SIGACT News Complexity Theory Column 25

Lane A. Hemaspaandra
Dept. of Computer Science, University of Rochester
Rochester, NY 14627, USA lane@cs.rochester.edu

Introduction to Complexity Theory Column 25

The 2000s are upon us, and your Complexity Theory Column celebrates their start with Madhu Sudan writing on list decoding. Up next issue is Ulrich Hertrampf writing on algebraic acceptance mechanisms for polynomial-time machines.

Guest Column: List Decoding: Algorithms and Applications.

Madhu Sudan[1]

Abstract

Over the years coding theory and complexity theory have benefited from a number of mutually enriching connections. This article focuses on a new connection that has emerged between the two topics in the recent years. This connection is centered around the notion of "list-decoding" for error-correcting codes. In this survey we describe the list-decoding problem, the algorithms that have been developed, and a diverse collection of applications within complexity theory.

1 Introduction

The areas of coding theory and complexity theory have had a long and sustained history of interesting connections. Early work on computation in the presence of noise built on these connections. Recent successes of complexity theory, showing IP=PSPACE and giving PCP characterizations of NP have relied on connections with coding theory either implicitly or explicitly. The survey article of Feigenbaum [10] gives a detailed account of many connections and consequences.

Over the last few years a new strain of connections has emerged between coding theory and complexity theory. These connections are different from the previous ones in that they rely especially on the qualitative strength of the decoding algorithms; and in particular on the ability to recover from large amounts of noise. The first work in this vein seems to be that of Goldreich and Levin [12], whose work describes (implicitly) an error-correcting code and gives a highly efficient algorithm to decode the code from even the slightest non-trivial amount of information. They then use the algorithm to give a generic construction of hard-core predicates from an arbitrary one-way function. Subsequently, there have been a number of other such results providing even more powerful decoding algorithms and deriving other applications from these algorithms to complexity theory.

---

[1]© Madhu Sudan, 2000. Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA. E-mail: madhu@mit.edu.

The main theme common to these works is the application of a new notion for decoding of error-correcting codes called *list decoding*. List decoding formalizes the notion of error-correction, when the number of errors is potentially very large. Borrowing the terminology from the area of information communication, recall that to transmit information over a noisy channel, the transmitter transmits a codeword of an error-correcting code. This transmitted word is corrupted by the noisy channel, and the receiver gets some corrupted word that we will call "the received word." If the number of errors that occur during transmission is very large, then the received word may actually be closer to some codeword other than the transmitted one. Under the mandate of list-decoding, the receiver is required to compile a list of all codewords within a reasonable sized Hamming ball around the received word (and not just *the* nearest one). The list-decoding is declared to be successful if this list includes the transmitted word.

This notion of list decoding was proposed by Elias [9] in the 1950's. However till recently no non-trivial[2] list decoding algorithms were known for any error-correcting code. Of late, we have seen a spurt of efficient list-decoding algorithms; and equally interestingly, a diverse collection of applications of these list-decoders to complexity theoretic problems. In this survey, we describe some of these results. First we start with some definitions.

2 Error-correcting codes and list-decoding

A block error-correcting code $\mathcal{C}$ is a collection of strings called codewords, all of which have the same length, over some finite alphabet Σ. The three basic parameters describing the code are the size of the alphabet, denoted q; the length of the codewords n; and an information parameter k, where the number of codewords is q^k. Such a code is succinctly referred to as an $(n, k)_q$ code[3]. If Σ has a field structure imposed on it, then Σ^n may be viewed as a vector space. If additionally $\mathcal{C}$ forms a linear subspace of Σ^n, then $\mathcal{C}$ is termed a linear code and denoted an $[n, k]_q$ code[4]. Almost all codes dealt with in this article will be linear codes.

In order to ensure that the code helps in the recovery from errors, one designs codes in which any two codewords differ from each other in large number of locations. Formally, let the *Hamming distance* between strings x and y from Σ^n, denoted $\Delta(x, y)$, be the number of coordinates where x and y differ from each other. The the distance of a code $\mathcal{C}$, typically denoted $d(\mathcal{C})$, is the minimum, over all pairs of non-identical codewords in $\mathcal{C}$, of the distance between the pair.

One of the first observations that can be made about a code $\mathcal{C}$ with distance d is that it can unambiguously correct $\frac{d-1}{2}$ errors, i.e., given any word $r \in \Sigma^n$, there exists at most one codeword $c \in \mathcal{C}$ such that $\Delta(r, c) \leq \frac{d-1}{2}$. It is also easy to find a word r such there exist two codewords at distance $\frac{d+1}{2}$ from it, so one can not improve the error bound for unambiguous decoding. However it was realized early on that unambiguous decoding is not the only useful notion of recovery from error. Elias [9] proposed the notion of *list decoding* in which a decoding algorithm is expected to output a list of all codewords within a given distance e from a received word $r \in \Sigma^n$. If the list of words output is relatively small, then one could consider this to be a reasonable recovery from error. Algorithmically, this problem is stated as follows:

Definition 2.1 (List decoding problem for a code $\mathcal{C}$)
INPUT: *Received word $r \in \Sigma^n$, error bound e.*

[2]Here triviality is used to rule out both brute-force search algorithms and the unique decoding algorithms.

[3]Sometimes in the literature this would be referred to as an $(n, q^k)_q$ code, where the second parameter counts the number of messages as opposed to say the "length" of the message.

[4]Note the subtle change in the notation.

OUTPUT: *A list of all codewords $c_1, \ldots, c_m \in \mathcal{C}$ that differ from r in at most e places.*

As usual, the goal is to solve the list decoding problem efficiently: i.e., in time polynomial in n. However this is only possible if the output size is polynomially bounded in n. This motivates the following, purely combinatorial, question.

Definition 2.2 (List decoding problem: Combinatorial version)
For every c, determine the function $e_c(n, k, d, q)$ such that for every $(n, k)_q$ code $\mathcal{C}$ of distance $d(\mathcal{C}) = d$, and for every received word $r \in \Sigma^n$, there are at most $(qn)^c$ codewords in the Hamming ball of radius e around r.

We would like to study the asymptotic growth of e_c when we say fix the ratio k/n and d/n and let $n \to \infty$. If it makes sense, we would then like to study e_∞, the limit of e_c as $c \to \infty$. It turns out that e_∞ is fairly well-understood and this will be described in Section 3. Somewhat coincidentally, for a variety of codes, the list decoding problem can be solved in polynomial time provided $e < (1 - o(1))e_\infty$. These results will be described in Section 4.

Before concluding this section, we present one more version of the algorithmic list-decoding problem that has been studied in the literature. This version is motivated by the question: Are there sub-linear time list-decoding algorithms for any error-correcting code? At first glance, linear time in n appears to be a lower bound on the running time since that is the amount of time it takes to read the input, or even the time to output one codeword. However, by being specifying the input implicitly and allowing the output also to be specified implicitly, one is no longer subject to these trivial lower bounds on computation time. The notion of implicit representation of the input can be formalized by using an "oracle" to represent the input—when queried with an index i, the oracle responds with the ith bit of the received word. The notion of implicit representation of the output is somewhat more involved. Roughly we would each element of the output list to be described succinctly by a program that computes any one coordinate of the codeword. However these programs are allowed to be randomized; furthermore, they are allowed to make oracle calls to the implicit input when attempting to compute any one coordinate of the output. The notion of implicit representation of an output (codeword) is thus formalized by the concept of "probabilistic oracle machines," machines that are allowed to make oracle calls (to the received word). Under this formalism, the list decoding problem may now be rephrased as:

Definition 2.3 (List decoding problem: Implicit version)
IMPLICIT INPUT: *Oracle access to received word $r : \{1 \ldots, n\} \to \Sigma$, error bound e.*
OUTPUT: *A list of all codewords $c_1, \ldots, c_m \in \mathcal{C}$, represented implicitly by probabilistic oracle machines $M_1, \ldots, M_m$ working with oracle access to r, that differ from r in at most e places.*

We remark that these implicit representations have now become common and useful in the theory of computation (e.g., in works on program testing/self-correcting, PCPs etc.). They allow for more modular use of algorithmic ideas; and results expressed in these terms deserve attention. It turns out that for the list decoding problem highly efficient solutions exist in this model for some codes—essentially in time polynomial in $\log n$. This efficiency translates into some very useful applications in complexity, and this will be described in the forthcoming sections.

3 Status of the combinatorial problem

We first sketch the status of the combinatorial problem described above. It is easily seen that $e_\infty(n, k, d, q) \geq e_0(n, k, d, q) = \frac{d-1}{2}$ (the unambiguous error-correction radius). Also, if the parameters are such that an $(n, k)_q$ code with distance d does exist, then it is also possible to get one along with a received word that has exponentially many codewords at distance d from it. Informally, this suggests $e_\infty(n,k,d,q) \le d$ (though to be formal, we should first let n go to infinity, and then let c go to infinity!). Thus it seems reasonable to believe that e_c may be of the form αd, where α is some universal constant between 1/2 and 1, and possibly a function of c. Unfortunately, the answer is not so simple: e_c turns out to be a function also of n and q and surprisingly is not very dependent on c. Roughly, (if q is very large), then $e_c \approx n - \sqrt{n(n-d)}$. (Even the task of performing a sanity check on this expression, i.e., to verify that $d/2 \le n - \sqrt{n(n-d)} \le d$, takes a few moments!) Some insight into this expression: If $d = o(n)$, then $n - \sqrt{n(n-d)}$ is well approximated by $d/2$. However, if d is large, i.e., $d = n - o(n)$, then the bound on e is also $n - o(n)$ and so the list-decoding radius is not limited by "half the distance" barrier in this case.

The following theorem essentially refines the above expression to take into account small values of q. Recall that the "Plotkin bound" of coding theory shows that error-correcting codes with $d \ge (1-1/q)n$ have only polynomially many codewords and hence are not very interesting. Thus it makes sense to compare d and e as fractions of n' rather than n. The theorem statement replaces all occurrences of n in the expression above by $n' = (1-1/q)n$.

Theorem 1

1. *Let n,k,d,q,e satisfy $d \le n'$ and $e < \left(1-\sqrt{1-\frac{d}{n'}}\right)n'$ where $n' = \left(1-\frac{1}{q}\right)n$. Then, for every $(n,k)_q$ code $\mathcal{C}$ with $d(\mathcal{C}) \ge d$ and for every received word r, there are at most qn^2 codewords within a Hamming distance of e from r.*

2. *For every n,d,q,e such that $d < n'$ and $e \ge (1+\epsilon)\left(1-\sqrt{1-\frac{d}{n'}}\right)n'$ where $n' = \left(1-\frac{1}{q}\right)n$, there exists a (non-linear) $(n,k)_q$ code $\mathcal{C}$ of distance at least d and a received word r, such that there are exponentially many codewords (with the exponent growing with ϵn) within a Hamming distance of e from r.*

Note: The theorem above appears explicitly in Goldreich, Rubinfeld, and Sudan [13]. The crucial direction, Part (1) above, is a q-ary extension of the "Johnson bound" in coding theory. Johnson proves this bound only for the binary case, but the extension to the q-ary case seems to be implicitly known to the coding theory community [27, Chapter 4, page 301].

Theorem 1 yields an asymptotically tight result on e_∞. To study this asymptotic limit, let us minimize some of the parameters above. First notice that k does not play any role in Part (1) of the theorem. So let $e_c(n,\cdot,d,q)$ denote the minimum over k of $e_c(n,k,d,q)$. Now further fix $d = n(1-\frac{1}{q})\delta$ and let $q = q(n)$ be any function of n. Now let

$$\epsilon_c(\delta) = \lim_{n\to\infty} \frac{e(n,\cdot,(1-\frac{1}{q})\delta n, q)}{(1-\frac{1}{q})n}.$$

Let $\epsilon_\infty(\delta) = \lim_{c\to\infty} \epsilon_c$. Then Theorem 1 above can be summarized as:

Corollary 1 *For $\delta \in [0,1]$, $\epsilon_2 = \epsilon_\infty(\delta) = 1 - \sqrt{1-\delta}$.*

In the next section, we will describe algorithmic results which come close to matching the combinatorial results above.

4 Specific codes and performance of list-decoding algorithms

We start by introducing the reader to a list of commonly used (and some not so commonly used) error-correcting codes. In the first five codes below, q will be assumed to be a prime power, and Σ will be a finite field on q elements.

Hadamard codes. For any k, the Hadamard code $\mathcal{H}_k$ is a $(n = q^k, k)_q$ code with distance $(1-\frac{1}{q})q^k$, obtained as follows: The message is a k dimensional vector α. The codeword is indexed by space of k-dimensional vectors. The β-th symbol in the encoding of α is their inner product $\sum_{i=1}^{k} \alpha_i \cdot \beta_i$.

(Generalized) Reed Solomon codes. Here the message is thought of as specifying a degree $k-1$ polynomial by giving it k coefficients. The encoding evaluates the polynomial at n distinct points in the finite field. (It follows that q has to be at least n.) The fact that two distinct degree $k-1$ polynomials may agree on at most $k-1$ points yields that the distance is at least $n-k+1$. The fact that there do exist distinct degree $k-1$ polynomials that agree at any given subset of $k-1$ places shows that the distance is exactly $n-k+1$.

Reed Muller codes. Reed Muller codes may be viewed as a common generalization of Reed Solomon codes and Hadamard codes. For parameters m and l, the Reed Muller code has $k = \binom{m+l}{m}$ and $n = q^m$. The message is viewed as specifying a degree l polynomial over m variables. The encoding gives the evaluation of this polynomial at every possible input. For $l < q$, the codewords are at a distance of at least $(1 - l/q)n$ from each other.

Algebraic geometric codes. These codes are also generalizations of the generalized Reed Solomon codes. Description of the construction of these codes is out of scope. All we will say is that they yield $(n,k)_q$ codes with distance at least $d = n - k - n/(\sqrt{q} - 1)$ when q is a square. This beats the parameters achieved by the random code, provided q is large enough!

Concatenated codes. This term refers to any code obtained by a certain process, called concatenation of codes, that derives a new code from two given codes. Specifically, given an "outer" code over a q^k-ary and an "inner" code of dimension k over a q-ary alphabet, the concatenated codeword corresponding to a given message is obtained by first encoding the message using the outer code, and then encoding each symbol of the resulting string by the inner code. If the outer code is an $(n_1, k_1)_{q^{k_2}}$ code and the inner code is an $(n_2, k_2)_q$ code, then the concatenated code is an $(n_1 n_2, k_1 k_2)_q$ code. If d_1 is the distance of the outer code and d_2 is the distance of the inner code, then the concatenated code has minimum distance $d_1 d_2$. In this section we will consider codes obtained by concatenating a Reed-Solomon, Reed-Muller or Algebraic-Geometry code as the outer code with a Hadamard code as the inner code.

Chinese remainder codes. These codes are an aberration in the class of codes we consider in that they are not defined over any single alphabet. Rather the i-th symbol is from an alphabet p_i, where $p_1, \ldots, p_n$ are n distinct primes arranged in increasing order. The messages of this code are integers between 0 and $K-1$, where $K = \prod_{j=1}^{k} p_j$. The encoding of a message is the n-tuple of its residues modulo $p_1, \ldots, p_n$. In the coding theory literature, this code is often referred to as the Redundant Residue Number System code. As an easy consequence of the Chinese Remainder Theorem, we have that the message can be inferred given any k of the n residues making this a code of distance $n-k+1$. If $p_1 \approx p_n \approx p$, then one may view this code as approximately an $(n,k)_p$ code.

4.1 List decoding results: Explicit version

For some families of codes, it is possible to get algorithms that perform list-decoding in polynomial time for e, the number of errors, as large as the bound in Theorem 1. The following theorem lists this family of results.

Theorem 2 *Let $\mathcal{C}$ be an $(n, k)_q$ code with designed distance*[5] $d = \delta n'$, *where* $n' = (1 - 1/q)n$. *Further, if $\mathcal{C}$ is either a (1) Hadamard code, (2) Reed-Solomon code, (3) Algebraic-geometric code, (4) Reed-Solomon or algebraic-geometry code concatenated with a Hadamard code, or (5) Chinese remainder code, then it has a polynomial time list decoding algorithm that decodes from* $e < (1 - \sqrt{1-\delta})n'$ *errors.*

Remarks:

1. Proofs of any of these results is out of scope. We will simply give some pointers here.
2. Note that the result for Hadamard codes is trivial, since this code has only n codewords. Thus a brute force search algorithm that lists all codewords and then evaluates their distance against the received word to prune this list, runs in time $O(n^2)$.
3. An algorithm for list-decoding the Reed-Solomon codes when $e < n - \sqrt{2n(n-d)}$ was given by Sudan[31] based on earlier work of Ar, Lipton, Rubinfeld, and Sudan [2]. For the case of explicit list-decoding problem this was the first non-trivial list-decoder that was constructed for any code. The tight result above is from the work of Guruswami and Sudan [17].
4. The first list-decoder for algebraic-geometry codes was given by Shokrollahi and Wasserman [29]. There error bound matched that of [31]. The tight result above is again from [17].
5. It is easy to combine non-trivial list-decoders for the outer code and inner code to get some non-trivial list decoding of a concatenated code. However, such results will not obtain the tight result described above. The tight result above is also due to Guruswami and Sudan [18].
6. A list decoder correcting $n - \sqrt{2kn\frac{\log p_n}{\log p_1}}$ errors for the Chinese remainder codes was given by Goldreich, Ron, and Sudan [14]. Boneh [7] recently improved this bound to correct $n - \sqrt{kn\frac{\log p_n}{\log p_1}}$ errors. Even more recently Guruswami, Sahai, and Sudan [16] improve this to correct $n - \sqrt{nk}$ errors.

4.2 List decoding results: Implicit version

For the implicit list-decoding problem, some fairly strong results are known for the cases of Hadamard codes, Reed-Muller codes and consequently for concatenated codes. We describe these results in the next two theorems. For the case of binary Hadamard codes, Goldreich and Levin [12] gave a list-decoding algorithm when the received word is specified implicitly. They consider the case when the number of errors is arbitrarily close to the limit of Theorem 1, and in particular takes the form $e = (\frac{1}{2} - \gamma)n$. They give a randomized algorithm whose running time is $\mathrm{poly}(\log n, \frac{1}{\gamma})$

[5] In at least two cases, that of algebraic-geometry codes and algebraic-geometry codes concatenated with Hadamard code, the code designed to have distance d, may turn out to have larger minimum distance. Typical decoding algorithms are unable to exploit this extra bonus, and work only against the designed distance of the code; in fact, there may be no short proof of the fact that the code has this larger minimum distance. This explains the term "designed distance" of a code.

and reconstructs explicitly a list of all messages (note that message lengths are $O(\log n)$ for the Hadamard code) that come within this error of the received word. Subsequently, this algorithm was generalized to general q-ary Hadamard codes by Goldreich, Rubinfeld, and Sudan [13]. This yields the following theorem.

Theorem 3 *There exists a probabilistic list decoding algorithm in the implicit input model for Hadamard codes that behaves as follows: For an $(n,k)_q$ code $\mathcal{C}$, given oracle access to a received word r, the algorithm outputs a list that includes all messages that lie within a distance of e from the received word. The running time of the algorithm is a polynomial in $\log n, \log q$ and $\frac{n}{n-\frac{q}{q-1}e}$.*

For the case of Reed-Muller, equally strong list-decoding results are known, now with the output representation also being implicit. Arora and Sudan [3] provided such a list-decoder provided the error bound satisfies $e < n(1-(1-d/n)^{\epsilon})$, for some positive ϵ. Sudan, Trevisan, and Vadhan [32] improved this bound to a tighter bound of $e < (1-\sqrt{1-d/n})n$, thus yielding the following theorem.

Theorem 4 *There exists a probabilistic list decoding algorithm in the implicit input and implicit output model for Reed-Muller codes that behaves as follows: For a $(n = q^m, k = \binom{m+l}{l})_q$ Reed-Muller code $\mathcal{C}$, given oracle access to a received word r, the algorithm outputs a list of randomized oracle programs that includes one program for each codeword that lies within a distance of e from the received word, provided $e < (1 - O(\sqrt{l/q}))n$. The running time of the algorithm is a polynomial in m, l and $\log n$.*

As pointed out earlier. it is easy combine list-decoding algorithms for outer and inner codes to get a list-decoding algorithm for a concatenated code. By concatenating a Reed-Muller code with some appropriately chosen Hadamard code, one also obtains the following result, which turns out to be a handy result for many applications. In fact, all results of Section 5 use only the following theorem.

Theorem 5 *For every q, ϵ and k, if $n \geq \mathrm{poly}(k, q, \frac{1}{\epsilon})$ there exists an $(n,k)_q$ code with a polynomial time list-decoding algorithm for errors up to $(1 - 1/q - \epsilon)n$. Furthermore, the algorithm runs in time polynomial in $\log k$ and $1/\epsilon$ if the input and output are specified implicitly.*

The above result, specialized to $q = 2$ is described explicitly in [32]. The general codes and list-decoding algorithm can be inferred from their proof.

5 Applications in complexity theory

Algorithms for list-decoding have played a central role in a variety of results in complexity theory. Here we enumerate some (all?) of them.

Hardcore predicates from one-way permutations. A classical question lying at the very foundations of cryptography is the task of extracting one hard Boolean function (predicate), given any hard one-way function. Specifically given a function $f : \{0,1\}^k \rightarrow \{0,1\}^k$ that is easy to compute but hard to invert, obtain a predicate $P : \{0,1\}^k \rightarrow \{0,1\}$ such that $P(x)$ is hard to predict given $f(x)$. Blum and Micali citeBM showed how it was possible to extract one such hard predicate from the Discrete Log function and used it to generate cryptographically strong sequences of pseudo-random bits. A natural question raised is whether this ability to extract hard predicates is special to the Discrete Log function, and if not, could such a hard predicate be extracted from every one-way function f.

At first glance this seems impossible. In fact, given any predicate P, it is possible to construct one-way functions f such that $f(x)$ immediately gives $P(x)$. However, this limitation is inherited from the deterministic nature of P. Goldreich and Levin [12] modify the setting to allow the predicate P to be randomized. Specifically, they allow the predicate P to be a function of x and an auxiliary random string r. P is considered hardcore for f if $P(x, r)$ is hard to predict with accuracy better than $\frac{1}{2} + \epsilon$ given $f(x)$ and r. (The function P is said to be predictable with accuracy α if the output of some polynomial sized circuit agrees P on α fraction of the inputs.) They then show that this minor modification to the problem statement suffices to construct hardcore predicates from any one-way function.

One parameter of some interest in the construction of hardcore predicates is the length of the auxiliary random string. Let l denote this parameter. The initial construction of [12] (which was based on their list-decoding algorithm for the Hadamard code) sets $l = k$. Impagliazzo [19] gives an substantial improvement to this parameter, achieving $l = O(\log k + \log \frac{1}{\delta})$, by using the list-decoders for Reed-Solomon and Hadamard codes. It turns out that both constructions can be described as a special case of a generic construction using list-decodable codes. The construction goes as follows: Let $\mathcal{C}$ be a $(n, k)_2$ binary code as given by Theorem 5 with ϵ set to some poly(δ). Then the predicate $P(x, r) = (\mathcal{C}(x))_r$ (the rth bit of the encoding of x) is as hard as required. The proof follows modularly from the list-decodability property of $\mathcal{C}$. Specifically, if for some x, the prediction of the circuit agrees with $P(x, \cdot)$ for a $\frac{1}{2} + \epsilon$ fraction of the values of $i \in \{1, \ldots, n\}$, then one can use the list decoder to come up with a small list of candidates that includes x. Further, the knowledge of $f(x)$ tells us how to find which element of the list is x. The hardness of inverting f thus yields that there are not too many x's for which the circuit can predict $P(x, \cdot)$ with this high an accuracy. Now to see the effectiveness of Theorem 5, note that the extra input has length $\log n$, which by the theorem is only $O(\log k + \log \frac{1}{\delta})$.

Aside: Recall that the early results of Blum and Micali [6] and Alexi, Chor, Goldreich, and Schnorr [1] that gave hardcore predicates for specific one-way functions (namely, Discrete Log and RSA) actually use $l = 0$ extra randomness. It would be interesting to see if these specific results can also be explained in terms of list-decoding.

Predicting witnesses for NP-search problems. Consider an NP-complete relation such as 3-SAT. Kumar and Sivakumar [24], based on earlier work of Gal, Halevi, Lipton, and Petrank [11], raise the question of whether it is possible to efficiently construct a string x that has non-trivial proximity to a witness of the given instance. For general relations in NP they show that if some string with distance $\frac{1}{2} + \epsilon$ can be found; then NP=P. They show this result using the list-decoding algorithms for Reed Solomon and Hadamard codes. Again this result can be explained easily using Theorem 5 as follows: Construct an NP relation whose instances are, say, instances of satisfiability but whose witnesses are encodings, using a code obtained from Theorem 5, of satisfying assignments. Given a string that has close proximity to a valid witness, one can recover a small set of strings one of which includes the witness.

Amplifying hardness of Boolean functions. One of the recent success stories in complexity theory is in the area of finding complexity theoretic assumptions that suffice to derandomize BPP. In a seminal result in this direction, Impagliazzo and Wigderson [22], show that a strong form of the assumption "E does not have subexponential sized circuits" implies BPP=P. One important question raised in this line of research is on amplification of the hardness of Boolean functions. Specifically, given a Boolean function $f : \{0,1\}^l \to \{0,1\}$, transform it into a Boolean function $f' : \{0,1\}^{l^{O(1)}} \to \{0,1\}$ such that if no small circuit computes f, then no small circuit computes f' on more than $\frac{1}{2} + \epsilon$ fraction of the inputs.

[22] give such a transformation which goes through a sequence of of transformations: one from Babai, Fortnow, Nisan and Wigderson [4], one from Impagliazzo [20], and a new one original to [22]. Again this step can be modularly achieved from error-correcting codes efficiently list-decodable under the implicit input/output model, as follows (from Sudan, Trevisan, and Vadhan [32]): Think of f as a 2^l bit string and encode this string using an error correcting code. Say the encoded string is a $2^{l'}$ bit string. Then this function can be thought of as the truth table of a function $f' : \{0,1\}^{l'} \to \{0,1\}$. It follows from the list-decodability properties of the error-correcting code that f' is highly unpredictable. Specifically, suppose C is a circuit predicting f'. Then C is an implicit representation of a received word that is close to f'; thus list-decoding, in the implicit output model, yields a small circuit computing f' (and with some work, a small circuit encoding f). The strength of this transformation is again in its efficiency. For example, Impagliazzo, Shaltiel, and Wigderson [21], note that this construction is also significantly more efficient in some parameters and use this aspect in a (more) optimal derandomization of BPP.

Direct product of NP-complete languages. Let SAT be the characteristic function of the satisfiability language. I.e., $\mathrm{SAT}(\phi) = 1$ if ϕ is a satisfiable formula and 0 otherwise. Let SAT^l be the l-wise direct product of the SAT function. I.e., it takes as input l formulae $\phi_1, \ldots, \phi_l$ and outputs the l-bit vector $\mathrm{SAT}(\phi_1), \ldots, \mathrm{SAT}(\phi_l)$. Clearly SAT^l is at least as hard to compute as SAT. Presumably it is much harder. In fact if SAT were hard to compute on more than $1 - \delta$ fraction of the instances chosen from some distribution, then SAT^l would be hard to compute with probability more than $(1-\delta)^l$ on the product distribution. Unfortunately, no NP-complete problem is known to be hard when the inputs are chosen at random. In the face of this lack of knowledge, what can one say about SAT^l? This topic is studied in the complexity theory literature under the label of membership comparability. Sivakumar [30] gives a nice hardness for this problem. He shows that if it is even possible to efficiently compute the least amount of information about SAT^l, for $l(n) = O(\log n)$ then NP=RP. Specifically, if some polynomial time algorithm, on input an instance $\vec{\phi}$ of SAT^l, rules out even one string out 2^l as the value of $\mathrm{SAT}^l(\vec{\phi})$, then it can be used to decide satisfiability. [30] uses the list-decodability properties of the Reed Solomon codes and a version of Sauer's lemma. Simplifying the proof slightly it is possible to get it as a consequence of Theorem 5 applied to $q = 2^l$ and $\epsilon = 2^{-2l}$.

Permanent of random matrices. In a striking result, Lipton [25], showed how it is possible to use the fact that the permanent is a low-degree polynomial to conclude the following. If it is easy to compute the permanent of an $n \times n$ matrix modulo a prime $p > n$, with high probability when the matrix is chosen at random, then it is also easy to compute the permanent of any $n \times n$ matrix modulo p. One of the first results to establish the average-case hardness of a computationally hard problem, this result laid down the seed for a series of interesting results in complexity theory including IP=PSPACE and the PCP characterizations of NP.

Subsequent results strengthened the average case hardness of the permanent to the point where it suffices to have an algorithm that computes the permanent modulo p on an inverse polynomially small fraction of the matrices, as shown by Cai, Pavan and Sivakumar [8]. Their result uses the list-decoding algorithm for Reed Solomon codes. Independently Goldreich, Ron and Sudan [13] strengthened this result in different direction. They show it suffices to have an algorithm that computes the permanent correctly with inverse polynomial probability when both the matrix and the prime are chosen at random. Their result uses the list decoding algorithm for the Reed Solomon codes as well as that for the Chinese Remainder code. It turns out that the techniques of [8] extend to this problem also, thus giving an alternate proof that does not use the list-decoder for the Chinese remainder code (but still uses the list-decoder for the Reed-Solomon code).

6 Concluding thoughts

By now we have seen many applications of algorithms for list-decoding. The notion of list-decoding itself, never mind the algorithmic results, is a very important one for complexity theory. The recent beautiful result of Trevisan [34], gives strong evidence of the role that this theme can play in central questions in complexity/extremal combinatorics. (For those few of you who may have missed this development, Trevisan showed how to construct a strong family of extractors by combining binary codes that have very good combinatorial list-decoding properties, with a pseudo-random generator of Nisan and Wigderson [26]. This construction and its successors, see Raz, Reingold and Vadhan [28], and Impagliazzo, Shaltiel, and Wigderson [21], reach optimal characteristics for various choices of parameters.) We hope other applications of list-decoding will continue to emerge as the notion becomes more popular.

We conclude with some open questions relating to combinatorial list-decoding performance of error-correcting codes. The combinatorial question relating to list-decoding posed in this survey was chosen carefully to allow for a tight presentation of results (in Theorem 1 and its corollary). However, it is much more interesting to study list-decoding characteristics of specific codes and here we know very little (Part (1) of Theorem 1 applies, of course, but Part (2) is irrelevant to specific codes). For example, Ta-Shma and Zuckerman [33], have shown that the random (non-linear) code has polynomially many codes in any ball of radius e, for e very close to the minimum distance of the code. The existence of such codes with good list-decoding properties raises the question of whether such codes exist with small description size; and if so can they be constructed and/or decoded efficiently. One could ask such questions about the classes of codes described in this paper. For example, what is the largest error e for an $(n, k)_q$ Reed-Solomon code for which the Hamming ball of radius e around any received word has only $\mathrm{poly}(n)$ codewords. The best known bound is still given by Part (1) of Theorem 1. This remains the case for *every* linear code we know of! In fact the following question remains very interesting: Let $\epsilon_\infty^{\mathrm{Lin}}(\delta)$ be defined analogously to $\epsilon_\infty(\delta)$, however restricted to linear codes. As earlier we know $1 - \sqrt{1-\delta} \le \epsilon_\infty^{\mathrm{Lin}}(\delta) \le \delta$. However we know very little beyond this point. (In some recent work in progress, Guruswami, Håstad, Sudan, and Zuckerman [15], have shown that the analogous quantity $\epsilon_c^{\mathrm{Lin}}(\delta)$ is strictly smaller than δ for every choice of δ and c, however the difference in their proof vanishes as $c \to \infty$. Thus a number of questions relating to the combinatorics of the list-decoding problem remain open. Depending on the answers to these, a number of algorithmic challenges could also open up. Thus the area seems rife for further exploration.

Acknowledgments

I'd like to thank Oded Goldreich, Venkatesan Guruswami, Johan Håstad, Luca Trevisan, Salil Vadhan, and David Zuckerman for sharing with me many of their thoughts and ideas on the topic of list-decoding.

References

[1] Werner Alexi, Benny Chor, Oded Goldreich, and Claus P. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194-209, April 1988.

[2] Sigal Ar, Richard J. Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from erroneous data. *SIAM Journal on Computing*, 28(2): 487–510, April 1999.

[3] Sanjeev Arora and Madhu Sudan. Improved low degree testing and its applications. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, El Paso, Texas, 4–6 May 1997.

[4] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[5] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of Lecture Notes in Computer Science, pages 37-48, Rouen, France, 22-24 February 1990. Springer.

[6] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[7] Dan Boneh. Finding smooth integers in short intervals using CRT decoding. (To appear) *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, Portland, Oregon, 21-23 May 2000.

[8] Jin-Yi Cai, A. Pavan, and D. Sivakumar. On the hardness of the permanent. In *16th International Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, Trier, Germany, March 4–6 1999. Springer-Verlag.

[9] Peter Elias. List decoding for noisy channels. In *1957-IRE WESCON Convention Record*, Pt. 2, pages 94-104, 1957.

[10] Joan Feigenbaum. The use of coding theory in computational complexity. In *Proceedings of Symposia in Applied Mathematics*, R. Calderbank (ed.), American Mathematics Society, Providence, pages 203–229, 1995.

[11] Anna Gal, Shai Halevi, Richard Lipton, and Erez Petrank. Computing from partial solutions In *Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, Atlanta, Georgia, 4-6 May 1999.

[12] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.

[13] Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries—the highly noisy case. Technical Report TR98-060, Electronic Colloquium on Computational Complexity, 1998. Preliminary version in FOCS '95.

[14] Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese remaindering with errors. Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, pages 225–234, Atlanta, Georgia, 1-4 May 1999.

[15] Venkatesan Guruswami, Johan Håstad, Madhu Sudan, and David Zuckerman. Untitled Manuscript, January 2000.

[16] Venkatesan Guruswami, Amit Sahai, and Madhu Sudan. Untitled Manuscript, January 2000.

[17] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(6): 1757–1767, September 1999.

[18] Venkatesan Guruswami and Madhu Sudan, Low-rate codes with high error correction capabilities. (To appear) *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, Portland, Oregon, 21-23 May 2000.

[19] Russell Impagliazzo. Personal Communication, July 1997.

[20] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science*, pages 538–545, Milwaukee, Wisconsin, 23–25 October 1995. IEEE.

[21] Russell Impagliazzo, Ronen Shaltiel, and Avi Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the Thirty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, New York City, New York, 17-19 October 1999. (To appear.)

[22] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.

[23] Erich Kaltofen. Polynomial factorization 1987-1991. In I. Simon, editor, *Proc. LATIN '92*, volume 583 of Lect. Notes Comput. Sci., pages 294-313, Heidelberg, Germany, 1992. Springer Verlag.

[24] S. Ravi Kumar and D. Sivakumar. Proofs, codes, and polynomial-time reducibilities. In *Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, Atlanta, Georgia, 4-6 May 1999.

[25] Richard Lipton. New directions in testing. In *Proceedings of DIMACS Workshop on Distributed Computing and Cryptography*, 1989.

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

[27] Vera Pless, W. Cary Huffman, and Richard A. Brualdi (Eds.). *Handbook of Coding Theory*, North-Holland, 1998.

[28] Ran Raz, Omer Reingold, and Salil Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, Atlanta, GA, May 1999.

[29] M. Amin Shokrollahi and Hal Wasserman. List decoding of algebraic-geometric codes *IEEE Transactions on Information Theory*, 45:432–437, March 1999.

[30] D. Sivakumar. On membership comparable sets. *Journal of Computer and System Sciences*, 59(2): 270–280, October 1999.

[31] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, March 1997.

[32] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma [extended abstract]. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 537–546, Atlanta, Georgia, 1-4 May 1999.

[33] Amnon Ta-Shma and David Zuckerman. Personal communication, November 1999.

[34] Luca Trevisan. Construction of extractors using pseudorandom generators. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, Atlanta, Georgia, 1-4 May 1999.