-case algorithms. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 298–303. IEEE Computer Society, 2009.

[AFvMV06] Luis Antunes, Lance Fortnow, Dieter van Melkebeek, and N. V. Vinodchandran. Computational depth: Concept and applications. *Theor. Comput. Sci.*, 354(3):391–404, 2006.

[AGGM06] Adi Akavia, Oded Goldreich, Shafi Goldwasser, and Dana Moshkovitz. On basing one-way functions on NP-hardness. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 701–710, 2006.

[All20] Eric Allender. The new complexity landscape around circuit minimization. In *Language and Automata Theory and Applications - 14th International Conference, LATA 2020, Milan, Italy, March 4-6, 2020, Proceedings*, volume 12038 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2020.

[BB15] Andrej Bogdanov and Christina Brzuska. On Basing Size-Verifiable One-Way Functions on NP-Hardness. In *Proceedings of the Theory of Cryptography Conference (TCC)*, pages 1–6, 2015.

[BBB19] Enric Boix-Adserà, Matthew S. Brennan, and Guy Bresler. The Average-Case Complexity of Counting Cliques in Erdős-Rényi Hypergraphs. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 1256–1280, 2019.

[BCGL92] Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the Theory of Average Case Complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, 1992.

[BFKR97] Donald Beaver, Joan Feigenbaum, Joe Kilian, and Phillip Rogaway. Locally Random Reductions: Improvements and Applications. *J. Cryptol.*, 10(1):17–36, 1997.

[BFP05] Harry Buhrman, Lance Fortnow, and Aduri Pavan. Some results on derandomization. *Theory Comput. Syst.*, 38(2):211–227, 2005.

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short pcps verifiable in polylogarithmic time. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 120–134. IEEE Computer Society, 2005.

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 483–496, 2017.

[BRSV18] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of Work From Worst-Case Assumptions. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 789–819, 2018.

[BSV21] Zvika Brakerski, Noah Stephens-Davidowitz, and Vinod Vaikuntanathan. On the hardness of average-case k-sum. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 29:1–29:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[BT06a] Andrej Bogdanov and Luca Trevisan. Average-Case Complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1), 2006.

[BT06b] Andrej Bogdanov and Luca Trevisan. On Worst-Case to Average-Case Reductions for NP Problems. *SIAM J. Comput.*, 36(4):1119–1159, 2006.

[BV14] Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In *Proc. 41st Internat. Colloq. on Automata, Languages and Programming (ICALP'14)*, pages 163–173. Springer, 2014.

[CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1–12. IEEE, 2020.

[CT21] Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: eliminating randomness at almost no cost. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 283–291. ACM, 2021.

[DLW20] Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New Techniques for Proving Fine-Grained Average-Case Hardness. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 774–785, 2020.

[FF93] Joan Feigenbaum and Lance Fortnow. Random-Self-Reducibility of Complete Sets. *SIAM J. Comput.*, 22(5):994–1005, 1993.

[GMR05] Venkatesan Guruswami, Daniele Micciancio, and Oded Regev. The complexity of the covering radius problem. *Comput. Complex.*, 14(2):90–121, 2005.

[GR18] Oded Goldreich and Guy N. Rothblum. Counting t-Cliques: Worst-Case to Average-Case Reductions and Direct Interactive Proof Systems. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 77–88, 2018.

[GS87] Yuri Gurevich and Saharon Shelah. Expected Computation Time for Hamiltonian Path Problem. *SIAM J. Comput.*, 16(3):486–502, 1987.

[GS88] Joachim Grollmann and Alan L. Selman. Complexity Measures for Public-Key Cryptosystems. *SIAM J. Comput.*, 17(2):309–335, 1988.

[Hir18] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 247–258. IEEE Computer Society, 2018.

[Hir20a] Shuichi Hirahara. Characterizing average-case complexity of PH by worst-case meta-complexity. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 50–60. IEEE, 2020.

[Hir20b] Shuichi Hirahara. Non-disjoint promise problems from meta-computational view of pseudorandom generator constructions. In *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 20:1–20:47. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Hir21] Shuichi Hirahara. Average-case hardness of NP from exponential worst-case hardness assumptions. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 292–302. ACM, 2021.

[HN21] Shuichi Hirahara and Mikito Nanashima. On worst-case learning in relativized heuristica. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, 2021.

[HS65] Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[HS21] Shuichi Hirahara and Nobutaka Shimizu. Nearly Optimal Average-Case Complexity of Counting Bicliques Under SETH. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2346–2365, 2021.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.

[IL89] Russell Impagliazzo and Michael Luby. One-way Functions are Essential for Complexity Based Cryptography (Extended Abstract). In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 230–235, 1989.

[Imp95] Russell Impagliazzo. A Personal View of Average-Case Complexity. In *Proceedings of the Structure in Complexity Theory Conference*, pages 134–147, 1995.

[Imp11] Russell Impagliazzo. Relativized Separations of Worst-Case and Average-Case Complexities for NP. In *Proceedings of the Conference on Computational Complexity (CCC)*, pages 104–114, 2011.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[IW97] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proc. 29th STOC*, pages 220–229. ACM Press, 1997.

[JMV15] Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local Reductions. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 749–760, 2015.

[Ko85] Ker-I Ko. On Some Natural Complete Operators. *Theor. Comput. Sci.*, 37:1–30, 1985.

[Liv10] Noam Livne. On the Construction of One-Way Functions from Average Case Hardness. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 301–309, 2010.

[LLW19] Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. Public-Key Cryptography in the Fine-Grained Setting. *IACR Cryptology ePrint Archive*, 2019:625, 2019.

[MW20] Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM J. Comput.*, 49(5), 2020.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. System Sci.*, 49(2):149–167, 1994.

[RRV02] Ran Raz, Omer Reingold, and Salil P. Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. *J. Comput. Syst. Sci.*, 65(1):97–128, 2002.

[STV01] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001.

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

[SW15] Rahul Santhanam and Richard Ryan Williams. Beating Exhaustive Search for Quantified Boolean Formulas and Connections to Circuit Complexity. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 231–241, 2015.

[Vio05a] Emanuele Viola. On Constructing Parallel Pseudorandom Generators from One-Way Functions. In *Proceedings of the Conference on Computational Complexity (CCC)*, pages 183–197, 2005.

[Vio05b] Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. *Computational Complexity*, 13(3-4):147–188, 2005.

[Wee06] Hoeteck Wee. Finding Pessiland. In *Proceedings of the Theory of Cryptography Conference (TCC)*, pages 429–442, 2006.

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

[Wil16] R. Ryan Williams. Natural proofs versus derandomization. *SIAM J. Comput.*, 45(2):497–529, 2016.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il