Hardness Amplification within NP against Deterministic Algorithms[*]

Parikshit Gopalan[†]
Microsoft Research - Silicon Valley
Mountain View, CA 94043.

Venkatesan Guruswami[‡]
Computer Science Department
Carnegie Mellon University.

Abstract

We study the average-case hardness of the class NP against algorithms in P. We prove that there exists some constant $\mu > 0$ such that if there is some language in NP for which no deterministic polynomial time algorithm can decide L correctly on a $1 - (\log n)^{-\mu}$ fraction of inputs of length n, then there is a language L' in NP for which no deterministic polynomial time algorithm can decide L' correctly on a $3/4 + (\log n)^{-\mu}$ fraction of inputs of length n. In coding theoretic terms, we give a construction of a monotone code that can be uniquely decoded up to error rate $\frac{1}{4}$ by a *deterministic* local decoder.

1 Introduction

The relationship between the complexity classes NP and P has been arguably the central open problem in theoretical computer science, starting from the seminal work of Cook, Levin [7, 23] and Karp [22]. In this paper, we explore one (tiny) fragment of the relation between these classes, namely we ask:

> How hard are languages in NP on *average* for deterministic polynomial time algorithms?

The average case hardness of complexity classes such as EXP and NP has been studied intensively. There are several motivations for this study: in addition to being a natural alternative to worst-case hardness, average-case hardness is also necessary in cryptography. Functions with extreme average-case hardness also play a key role in the construction of pseudorandom generators and derandomization. This hardness is measured against a certain complexity class $\mathcal{C}$. We consider

[*] A version of this paper was presented at the *23rd IEEE Conference on Computational Complexity*, June 2008.

[†] Work done while the author was a postdoc at the University of Washington, supported in part by NSF CCF-0343672 and Venkatesan Guruswami's Packard Foundation fellowship. Email: parik@microsoft.com

[‡] Some of this work done while visiting the School of Mathematics, Institute for Advanced Study, Princeton, on leave from the University of Washington. Research supported in part by NSF grants CCF-0343672 and CCF-0953155, and a David and Lucile Packard Foundation Fellowship. Email: guruswami@cmu.edu

Boolean functions $f : \{0,1\}^* \to \{0,1\}$. The function f may not be defined for all input lengths n, but it should be defined for infinitely many n. We only consider such n in the definition below.

Definition 1. *We say that f is $(1-\delta)$-hard* almost everywhere (a.e) *for $\mathcal{C}$ if for every $\mathcal{A} \in \mathcal{C}$ there is some n_0 so that for all $n \geqslant n_0$, $\Pr_{x \in \{0,1\}^n}[\mathcal{A}(x) \neq f(x)] \geqslant \delta$. We say f is $(1-\delta)$-hard* infinitely often (i.o) *for $\mathcal{C}$ if for every $\mathcal{A} \in \mathcal{C}$ there are infinitely many n such that $\Pr_{x \in \{0,1\}^n}[\mathcal{A}(x) \neq f(x)] \geqslant \delta$.*

Our main focus in this work is on the average-case hardness of NP languages, and specifically on *hardness amplification* results that convert an NP language with mild average-case hardness into another NP language that is much harder. Previous hardness amplification results for NP focus on the case when the class $\mathcal{C}$ is either the class P/poly of poly-size circuits [24, 15] or the class BPP of randomized polynomial time algorithms [26, 27, 6]. In this work we study the hardness of NP for the class P of *deterministic* polynomial time algorithms.[1] The following is our main result.

Theorem 1. *There exist constants μ and c_1, c_2 such that if there is some balanced function in* NP *which is $(1 - c_1(\log n)^{-\mu})$-hard a.e/i.o for* P *then there is a function in* NP *which is $\left(\frac{3}{4} + c_2(\log n)^{-\mu}\right)$-hard a.e/i.o for* P*.*

This is the first hardness amplification result for NP against P. This matches the hardness parameters shown by Trevisan in [26] for NP against BPP. Subsequently, this was improved to an amplification from $(1 - \frac{1}{\mathrm{poly}(n)})$-hardness to $(\frac{1}{2} + o(1))$-hardness by [27, 6]. Stronger results are known for amplification against P/poly (see Section 1.1).

A statement similar to Theorem 1 still holds if the original mildly hard function is only close to balanced, and has bias bounded by $(\log n)^{-C}$ for some constant $C = C(\mu) > 0$ (with $C(\mu) \to 0$ for $\mu \to 0$). If we assume hardness *almost everywhere* against slightly non-uniform algorithms, we can eliminate the balance hypothesis entirely, and still conclude $(\frac{3}{4} + o(1))$-hardness *infinitely often.*

Theorem 2. *There exist constants μ and c_1, c_2 such that if there is some function in* NP *which is $(1-2c_1(\log n)^{-\mu})$-hard a.e for* $\mathrm{P}/\log n$ *then there is a function in* NP *which is $(\frac{3}{4}+c_2(\log n)^{-\mu})$-hard i.o for* P*.*

In fact, it suffices to assume hardness against $\mathrm{P}/(\log n)^{\varepsilon}$ for any fixed $\varepsilon > 0$, and the claimed statement will hold with some $\mu = \mu(\varepsilon) > 0$.

Our techniques also yield a simple proof of amplification from $(1 - 1/\mathrm{poly}(n))$-hardness to $(3/4 + 1/\mathrm{poly}(n))$-hardness, for languages in PSPACE (or any class closed under XOR) against P. This improves on the amplification from $1 - \frac{1}{\mathrm{poly}(n)}$ to $\frac{7}{8} + \frac{1}{\mathrm{poly}(n)}$ shown by Trevisan [26]. Stronger amplification results are known for PSPACE and EXP against BPP and P/poly (see Section 1.1).

Theorem 3. *If there is some function in* PSPACE *which is $(1 - 1/\mathrm{poly}(n))$-hard a.e/i.o for* P *then there is a function in* PSPACE *which is $(\frac{3}{4} + 1/\mathrm{poly}(n))$-hard a.e/i.o for* P*.*

One can replace PSPACE by any complexity class which is closed under XOR. Though we note that for EXP and higher classes, very strong average case hardness is known unconditionally via diagonalization [10].

[1] Technically, P is the class of languages that admit deterministic polynomial time algorithms, and not the algorithms themselves. Likewise for BPP and P/poly. For ease of notation, we blur this distinction.

There is a well-studied connection between hardness amplification and error-correcting codes [25, 26]. Black-box hardness amplification amounts to taking a code with weak error-correction properties and converting it into a better error-correcting code which has a local decoding algorithm. A local decoding algorithm is one that can recover any bit of the message by querying the received word at a few locations. Buresh-Oppenheim *et al.* [6] define a family of codes called monotone codes and showed that error-correction for these codes can give amplification within NP. In what follows it will be helpful to think of the message $f \in \{0,1\}^N$ as the truth-table of a Boolean function on n bits, where $N = 2^n$, indexed by strings in $\{0,1\}^n$.

Definition 2. *An $[N, M]$ monotone code is a mapping $\mathrm{C} : \{0,1\}^N \to \{0,1\}^M$ such that each bit of the codeword $\mathrm{C}(f)$ is a monotone function in the bits of the message f.*

Typically, we want $M = 2^{\mathrm{poly}(n)}$. Also, we want our local decoder to run in time $\mathrm{poly}(n)$ when asked for any bit of the message, which implies that it can only read a tiny fraction of the received word. It is easy to show that monotone codes cannot have good distance for all messages. However it is possible to have good distance if we restrict ourselves to balanced messages. Further one needs to settle for recovering the message approximately as opposed to exactly, we will show in Section 4 that this is an inherent limitation of such codes.

To show hardness amplification against P, one needs the local decoding algorithm $\mathcal{A}$ to be *deterministic.* This is a non-trivial requirement since in the conventional setting where we wish to recover the message exactly, it is impossible for a deterministic local decoder to correct a constant fraction of errors. This is because an adversary could corrupt all the bits of the input that are probed by the decoder to recover a particular bit of the message. The locality of the decoder implies that this is a tiny fraction of the entire message. However, this does not rule out a deterministic local decoder that can recover all but a small fraction of the message bits, even at large error rates. Indeed, our result gives a monotone code with a deterministic local decoder that can recover $1 - o(1)$ of the message bits from $\frac{1}{4} - o(1)$ errors.

Theorem 4. *There exist constants μ, c_1, c_2 such that there is a monotone $[N, M]$ code C for every $N = 2^n$ where n is even, with $M = 2^n o(n)$ and a deterministic local decoder $\mathcal{A}$ which can $c_1(\log n)^{-\mu}$-approximately decode C up to distance $\frac{1}{4} - c_2(\log n)^{-\mu}$ on balanced messages. Further, $\mathcal{A}$ runs in time $O(n^{8+o(1)})$.*

We also prove limitations to the error-correcting capabilities of monotone codes which can be used in hardness amplification. For EXP, one can obtain optimal average-case hardness starting from worst-case hardness assumptions [25, 28]. One does not expect similar black-box reductions for NP, under standard complexity theoretic assumptions [5, 29, 30]. The crux of the reduction for EXP is the existence of codes which can be decoded exactly even at very high error-rates. We observe that there exist monotone codes which have similar error-correcting properties. However, for use in black-box amplification, one requires an upper bound on the complexity of encoding each bit; namely that the functions used have small certificates. We show that this places certain restrictions on the error-correcting capacity of the code. On one hand we show that starting from worst case hardness, one can only get very weak-average case hardness $(1 - 2^{-n(1-o(1))})$. At the other end of the spectrum, to get a hardness of $1 - \eta$ for some constant $\eta > 0$, we show that one must start by assuming $1 - \frac{1}{\mathrm{poly}(n)}$ hardness (see Section 4 for precise statements).

Our work raises the question of whether it is possible to deterministically amplify hardness from $1-\varepsilon$ to $\frac{1}{2}+\delta$ for NP and even for PSPACE. Proving such a result using the framework of error-correcting codes requires breaking the unique decoding barrier with a deterministic local decoder, and we believe that this will be fairly hard. In Section 5, we point out some of the technical obstacles that need to be overcome in proving such a result.

1.1 Previous work

There has been a long body of work in computer science devoted to studying hardness amplification in various scenarios. In addition to being a natural question in its own right, hardness amplification has important applications in cryptography and derandomization. The first such result is the famous XOR Lemma due to Yao, which asserts that computing the XOR of k independent copies of a mildly hard function f is much harder than computing f [31, 9]. There has been a long line of research that studies amplification for EXP against BPP and P/poly, motivated by derandomization [4, 17, 20, 21, 25, 28, 18, 19].

The study of hardness amplification within NP was initiated in a beautiful paper by O'Donnell [24] who showed that one can amplify $(1-1/\operatorname{poly}(n))$-hardness to $(1/2+1/n^{1/3})$-hardness for NP against polynomial size circuits. The key technical ingredient in this work was the analysis of a variant of Yao's XOR Lemma when the XOR function is replaced by a monotone function. The efficacy of a particular monotone function in such a setting was rather precisely tied to the noise sensitivity of the function. O'Donnell's result was improved by Healy, Vadhan, and Viola [15] who showed how to amplify $(1-1/\operatorname{poly}(n))$-hardness to $(1/2+1/\operatorname{poly}(n))$-hardness, also against polynomial size circuits. The non-uniformity in these proofs seemed inherent due to the use of Impagliazzo's powerful hard-core set lemma [17].

The first *uniform* hardness amplification result for NP, albeit against randomized algorithms, was due to Trevisan [26]. He was able to amplify $(1-1/(\log n)^{\alpha})$-hardness to $(3/4+1/(\log n)^{\alpha})$-hardness (which is identical to what we achieve for deterministic polynomial time algorithms in this work). Trevisan's proof was based on a "more uniform" version of the hard-core set lemma, but the amount of non-uniformity was large enough to preclude amplifying from hardness fractions greater than $1-1/(\log n)^{\alpha}$.

In [27], Trevisan built upon the result of [26] to achieve amplification from $(1-1/\operatorname{poly}(n))$-hardness to $(1/2+1/(\log n)^{\alpha})$-hardness, for NP against randomized polynomial time algorithms. An alternate proof of this result was given by [6] based on monotone codes, and the powerful direct product theorem of Impagliazzo *et al.* [18].

Our work seems to be the first to address the average-case hardness of NP against deterministic uniform algorithms. Hardness amplification against P was considered previously by Trevisan [26] who proved an amplification from $(1-1/\operatorname{poly}(n))$-hardness to $(7/8+1/\operatorname{poly}(n))$-hardness for PSPACE (or any class that is closed under XOR). Goldreich and Wigderson use diagonalization to prove the existence of languages in EXP which are hard on average a.e for P [10].

There has been a body of work exploring limitations to the hardness amplification parameters that are achievable via various kinds of black-box reductions [28, 24, 15, 29, 30, 6]. Our negative results in Section 4 complement these results.

1.2 Technical Contributions

Just as amplification against BPP requires uniform local decoding algorithms, amplification against P requires derandomized local decoding. Our main technical contribution is the construction of a monotone code that has an efficient, deterministic local decoder that can correct up to $\frac{1}{4} - o(1)$ errors. Note that $\frac{1}{4}$ is an upper bound on the unique-decoding radius of any non-trivial binary code, and due to the well-known connection between hardness amplification and coding theory, it is also the limit for results shown using uniform "black-box" reductions (see, for instance, the discussion in [27] or [28, Sec. 6]). Thus the $(3/4 + o(1))$-hardness we achieve is a natural barrier for our techniques.

Our construction and proof is based on majority logic decoding of an expander-based code [3, 11] and *Generalized Minimum Distance* (GMD) decoding of concatenated codes [8]. The first step in the encoding is to use the expander-based construction of Alon *et al.*[3] which gives a good distance code over a larger alphabet as the outer code. A standard way to reduce the alphabet size is to use concatenation with an *inner* binary code. The advantage is that the binary code is now applied to a small message size, and thus can be decoded even by brute force. By adapting GMD decoding to this setting, Guruswami and Indyk [11, 12] constructed binary codes that can decoded up to a $(1/4 - \varepsilon)$ fraction of errors in time linear in the block length. Trevisan [26, Theorem 7] used the same construction with a simpler (but less powerful) local decoder to amplify $(1 - 1/\operatorname{poly}(n))$-hardness to $(7/8 + 1/\operatorname{poly}(n))$-hardness for PSPACE against the class P. However, translating this to the monotone setting and achieving a decoding radius of $\frac{1}{4}$ locally and deterministically require overcoming some obstacles which we detail below.

A significant barrier is that the naive decoder will only correct a fraction 1/8 of errors and in order to decode up to a fraction $(1/4 - o(1))$ of errors, one needs to implement GMD decoding in a *local* manner. It is mentioned in [13, Remark 4.9] that the binary code construction from [11] can be used to boost hardness to $(3/4 + \varepsilon)$ for PSPACE in a uniform, deterministic way. However, this point is rather subtle and it is unclear if usual GMD algorithm can be made to work locally. The standard approach behind GMD decoding is to pass weights from the inner decodings along with the decoded symbols. The outer decoder then picks a threshold, erases all symbols with weights below the threshold, and decodes from the remaining symbols using a majority vote for each bit of the message. This is repeated for each possible threshold. This seems hard to implement locally and uniformly since the same threshold needs to be used for all bits of the message and we do not know in advance which threshold will work.

We bypass this problem by using the weights as *soft information* representing the confidence of each inner decoder and take a weighted majority. This scheme is local and deterministic since to recover a message bit, we only need to decode the inner codes for its neighbors, and then take a majority vote. To prove that this scheme works, we use the spectral properties of the underlying expander graph.

In order for the final function to belong to NP the inner code has to be monotone. However monotone codes are only guaranteed to have good distance when the messages are far apart, and when each of them is balanced or close to balanced. Indeed, the code we use has the property that nearby messages have similar encodings, and this is crucial to our analysis. On the decoding side, we can only guarantee that messages are recovered approximately even when the decoding of the inner code succeeds (unlike in the conventional setting), which makes the analysis of the outer decoder much harder.

Our local GMD decoding technique can also be used to amplify hardness against P within PSPACE (or any class closed under XOR) via concatenation with standard binary codes. In fact the proof is easier since none of the complications of monotone codes arise.

An alternative algorithm for soft-decoding expander-based codes of the kind we consider was given previously by Akavia and Venkatesan [2, 1]. They propose a deterministic, local error-reduction algorithm which gives similar results to our algorithm in the setting where the underlying expander graph has constant degree. However, their running times is 2^{k^2} where k is the degree, independent of the running time of the inner decoder. Our decoder uses poly(k) calls to the inner decoder. In the NP setting, the inner decoder is a brute-force search which takes time 2^k, while in the PSPACE setting the inner decoder runs in time poly(k), which lets us take $k = O(\log n)$ for NP, and $k = \text{poly}(n)$ for PSPACE. In the NP setting one could use their decoder and get similar results to what we get, but in the PSPACE setting, the results obtained would be weaker.

2 Preliminaries

2.1 Expander Graphs

A crucial ingredient in our construction is k-regular expander graphs with small second eigenvalue. We need graphs on N vertices where $N = 2^n$, whose degree is $k = \Theta(\log n)$ and whose second largest eigenvalue (in absolute value) λ is bounded by $k^{(1-\nu)}$ for some absolute constant $\nu > 0$. Finally, we need graphs which are highly explicit in the sense that the i'th neighbor of any vertex v can be computed in deterministic poly(n) time. The standard way to get graphs with good eigenvalue gap is to use the LPS construction of Ramanujan graphs. However, to construct such a graph with $N = 2^n$ vertices requires finding a large prime of size $N^{\Omega(1)}$ and it is not known how to perform this task in deterministic poly(n) time. Instead, we start with the construction of Margulis and its analysis due to Gabber-Galil [16, Chap. 8].

Definition 3. *Let N be a square. The vertex set of G_N is $\mathbb{Z}_{\sqrt{N}} \times \mathbb{Z}_{\sqrt{N}}$. Let*

$$T_1 = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix},\ T_2 = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix},$$

$$e_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\ e_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

The vertex $v = (x, y)$ is adjacent to $T_1 v, T_2 v, T_1 v + e_1, T_2 v + e_2$ and four other vertices obtained by the inverse transformations.

Note that since the transformations are invertible G_N has degree 8. The following result of Gabber and Galil bounds its second eigenvalue.

Theorem 5 ([16])**.** *The graph G_N satisfies $\lambda \leqslant 5\sqrt{2} < 8$.*

From this graph, we can construct the an expander with the parameters we need by taking a suitable power of the graph.

Theorem 6. *There is an absolute constant $\nu > 0$ such that for all even integers n, there exists a highly explicit graph on $N = 2^n$ vertices with degree k where $\log n \leqslant k \leqslant 8 \log n$ and second eigenvalue $\lambda \leqslant k^{(1-\nu)}$.*

Proof. Let $t = \lceil \frac{\log\log n}{3} \rceil$, and $k = 8^t$, so that

$$\log n \leqslant 8^t \leqslant 8 \log n. \tag{1}$$

Now take G_N^t, the t^{th} power of G_N, the (multi)graph where vertices are adjacent if they are connected by a length-t walk in G. It is easy to see that that this graph has N vertices, degree $k = 8^t$ and second eigenvalue $\lambda \leqslant (5\sqrt{2})^t = 8^{t(1-\nu)}$ where $\nu = 1 - \frac{\log 5\sqrt{2}}{\log 8} > 0$. □

The above construction works only for even n. Note that if $f : \{0,1\}^n \to \{0,1\}$ is $(1-\delta)$-hard for $\mathcal{C}$, then the function $\tilde{f} : \{0,1\}^{n+1} \to \{0,1\}$ defined by $\tilde{f}(x,b) = f(x)$ for $b \in \{0,1\}$ is $(1-\delta/2)$-hard for class $\mathcal{C}$. For our application to hardness amplification, we can thus assume that the input length n of f is even, and thus use the above expanders.

Finally, let $G(A,B,E)$ be the $k = d^t$-regular bipartite multigraph obtained by taking the double cover of G_N^t. The double cover of a graph $G(V,E)$ is the bipartite graph on the vertex set $V \times V$ which has edge (u,v) iff $(u,v) \in E$. By the expander mixing lemma [16], for $S \subseteq A$ and $T \subseteq B$,

$$\left| |E(S,T)| - \frac{k|S||T|}{N} \right| \leqslant \lambda \sqrt{|S||T|}.$$

In the above, $|E(S,T)|$ is the number of edges between vertices in S and T, where multiple edges are counted as many times.

2.2 Monotone Codes

We begin by formally defining distance and decodability for monotone codes. We use Δ to denote the normalized Hamming distance between two strings of equal lengths.

Definition 4. *A monotone code* C *has distance (α,β) if for any two balanced messages f and g such that $\Delta(f,g) \geqslant \alpha$, $\Delta(\mathrm{C}(f),\mathrm{C}(g)) \geqslant \beta$.*

Notice that we only require good distance between balanced messages that are far apart. This is not an inherent limitation of all monotone codes, [6] prove the existence of monotone codes that have good distance for any pair of balanced messages. However in Section 4, we show that monotone codes which can be used in black-box hardness amplification do have inherent limitations; they can only have good distance for messages which are far apart.

Definition 5. *Algorithm $\mathcal{A}$ α-approximately decodes* C *up to distance γ if when given a received word R, if there is a balanced f such that $\Delta(\mathrm{C}(f), R)) < \gamma$, $\mathcal{A}$ outputs g such that $\Delta(f, g) < \alpha$.*

The following generic construction of monotone codes is from [6].

Definition 6. *Given a monotone function $g : \{0,1\}^r \to \{0,1\}$, we define the $[k, k^r]$ code $g^{k,r}$ as follows. The codeword is indexed by $i = (i_1, \dots, i_r) \in [k]^r$. Given a message $x \in \{0,1\}^k$, the bit of $g^{k,r}(x)$ for index i is given by*

$$g^{k,r}(x)_i = g(x_{i_1}, \dots, x_{i_r}).$$

To compute a random index of $g^{k,r}(x)$, we sample an r-tuple of coordinates in x with repetition, and apply the function g. To analyze the properties of the code, we use the notion of noise-sensitivity of a Boolean function. Given $x \in \{0,1\}^r$ let $y \in \mathrm{N}_\delta(x)$ denote sampling a random vector y by flipping each bit of x independently with probability δ.

Definition 7. *For a function $g : \{0,1\}^r \to \{0,1\}$, the noise-sensitivity of g at δ denoted* $\mathrm{NS}_\delta(g)$ *is defined as*

$$\mathrm{NS}_\delta(g) = \Pr_{x \in \{0,1\}^r, y \in N_\delta(x)}[g(x) \neq g(y)].$$

The following Lemma is implicit in [6].

Lemma 7. *The code $g^{k,r}$ has distance $(\delta, \mathrm{NS}_\delta(g))$.*

Proof. Take two balanced messages $x, y \in \{0,1\}^k$ with $\Delta(x, y) = \delta$. If we pick a random $i \in [k]$,

$$\Pr[x_i = 1] = \Pr[y_i = 1] = \frac{1}{2}, \quad \Pr[x_i \neq y_i] = \delta.$$

Thus it follows that $\Delta(g^{k,r}(x), g^{k,r}(y)) = \mathrm{NS}_\delta(g)$. □

Note that Claim 7 of [6] about the list-decodability of $g^{k,r}$ can be derived from Lemma 7 by applying the Johnson bound. Another useful property of these codes is that nearby messages (even unbalanced) have similar encodings. This allows us to reason about the distance between the encodings of nearly-balanced messages by replacing them with nearby balanced messages.

Lemma 8. *For all $x, y \in \{0,1\}^k$ such that $\Delta(x, y) \leqslant \delta$, $\Delta(g^{k,r}(x), g^{k,r}(y)) \leqslant \delta r$.*

Proof. Pick a random index of $g^{k,r}$. Since x and y differ at no more than δk indices, the probability that sampling a random r-tuple gives different strings is bounded by δr, from which the claim follows. □

For each $r = 3^\ell$, the Recursive-Majority function $\mathrm{RecMaj} : \{0,1\}^r \to \{0,1\}$ is a function on r variables given by a depth-ℓ ternary tree of majority gates with the inputs at the leaves.

Fact 9 ([24]). *There is a constant $\gamma \in (0,1)$, such that* $\mathrm{NS}_{r^{-\gamma}}(\mathrm{RecMaj}) \geqslant \frac{1}{2} - r^{-\gamma}$.

In particular, one can take $\gamma = \frac{1}{20}$. Since this exact statement does not appear in [24], we provide a proof below, but the claim is well-known. We use the following Lemma about the noise sensitivity of the Recursive Majority .

Lemma 10. *(Proposition 11, [24]) Let $r = 3^\ell$ and let $\delta \geqslant (1.1)^{-\ell}$. Then*

$$\mathrm{NS}_\delta(\mathrm{RecMaj}) \geqslant \frac{1}{2} - \frac{1}{2\delta^{1.1}r^{0.15}}.$$

Proof. We set $\delta = r^{-0.05} = (3^{0.05})^{-\ell} \geqslant 1.1^{-\ell}$. We get that

$$\mathrm{NS}_\delta(\mathrm{RecMaj}) \geqslant \frac{1}{2} - \frac{r^{0.055}}{2r^{0.15}} \geqslant \frac{1}{2} - \frac{1}{r^{0.05}}.$$

□

The bound holds for all r, however it is meaningful only when $r > 2^{\frac{1}{\gamma}}$.

3 The Monotone Code construction

3.1 The expander-based construction

There is a standard way, originally due to [3], to use an expander graph to map a binary string into a string over a larger alphabet, such that a small Hamming distance between two binary strings is amplified into a much larger Hamming distance between their images.

Definition 8. *Given a string $f \in \{0,1\}^N$ and a k-regular bipartite (multi)-graph $G(A, B, E)$ with $|A| = |B| = N$ (we assume the elements of A and B are identified with $\{1, 2, \ldots, N\}$ in some fixed order), the string $G(f) \in (\{0,1\}^k)^N$ is defined as follows. For $j \in B$, let $G(f)_j$, the j'th symbol of $G(f)$, be the vector obtained by concatenating the symbols of f corresponding to positions in the neighborhood of j in A in some fixed order. In other words, if $\Gamma_1(j), \ldots, \Gamma_k(j)$ are the k neighbors of $j \in B$, then*

$$G(f)_j = f_{\Gamma_1(j)} \circ f_{\Gamma_2(j)} \circ \cdots \circ f_{\Gamma_k(j)}.$$

In other words, we associate the vertices of the LHS with the message symbols. To each vertex on the RHS we associate the concatenation of the symbols of to its neighbors (considered in some fixed order). To reduce the alphabet size, we will further encode the k bits in each symbol of $G(f)$ by an "inner" code.

We can now state the construction of our code C. For some constant $\tau > 1$ that we fix later, we pick a graph $G(A, B, E)$ with $|A| = |B| = 2^n$, degree $k = \Theta(\log n)$, and second largest eigenvalue at most $k^{1-\nu}$, as guaranteed by Theorem 6. We associate the message f of length N with the vertices on the LHS. We concatenate each symbol on the RHS (which lies in $\{0,1\}^k$) with the code $\mathrm{RecMaj}^{k,r}$, where we specify r shortly. Let $C(f)_j = \mathrm{RecMaj}(G(f)_j)$ denote the codeword obtained by applying this concatenation operation to the message $G(f)_j$. Thus the final encoding of f by C is given by $C(f) = \{C(f)_j\}_{j=1}^N$.

The parameter r is chosen to be a power of 3 which satisfies

$$\frac{k^{\frac{\nu}{2(1+\gamma)}}}{3} \leqslant r \leqslant k^{\frac{\nu}{2(1+\gamma)}} . \tag{2}$$

Thus $r = \Theta((\log n)^\rho)$ for some constant $\rho \in [0, 1]$. From the choice of parameters, each codeword of the inner code has block-length $k^r \leqslant (8 \log n)^{(\log n)^\rho} = o(n)$. Overall, the encoding length is $M = Nk^r = 2^n o(n)$. One can show that the distance of this code is close to $\frac{1}{2}$, as long as the messages are at distance at least $4r^{-\gamma}$ (here $\gamma > 0$ is the constant from Fact 9). We defer this proof.

Lemma 11. *The code C has distance $(4r^{-\gamma}, \frac{1}{2} - 4r^{-\gamma})$.*

3.2 Deterministic Local Decoding

We analyze the following decoding procedure. Let R_j denote the portion of the received word corresponding to $C(f)_j$. We decode R_j to a balanced string y_j such that $\Delta_j = \Delta(\mathrm{RecMaj}^{k,r}(y_j), R_j)$ is minimized. We then define a confidence estimate $\beta_j \in [0, 1]$ as

$$\beta_j = \max(1 - 4\Delta_j, 0) \ . \tag{3}$$

Note that if Δ_j is small, then the confidence estimate is close to 1. As Δ_j approaches $\frac{1}{4}$, which is approximately half the relative distance of the inner code, β_j drops to 0. This decoding is done by exhaustive search over all balanced messages, which takes time $2^k k^r = n^{8+o(1)}$.

After decoding the inner codes, for each vertex i on the LHS, we get a *vote* for the value of f_i from every vertex j in its neighborhood $\Gamma(i)$. We take a weighted majority of these votes, using the β_js are weights. Let $i \in A$, and let $\Gamma_0(i), \Gamma_1(i)$ denote the subset of $\Gamma(i)$ on the RHS which vote 0 and 1 respectively. We set $f_i = 1$ if

$$\sum_{j \in \Gamma_1(i)} \beta_j \geqslant \sum_{j \in \Gamma_0(i)} \beta_j \ , \tag{4}$$

and $f_i = 0$ otherwise.

The decoding procedure outlined above is deterministic. To recover a bit f_i, we only need to decode the inner codes for the k vertices in $\Gamma(i)$. The time taken to decode each inner code is bounded by $2^k k^r$. Thus the time needed to recover f_i is $2^k k^{r+1} = O(n^{8+o(1)})$.

We now analyze the error-correction. Let f be a balanced message. Assume that an adversary corrupts $\Delta(R, C(f)) = \eta \leqslant \frac{1}{4} - 4r^{-\gamma/2}$ fraction of the indices. Also, let η_j denote the error-rate on the inner code for a vertex $j \in B$. We partition B into two sets, $\mathsf{Balanced} \subset B$ is the set of vertices j such that the bias of $G(f)_j$ is at most $\frac{1}{k^{\nu/2}}$, and its complement $\mathsf{UnBalanced}$.

Lemma 12. *We have $|\mathsf{UnBalanced}| \leqslant \frac{1}{k^\nu} N$.*

Proof. Take S to be the set of 1s on the LHS, so that $|S| = \frac{1}{2}N$. Let

$$U_1 = \left\{ j \in B \mid \mathrm{wt}(G(f)_j) > \frac{1}{2} + \frac{1}{k^{\nu/2}} \right\},$$
$$U_0 = \left\{ j \in B \mid \mathrm{wt}(G(f)_j) < \frac{1}{2} - \frac{1}{k^{\nu/2}} \right\}.$$

Note that $|E(S, U_1)| > (\frac{1}{2} + \frac{1}{k^{\nu/2}})k|U_1|$. Hence $|E(S, U_1)| - \frac{k|U_1|}{2} > k^{1-\nu/2}|U_1|$. Applying the expander mixing lemma,

$$k^{1-\nu/2}|U_1| \leqslant k^{1-\nu}\sqrt{|S||U_1|} \ \Rightarrow\ |U_1| \leqslant \frac{1}{2k^{\nu}}N.$$

A similar calculation for U_0 gives $|\mathsf{UnBalanced}| = |U_0| + |U_1| \leqslant \frac{N}{k^{\nu}}$. □

Remark: We note that for this lemma, $|S|$ need not be perfectly balanced, it follows that $|\mathsf{UnBalanced}| \leqslant O(\frac{1}{k^{\nu}}N)$ as long as the bias of f is $o(\frac{1}{k^{\nu/2}})$.

Thus $|\mathsf{Balanced}| \geqslant N(1 - \frac{1}{k^{\nu}})$. We further divide Balanced into two parts, Good and Bad. We say that $j \in \mathsf{Balanced}$ lies in Good if $\Delta(y_j, G(f)_j) \leqslant 2r^{-\gamma}$ else $j \in \mathsf{Bad}$. These parts correspond to vertices where the inner decoding succeeds or fails respectively.

Lemma 13. *We have*

$$\sum_{j \in \mathsf{Good}} \beta_j - \sum_{j \in \mathsf{Bad}} \beta_j \geqslant r^{-\gamma}N.$$

Proof. Fix a $j \in \mathsf{Balanced}$. Let x_j denote the balanced message so that $\Delta(x_j, G(f)_j) \leqslant \frac{1}{k^{\nu/2}}$. By Lemma 8,

$$\Delta(\mathrm{RecMaj}^{k,r}(x_j), C(f)_j) \leqslant \frac{r}{k^{\nu/2}} \leqslant r^{-\gamma}. \tag{5}$$

where the last inequality follows from $r \leqslant k^{\frac{\nu}{2(1+\gamma)}}$.

Case 1: $j \in \mathsf{Bad}$. In this case, we show that if vertex j has a high confidence β_j, then η_j needs to be large. Formally, we show that if $j \in \mathsf{Bad}$, then

$$\eta_j \geqslant \frac{1 + \beta_j}{4} - 2r^{-\gamma}. \tag{6}$$

Since j is a bad vertex, $\Delta(y_j, G(f)_j) \geqslant 2r^{-\gamma}$, hence by the triangle inequality

$$\Delta(x_j, y_j) > 2r^{-\gamma} - \frac{1}{k^{\nu/2}} \geqslant 2r^{-\gamma} - r^{-(1+\gamma)} \geqslant r^{-\gamma} \quad (\text{since } r^{1+\gamma} < k^{\nu/2})$$

Hence their encodings are far apart,

$$\Delta(\mathrm{RecMaj}^{k,r}(x_j), \mathrm{RecMaj}^{k,r}(y_j)) \geqslant \frac{1}{2} - r^{-\gamma}$$

which together with (5) gives

$$\Delta(C(f)_j, \mathrm{RecMaj}^{k,r}(y_j)) \geqslant \frac{1}{2} - 2r^{-\gamma}\ .$$

Now we consider two cases. If $\beta_j > 0$, then by definition

$$\Delta(R_f, \mathrm{RecMaj}^{k,r}(y_j)) = \frac{1-\beta_j}{4}$$

hence

$$\eta_j = \Delta(C(f)_j, R_j) \geqslant \frac{1}{2} - 2r^{-\gamma} - \frac{1-\beta_j}{4} = \frac{1+\beta_j}{4} - 2r^{-\gamma}.$$

When $\beta_j = 0$, we know that

$$\Delta(\mathrm{RecMaj}^{k,r}(x_j), R_j) \geqslant \Delta(\mathrm{RecMaj}^{k,r}(y_j), R_j) \geqslant \frac{1}{4}.$$

Together with (5), this gives

$$\eta_j = \Delta(C(f)_j, R_j) \geqslant \frac{1}{4} - r^{-\gamma}.$$

Thus, in either case Equation 6 holds.

Case 2: $j \in \mathsf{Good}$. We now show that if a good vertex has low confidence, then η_j is close to $\frac{1}{4}$. When $j \in \mathsf{Good}$, we have

$$\Delta(R_f, \mathrm{RecMaj}^{k,r}(x_j)) \ \geqslant\ \frac{1-\beta_j}{4}.$$

and hence

$$\Delta(C(f)_j, R_j) \ \geqslant\ \frac{1-\beta_j}{4} - \frac{r}{k^{\nu/2}} \ \geqslant\ \frac{1-\beta_j}{4} - r^{-\gamma}.$$

Since the total fraction of errors is bounded by $\frac{1}{4} - 4r^{-\gamma/2}$,

$$\sum_{j\in\mathsf{Good}} \left(\frac{1-\beta_j}{4} - r^{-\gamma}\right) + \sum_{j\in\mathsf{Bad}} \left(\frac{1+\beta_j}{4} - 2r^{-\gamma}\right) \ \leqslant\ \left(\frac{1}{4} - 4r^{-\gamma/2}\right) N.$$

Rearranging terms, and using the fact that all but $\frac{1}{k^\nu}N$ vertices are balanced,

$$\sum_{j\in\mathsf{Good}} \beta_j - \sum_{j\in\mathsf{Bad}} \beta_j \geqslant N\left(2r^{-\gamma/2} - \frac{1}{k^\nu}\right) \geqslant r^{-\gamma/2}N$$

□

Thus overall, the total mass of the β_j's is higher on the good vertices than the bad vertices. Using the spectral properties of the graph, we can show that even locally, most vertices will receive more votes from good than bad vertices.

Lemma 14. *Define the set*

$$I_1 = \left\{ i \in A \;\middle|\; \sum_{j \in \Gamma(i) \cap \mathsf{Good}} \beta_j - \sum_{j \in \Gamma(i) \cap \mathsf{Bad}} \beta_j < \frac{r^{-\gamma/2}}{2} k \right\}$$

Then $|I_1| \leqslant 4r^{-(4+3\gamma)}N$.

Proof. Define the vector $z = (z_1, \ldots, z_N)$ where $z_j = \beta_j$ for $j \in \mathsf{Good}$, $z_j = -\beta_j$ for $j \in \mathsf{Bad}$ and $z_j = 0$ otherwise. Define the vector χ to be the indicator of the set I_1. Let A_G be the adjacency matrix of G. Then

$$\begin{aligned} \chi^t A_G z &= \sum_{i \in I_1} (A_G z)_i = \sum_{i \in I_1} \sum_{j \in \Gamma(i)} z_j \\ &= \sum_{i \in I_1} \left(\sum_{j \in \Gamma(i) \cap \mathsf{Good}} \beta_j - \sum_{j \in \Gamma(i) \cap \mathsf{Bad}} \beta_j \right) \\ &< \frac{r^{-\gamma/2}}{2} k |I_1|. \end{aligned}$$

On the other hand, let us expand z and χ in the orthonormal basis of the eigenvectors $\{v_1, \ldots, v_N\}$ of A (with corresponding eigenvalues $k = \lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_N$ with $\lambda = \max\{\lambda_2, |\lambda_N|\}$) as: $z = \sum_N \zeta_\ell v_\ell$ and $\chi^T = \sum_\ell \xi_\ell v_\ell$. Note that $\xi_1 = |I_1|/\sqrt{n}$, and $\zeta_1 = (\sum_i z_i)/\sqrt{N} \geqslant r^{-\gamma}\sqrt{N}$. Now

$$\begin{aligned} \chi^t A_G z &= \sum_\ell \xi_\ell \zeta_\ell \lambda_\ell \\ &= \xi_1 \zeta_1 d + \sum_{\ell \geqslant 2} \xi_\ell \zeta_\ell \lambda_\ell \\ &\geqslant \frac{|I_1|}{\sqrt{N}} \cdot r^{-\gamma/2} \sqrt{N} \cdot k - \lambda \langle \chi, z \rangle \\ &\geqslant r^{-\gamma/2} k |I_1| - \lambda \|\chi\| \|z\| \\ &\geqslant r^{-\gamma/2} k |I_1| - \lambda \sqrt{|I_1| N}. \end{aligned}$$

We thus conclude

$$\begin{aligned} & r^{-\gamma/2} k |I_1| - \lambda \sqrt{|I_1| N} < \frac{r^{-\gamma/2}}{2} k |I_1| \\ \Rightarrow \quad & |I_1| < \frac{4r^\gamma}{k^{2\nu}} N \leqslant 4r^{-(4+3\gamma)} N. \end{aligned}$$

□

Lemma 15. *The local decoding procedure* $24r^{-\gamma/2}$*-approximately decodes* C *up to distance* $\frac{1}{4} - 4r^{-\gamma/2}$.

Proof. After decoding the inner code, each vertex j on the right sends the symbol in y_j to the vertices on the right, as its estimate for $G(f)_j$. We say that an edge (i, j) is incorrect if the wrong symbol is sent to vertex i on the LHS by this procedure. We assume that all edges from vertices in Bad and vertices in $\mathsf{UnBalanced}$ are incorrect. If vertex $j \in \mathsf{Good}$, then we know that $\Delta(y_j, G(f)_j) \leqslant 2r^{-\gamma}$. Thus each vertex in Good can have some incorrect edges, but the number of such edges is at most $2r^{-\gamma}$ incorrect edges.

Let $I_2 \subset A$ be the set of vertices which have at least $\frac{r^{-\gamma/2}}{4}k$ neighbors in $\mathsf{UnBalanced}$. Since the total out-degree of the set $\mathsf{UnBalanced}$ is bounded by $k^{1-\nu}N$, we have

$$|I_2|\frac{r^{-\gamma/2}}{4}k \leqslant k^{1-\nu}N \;\Rightarrow\; |I_2| \leqslant \frac{4r^{\gamma/2}}{k^{\nu}}N \leqslant 4r^{-(2+1.5\gamma)}N.$$

Hence for any vertex in $A \setminus I_1 \cup I_2$,

$$\sum_{j\in\Gamma(i)\cap\mathsf{UnBalanced}} \beta_j \leqslant |\Gamma(i)\cap\mathsf{UnBalanced}| \leqslant \frac{r^{-\gamma/2}}{4}k.$$

$$\sum_{j\in\Gamma(i)\cap\mathsf{Good}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{Bad}} \beta_j \geqslant \frac{r^{-\gamma/2}}{2}k.$$

Hence

$$\sum_{j\in\Gamma(i)\cap\mathsf{Good}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{Bad}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{UnBalanced}} \beta_j \;\geqslant \frac{r^{-\gamma/2}}{4}k. \tag{7}$$

Finally, partition $\Gamma(i)\cap\mathsf{Good}$ into $\mathsf{Correct_i}$ and $\mathsf{Incorrect_i}$ based on whether or not the edge (i,j) is correct. The advantage of correct votes over incorrect votes at vertex i is given by

$$\begin{aligned}
\mathrm{adv}_i &\geqslant \sum_{j\in\mathsf{Correct_i}} \beta_j - \sum_{j\in\mathsf{Incorrect_i}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{Bad}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{UnBalanced}} \beta_j \\
&= \sum_{j\in\mathsf{Good}} \beta_j - 2\sum_{j\in\mathsf{Incorrect_i}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{Bad}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{UnBalanced}} \beta_j \\
&\geqslant \sum_{j\in\mathsf{Good}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{Bad}} \beta_j - \sum_{j\in\Gamma(i)\cap\mathsf{UnBalanced}} \beta_j - 2|\mathsf{Incorrect_i}| \\
&\geqslant \frac{r^{-\gamma/2}}{4}k - 2|\mathsf{Incorrect_i}|.
\end{aligned}$$

Hence for an incorrect decoding of i, we must have $|\mathsf{Incorrect_i}| \geqslant \frac{r^{-\gamma/2}}{8}k$. Denote the set of such vertices by I_3. The number of incorrect edges leaving all the vertices of Good is bounded by $(2r^{-\gamma})kN$. Thus,

$$|I_3|\frac{r^{-\gamma/2}}{8}k \;\leqslant\; (2r^{-\gamma})kN \;\Rightarrow\; |I_3| \leqslant 16r^{-\gamma/2}N.$$

Thus overall, the fraction of vertices that are wrongly decoded is bounded by

$$\delta \leqslant \frac{|I_1|+|I_2|+|I_3|}{N} \leqslant 4r^{-(4+3\gamma)} + 4r^{-(2+1.5\gamma)} + 16r^{-\gamma/2} \leqslant 24r^{-\gamma/2}.$$

□

This concludes our analysis of the code C. Note that the statement of Lemma 15 holds true for every r. However, it is only meaningful once the error in the decoding $24r^{-\gamma/2} \leqslant \frac{1}{2}$, which requires $r > (48)^{\frac{2}{\gamma}}$. Since $r = \Theta((\log n)^{\rho})$, this in turn requires n to be sufficiently a large constant.

We now complete the proof of Theorem 4.

Proof. We have bounded the encoding length by $2^n k^r = 2^n o(n)$, and the run-time of the decoder by $O(n^{8+o(1)})$. By our settings of k, r (Equations 1 and 2), there are constants μ, c_1', c_2' such that

$$c_1'(\log n)^{-\mu} \leqslant r^{-\gamma/2} \leqslant c_2'(\log n)^{-\mu}.$$

Hence by Lemma 15, it follows that the code C is $16c_1'(\log n)^{-\mu}$ approximately decodable up to distance $\frac{1}{4} - 4c_2'(\log n)^{-\mu}$. □

3.3 Proofs of the Hardness Amplification Results

We now prove Theorem 1. We note that the argument follows the outline suggested in [26, 6], and is provided here for completeness.

Proof. Let f be a balanced function in NP which is $1 - (\log n)^{-\mu}$ hard for P. Define a new function $g : \{0,1\}^n \times [k]^r \to \{0,1\}$ by $g = C(f)$. We first verify that $g \in \mathrm{NP}$.

We can think of g as indexed by $j \in [2^n]$ and $d = (d_1, \ldots, d_r) \in [k]^r$. Given j, we compute its neighbors in G_N^t, which takes time poly$(\log n)$ since G_N^t is highly explicit. Each neighbor d_i corresponds to a string $x_i \in \{0,1\}^n$. We have $g(j,d) = \mathrm{RecMaj}(f(x_1), \ldots, f(x_r))$. To certify that $g(j,d) = 1$, we pick all indices S where $f(x_i) = 1$ and provide certificates for these values. Since $f \in \mathrm{NP}$, these certificates can be verified in polynomial time. The verifier checks these certificates in polynomial time and then checks that the RecMaj function evaluates to 1, regardless of the other bits. It is easy to see that this also possible in polynomial time.

Assume that there is a deterministic polynomial time algorithm $\mathcal{R}$ that computes g within accuracy better than $\frac{3}{4} + 4(\log n)^{-\mu}$ and runs in time $O(n^c)$. Identifying $\mathcal{R}$ with the truth-table of the function that it computes, we have $\Delta(\mathcal{R}, C(f)) \leqslant \frac{1}{4} - 4(\log n)^{-\mu}$.

Then running the local decoding procedure $\mathcal{A}$ from Theorem 4 on $\mathcal{R}$ gives $\mathcal{R}'$ such that $\Delta(\mathcal{R}', f) \leqslant 24(\log n)^{-\mu}$. We further claim that $\mathcal{R}'$ is in computable in deterministic polynomial time. To prove this, note that while we do not have the truth-table of $\mathcal{R}$, every time we need to access a bit, we can run the algorithm $\mathcal{R}$. Thus the algorithm $\mathcal{R}'$ is a deterministic polynomial time algorithm which runs in time $n^{c+8+o(1)}$ and computes f with accuracy $\frac{3}{4} + 4(\log n)^{-\mu}$, contradicting our assumption about the hardness of f. □

To eliminate the balance hypothesis and prove Theorem 2, we use the padding argument used by Trevisan [27, Lemma 7], which in turn follows an idea from [24].

Proof. Assume that the function $f \in \mathrm{NP}$ is $(1 - c_1(\log n)^{-\mu})$-hard a.e for $\mathrm{P}/\log n$. We will define a padded version of f which we call f_{Pad}. Infinitely often, this function will be balanced and nearly as hard as f.

Consider input length $n = 2^t$ for all t large enough that $t + 1 < 2^t$. For input lengths $N \in \{2^t + 1, \ldots, 2^t + t + 1\}$, the function f_{Pad} is a suitably padded version of f on input length $n = 2^t$. For other input lengths, $f = 1$ is constant. Given a string x of length $N \in \{2^t + 1, \ldots, 2^t + t + 1\}$, we write it as $x = byz$ where b, y, z have length $1, n, r = N - (n + 1)$ respectively, where $r \in \{0, t\}$. We define $f_{\mathsf{Pad}}(byz)$ as follows:

- If $b = 0$, then $f_{\mathsf{Pad}}(x) = 1$ if y is one of the lexicographically first $\lfloor \frac{r}{t} 2^n \rfloor$ strings in $\{0,1\}^n$.
- If $b = 1$, then $f_{\mathsf{Pad}}(x) = f(y)$.

We observe that (ignoring rounding issues),

$$\Pr_{x \in \{0,1\}^N}[f_{\mathsf{Pad}}(x) = 1] = \frac{1}{2}\left(\frac{r}{t} + \Pr_{y \in \{0,1\}^n}[f(y) = 1]\right)$$

so there exists an r and a length N so that the function f_{Pad} is $\frac{1}{t}$-biased. Assume that for this N, we have an algorithm $\mathcal{R}$ so that $\Pr_{x \in \{0,1\}^N}[\mathcal{R}(x) = f_{\mathsf{Pad}}(x)] \geqslant 1 - \delta$. If follows that there exists a $z \in \{0,1\}^r$ such that

$$\Pr_{by \in \{0,1\}^{n+1}}[\mathcal{R}(byz) = f_{\mathsf{Pad}}(byz)] \geqslant 1 - \delta \quad \Rightarrow \quad \Pr_{y \in \{0,1\}^n}[\mathcal{R}(1yz) = f(y)] \geqslant 1 - 2\delta.$$

We claim that the function $C(f_{\mathsf{Pad}})$ is $(\frac{3}{4} + c_2(\log n)^{-\mu})$ hard for P i.o. Assume for contradiction that there is an algorithm $\mathcal{R}$ that can compute $C(f_{\mathsf{Pad}})$ for all input lengths $n > n_0$ (of f_{Pad}) with accuracy better than $(\frac{3}{4} + c_2(\log n)^{-\mu})$. Our goal is to show that this gives an algorithm in $\mathrm{P}/\log n$ which computes f with accuracy $1 - 2c_1(\log n)^{-\mu}$ i.o.

For $n = 2^t > n_0$, fix N so that f_{Pad} is $\frac{1}{t}$-biased. Applying Theorem 4, there is an algorithm $\mathcal{R}'$ (obtained by running the decoder $\mathcal{A}$ of C on the truth-table of $\mathcal{R}$) which computes f_{Pad} on length N with accuracy $1 - c_1(\log n)^{-\mu}$. Further, there is a string z with $|z| = r \leqslant \log n$ such that

$$\Pr_{y \in \{0,1\}^n}[\mathcal{R}'(1yz) = f(y)] \geqslant 1 - 2c_1(\log n)^{-\mu}.$$

This gives an algorithm $\mathcal{R}'' \in \mathrm{P}/\log n$ to compute f on input length $2^t > n_0$. The algorithm $\mathcal{R}''$ will receive as advice the string $z \in \{0,1\}^r$. To compute $f(y)$ it will return $\mathcal{R}'(1yz)$. If follows that $\Pr_{y \in \{0,1\}^n}[\mathcal{R}''(y) = f(y)] \geqslant 1 - 2c_1(\log n)^{-\mu}$. This contradicts the assumption that f is $1 - 2c_1(\log n)^{-\mu}$ hard a.e. for $\mathrm{P}/\log n$. □

The argument above is similar to ones used in [27], with one deviation: the advice string there consists only of the length r. The padding string z is not needed in the BPP setting as one can pad the input with a random string.

Finally, we sketch the proof of the amplification result for PSPACE (Theorem 3).

Proof. Let $f \in \{0,1\}^N$ with $N = 2^n$ be a function in PSPACE that is $\frac{1}{\mathrm{poly}(n)}$-hard for deterministic polynomial time algorithms. We will apply a similar construction to Section 3.1, but now with an expander graph of degree $k = n^{\Theta(1)}$. The k-bit symbols of the codeword $G(f)$ will now be encoded by a (usual) binary linear code C (instead of the monotone Recursive Majority code) that maps k bits to $k^{O(1)}$ bits and which can be unique decoded up to error rate $\frac{1}{4} - n^{-\Omega(1)}$ in time $\mathrm{poly}(k)$. (There are several choices for such a code, for example Reed-Solomon concatenated with a Hadamard code would work [25].) Note that each bit of the final codeword depends only on $n^{O(1)}$ bits of f and thus can be computed in PSPACE if f can be computed in polynomial space.

For decoding, we define the confidence parameter β_j for the decoding (of the code C) at each vertex on the RHS of the bipartite expander as we did in Equation (3) before. In the second step of the decoding, each vertex on the left recovers the bit given by a weighted majority vote over its neighbors, as in (4).

A vertex on the right is now considered good only if it recovers the corresponding symbol of $G(f)$ *exactly*, and is bad otherwise. We can prove statements similar to Lemma 13 showing that globally good vertices have more confidence overall, and Lemma 14 showing that, due to the expansion properties of the graph, this is also true locally for the neighborhoods of most vertices on the LHS. This implies that all but a small ($n^{-\Omega(1)}$) fraction of vertices on the LHS recover the correct bit of f when taking the weighted majority (according to weights β_j) of the bits suggested by their k neighbors. Note that the analysis is actually simpler since unlike in the monotone case, we do not have to worry about unbalanced vertices or wrong votes from good vertices. This gives a deterministic local decoder that runs in time $\mathrm{poly}(n)$ and can $\frac{1}{\mathrm{poly}(n)}$-approximately decode the message f from noisy codewords with up to a fraction $\frac{1}{4} - \frac{1}{\mathrm{poly}(n)}$ of errors. The natural connection between such codes and hardness amplification now implies Theorem 3. □

4 Limitations to Hardness Amplification via Monotone Codes

For EXP, it is possible to show optimal average case hardness starting from worst-case hardness assumptions. Roughly, amplifying worst-case hardness to $(1-\delta)$-hardness requires codes that are exactly locally decodable up to error rate δ. The crux of this reduction for EXP is the existence of codes which can be locally list-decoded (exactly) up to $\frac{1}{2} - o(1)$ errors (the notion of local list-decoding is subtle and needs to be defined carefully, see [25]).

In contrast, for NP we do not expect to amplify worst case hardness to even $1 - 1/\mathrm{poly}(n)$-hardness in a black-box manner, assuming standard complexity-theoretic hypotheses [5, 29, 30]. This suggests limitations to the error-correcting properties of monotone codes that can be used for amplification, namely that they cannot be decoded exactly. Our goal in this section is to prove these limitations directly. Our lower bounds are purely combinatorial in nature, hence they apply also to randomized and non-uniform decoders.

The outline of our argument is as follows: Buresh-Oppenheim *et al.* [6] show that monotone codes can have distance close to $\frac{1}{2}$ when restricted to balanced messages [6]. We observe that there exist such monotone codes with efficient, exact local decoders. However, for use in hardness amplification, we also need upper bounds on the complexity of encoding: specifically, one needs monotone codes that have small certificate complexity (see Definition 9). We show that with this requirement, exact decoding becomes impossible, and one has to settle for recovering the message approximately (Lemmas 17 and 18).

Lemma 16. *[6] For any $\eta < \frac{1}{4}$, there exist monotone codes that can be locally decoded exactly up to error rate η.*

Proof. Let $\mathrm{C} : \{0,1\}^N \to \{0,1\}^M$ be a (not necessarily monotone) code which is locally decodable up to error rate η. We now define a monotone code $\mathrm{C}' : \{0,1\}^N \to \{0,1\}^M$ as follows: on balanced messages, $\mathrm{C}'(x) = \mathrm{C}(x)$. For each $i \in [M]$ and balanced string $x \in \{0,1\}^N$, this defines a function $\mathrm{C}'_{\mathrm{i}}(x)$ taking values in $\{0,1\}$. We can now extend C'_{i} to a monotone function that is defined on the entire hypercube $\{0,1\}^N$. The local decoder for C' is just the local decoder for C. On any balanced message, it is guaranteed to recover from up to η fraction of errors. □

The reason why this construction will not give hardness amplification is because we do not have an upper bound on the complexity of encoding. While it is certainly sufficient if the encoding is local, this is a strong condition and is in fact restrictive. Healy *et al.*[15] show that one cannot hope for better that $\frac{1}{2} + \frac{1}{\mathrm{poly}(n)}$ hardness via such reductions. Healy *et al.* [15] observe that for black-box hardness amplification, the monotone code C must be such that each function C_i has small certificate complexity. Using this, they are able to break the $\frac{1}{2} + \frac{1}{\mathrm{poly}(n)}$ barrier.

Definition 9. *For $S \subseteq N$ and $x \in \{0,1\}^N$, let x_S denote the projection of x onto the coordinates in S. For a Boolean function $f : \{0,1\}^N \to \{0,1\}$, and $x \in f^{-1}(1)$, we say that x_S is a certificate for x if $f(y) = 1$ for all $y \in \{0,1\}^N$ such that $y_S = x_S$. We define*

$$N_1(f) = \max_{x \in f^{-1}(1)} \min |x_S|$$

over all certificates x_S for x.

Alternately $N_1(f)$ is the non-deterministic complexity of f, it is the maximum number of bits of x that a prover has to reveal to a verifier to convince her that $f(x) = 1$. It is easy to see that if f is monotone, then a minimal certificate for $x \in f^{-1}(1)$ can only consist of 1's. For a monotone code $\mathrm{C} : \{0,1\}^N \to \{0,1\}^M$, we define

$$N_1(\mathrm{C}) = \max_{i \in [M]} N_1(\mathrm{C}_i).$$

For black-box hardness amplification, we need $N_1(\mathrm{C})$ to be bounded by $\mathrm{poly}(n)$ and $N = 2^n$. This still allows each C_i to depend on all the 2^n message bits. Indeed this assumption that each bit can be encoded with small non-deterministic complexity seems to be the natural definition for amplification within NP. We will show that the condition $N_1(\mathrm{C}) = \mathrm{poly}(n)$ places bounds on the distance of the resulting monotone code.

Lemma 17. *A monotone code with $N_1(\mathrm{C}) \leqslant t$ can be exactly decoded from at most a fraction $\delta = \frac{t}{N}$ of errors.*

Proof. Let $x \in \{0,1\}^N$ be a random balanced message. Let A_0 and A_1 denote the subsets of indices where x_i is 0 and 1 respectively. Let $y \in \{0,1\}^N$ be generated from x by flipping a random bit from each of A_0 and A_1. It is clear that y is also a random balanced message.

For each $i \in [M]$, we bound $\Pr[\mathrm{C}_i(x) \neq \mathrm{C}_i(y)]$:

$$\begin{aligned}\Pr[\mathrm{C}_i(x) \neq \mathrm{C}_i(y)] &= \Pr[\mathrm{C}_i(x) = 0, \mathrm{C}_i(y) = 1] + \Pr[\mathrm{C}_i(x) = 1, \mathrm{C}_i(y) = 0] \\ &= 2\Pr[\mathrm{C}_i(x) = 1,\ \mathrm{C}_i(y) = 0]\end{aligned}$$

by symmetry. By monotonicity, flipping a bit from A_0 cannot cause C_i to change from 1 to 0. Let S be a certificate that $\mathrm{C}_i(x) = 1$ of size at most t. The bit flipped from A_1 must belong to S, else $\mathrm{C}_i(y) = \mathrm{C}_i(x) = 1$ since $y_S = x_S$. The probability of this event is bounded by $\frac{t}{N}$. Thus,

$$\Pr[\mathrm{C}_i(x) \neq \mathrm{C}_i(y)] \leqslant \frac{2t}{N}.$$

By linearity of expectation,

$$\Delta(\mathrm{C}(x), \mathrm{C}(y)) \leqslant \frac{2t}{N}.$$

Thus there exists a pair of balanced messages x, y satisfying this condition. Thus any unique decoder for C can tolerate at most an error rate of $\frac{t}{N}$, which is exponentially small in n. □

The above lemma rules out getting $1 - \frac{1}{\mathrm{poly}(n)}$ hardness starting from worst-case hardness via black-box reductions. One can use a similar argument to also bound from below the error of any *approximate* decoder that decodes from a constant fraction of error.

Lemma 18. *Let* C *be a monotone code with* $N_1(\mathrm{C}) \leqslant t$*. Any local decoder that can tolerate* η *fraction of errors can only recover balanced messages* $\frac{\eta}{2t}$*-approximately.*

Proof. Define $x \in \{0,1\}^N, A_0 \subset [N], A_1 \subset [N]$ as before. Pick random subsets $B_0 \subset A_0$ and $B_1 \subset A_1$ of size $\frac{\eta}{2t}N$ each and flip these bits to obtain y. As before, we have

$$\Pr[\mathrm{C}_i(x) \neq \mathrm{C}_i(y)] = 2\Pr[\mathrm{C}_i(x) = 1,\ \mathrm{C}_i(y) = 0]$$

Let S be a certificate that $\mathrm{C}_i(x) = 1$ of size at most t. Some bit from S must lie in B_1 in order for $\mathrm{C_i}(y)$ to equal 0. It is easy to show that the expected size of $B_1 \cap S$ is at most η, hence the probability that it is non-empty is bounded by η. Hence using linearity of expectation,

$$\Pr[\mathrm{C}_i(x) \neq \mathrm{C}_i(y)] \leqslant 2\eta \ \Rightarrow \ \Delta(\mathrm{C}(x), \mathrm{C}(y)) \leqslant 2\eta$$

Thus there exist two balanced messages x, y such that

$$\Delta(x, y) = \frac{\eta}{t}, \ \ \Delta(\mathrm{C}(x), \mathrm{C}(y)) \leqslant 2\eta.$$

Hence there exists a received word $R \in \{0,1\}^M$ such that

$$\Delta(R, \mathrm{C}(x)) \leqslant \eta, \ \ \Delta(R, \mathrm{C}(y)) \leqslant \eta.$$

Now consider the output of the decoder on input R. The decoded message is always at distance at least $\frac{\eta}{2t}$ from one of x and y. □

This lemma shows that to achieve $(1-\eta)$-hardness for any constant η by a black-box reduction, one has to start by assuming $1 - \frac{1}{\mathrm{poly}(n)}$ hardness.

5 Towards Stronger Hardness Amplification

Our work raises the question of whether it is possible to deterministically amplify hardness from $1-\varepsilon$ to $\frac{1}{2}+\delta$ for NP and even for PSPACE. Proving such a result in the error-correcting codes framework requires breaking the unique decoding barrier of $\frac{3}{4}$ with a deterministic local decoder. We believe that this calls for significantly new ideas and might be beyond the reach of present techniques.

Let us first consider the problem for PSPACE, where none of the added complications of monotone codes arise. While worst case to average case amplification even for PSPACE is impossible with a deterministic local decoder, amplification from $1-\varepsilon$ to $\frac{1}{2}+\delta$ might be possible. We begin by defining a deterministic local list-decoder; which in our setting is simply a list of local decoding algorithms that have oracle access to the received word R. Every message whose encoding is close to R is computed approximately by some machine in this list.

Definition 10. *An $[N, M]$ code* C *is α-approximately locally list-decodable up to error-rate γ if there exist $\ell(N) = (\log N)^{O(1)}$ deterministic algorithms $\mathcal{A}_1, \cdots, \mathcal{A}_\ell$ that when given oracle access to a received word $R \in \{0,1\}^M$ each compute functions $\mathcal{A}_i^R : [N] \to \{0,1\}$ such that:*

- *On a query $j \in [N]$, $\mathcal{A}_i$ returns $\mathcal{A}_i^R(j)$ in time $t(N) = (\log N)^{O(1)}$.*
- *For any f $in\{0,1\}^N$ such that $\Delta(R, C(f)) \leqslant \gamma$, there exists $[i] \in \ell$ so that $\Delta(\mathcal{A}_i^R, f) \leqslant \alpha$.*

Our definition is a restricted version of the one used by Sudan *et al.*[25], the main difference being that in their setting, the oracle machines are produced by a (randomized) decoding algorithm Decode which has oracle access to R and runs in time $(\log N)^{O(1)}$. If we restrict Decode to be deterministic, oracle access to R is of no use since an adversary can ensure that all its queries to R always return 0 by corrupting only $(\log N)^{O(1)}$ indices. This observation reduces their definition to our definition.

Removing Errors in List-Decoding: Proving a $1-\varepsilon$ to $\frac{1}{2}+\delta$ amplification result for PSPACE via the local decoding approach requires constructing a code C that is δ-approximately list-decodable deterministically at error rate $\frac{1}{2}-\varepsilon$. Currently, all the local list-decoders we are aware of have some error probability, there is some chance over the random choices of Decode that a received word is not computed by any algorithm $\mathcal{A}_i$. Thus a first step towards derandomization would be to eliminate errors in this stage of the decoder. It is not clear if this is even possible.

Getting Near-Linear Encoding Length: The property of having a deterministic local decoder enforces a very strong restriction on the rate of the code C (this was observed by Salil Vadhan).

Lemma 19. *If $\mathcal{C}$ is α-approximately locally decodable up to error rate γ for constants α, γ, then $M \leqslant N(\log N)^{O(1)}$.*

Proof. Let us pick a random message $f \in \{0,1\}^N$. We begin with $C(f)$ and corrupt it to get a received word R, such that every algorithm $\mathcal{A}_i$ makes at least αN mistakes in computing f.

We pick the locations probed by $\mathcal{A}_1$ on the first $3\alpha N$ indices and set them at random. It is easy to show that $\Delta(\mathcal{A}_1^R, f) \geqslant \alpha N$ with good probability. By the locality of $\mathcal{A}_1$, this set is of size at most $3\alpha N t(N)$. We repeat this for every $\mathcal{A}_i$ where $i \in [\ell(N)]$. Thus overall by corrupting only $3\alpha N t(N)\ell(N)$ indices, we can ensure that f is at distance αN from every $\mathcal{A}_i^R$. Hence we must have $\gamma M \leqslant 3\alpha N t(N)\ell(N)$. Since $t(N)$ and $\ell(N)$ are both $(\log N)^{O(1)}$, the claim follows. □

On one hand, we have relaxed the usual setting of locally decodable codes by allowing some errors in the decoding. But we are not aware of any codes in this setting that achieve $M = N(\log N)^{O(1)}$ even if we allow randomized decoders. To the best of our knowledge, currently the best encoding length achievable for locally list-decodable codes even with randomized decoders is $M = N^2$ [25, 26, 18, 19]. If we relax the locality requirement however, there are constant-rate codes which can be list-decoded at error rates close to $\frac{1}{2}$ in time polynomial in the block-length [14]. If we relax the list-decoding requirement, the construction in this paper achieves the desired parameters.

Deterministic Reduction from Search to Decision: Given a list of candidate functions one of which is correct, hardness amplification results typically identify the correct function using a search to decision reduction for that complexity class [28, 27]. Currently known reductions of this kind are randomized. One would need to derandomize them for deterministic hardness amplification.

All the obstacles mentioned above would also occur in the NP setting, with the additional restriction that the code is monotone. A more reasonable goal might be to amplify from $1 - 1/\operatorname{poly}(n)$ hardness to $\frac{3}{4} + 1/\operatorname{poly}(n)$ hardness for NP. The reason we are unable to achieve this is due to the fact that we do not know anything but a brute-force exponential decoder for the inner monotone code that can correct a linear fraction of errors. In order to prove such a result via our concatenation scheme, we would need an inner monotone code with a deterministic decoder that on messages of size k, can correct up to $\frac{1}{4}$ fraction of errors with accuracy $k^{-\gamma}$ for some constant $\gamma > 0$ in $\operatorname{poly}(k)$ time. Of course, we would need some additional properties of the inner code (like Lemma 8) to deal with imbalance in the messages.

Acknowledgments

We are thankful to Dang-Trinh Huynh-Ngoc for several discussions during initial stages of this work. We thank Valentine Kabanets and Rahul Santhanam for useful comments, and Rahul again for pointing us to [10]. We thank Salil Vadhan for many helpful discussions, and for pointing out Lemma 19. We thank an anonymous referee for the pointers to [2, 1]. We are grateful to another referee for pointing out an error in the choice of parameters in our earlier proof of Theorem 4, and for several useful comments that have improved the quality of our presentation.

References

[1] A. Akavia. *Learning Noisy Characters, Multiplication Codes and Cryptographic Hardcore Predicates.* PhD dissertation, MIT, EECS, Feb. 2008.

[2] A. Akavia and R. Venkatesan. Perturbation codes. Presented at IPAM Workshop on Locally Decodable Codes, 2006.

[3] N. Alon, J. Bruck, J. Naor, M. Naor, and R. M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, 1992.

[4] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[5] A. Bogdanov and L. Trevisan. On worst-case to average-case reductions for NP problems. *SIAM Journal of Computing*, 36(4):1119–1159, 2006.

[6] J. Buresh-Oppenheim, V. Kabanets, and R. Santhanam. Uniform hardness amplification in NP via monotone codes. ECCC TR06-154, 2006.

[7] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3^{rd} annual ACM symposium on Theory of computing (STOC'71)*, pages 151–158, 1971.

[8] G. D. Forney. Generalized minimum distance decoding. *IEEE Transactions on Information Theory*, 12:125–131, 1966.

[9] O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR-Lemma. ECCC TR95-050, 1995.

[10] O. Goldreich and A. Wigderson. On pseudorandomness with respect to deterministic observers. *ICALP Satellite Workshops*, pages 77–84, 2000.

[11] V. Guruswami and P. Indyk. Expander-based constructions of efficiently decodable codes. In *Proc. 42^{nd} IEEE Symposium on Foundations of Computer Science (FOCS'01)*, pages 658–667, 2001.

[12] V. Guruswami and P. Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005.

[13] V. Guruswami and V. Kabanets. Hardness amplification via space-efficient direct products. In *Proc. of the 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, pages 556–568, 2006. Citations refer to journal version to appear in *Computational Complexity.*

[14] V. Guruswami and M. Sudan. List decoding algorithms for certain concatenated codes. In *Proc. 32^{nd} annual ACM symposium on Theory of computing (STOC'00)*, pages 181–190, 2000.

[15] A. Healy, S. Vadhan, and E. Viola. Using nondeterminism to amplify hardness. *SIAM Journal on Computing*, 35(4):903–931, 2006.

[16] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math Soc.*, 43:439–561, 2006.

[17] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proc. 36^{th} Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 538–545. IEEE Computer Society, 1995.

[18] R. Impagliazzo, R. Jaiswal, and V. Kabanets. Approximately list-decoding direct product codes and uniform hardness amplification. In *Proc. 47^{th} Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 187–196, 2006.

[19] R. Impagliazzo, R. Jaiswal, V. Kabanets, and A. Wigderson. Uniform direct product theorems: Simplified, unified and derandomized. In *Proc. 40^{th} Ann. ACM Symposium on Theory of Computing (STOC'08)*, 2008.

[20] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: derandomizing the XOR lemma. In *Proc. 27^{th} annual ACM symposium on Theory of computing (STOC'97)*, pages 220–229, 1997.

[21] R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *Proc. 39^{th} Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 734–743, 1998.

[22] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[23] L. Levin. Universal search problems. *Problemy Peredachi Informatsii*, 9(3):265–266, 1973.

[24] R. O'Donnell. Hardness amplification within NP. *Journal of Computer and System Sciences*, 69(1):68–94, 2004.

[25] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62 (2):236–266, 2001.

[26] L. Trevisan. List-decoding using the XOR lemma. In *Proc. 44^{th} Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, page 126, 2003.

[27] L. Trevisan. On uniform amplification of hardness in NP. In *Proc. 37^{th} annual ACM symposium on Theory of computing (STOC'05)*, pages 31–38, 2005.

[28] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *IEEE Conference on Computational Complexity (CCC'02)*, pages 129–138, 2002. Citations refer to journal version to appear in *Computational Complexity.*

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

[30] E. Viola. On constructing parallel pseudorandom generators from one-way functions. In *Proc. IEEE Conference on Computational Complexity (CCC'05)*, pages 183–197, 2005.

[31] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.