The Complexity of Local List Decoding

Dan Gutfreund[1]* and Guy N. Rothblum[2]**

[1] Department of Mathematics and CSAIL, MIT
danny@math.mit.edu
[2] CSAIL, MIT
rothblum@csail.mit.edu

Abstract. We study the complexity of locally list-decoding binary error correcting codes with good parameters (that are polynomially related to information theoretic bounds). We show that computing majority over $\Theta(1/\epsilon)$ bits is essentially equivalent to locally list-decoding binary codes from relative distance $1/2 - \epsilon$ with list size at most poly$(1/\epsilon)$. That is, a local-decoder for such a code can be used to construct a circuit of roughly the same size and depth that computes majority on $\Theta(1/\epsilon)$ bits. On the other hand, there is an explicit locally list-decodable code with these parameters that has a very efficient (in terms of circuit size and depth) local-decoder that uses majority gates of fan-in $\Theta(1/\epsilon)$.
Using known lower bounds for computing majority by constant depth circuits, our results imply that every constant-depth decoder for such a code must have size almost exponential in $1/\epsilon$ (this extends even to sub-exponential list sizes). This shows that the list-decoding radius of the constant-depth local-list-decoders of Goldwasser *et al.* [STOC07] is essentially optimal.

Key words: locally-decodable codes, list-decodable codes, constant-depth circuits

1 Introduction

Error correcting codes are highly useful combinatorial objects that have found numerous applications both in practical settings as well as in many areas of theoretical computer science and mathematics. In the most common setting of error-correcting codes we have a message space that contains strings over some finite alphabet Σ (for simplicity we assume that all strings in the message space are of the same length). The goal is to design a function, which we call the *encoding function*, that encodes every message in the message space into a *codeword* such that even if a fairly large fraction of symbols in the codeword are

* Research was partially supported by NSF grant CCF-0514167. Part of this research was done while the author was at Harvard University and supported by ONR grant N00014-04-1-0478 and NSF grant CNS-0430336.
** Supported by NSF Grants CCF-0635297, NSF-0729011, CNS-0430336 and by a Symantec Graduate Fellowship, rothblum@csail.mit.edu.

corrupted it is still possible to recover from it the original message. The procedure that recovers the message from a possibly corrupted codeword is called *decoding*.

It is well known that beyond a certain fraction of errors, it is impossible to recover the original message, simply because the relatively few symbols that are not corrupted do not carry enough information to specify (uniquely) the original message. Still, one may hope to recover a list of candidate messages, one of which is the original message. Such a procedure is called *list-decoding*.

Typically, the goal of the decoder is to recover the entire message (or list of candidate messages) by reading the entire (possibly corrupted) codeword. There are settings, however, in which the codeword is too long to be read as a whole. Still, one may hope to recover any given individual symbol of the message, by reading only a small number of symbols from the corrupted codeword. This setting is called *local-decoding*, and both the unique and list decoding variants (as discussed above) can be considered.

Locally decodable codes, both in the unique and list decoding settings, have found many applications in theoretical computer science, most notably in private information retrieval [3, 12], and worst-case to average-case hardness reductions [17] (we elaborate on this application below). Furthermore, they have the potential of being used for practical applications, such as reliably storing a large static data file, only small portions of which need to be read at a time.

1.1 This Work

In this work we study the complexity of locally list decoding binary codes (i.e. where the alphabet is $\{0,1\}$). Let us proceed more formally. Let $C : \{0,1\}^M \to \{0,1\}^N$ be the encoding function of an error-correcting code.[3] A local list-decoder D for a code C gets oracle access to a corrupted codeword, and outputs a "list" of ℓ local-decoding circuits $D_1, \ldots, D_\ell$. Each D_a is itself a probabilistic circuit with oracle access to the corrupted codeword. On input an index $j \in [M]$, a circuit D_a from the list tries to output the j-th bit of the message. We say that the decoder is a $(1/2 - \epsilon, \ell)$-local-list-decoder, if for every $y \in \{0,1\}^N$ and $m \in \{0,1\}^M$, such that the fractional Hamming distance between $C(m)$ and y is at most $1/2 - \epsilon$, with high probability at least one of the D_a's successfully decodes *every* bit of the message y. Note that here $1/2 - \epsilon$ refers to the "noise rate" (or the list-decoding radius) from which the decoder recovers, and ℓ is the "list size": the number of decoding circuits, one of which makes the decoder recover every index correctly (with high probability). The quantity N/M which measures the amount of redundancy in the code is called *the rate* of the code.

Throughout this paper we think of a local list-decoder as receiving an "advice" index $a \in [\ell]$, running D to output D_a, and then running D_a to retrieve the j-th message bit. Note that by giving both D and the D_a's oracle access to the received word, and requiring them to decode individual symbols, we can

[3] Formally, we consider a family of codes one for each message length M. The parameters listed above and below, e.g. N, ϵ, ℓ, should all be thought of as functions of M. For the exact definition of locally list-decodable codes see Definition 3.

hope for decoders whose size is much smaller than N (in particular we can hope for size that is poly-logarithmic in N). See Definition 3 for a formal definition of locally list decodable codes.[4]

It is well known that for every (non-trivial) $(1/2-\epsilon, \ell)$-locally-list-decodable code, it must hold that $\ell = \Omega(1/\epsilon^2)$ [1, 9] (in fact this bound holds even for standard, non-local, list decoding). Thus, aiming to stay within polynomial factors of the best possible information theoretic parameters, our primary goal is to understand the complexity of decoding $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-locally-list-decodable binary codes that have polynomial rate (i.e. where $N(M) = \mathrm{poly}(M)$). We consider such codes to have "good" parameters (we elaborate on this choice below).

An explicit code with good parameters was given by Sudan, Trevisan and Vadhan [17]. The local-decoder for this code (namely the algorithm D as well as the circuits D_a) is in the complexity class NC^2 (i.e. its depth is poly-logarithmic in its input length). Explicit codes with local-decoders in the (strictly lower) class AC^0 (i.e. constant depth unbounded fan-in decoders) are also known [7, 5]. However, these codes do not have good parameters.[5] Specifically the Hadamard code has such a decoder [7], but its rate is exponential in M. This was improved by Goldwasser et al. [5] who showed codes with AC^0 local-list-decoders in which the rate is exponential in $1/\epsilon$ (but not in M). Furthermore, the circuit size of the AC^0 decoders for both these codes is exponential in $1/\epsilon$. For comparison, the size of the NC^2 decoder of [17] is $\mathrm{poly}(\log M, 1/\epsilon)$. In other words, the constant-depth decoder of [5] only matches the parameters of [17] (in terms of circuit size and information theoretic parameters) when $\epsilon \geq 1/\mathrm{poly}\log\log M$ (while in general ϵ can be as small as $1/\mathrm{poly}(M)$).

Our results. Our goal in this work is to understand the complexity of local-list-decoders of binary codes with good parameters. Specifically, we ask whether the exponential dependency on $1/\epsilon$ in both the rate and decoder size in [5] can be improved, and whether the parameters of [17] can be achieved by (small) constant-depth decoders. We show that while the rate of codes with constant-depth local-list-decoders can be improved, their circuit size cannot.

These conclusions follow from our main technical result, which shows that computing the majority function on $\Theta(1/\epsilon)$ bits is essentially equivalent to $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-local-list-decoding binary codes: Any circuit for a local-decoder of such a code can be used to construct a circuit of roughly the same size and depth that computes majority on $\Theta(1/\epsilon)$ bits. In the other direction, there is an explicit $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-locally-list-decodable code with a very efficient (in terms of size and depth) local-decoder that uses majority gates of fan-in $\Theta(1/\epsilon)$. This is stated (informally) in the following theorem.

[4] We would like to point out that we use $(1/2-\epsilon, \ell)$ to denote the relative distance and list size, whereas previous work (e.g. [17]) used (ϵ, ℓ) to denote the same quantities (for binary codes). We find this notation more useful, especially when we work with non-binary codes (which come up in our construction).

[5] We note that for non-binary codes, i.e. codes with large alphabets, one can construct codes with constant-depth local list-decoders and "good" parameters, see [5].

Theorem 1 (Informal). *If there exists a binary code with a* $(1/2-\epsilon, poly(1/\epsilon))$*-local-list-decoder of size* s *and depth* d*, then there exists a circuit of size* $poly(s)$ *and depth* $O(d)$ *that computes majority on* $\Theta(1/\epsilon)$ *bits.*

In the other direction, there exist (for every $\epsilon \geq 1/2^{\sqrt{\log(M)}}$*) explicit binary codes of polynomial rate, with a* $(1/2-\epsilon, poly(1/\epsilon))$*-local-list-decoder. The decoder is a constant depth circuit of size* $poly(\log M, 1/\epsilon)$ *with majority gates of fan-in* $\Theta(1/\epsilon)$*.*

The upper bound follows by replacing one of the ingredients in the construction of Goldwasser et al. [5], with a modification of the recent de-randomized direct-product construction of Impagliazzo *et al.* [10], thus improving the code's rate. Our main technical contribution is in the lower bound, where we show a reduction from computing majority over inputs of size $\Omega(1/\epsilon)$ to local-list-decoding binary codes with good parameters. In fact, our lower bound holds for any $(1/2-\epsilon, \text{poly}(1/\epsilon))$-local-list-decodable binary code, regardless of its rate. By known lower bounds on the size of constant-depth circuits that compute majority [15, 16], we obtain the following corollary.

Corollary 2 (Informal). *Any constant-depth* $(1/2-\epsilon, poly(1/\epsilon))$*-local list decoder for a binary code, must have size almost exponential in* $1/\epsilon$*. This holds even if the decoder is allowed* $\mod q$ *gates, where* q *is an arbitrary prime number.*

In particular, this result shows that the noise rate from which the constant-depth local-list-decoders of [5] recover is essentially optimal. And thus we get an exact characterization of what is possible with constant-depth decoders: up to radius $1/2 - 1/\text{poly}\log\log M$ locally-list-decodable codes with constant-depth decoders and good parameters exist, and beyond this radius they do not. We note that in fact we prove a stronger result in terms of the list size. We show that $(1/2-\epsilon, \ell)$-local-list-decoding with a decoder of size s and depth d, implies a circuit of size $\text{poly}(s, \ell)$ and depth d that computes majority on $O(1/\epsilon)$ bits. This means that even if the list size is *sub-exponential* in $1/\epsilon$, the size of the decoder still must be nearly exponential in $1/\epsilon$ (even if the decoder is allowed $\mod q$ gates).

Hardness amplification. Hardness amplification is the task of obtaining from a Boolean function f that is somewhat hard on the average, a Boolean function f' that is very hard on the average. By a beautiful sequence of works [17, 20, 19, 21], it is well known that there is a tight connection between binary locally (list) decodable codes and hardness amplification. Using this connection, we obtain limits (in the spirit of Corollary 2) on (black-box) hardness amplification procedures. We defer the statement of these results and a discussion to Section 5.

1.2 Related Work

The question of lower bounding the complexity of local-list-decoders was raised by Viola [22]. He conjectured that $(1/2-\epsilon, \ell)$-locally-list-decodable codes require computing majority over $O(1/\epsilon)$ bits,[6] even when the list size ℓ is exponential in $1/\epsilon$. Note that while exponential lists are not commonly considered in the coding setting (the focus instead is on polynomial or even optimal list sizes), they do remain interesting for applications to (non-uniform) worst-case to average-case hardness reductions. In particular, lower bounds for local-list-decoding with exponential lists, imply impossibility results for *non-uniform* black-box worst-case to average-case hardness reductions (see Section 5). In this paper we prove the conjecture for the case of sub-exponential size lists. While a proof of the full-blown conjecture remains elusive, there are results for other (incomparable) special cases:

Known Results for Non-Local *Decoders.* Viola [22] gave a proof (which he attributed to Madhu Sudan) of the conjecture for the special case of the standard *non-local* list-decoding setting. It is shown that a list-decoder from distance $1/2-\epsilon$ can be used to compute majority on $\Theta(1/\epsilon)$ bits, with only a small blow-up in the size and depth of the decoder. This result rules out, for example, constant-depth list-decoders whose size is $\mathrm{poly}(1/\epsilon)$. Note, however, that in the non-local list decoding setting the size of the decoder is at least N (the codeword length) because it takes as input the entire (corrupted) codeword. This means that the bound on the size of constant-depth decoders does not have consequences for fairly large values of ϵ. For example, when $\epsilon \geq 1/\log N$, the only implication that we get from [22], is that there is a constant-depth circuit of size at least $N \geq 2^{1/\epsilon}$ that computes majority on instances of size $1/\epsilon$. But this is trivially true, and thus we do not get any contradiction. In the local-decoding setting the decoders' circuits are much smaller and thus we can obtain limitations for much larger ϵ's. Indeed in this paper we rule out constant-depth decoders for $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-local-list-decoders for any ϵ smaller than $1/\mathrm{poly}\log\log N$ (and recall that this matches the construction of [5]).

Known Results for Specific *Codes.* Viola [22] also proved that there are no constant-depth decoders (with polynomial-size lists) for *specific* codes, such as the Hadamard and Reed-Muller codes. We, on the other hand, show that there are no such decoders for *any* code (regardless of the code's rate, and even with sub-exponential list size).

Known Results for Non-Adaptive *Decoders.* Recently (simultaneously and independently of our work), Shaltiel and Viola [18] gave a beautiful proof of the conjecture for the local-decoding setting, with ℓ exponential in $1/\epsilon$, but for the special case that the decoder is restricted to have *non-adaptive* access to the received word. (I.e., they give a lower bound for decoders that make all their queries to the received word simultanuously.) Our result is incomparable to [18]: we prove Viola's conjecture only for the case that ℓ is sub-exponential in $1/\epsilon$, but do so for *any* decoder, even an adaptive one. We emphasize that for important

[6] By "require" we mean that the decoding circuit can be used to construct a circuit of comparable size and depth that computes the majority function on $O(1/\epsilon)$ bits.

ranges of parameters the best codes known to be decodable in constant depth use *adaptive* decoders. In particular, the constant depth decoder of [5], as well as its improvement in this work, are adaptive. In light of this, it is even more important to show lower bounds for adaptive decoders.

1.3 On the Choice of Parameters

In this work codes with polynomial-rate are considered to have "good" parameters. Usually in the standard coding-theory literature, "good" codes are required to have *constant* rate.[7]. We note that, as far as we know, there are no known locally-decodable codes (both in the unique and list decoding settings) with constant rate (let alone codes that have both constant rate *and* have decoders that are in the low-level complexity classes that we consider here). The best binary locally decodable codes known have polynomial rate [17]. It is an interesting open question to find explicit codes with constant or even polylogarithmic rate.

Finally, we note that in this work we do not (explicitly) consider the query complexity of the decoder. The only bound on the number of queries the decoder makes to the received word comes from the bound on the size of the decoding circuit. The reason is that known codes with much smaller query complexity than the decoder size (in particular constant query complexity) have a very poor rate (see e.g. [25]). Furthermore, there are negative results that suggest that local-decoding with small query complexity may *require* large rate [12, 4, 14, 11, 23, 6].

2 Preliminaries

For a string $m \in \{0,1\}^*$ we denote by $m[i]$ the i'th bit of m. $[n]$ denotes the set $\{1,\ldots,n\}$. For a finite set S we denote by $x \in_R S$ that x is a sample uniformly chosen from S. For a finite alphabet Γ we denote by Δ_Γ the relative (or fractional) Hamming distance between strings over Γ. That is, let $x, y \in \Gamma^n$ then $\Delta_\Gamma(x,y) = \Pr_{i \in_R [n]}[x[i] \neq y[i]]$, where $x[i], y[i] \in \Gamma$. Typically, Γ will be clear from the context, we will then drop it from the subscript.

2.1 Circuit Complexity Classes

Boolean circuits in this work always have NOT gates at the bottom and unbounded AND and OR gates. Such circuits may output more than one bit. Whenever we use circuits with gates that compute other functions, we explicitly state so. For a positive integer $i \geq 0$, AC^{i} circuits are Boolean circuits of size poly(n), depth $O(\log^i n)$, and unbounded fan-in AND and OR gates (where n is the length of the input). $AC^i[q]$ (for a prime q) are similar to AC^{i} circuits, but augmented with $\bmod q$ gates. Throughout, we extensively use oracle circuits:

[7] We do remark that for applications such as worst-case to average-case reductions, polynomial or even quasi-polynomial rates suffice.

circuits that have (unit cost) access to an oracle computing some function. We sometimes interpret this function as a string, in which case the circuit queries and index and receives from the oracle the symbol in that location in the string.

2.2 Locally List-Decodable Codes

Definition 3 (Locally list-decodable codes). *Let Γ be a finite alphabet. An ensemble of functions $\{C_M : \{0,1\}^M \to \Gamma^{N(M)}\}_{M\in\mathbb{N}}$ is a $(d(M), \ell(M))$-locally-list-decodable code, if there is an oracle Turing machine $D[\cdot,\cdot,\cdot,\cdot]$ that takes as input an index $i \in [M]$, an "advice" string $a \in [\ell(M)]$ and two random strings r_1, r_2,*[8] *and the following holds: for every $y \in \Gamma^{N(M)}$ and $x \in \{0,1\}^M$ such that $\Delta_\Gamma(C_M(x), y) \le d(M)$,*

$$\Pr_{r_1}\left[\exists a \in [\ell] \text{ s.t. } \forall i \in [M] \ \Pr_{r_2}[D^y(a,i,r_1,r_2) = x[i]] > 9/10\right] > 99/100 \qquad (1)$$

If $|\Gamma| = 2$ we say that the code is binary. If $\ell = 1$ we say that the code is uniquely decodable. We say that the code is explicit if C_M can be computed in time poly$(N(M))$.

Remark 4. One should think of the decoder's procedure as having two stages: first it tosses coins r_1 and generates a sequence of ℓ circuits $\{C_a(\cdot,\cdot)\}_{a\in[\ell]}$, where $C_a(i, r_2) = D(a, i, r_1, r_2)$. In the second stage, it uses the advice a to pick the probabilistic circuit C_a and use it (with randomness r_1) to decode the message symbol at index i. In [17] the two-stage process is part of the definition, for us it is useful to encapsulate it in one machine (D).

In the sequel it will be convenient to simplify things by ignoring the first stage, and consider D as a probabilistic circuit (taking randomness r_2) with two inputs: the advice a and the index to decode i, with the property that (always) for at least one $a \in [\ell]$, $D(a,\cdot)$ decodes correctly every bit of the message (with high probability over r_2). Indeed if we hardwire any "good" r_1 (chosen in the first stage) into D then we are in this situation. This happens with probability at least 99/100. Thus in our proofs we will assume that this is the case, while (implicitly) adding 1/100 to the bound on the overall probability that the decoder errs. This simplification makes our proofs much clearer (since we do not have to deal with the extra randomness r_1).

2.3 Majority and Related Functions

We use the promise problem Π, defined in [22] as follows:
$\Pi_{Yes} = \{x : x \in \{0,1\}^{2k}$ for some $k \in \mathbb{N}$ and $weight(x) \le k-1\}$
$\Pi_{No} = \{x : x \in \{0,1\}^{2k}$ for some $k \in \mathbb{N}$ and $weight(x) = k\}$
where $weight(x)$ is the number of bits in x which are 1.

[8] The length of these random strings lower-bounds D's running time. Later in this work, when we consider D's with bounded running time, the length of these random strings will also be bounded.

We will extensively use the fact, proven in [22], that computing the *promise* problem Π on $2k$ bit inputs is (informally) "as hard" (in terms of circuit depth) as computing majority of $2k$ bits. This is stated formally in the claim below:

Lemma 5 ([22]). *Let $\{C\}_{M\in\mathbb{N}}$ be a circuit family of size $S(M)$ and depth $d(M)$ that solves the promise problem Π on inputs of size M. Then, for every $M \in \mathbb{N}$, there exists a circuit B_M of size poly$(S(M))$ and depth $O(d(M))$ that computes majority on M bits. The types of gates used by the B_M circuit are identical to those used by C_M. E.g., if C_M is an $AC^0[q]$ circuit, then so is B_M.*

3 Local-List-Decoding Requires Computing Majority

Theorem 6. *Let $\{C_M : \{0,1\}^M \to \{0,1\}^{N(M)}\}_{M\in\mathbb{N}}$ be a $(1/2 - \epsilon(M), \ell(M))$-locally-list-decodable code, such that $\ell(M) \leq 2^{\kappa\cdot M}$, and $1/N^{\delta_1} \leq \epsilon(M) \leq \delta_2$ for universal constants $\kappa, \delta_1, \delta_2$. Let D be the local decoding machine, of size $S(M)$ and depth $d(M)$.*

Then, for every $M \in \mathbb{N}$, there exists a circuit A_M of size poly$(S(M), \ell(M))$ and depth $O(d(M))$, that computes majority on $\Theta(1/\epsilon(M))$ bits. The types of gates used by the circuit A_M are identical to those used by D. E.g., if D is an $AC^0[q]$ circuit, then so is A_M.

Proof Intuition for Theorem 6. Fix a message length M and $\epsilon = \epsilon(M)$. We will describe a circuit B with the stated parameters that decides the promise problem Π on inputs of length roughly $1/\epsilon$. By Lemma 5 this will also give a circuit for computing majority.

We start with a simple case: assume that the (local) decoder D makes only non-adaptive queries to the received word. In this case we proceed using ideas from the proof of Theorem 6.4 in [22]. Take m to be a message that cannot be even approximately decoded[9] from random noise with error rate $1/2$. Such a word exists by a counting argument. Let $C(m)$ be the encoding of m. Let $x \in \Pi_{Yes} \cup \Pi_{No}$ be a Π-instance of size $1/2\epsilon$ (we assume w.l.o.g. throughout that $1/\epsilon$ is an integer). B uses x to generate a noisy version of $C(m)$, by XORing each one of its bits with some bit of x that is chosen at random. It then uses D to decode this noisy version of $C(m)$. If $x \in \Pi_{No}$, this adds random noise (error rate $1/2$), and the decoding algorithm cannot recover most of m's bits. If $x \in \Pi_{Yes}$, then each bit is noisy with probability less than $1/2 - 2\epsilon$, which means that w.h.p. the fraction of errors is at most $1/2 - \epsilon$, and the decoding algorithm successfully recovers every bit of m.

By comparing the answers of the decoding algorithm (or more precisely, every decoding algorithm in the list, by trying every possible advice) and the real bits of m in a small number of random locations, the algorithm B distinguishes w.h.p. whether $x \in \Pi_{Yes}$ or $x \in \Pi_{No}$.

[9] By this we mean that no decoder can recover (w.h.p.) a string that is, say, 1/3-close to m.

Note, however, that B as described above is *not* a standard algorithm for Π. This is because we gave B access to the message m as well as its encoding. Both of these are strings that are much larger than we want B itself to be. So our next goal is to remove (or at least minimize) B's access to m and $C(m)$, making B a standard circuit for Π. Observe that B as described above distinguishes whether x is in Π_{Yes} or in Π_{No} with high probability over the choices of D's random coins, the random locations in which we compare D's answers against m, and the random noise generated by sampling bits from x. In particular, there exists a fixing of D's random string as well as the (small number of) testing locations of m that maintains the advantage in distinguishing whether x comes from Π_{Yes} or Π_{No}, where now the probability is only over the randomness used to sample bits from x. So now we can hardwire the bits of m used to test whether D decodes the noisy version of $C(m)$ correctly (i.e. we got rid of the need to store the whole string m). Furthermore, after we fix D's randomness, *by the fact that it is non-adaptive*, we get that the positions in which B queries the noisy $C(m)$ are now also fixed, and *independent of* x. So we also hardwire the values of $C(m)$ in these positions (and only these positions) into B. For any x, we now have all the information to run B and conclude whether x is in Π_{Yes} or Π_{No}.

Next we want to deal with adaptive decoders. If we proceed with the ideas described above, we run into the following problem: suppose the circuit has two (or more) levels of adaptivity. The queries in the second level do not only depend on the randomness of the decoder, but also on the values read from the received word at the first level, and in particular they also depend on *the noise*. The noise in our implementation depends on the specific Π-instance x. This means that we cannot hardwire the values of $C(m)$ that are queried at the second level because they depend on x!

To solve this problem, we analyze the behavior of the decoder when its error rate changes in the middle of its execution. Specifically, suppose that the decoder D queries the received word in d levels of adaptivity. For every $0 \leq k \leq d$, we consider the behavior of the decoder when up to level k we give it access to the encoded message corrupted with error-rate $1/2 - 2\epsilon$, and above the k'th level we give it access to the encoded message corrupted with error-rate $1/2$. By a hybrid argument, there exists some level k, in which the decoder has a significant advantage in decoding correctly when up to the k'th level it sees error rate $1/2 - 2\epsilon$ (and error-rate $1/2$ above it), over the case that up to the $(k-1)$'th level the error-rate is $1/2-2\epsilon$ (and $1/2$ from k and up). We now fix and hardwire randomness for the decoder, as well as noise for the first $k-1$ levels (chosen according to error-rate $1/2 - 2\epsilon$), such that this advantage is preserved. Once the randomness of D and the noise for the first $k-1$ levels are fixed, the queries at the k-th level (but not their answers) are also fixed. For this k-th level we can proceed as in the non-adaptive case (i.e. choose noise according to x and hardwire the fixed positions in $C(m)$). We now have to deal with queries above the k'th level. At first glance it is not clear that we have gained anything, because we still have to provide answers for these queries, and as argued above, these may now depend on the input x and therefore the query locations as well as the restriction of $C(m)$ to these locations cannot be hard-wired. The key point is that for these "top" layers the error rate has changed to 1/2. So while we have no control on the query locations (as they depend on x) we do know their answers: they are completely random bits that have nothing to do with m or $C(m)$! Thus, B can continue to run the decoder, answering its queries (in the levels above the k'th) with random values. We thus obtain a circuit that decides membership in Π correctly with a small advantage. Since the number of adaptivity levels is only d (the circuit depth of the decoder), the distinguishing advantage of the k-th hybrid is at least $O(1/d)$, and in particular this advantage can now be amplified by using only additional depth of $O(\log(d))$. This gives a circuit that computes Π and concludes the proof. Due to space constraints we defer the formal proof of Theorem 6 to the full version of this paper [8].

By using known lower bounds for computing the majority function by $AC^0[q]$ circuits (for a prime q) [15, 16], we obtain the following corollary.

Corollary 7. *Let $\{C_M : \{0,1\}^M \to \{0,1\}^{N(M)}\}_{M\in\mathbb{N}}$ be a $(1/2-\epsilon, \ell)$-locally-list-decodable code (where ϵ is in the range specified in Theorem 6) with a decoder that can be implemented by a family of $AC^0[q]$ circuits of size $s = s(M)$ and depth $d = d(M)$. Then $s = 2^{(1/\epsilon)^{\Omega(1/d)}}/poly(\ell)$.*

4 Majority Suffices for Local-List-Decoding

Theorem 8. *For every $2^{-\Theta(\sqrt{\log M})} \leq \epsilon = \epsilon(M) < 1/2$, there exists a $(1/2 - \epsilon, poly(1/\epsilon))$-locally-list-decodable code $\{C_M : \{0,1\}^M \to \{0,1\}^{poly(M)}\}_{M\in\mathbb{N}}$ with a local-decoder that can be implemented by a family of constant depth circuits of size $poly(logM, 1/\epsilon)$ using majority gates of fan-in $\Theta(1/\epsilon)$ (and AND/OR gates of unbounded fan-in).*

Remark 9. The construction above only applies for $\epsilon \geq 2^{-\Theta(\sqrt{\log M})}$. Thus we fall slightly short of covering the whole possible range (since one can hope to get such codes for $\epsilon = 1/M^{\delta}$ for a small constant δ). We note, however, that the range of ϵ which is most interesting for us is between $1/poly \log M$ and $1/poly \log\log M$ (see the discussion in the introduction) which we do cover. We also mention that if one insists on codes with $\epsilon = 1/M^{\delta}$, then we can construct such codes with quasi-polynomial rate (in the full version [8] we state without proof the exact parameters of these codes).

The proof of Theorem 8 is omitted due to lack of space. In a nutshell, to prove the theorem we combine three codes. The first, by [5], is a binary locally-decodable code that can be uniquely decoded from a constant relative distance. The second code that we need is a modification of the de-randomized direct product code of [10]. The main reason that we need to modify the code of [10], is that in their construction the decoder needs to manipulate small dimension affine subspaces over finite fields. We do not know of a concise and unique representation of low-dimensional affine sub-spaces that can be computed and manipulated in AC^0. We instead represent such subspaces using randomly selected basis vectors and a shift vector (a concise, but not unique representation). This changes the code and allows decoding in AC^0. The third code in our construction is the well known Hadamard code with its local list-decoder given by Goldreich and Levin [7].

Informally, our code for Theorem 8 first encodes the message using the first code, it then encodes this encoding using the second code. Finally, it concatenates this code (i.e. encodes every symbol of it) using the third and final code. For details of the construction and the parameters it achieves see the full version of this work [8].

5 Hardness Amplification

In this section we describe our results regarding hardness amplification of Boolean functions. A more thorough discussion and extensions of our results appear in the full version [8].

Functions that are hard to compute *on the average* (by a given class of algorithms or circuits) have many applications, for example in cryptography or for de-randomization via the construction of pseudo-random generators (the "hardness vs. randomness" paradigm [2, 24, 13]). Typically, for these important applications, one needs a function that no algorithm (or circuit) in the class can compute on random inputs much better than a random guess. Unfortunately, however, it is often the case that one does not have or cannot assume access to such a "hard on the average" function, but rather only to a function that is "somewhat hard": every algorithm in the class fails to compute it and errs, but only on relatively few inputs (e.g. a small constant fraction, or sometimes even just a single input for every input length). A fundamental challenge is to obtain functions that are very hard on the average from functions that are somewhat hard on the average (or even just hard on the worst-case).

Let us be more precise. We say that a Boolean function $f : \{0,1\}^* \to \{0,1\}$ is δ-hard on the average for a circuit class $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ (where circuits in the set $\mathcal{C}_n$ have input length n), if for every large enough n, for every circuit $C_n \in \mathcal{C}_n$;

$$\Pr_{x\in_R U_n}[C_n(x) = f(x)] \leq 1 - \delta$$

The task of obtaining from a function f that is δ-hard for a class $\mathcal{C}$, a function f' that is δ'-hard for the class $\mathcal{C}$, where $\delta' > \delta$ is called hardness amplification from δ-hardness to δ'-hardness (against the class $\mathcal{C}$). Typical values for δ are small constants (close to 0), and sometimes even 2^{-n}, in which case the hardness amplification is from worst-case hardness. Typical values for δ' (e.g. for cryptographic applications) are $1/2 - n^{-\omega(1)}$.

The most commonly used approach to prove hardness amplification results is via reductions, showing that if there is a sequence of circuits in $\mathcal{C}$ that computes f' on more than a $1-\delta'$ fraction of the inputs, then there is a sequence of circuits in $\mathcal{C}$ that computes f on more than $1-\delta$ fraction of the inputs. An important family of such reductions are so-called fully-black-box reductions which we define next.

Definition 10. *A (δ, δ')-fully-black-box hardness amplification from input length k to input length $n = n(k, \delta, \delta')$, is defined by an oracle Turing machine Amp that computes a Boolean function on n bits, and an oracle Turing machine Dec that takes non-uniform advice of length $a = a(k, \delta, \delta')$. It holds that for every $f : \{0,1\}^k \to \{0,1\}$, for every $A : \{0,1\}^n \to \{0,1\}$ for which*

$$\Pr_{x \in_R U_n}[A(x) = Amp^f(x)] > 1 - \delta'$$

there is an advice string $\alpha \in \{0,1\}^a$ such that

$$\Pr_{x \in_R U_k}[Dec^A(\alpha, x) = f(x)] > 1 - \delta$$

where $Dec^A(\alpha, x)$ denotes running Dec with oracle access to A on input x and with advice α.

If Dec does not take non-uniform advice ($a = |\alpha| = 0$), then we say that the hardness amplification is uniform. *If Dec can ask all its queries to A in parallel (i.e. no query depends on answers to previous queries) then we say that the hardness amplification is non-adaptive.*

The complexity of *Dec* determines against which classes (of Boolean circuits or algorithms) we measure hardness (when we translate the reduction to a hardness amplification result). In particular if one wants to obtain hardness amplification against AC^0 or $AC^0[q]$ circuits, *Dec* must be implemented by such circuits.

It is well known [17, 20, 19, 21] that there is a tight connection between $(2^{-k}, \delta')$-fully-black-box hardness amplification (or in other words worst-case to average-case reductions) and binary locally (list) decodable codes. In particular a lower bound on the complexity of local-list-decoders implies a lower bound on the complexity of *Dec* in Definition 10. Using Theorem 6 we can show that worst-case to average-case hardness amplification with small non-uniform advice requires computing majority. This is stated formally in the theorem below:

Theorem 11. *If there is a $(2^{-k}, 1/2 - \epsilon(k))$-fully-black-box hardness amplification from length k to length $n(k)$ where Dec takes $a(k)$ bits of advice and can be implemented by a circuit of size $s(k)$ and depth $d(k)$, then for every $k \in \mathbb{N}$ there exists a circuit of size poly$(s(k), 2^{a(k)})$ and depth $O(d(k))$, that computes majority on $O(1/\epsilon(k))$ bits.*

It is known [15, 16] that low complexity classes cannot compute majority. Thus, Theorem 11 shows limits on the amount of hardness amplification that can be achieved by fully-black-box worst-case to average-case reductions (that do not use too many bits of advice), in which *Dec* can be implemented in low-level complexity classes.

Finally, we note that the worst-case lower bounds (which are actually mildly average-case lower bounds) of [15, 16] hold against *non-uniform* $AC^0[q]$. This means that it may be possible to get strong average-case hardness (e.g. as required for pseudo-randomness) by using a lot of non-uniformity in a fully-black-box reduction (i.e. a reduction in which Dec takes poly(k) bits of advice). Shaltiel and Viola [18] rule out such non-uniform fully-black-box reductions in the special case that Dec has only non-adaptive access to A.

As we mentioned, in the full version [8] we give a more thorough discussion of hardness amplification, together with extensions of Theorem 11 to hardness amplification from functions that are mildly hard on the average (rather than worst-case hard), as well as to reductions that are black-box but not necessarily fully-black-box.

Acknowledgements

We thank Shafi Goldwasser and Salil Vadhan for their assistance and insightful comments. Thanks to Russell Impagliazzo for helpful discussions and for supplying us with a manuscript of [10]. We also thank Ronen Shaltiel and Emanuele Viola for helpful discussions on the topics addressed in this work, and for supplying us with a manuscript of [18]. Finally, we thank the anonymous referees for their many helpful and insightful comments.

References

1. V. M. Blinkovsky. Bounds for codes in the case of list decoding of finite volume. *Problems of Information Transmission*, 22(1):7–19, 1986.
2. M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.
3. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.
4. A. Deshpande, R. Jain, T. Kavitha, J. Radhakrishnan and S. V. Lokam Better Lower Bounds for Locally Decodable Codes In *Proceedings of the IEEE Conference on Computational Complexity*, pages 184–193, 2002.
5. S. Goldwasser, D. Gutfreund, A. Healy, T. Kaufman, and G. N. Rothblum. Verifying and decoding in constant depth. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 440–449, 2007.
6. O. Goldreich, H. J. Karloff, L. J. Schulman, and L. Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. *Computational Complexity*, 15(3):263–296, 2006.
7. O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.
8. D. Gutfreund and G. N. Rothblum. The complexity of local list decoding. Technical Report TR08-034, Electronic Colloquium on Computational Complexity, 2008.
9. V. Guruswami and S. Vadhan. A lower bound on list size for list decoding. In *Proceedings of the 8th International Workshop on Randomization and Computation (RANDOM)*, pages 318–329, 2005.

10. R. Impagliazzo, R. Jaiswal, V. Kabanets, and A. Wigderson. Uniform direct-product theorems: Simplified, optimized, and derandomized. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 579–588, 2008.
11. I. Kerenidis and R. de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *Journal of Computer and System Sciences*, 69(3):395–420, 2004.
12. J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 80–86, 2000.
13. N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.
14. Kenji Obata. Optimal Lower Bounds for 2-Query Locally Decodable Linear Codes. In *Proceedings of the 5th International Workshop on Randomization and Computation (RANDOM)*, pages 39–50, 2002.
15. A. A. Razborov. Lower bounds on the dimension of schemes of bounded depth in a complete basis containing the logical addition function. *Akademiya Nauk SSSR. Matematicheskie Zametki*, 41(4):598–607, 623, 1987.
16. R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 77–82, 1987.
17. M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.
18. R. Shaltiel and E. Viola. Hardness amplification proofs require majority. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 589–598, 2008.
19. L. Trevisan. List-decoding using the XOR lemma. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 126–135, 2003.
20. L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):361–364, 2007.
21. E. Viola. The complexity of constructing pseudorandom generators from hard functions. *Computational Complexity*, 13(3-4):147–188, 2005.
22. E. Viola. *The complexity of hardness amplification and derandomization.* PhD thesis, Harvard University, 2006.
23. S. Wehner and R. de Wolf. Improved Lower Bounds for Locally Decodable Codes and Private Information Retrieval. In *Proceeding of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1424–1436, 2005.
24. A. C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.
25. S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 266–274, 2007.