Invited paper: *17th Conference on Foundations of Software Technology and Theoretical Computer Science*, Kharagpur, India, 1997.

Algorithmic issues in coding theory

Madhu Sudan

October 9, 1997

Abstract

The goal of this article is to provide a gentle introduction to the basic definitions, goals and constructions in coding theory. In particular we focus on the algorithmic tasks tackled by the theory. We describe some of the classical algebraic constructions of error-correcting codes including the Hamming code, the Hadamard code and the Reed Solomon code. We describe simple proofs of their error-correction properties. We also describe simple and efficient algorithms for decoding these codes. It is our aim that a computer scientist with just a basic knowledge of linear algebra and modern algebra should be able to understand every proof given here. We also describe some recent developments and some salient open problems.

1 Introduction

Error-correcting codes are combinatorial structures that allow for the transmission of information over a noisy channel and the recovery of the information without any loss at the receiving end. Error-correcting codes come in two basic formats. (1) The "block error-correcting code": Here the information is broken up into small pieces. Each piece contains a fixed finite amount of information. The encoding method is applied to each piece individually (independently). The resulting encoded pieces (or blocks) are sent over the noisy channel. (2) The "convolutional codes": Here the information is viewed as a potentially infinite stream of bits and the encoding method is structured so as to handle an infinite stream. This survey will be restricted to the coverage of some standard block error-correcting codes.

Formally a block error-correcting code may be specified by an encoding function C. The input to C is a message m, which is a k-letter string over some alphabet Σ (typically $\Sigma = \{0, 1\}$ but we will cover more general codes as well). E

maps m into a longer n-letter string over the same alphabet[1]. The mapped string is referred to as a codeword. The basic idea is that in order to send the message m over to the receiver, we transmit instead the codeword $C(m)$. By the time this message reaches the destination it will be corrupted, i.e., a few letters in $C(m)$ would have changed. Say the received word is R. Hopefully R will still be able to convey the original message m even if it is not identically equal to $C(m)$. The only way to preserve this form of redundancy is by ensuring that no two codewords are too "close" to each other. This brings us to the important notion of "close"ness used, namely the Hamming distance. The Hamming distance between two strings $x, y \in \Sigma^n$, denoted $\Delta(x, y)$, is the number of letters where x and y differ. Notice that Δ forms a metric, i.e., $\Delta(x, y) = 0 \Rightarrow x = y$, $\Delta(x, y) = \Delta(y, x)$ and $\Delta(x, y) + \Delta(x, z) \geq \Delta(x, z)$. A basic parameter associated with a code is its *distance* i.e., the maximum value d such that any two codewords are a Hamming distance of at least d apart. Given a code of distance d and a received word R that differs from $C(m)$ in at most $e \leq d-1$ places, the error in the transmission can be detected. Specifically, we can tell that some letter(s) has been corrupted in the transmission, even though we may not know which letters are corrupted. In order to to actually correct errors we have to be able to recover m uniquely based on R and a bound t on the number of errors that may have occurred. To get the latter property t has to be somewhat smaller than $d-1$. Specifically if $t \leq \lfloor (d-1)/2 \rfloor$, then we notice that indeed there can be at most one message m such that $\Delta(C(m), R) \leq t$. (If m_1 and m_2 both satisfy $\Delta(C(m_1), R), \Delta(C(m_2), R) \leq t$, then $\Delta(C(m_1), C(m_2)) \leq \Delta(m_1, R) + \Delta(R, m_2) \leq 2t \leq d - 1$, contradicting the distance of C.) Thus in an information theoretic sense R maintains the information contained in m. Recovering the information m efficiently from C is another matter and we will come back to this topic presently.

To summarize the discussion above we adopt the following terse notation that is standard in coding theory. A code C is an $[n, k, d]_q$ code if $C : \Sigma^k \rightarrow \Sigma^n$, where $|\Sigma| = q$ with $\min_{x,y \in \Sigma^k}\{\Delta(C(x), C(y))\} = d$. With some abuse of notation we will use C to denote the image of the map C (i.e., C may denote the collection of codewords rather than the map). C is called a e-error-detecting code for $e = d-1$ and a t-error correcting code for $t = \lfloor (d-1)/2 \rfloor$.

In the remaining sections of this article we will describe some common constructions of $[n, k, d]_q$ for various choices of the parameters n, k, d and q. We will also describe the algorithmic issues motivated by these combinatorial objects and try to provide some solutions (and summarize the open problems). (We assume some familiarity with algebra of finite fields [10, 19].) Before going on to these issues, we once again stress the importance of the theory of error-correcting codes and its relevance to computer science. The obvious applications of error-correcting codes are to areas where dealing with error becomes important such as storage of information on disks, CDs, and communication over modems

[1] The assumption that the message is a k-letter string over Σ is just made for notational convenience. As it will become obvious, the representation of the message space is irrelevant to the communication channel. The representation of the encoded string is however *very* relevant!

etc. Additionally, and this is where they become important to the theoretical computer scientist, error-correcting codes come into play in several ways in complexity theory — for example, in fault-tolerant computing, in cryptography, in the derandomization of randomized algorithms and in the construction of probabilistically checkable proofs. In several of these cases it is not so much the final results as the notions, methods and ingredients from coding theory that help. All of this makes it important that a theoretical computer scientist be comfortable with the methods of this field — and this is the goal of this article. A reader interested in further details may try one of the more classical texts [2, 11, 17]. Also, the article of Vardy [18] is highly recommended for a more detailed account of progress in coding theory. The article is also rich with pointers to topics of current interest.

2 Linear Codes

While all questions relating to coding theory can be stated in general, we will focus in our article on a subset of codes called *linear codes*. These codes are obtained by restricting the underlying alphabet Σ to be a finite field of cardinality q with binary operations "+" and "·". Thus a string in Σ^n can be thought of as a vector in n-dimensional space, with induced operations "+" (vector addition), and "·" (scalar multiplication). Thus a code $\mathcal{C} \subset \Sigma^n$ is now a subset of the vectors. If this subset of vectors forms a "subspace" then the code is linear, as made formal below:

Definition 1. $\mathcal{C} \subseteq \Sigma^n$ is a *linear code* if $\forall a \in \Sigma,\ x, y \in \mathcal{C},\ x + y, a \cdot x \in \mathcal{C}$.

Many of the parameters of error-correcting codes become very clean in the case of linear codes. For instance, how does one specify a code $\mathcal{C} \in \Sigma^n$? For general codes, succinct representations may not exist! However, for every linear code a succinct representation, of size polynomial in n does exist! In particular, we have the following two representations:

1. For every $[n, k, d]_q$ linear code $\mathcal{C}$ there exists an $n \times k$ "generator" matrix $G = G_{\mathcal{C}}$ with entries from Σ such that $\mathcal{C} = \{Gx | x \in \Sigma^k\}$.
2. For every $[n, k, d]_q$ code $\mathcal{C}$ there exists an $(n - k) \times n$ parity check matrix $H = H_{\mathcal{C}}$ over Σ such that $\mathcal{C} = \{y \in \Sigma^n \text{ s.t. } Hy = \mathbf{0}\}$.

Conversely, the following hold: Every $n \times k$ matrix G over Σ defines an $[n, k', d]_q$ code, for some $d \geq 1$ and $k' \leq k$, $\mathcal{C}_G$ having as codewords $\{Gx | x \in \Sigma^k\}$. Similarly every $(n-k) \times n$ matrix H defines an $[n, k', d]$ code $\mathcal{C}'_H$, for some $d \geq 1$ and $k' \leq k$, having as codewords $\{y \in \Sigma^n | Hy = 0\}$.
Exercise:

1. Prove properties (1) and (2) above.
2. Given the generator matrix $G_{\mathcal{C}}$ of a code $\mathcal{C}$, give a polynomial time algorithm to compute a parity check matrix $H_{\mathcal{C}}$ for $\mathcal{C}$.
3. Show that if G is of full column rank (H is of full row rank) then the code $\mathcal{C}_G$ ($\mathcal{C}_H$) is an $[n, k, d]_q$ code.

3 Some common constructions of codes

In this section we describe some common construction of codes. But first let us establish the goal for this section. In general we would like to find families of $[n,k,d]_q$ codes for infinitely many triples (n,k,d) for some fixed q. The property we would really like is that k/n and d/n are bounded away from zero as $n \to \infty$. Such a code is termed *asymptotically good* and the two properties $k/n > 0$ and $d/n > 0$ are termed *constant message-rate* and *constant distance-rate* respectively. Unfortunately we will not be able to get to this goal in this article. But we will settle for what we term *weakly good* codes. These are codes with *polynomial message-rate*, i.e., $k = \Omega(n^\epsilon)$ for some $\epsilon > 0$ and *constant distance-rate*.

3.1 Hamming code

Hamming codes are defined for every positive n such that there exists an integer l such that $n = 2^l - 1$. Then the Hamming code of block size n over the alphabet $\{0,1\}$ is given by an $l \times n$ parity check matrix H^{HMG} whose columns are all the distinct l-dimensional non-zero vectors. Notice that there are exactly $2^l - 1$ of these.

Lemma 2. *For every positive integer n such that $n = 2^l - 1$ for some integer l, the Hamming code of block size n is an $[n, n-l, 3]_2$ code.*

Proof Sketch. Notice that the rank of H^{HMG} is l. In particular the column vectors containing exactly one 1 are linearly independent and there are l of them. Thus we find that the Hamming code is an $[n,k,d]_2$ code for $k = n - l$.

We now move to showing that the distance of the Hamming code is 3. Notice that the code has no elements of weights since this would imply that two vectors in the parity check matrix are identical. This implies the distance is at least 3. Now consider any two column vectors v_1 and v_2 in H^{HMG}. Notice that the vector $v_1 + v_2$ is also a column vector of H^{HMG} and is distinct from v_1 and v_2. Now consider the n dimensional vector which is zero everywhere except in the coordinates corresponding to the vectors v_1, v_2 and $v_1 + v_2$. This vector has weight 3 and is easily seen to be an element of the Hamming code. Thus the distance of the Hamming code is exactly 3.

The Hamming code is a simple code with a very good rate. Unfortunately it can only correct 1 error, definitely far from our goal of constant error-rate. Next we move on to a code with good error-correcting properties, but with very low-rate.

3.2 Hadamard code

A Hadamard matrix is an $n \times n$ matrix M with entries from ± 1 such that $MM^T = n \cdot I_n$ where I_n is the $n \times n$ identity matrix. A Hadamard matrix immediately leads to an error correcting code where the rows of M are the codewords. This leads to a codeword over the alphabet $\Sigma = \{+1, -1\}$. We prove the distance property of the code first.

Lemma 3. *If M is a Hadamard matrix then any two rows agree is exactly $n/2$ places.*

Proof. Say the rows of interest are the ith and jth rows. Then consider the element $(MM^T)_{ij}$. This element is the sum of n terms, with the kth term being $m_{ik}m_{jk}$. Notice that this term evaluates to $+1$ if $m_{ik} = m_{jk}$ and -1 otherwise. Thus if the ith and jth rows disagree in t places, then $(MM^T)_{ij} = (n-t)+t$. Since $(MM^T)_{ij} = 0$, we have that $n - 2t = 0$ and hence the two rows (dis)agree in exactly $n/2$ places.

Thus the task of constructing a Hadamard code reduces to the task of constructing Hadamard matrices. Constructions of Hadamard matrices have been a subject of much interest in combinatorics. It is clear (from Lemma 3) that for an $n \times n$ Hadamard matrix to exists n must be even. The converse is not known to be true and is still an open question. What is known is that an $n \times n$ matrix exists for every n of the form $p-1$ where p is a prime. It is also known that if an $n_1 \times n_1$ Hadamard matrix exists and an $n_2 \times n_2$ Hadamard matrix exists, then an $n_1n_2 \times n_1n_2$ matrix exists. Many other such constructions are also known but not all possibilities are covered yet. Here we give the basic construction which applies when n is a power of 2. These constructions are described recursively as follows:

$$M_1^{\mathrm{HDM}} = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix} \qquad M_l^{\mathrm{HDM}} = \begin{bmatrix} +M_{l-1}^{\mathrm{HDM}} & +M_{l-1}^{\mathrm{HDM}} \\ +M_{l-1}^{\mathrm{HDM}} & -M_{l-1}^{\mathrm{HDM}} \end{bmatrix}.$$

Lemma 4. *For every l, the rows of M_l^{HDM} form a $[2^l, l, 2^{l-1}]_2$ code.*

Proof. Left as an exercise to the reader.

The Hadamard codes maintain a constant distance-rate. However their message-rate approaches zero very quickly. Next we describe a code with constant message-rate and distance-rate. The catch is that the code uses an alphabet of growing size.

3.3 Reed Solomon code

The Reed Solomon codes are a family of codes defined over an alphabet of growing size, with $n \leq q$. The more common definition of this code is not (we feel) as intuitive or as useful as the "folklore" definition. We present both definitions here, starting with the more useful one and then show the equivalence of the two.

Definition 5 (Reed Solomon codes). Let Σ be a field of size q, $n \le q$ and let $x_0, \dots, x_{n-1}$ be some fixed enumeration of n of the elements of Σ. (It is standard to pick $n = q-1$ and $x_i = \alpha^i$ for some primitive element α[2]. Then for every $k \le n$, the Reed Solomon code $\mathcal{C}^1_{\mathrm{RS},n,k,q}$ is defined as follows: A message $m = m_0 \dots m_{k-1}$ corresponds to the degree $k-1$ polynomial $M(x) = \sum_{i=0}^{k-1} m_i x^i$. The encoding of m, is $\mathcal{C}^1_{\mathrm{RS},n,k,q}(m) = c_0 \dots c_{n-1}$ where $c_j = M(x_j)$.

The distance properties of the Reed Solomon codes follow immediately from the fact that a degree $k-1$ polynomial may only have $k-1$ zeroes unless all of its coefficients are zero.

Lemma 6. *For every $n \le q$ and $k \le n$, the Reed Solomon code $\mathcal{C}^1_{\mathrm{RS},n,k}$ forms an $[n,k,n-k]_q$ linear code.*

Proof. The fact that the code is linear follows from the fact that if $M_0(x)$ and $M_1(x)$ are polynomials of degree at most $k-1$ then so is $M_0(x) + M_1(x)$. The distance follows from the fact that if $M_0(x_j) = M_1(x_j)$ for k values of j then $M_0 \equiv M_1$ (or equivalently if $M_0(x_j) - M_1(x_j)$ is zero for k values of j, then $M_0 - M_1$ is the zero polynomial).

Finally for the sake of completeness we present a second definition of Reed Solomon codes. This definition is more commonly seen in the texts, but we feel this part may be safely skipped at first reading.

Definition 7 (Reed Solomon codes). Let Σ be a field of size q with primitive element α, and let $n = q-1$, $k \le n$. Let $P_{k,q}(x)$ be the polynomial $(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{n-k})$. The Reed Solomon code $\mathcal{C}^2_{\mathrm{RS},n,k,q}$ is defined as follows: A message $m = m_0 \dots m_{k-1}$ corresponds to the degree $k-1$ polynomial $M(x) = \sum_{i=0}^{k-1} m_i x^i$. The encoding of m, is $\mathcal{C}^1_{\mathrm{RS},n,k,q}(m) = c_0 \dots c_{n-1}$ where c_j is the coefficient of x^j in the polynomial $P_{k,q}(x)M(x)$.

Viewed this way it is hard to see the correspondence between the two definitions (or the distance property). We prove an equivalence next.

Lemma 8. *The definitions of Reed Solomon codes given in Definitions 5 and 7 coincide for $n = q-1$ and the standard enumeration of the elements of* $\mathrm{GF}(q)$.

Proof. Notice that it suffices to prove that every codeword according to the first definition is a codeword according to the second definition. The fact that the sets are of the same size implies that they are identical.

Consider the encoding of $m = m_0 \dots m_{k-1}$. This encoding is $\mathcal{C}^1_{\mathrm{RS},n,k,q} = c_0 \dots c_{n-1}$ with $c_i = \sum_{j=0}^{k-1} m_j (\alpha^i)^j$. To show that this is a codeword according to the second definition we need to verify that the polynomial $C(x) = \sum_{i=0}^{n-1} c_i x^i$

[2] α is a primitive element of the field $\mathrm{GF}(q)$ if $\alpha^j \neq 1$ for any $j < q-1$.

has $(x - \alpha^l)$ as a factor for every $l \in \{1, \ldots, n-k\}$. Equivalently it suffices to verify that $C(\alpha^l) = 0$, which we do next:

$$
\begin{aligned}
C(\alpha^l) &= \sum_{i=0}^{n-1} c_i (\alpha^l)^i \\
&= \sum_{i=0}^{n-1} \sum_{j=0}^{k-1} m_j (\alpha^i)^j (\alpha^l)^i \\
&= \sum_{j=0}^{k-1} m_j \sum_{i=0}^{n-1} (\alpha^j \alpha^l)^i \\
&= \sum_{j=0}^{k-1} m_j \sum_{i=0}^{q-2} \gamma_{j,l}^i
\end{aligned}
$$

where $\gamma_{j,l} = \alpha^{j+l}$. Notice that for every j, l s.t. $j + l \neq q - 1$, $\gamma_{j,l} \neq 1$. Notice further that for every such $\gamma_{j,l}$ the summation $\sum_{i=0}^{q-2} \gamma_{j,l}^i = 0$[3]. Since $j \in \{0, \ldots, k-1\}$, we find that $\gamma_{j,l} \neq 1$ for every $l \in \{1, \ldots, q-1-k\}$. Thus for every $l \in \{1, \ldots, n-k\}$, we find that $C(\alpha^l) = 0$. This concludes the proof.

3.4 Multivariate polynomial codes

The next family of codes we describe are not very commonly used in coding theory, but have turned out to be fairly useful in complexity theory and in particular in the results on probabilistically checkable proofs. Surprisingly these codes turn out to be a common generalization of Hadamard codes and Reed Solomon codes!

Definition 9 (Multivariate polynomial code). For integer parameters m, l and q with $l < q$, the multivariate polynomial code $\mathcal{C}_{\mathrm{POLY},m,l,q}$ has as message a string of coefficients $m = \{m_{i_1,i_2,\ldots,i_m}\}$ with $i_j \geq 0$ and $\sum_j i_j \leq l$. This sequence is interpreted as the m-variate polynomial $M(x_1, \ldots, x_m) = \sum_{i_1,\ldots,i_j} m_{i_1,\ldots,i_j} x_1^{i_1} \cdots x_m^{i_m}$. The encoding of m is the string of letters $\{M(x_1, \ldots, x_m)\}$ with one letter for every $(x_1, \ldots, x_m) \in \Sigma^m$.

Obviously the multivariate polynomial codes form a generalization of the Reed Solomon codes (again using the first definition given here of Reed Solomon codes). The distance property of the multivariate polynomial codes follow also from the distance property of multivariate polynomials (cf. [5, 13, 21]).

Lemma 10. *For integers m, l and q with $l < q$, the code $\mathcal{C}_{\mathrm{POLY},m,l,q}$ is an $[n, k, d]_q$ code with $n = q^m$, $k = \binom{m+l}{m}$ and $d = (q - l)q^{m-1}$.*

---

[3] This identity is obtained as follows: Recall that Fermat's little theorem asserts that $\gamma^{q-1} - 1 = 0$ for every non-zero γ in GF(q). Factoring the left hand side, we find that either $\gamma - 1 = 0$ or $\sum_{i=0}^{q-2} \gamma^i = 0$. Since $\gamma \neq 1$, the latter must be the case.

Proof. The bound on n is immediate. The fact that the number of coefficients $i_1, \ldots, i_m$ s.t. $\sum_j i_j \leq l$ is at $\binom{m+l}{l}$ is a well-known exercise in counting. Finally the bound on the distance follows from the fact a degree l polynomial can only be zero for l/q fraction of its inputs. (This is an easy inductive argument based on the number of variables. The base case is well known and inductively one picks a random assignment to the variables $x_1, \ldots, x_{m-1}$ and argues that the resulting polynomial in x_m is non-zero with high probability. Finally one uses the base case again to conclude that the final polynomial in x_m is left non-zero by a random assignment to x_m.)

It is easy to see that the code $\mathcal{C}^1_{\mathrm{RS},q,k,q}$ is the same as the code $\mathcal{C}_{\mathrm{POLY},1,k-1,q}$. Also notice that the code $\mathcal{C}_{\mathrm{POLY},m,1,2}$ forms an $[2^m, m, 2^{m-1}]_2$ code, same as parameters of the Hadamard code given by the rows of M^{HDM}_m. It turns out that these two codes are in fact identical. The proof is left as an exercise to the reader.

3.5 Concatenated codes

Each code in the collection of codes we have accumulated above has some flaw or the other. The Hamming codes don't correct too many errors, the Hadamard codes are too low-rate, and the Reed Solomon codes depend on a very large alphabet. Yet it turns out it is possible to put some of these codes together and obtain a code with reasonably good behavior ("polynomially good"). This is made possible by a simple idea called "concatenation", defined next.

Definition 11 (Concatenation of codes). Let $\mathcal{C}_1$ be an $[n_1, k_1, d_1]_{q_1}$ code over the alphabet Σ_1 and let $\mathcal{C}_2$ be an $[n_2, k_2, d_2]_{q_2}$ code over the alphabet Σ_2. If $q_1 = q_2^{k_2}$ then the code $\mathcal{C}_1 \circ \mathcal{C}_2$ is defined as follows: Associate every letter in Σ_1 with a codeword of $\mathcal{C}_2$. Encode every message first using the code $\mathcal{C}_1$ and then encode every letter in the encoded string using the code $\mathcal{C}_2$. More formally, given a message $m \in \Sigma_1^{k_1} = \Sigma_2^{k_1 k_2}$, let $\mathcal{C}_1(m) = c_1 \ldots c_{n_1} \in \Sigma_1^{n_1}$. The encoding $\mathcal{C}_1 \circ \mathcal{C}_2(m)$ is given by $c_{11} \ldots c_{1n_2} c_{21} \ldots c_{n_1 n_2} \in \Sigma_2^{n_1 n_2}$, where for every $i \in \{1, \ldots, n_1\}$, $c_{i1} \ldots c_{in_2} = \mathcal{C}_2(c_i)$.

Almost immediately we get the following property of concatenation.

Lemma 12. *If $\mathcal{C}_1$ is an $[n_1, k_1, d_1]_{q_1}$ code and if $\mathcal{C}_2$ is an $[n_2, k_2, d_2]_{q_2}$ code with $q_1 = q_2^{k_2}$, then $\mathcal{C}_1 \circ \mathcal{C}_2$ is an $[n_1 n_2, k_1 k_2, d']_{q_2}$ code, for some $d' \geq d_1 d_2$.*

Proof. The block size and message size bounds follow from the definition. To see the distance property, consider two messages $m^1, m^2 \in \Sigma_1^{k_1}$. For $l \in \{1, 2\}$, let $c^l_1 \ldots c^l_{n_1}$ be the encoding of m^l using $\mathcal{C}_1$ and let $c^l_{11} \ldots c^l_{n_1 n_2}$ be its encoding using $\mathcal{C}_1 \circ \mathcal{C}_2$. Notice that there must exist at least d_1 values of i such that $c^1_i \neq c^2_i$ (by the distance of $\mathcal{C}_1$). For every such i, there must exist at least d_2 values of j such that $c^1_{ij} \neq c^2_{ij}$ (by the distance of $\mathcal{C}_2$). Thus we find that $\mathcal{C}_1 \circ \mathcal{C}_2(m^1)$ and $\mathcal{C}_1 \circ \mathcal{C}_2(m^2)$ differ in at least $d_1 d_2$ places.

To best see the power of concatenation, consider the following simple application: Let $\mathcal{C}_1$ be a Reed Solomon code with $q = 2^m$, $n = q$ and $k = .4n$. I.e., $\mathcal{C}_1$ is an $[n, .4n, .6n]_{2^m}$ code with $n = 2^m$. Let $\mathcal{C}_2$ be the Hadamard code $[2^m, m, 2^{m-1}]_2$. The concatenation $\mathcal{C}_1 \circ \mathcal{C}_2$ is an $[n^2, .4n \log n, .3n^2]_2$ code. I.e., the resulting code has constant distance-rate, polynomial rate and is over the binary alphabet! Thus this satisfies our weaker goal of obtaining a weakly-good code. Even the goal of obtaining an asymptotically good code is close now. In particular, the code of Justesen is obtained by an idea similar to that of concatenation. Unfortunately we shall not be able to cover this material in this article.

4 Algorithmic tasks

We now move on to the algorithmic tasks of interests: The obvious first candidate is *encoding*.

Problem 13 (Encoding).
INPUT: $n \times k$ matrix G and message $m \in \Sigma^k$.
OUTPUT: $\mathcal{C}(m)$, where $\mathcal{C} = \mathcal{C}_G$ is the code with G as the generator matrix.

It is clear that the problem as specified above is easily solved in time $O(nk)$ and hence in time polynomial in n. For specific linear codes such as the Reed Solomon codes it is possible to encode the codes faster, in time $O(n \log^c n)$ for some constant c. However till recently no asymptotically good code was known to be encodable in linear time. In a recent breakthrough. Spielman [15] presented the first known code that is encodable in linear time. We will discuss this more in a little bit.

The next obvious candidate problem is the decoding problem. Once again it is clear that if the received word has no errors, then this problem is only as hard as solving a linear system and thus can be easily solved in polynomial time. So our attention moves to the case where the received word has errors. We first define the error detection problem.

Problem 14 (Error detection).
INPUT: $n \times k$ generator matrix G for a code $\mathcal{C} = \mathcal{C}_G$; and a received word $R \in \Sigma^n$.
OUTPUT: Is R a codeword?

The error detection problem is also easy to solve in polynomial time. We find the parity check matrix H for the code $\mathcal{C}$ and then check if $HR = \mathbf{0}$. We now move to the problem of decoding in the presence of errors. This problem comes in several variants. We start with the simple definition first:

Problem 15 (Maximum likelihood decoding).
INPUT: $n \times k$ generator matrix G for a code $\mathcal{C} = \mathcal{C}_G$; and a received word $R \in \Sigma^n$.
OUTPUT: Find a codeword $x \in \mathcal{C}$, that is nearest to R in Hamming distance. (Ties may be broken arbitrarily.)

There are two obvious strategies for solving the maximum likelihood decoding problem:

Brute Force 1: Enumerate all the codewords and find the one that is closest to R.

Brute Force 2: For $t = 0, 1, \ldots,$, do: Enumerate all possible words within a Hamming distance of t from R and check if the word is a codeword. Output the first match.

Despite the naivete of the search strategies above, there are some simple cases where these strategies work in polynomial time. For instance, the first strategy above does work in polynomial time for Hadamard codes. The second strategy above works in polynomial time for Hamming codes (why?). However, both strategies start taking exponential time once the number of codewords becomes large, while distance also remains large. In particular, for "asymptotically good" or even "weakly good" codes, both strategies above run in exponential time. One may wonder if this exponential time behavior is inherent to the decoding problem. In perhaps the first "complexity" result in coding theory, Berlekamp, McEliece and van Tilborg [4] present the answer to this question.

Theorem 16 [4]. *The Maximum likelihood decoding problem for general linear codes is NP-hard.*

There are two potential ways to attempt to circumvent this result. One method is to define and solve the maximum likelihood decoding problem for specific linear codes. We will come to this question momentarily. The other hope is that we attempt to correct only a limited number of errors. In order to do so, we further parameterize the maximum likelihood decoding problem as follows:

Problem 17 (Bounded distance decoding).
INPUT: $n \times k$ generator matrix G for a code $\mathcal{C} = \mathcal{C}_G$; a received word $R \in \Sigma^n$ and a positive integer t.
OUTPUT: Find any/all codewords in $\mathcal{C}$ within a Hamming distance of t from R.

The hardness result of [4] actually applies to the Bounded distance decoding problem as well. However one could hope for a result of the form: "There exists an $\epsilon > 0$, such that for every $[n, k, d]_q$ linear code $\mathcal{C}$, the bounded distance decoding problem for $\mathcal{C}$ with $t = \epsilon d$ is solvable in polynomial time". One bottleneck to such a general result is that we don't know how to compute d for a generic linear code. This motivates the following problem:

Problem 18 (Minimum distance).
INPUT: $n \times k$ generator matrix G for a code $\mathcal{C} = \mathcal{C}_G$ and an integer parameter d.
OUTPUT: Is the distance of $\mathcal{C}$ at least d?

This problem was conjectured to be coNP-hard in [4]. The problem remained open for nearly two decades. Recently, in a major breakthrough, this problem was shown to be coNP-complete by Vardy [18]. While this does not directly rule out the possibility that a good bounded distance decoding algorithm may exist, the result should be ruled as one more reason that general positive results may be unlikely.

Thus we move from general results, i.e., where the code is specified as part of the input, to specific results, i.e., for well-known families of codes. The first question that may be asked is: "Is there a family of asymptotically-good $[n, k, d]_q$ linear code and $\epsilon > 0$, for which a polynomial time bounded distance decoding algorithm exists for $t \geq \epsilon d$?" For this question the answer is "yes". A large number of algebraic codes do have such polynomial time bounded distance decoding algorithms. In particular the Reed Solomon codes are known to have such a decoding algorithm for $t \leq \lfloor (d-1)/2 \rfloor$ (cf. [2, 11, 17]). This classical result is very surprising given the non-trivial nature of this task. This result is also very crucial for many of the known asymptotically good codes, since many of these codes are constructed by concatenating Reed Solomon codes with some other codes. In the next section we shall cover the decoding of Reed Solomon codes in more detail.

Lastly there is another class of codes, constructed by combinatorial means, for which bounded distance decoding for some $t \geq \epsilon d$ can be performed in polynomial time. These are the expander codes, due to Sipser and Spielman [14] and Spielman [15]. The results culminate in a code with very strong — linear time (!!!) — encoding and bounded distance decoding algorithms. In addition to being provably fast, the algorithms for the encoding and decoding of these codes are surprisingly simple and clean. However, the description of the codes and analysis of the algorithm is somewhat out of the scope of this paper. We refer the reader to the original articles [14, 15] for details.

5 Decoding of Reed Solomon code

As mentioned earlier a polynomial time algorithm for bounded distance decoding is known and this algorithm corrects up to $t \leq \lfloor (d-1)/2 \rfloor$ errors. Notice that this coincides exactly with the error-correction bound of the code (i.e., a Reed Solomon code of distance d is a t-error-correcting code for $t = \lfloor (d-1)/2 \rfloor$). This bound on the correction capability is inherent, if one wishes to determine the codeword uniquely. However in the bounded distance decoding problem we do allow for multiple solutions. Given this latitude it is reasonable to hope for a polynomial-time decoding algorithm that corrects more errors - say up to $t < (1-\epsilon)d$ where ϵ is some fixed constant. However no such algorithm is known for all possible values of $(n, k, d = n-k)$. Recently, in [16], we presented an algorithm which does correct up to $(1-\epsilon)d$ errors, provided $k/n \to 0$. This algorithm was inspired by an algorithm of Welch and Berlekamp [20, 3] for decoding Reed Solomon codes. This algorithm is especially clean and elegant. Our solution uses similar ideas to correct even more errors and we present this next.

Notice first that the decoding problem for Reed Solomon codes can be solved by solving the following cleanly stated problem:

Problem 19 (Reed Solomon decoding).
INPUT: n pairs of points $\{(x_i, y_i)\}$, $x_i, y_i \in \mathrm{GF}(q)$; and integers t, k.
OUTPUT: All polynomials p of degree at most $k-1$ such that $y_i \neq p(x_i)$ for at most t values of i.

The basic solution idea in Welch-Berlekamp and our algorithm is to find an algebraic description of *all* the given points, and to then use the algebraic description to extract p. The algebraic description we settle for is an "algebraic curve in the plane", i.e., a polynomial $Q(x, y)$ in two variables x and y such that $Q(x_i, y_i) = 0$ for every value of x and y. Given this basic strategy, the performance of the algorithm depends on the choice of the degree of Q which allows for such a curve to exist, and still be useful! (For example if we allow Q to be 0, or if we pick the degree of Q be n in x and 0 in y, the such polynomials do exist, but are of no use. On the other hand a non-zero polynomial Q of degree $n/10$ in x and 0 in y may be useful, but will probably not exist for the given data points.)

To determine what kind of polynomial Q we should search for, we pick two parameters l and m and impose the following conditions on $Q(x, y) = \sum_{i,j} q_{ij} x^i y^j$:

1. Q should not be the zero polynomial. (I.e., some q_{ij} should be non-zero.)
2. q_{ij} is non-zero implies $j \leq m$ and $i + (k-1)j \leq l$. (The reason for this restriction will become clear shortly.)
3. $Q(x_i, y_i) = 0$ for every given pair (x_i, y_i).

Now consider the task of searching for such a Q. This amounts to finding values for the unknown coefficients q_{ij}. On the other hand the conditions in (3) above amount to homogeneous linear equations in q_{ij}. By elementary linear algebra a solution to such a system exists and can be found in polynomial time provided the number of equations (n) strictly exceeds the number of unknowns (i.e., the number of (i, j) pairs such that $0 \leq i, j, j \leq m$ and $i + (k-1)j \leq m$). It is easy to count the number of such coefficients. The existence of such coefficients will determine our choice of m, l. Having determined such a polynomial we will apply the following useful lemma to show that p can be extracted from Q.

Lemma 20 [1]. *Let $Q(x, y) = \sum_{i,j} q_{ij} x^i y^j$ be such that $q_{ij} = 0$ for every i, j with $i + (k-1)j > l$. Then if $p(x)$ is polynomial of degree $k-1$ such that for strictly more than l values of i, $y_i = p(x_i)$ and $Q(x_i, y_i) = 0$, then $y - p(x)$ divides the polynomial $Q(x, y)$.*

Proof. Consider first the polynomial $g(x)$ obtained from Q by substituting $y = p(x)$. Notice that the term $q_{ij} x^i y^j$ becomes a polynomial in x of degree $i + (k-1)j$ which by property (2) above becomes a polynomial of degree at most l in x. Thus $g(x) = Q(x, p(x))$ becomes a polynomial in x of degree at most l. Now, for every i such that $y_i = p(x_i)$ and $Q(x_i, y_i) = 0$, we have that $g(x_i) = Q(x_i, p(x_i)) = 0$. But there are more than l such values of i. Thus g is identically zero. This immediately implies that $Q(x, y)$ is divisible by $y - p(x)$. (The division theorem for polynomials says that if a polynomial $h(y)$ evaluates to 0 at $y = \zeta$ then

$y - \zeta$ divides $h(y)$. Applying this fact to the polynomial $Q_x(y) = Q(x, y)$ and $y = p(x)$, we obtain the desired result. Notice in doing so, we are switching our perspective. We are thinking of Q as a polynomial in y with coefficients from the ring of polynomials in x.)

Going back to the choice of m and l, we have several possible choices. In one extreme we can settle for $m = 1$ and then if $l \approx (n+k)/2$, then we find that the number of coefficients is more than n. In this case the polynomial $Q(x, y)$ found by the algorithm is of the form $A(x)y + B(x)$. Lemma 20 above guarantees that if $t \le \lfloor (n-k)/2 \rfloor$ then $y - p(x)$ divides Q. Thus $p(x) = -B(x)/A(x)$ and can be computed easily by a simple polynomial division. Thus in this case we can decode from $\lfloor (n-k)/2 \rfloor$ errors thus recovering the results of [20]. In fact, in this case the algorithm essentially mimics the [20] algorithm, though the correspondence may not be immediately obvious.

At a different extreme one may pick $m \approx \sqrt{n/k}$ and $l \approx \sqrt{nk}$ and in this case Lemma 20 works for $t \approx n - 2\sqrt{nk}$. In this case to recover $p(x)$ from Q, one first factors the bivariate polynomial Q. This gives a list of all polynomial $p_j(x)$ such that $y - p_j(x)$ divides Q. From this list we pull out all the polynomials p_j such that $p_j(x_i) \ne y_i$ for at most t values of x_i. Thus in this case also we have a polynomial time algorithm provided Q can be factored in polynomial time. Fortunately, such algorithms are known, due to Kaltofen [8] and Grigoriev [7] (see Kaltofen [9] for a survey of polynomial factorization algorithms). For $k/n \to 0$, the number of errors corrected by this algorithm approaches $(1 - o(1))n$.

A more detailed analysis of this algorithm and the number of errors corrected by it appear in [16]. The result shows that this given an $[n, \kappa n, (1-\kappa)n]_q$ Reed Solomon code, the number of errors corrected by this algorithm approaches

$$n\left(1 - \frac{1}{1+\rho_\kappa} - \frac{\rho_\kappa}{2}\kappa\right) \text{ where } \rho_\kappa = \left\lfloor \sqrt{\frac{2}{\kappa} + \frac{1}{4}} - \frac{1}{2} \right\rfloor.$$

A plot of this curve against κ appears in Figure 1. Also shown in the figure are the distance of the code $((1-\kappa)n)$ and the classical-error correction bound $((1-\kappa)/2n)$.

6 Open questions

Given that the fundamental maximum likelihood decoding problem is NP-hard for a general linear code, the next direction to look to is a bounded distance decoding algorithm for every $[n, k, d]_q$ linear code. The bottleneck to such an approach is that in general we can't compute d in polynomial time, due to the recent result of Vardy [18]. Thus the next step in this direction seems to suggest an application of approximation algorithms:

Open Problem 1 *Given an $n \times k$ matrix G, approximate the distance d of the code $\mathcal{C}_G$ to within a factor of $\alpha(n)$.*

[t]

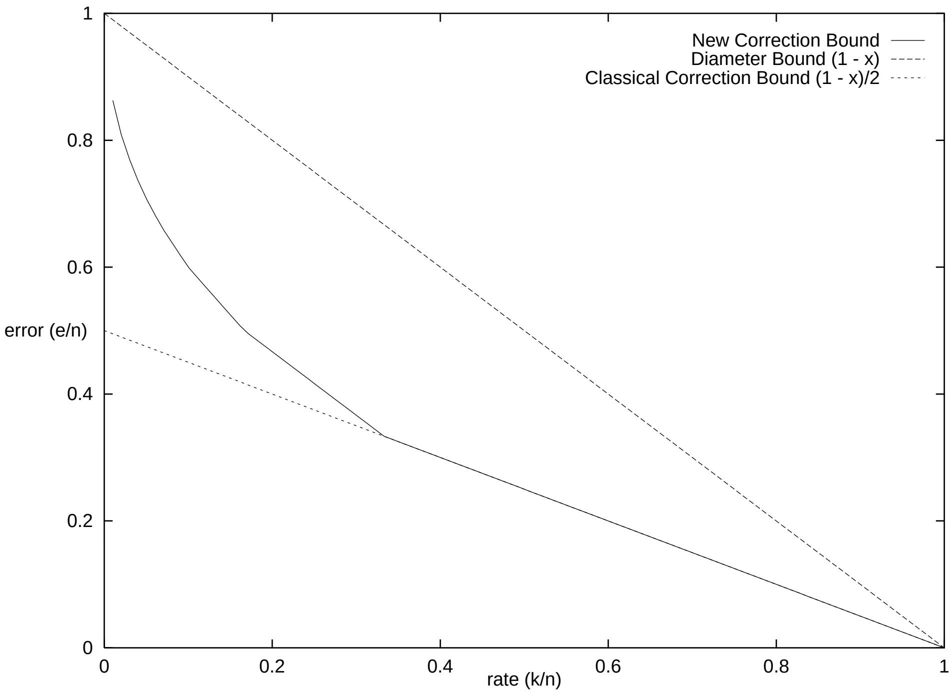

Fig. 1. Fraction of errors corrected by the algorithm from [16] plotted against the rate of the code. Also plotted are the distance of the code and the classical error-correction bound.

The goal here is to find the smallest factor $\alpha(n)$ for which a polynomial time approximation algorithm exists. Currently no non-trivial (i.e., with $\alpha(n) = o(n)$) approximation algorithm is known. A non-trivial $\alpha(n)$ approximation algorithm would then suggest the following candidate for bounded distance decoding:

Open Problem 2 *Given an $n \times k$ matrix G, a word $R \in \Sigma^n$ and an integer t, find all codewords within a Hamming distance of t from R, or show that the minimum distance of the code is less than $t\alpha_1(n)$.*

A similar problem is posed by Vardy [18] for $\alpha_1 = 2$. Here the hope would be to find the smallest value of α_1 for which a polynomial time algorithm exists. While there is no immediate formal reasoning to believe so it seems reasonable to believe that α_1 will be larger than α.

Next we move to the questions in the area of design of efficient codes, motivated by the work of Spielman [15].

Open Problem 3 *For every $\kappa > 0$, design a family of $[n, \kappa n, \delta n]_2$ codes $\mathcal{C}_n$ so that the bounded distance problem on $\mathcal{C}_n$ with parameter $t \leq \gamma n$ can be solved in* *linear time.*

The goal above is to make γ as large as possible for every fixed κ. Spielman's result allows for the construction codes which match the best known values of δ for any $[n, \kappa n, \delta n]_2$ linear code. However the value of γ is still far from δ in these results.

We now move towards questions directed towards decoding Reed-Solomon codes. We direct the reader's attention to Figure 1. Clearly every point above the solid curve and below the distance bound of the code, represents an open problem. In particular we feel that the following version maybe solvable in polynomial time:

Open Problem 4 *Find a bounded distance decoding algorithm for an* $[n, \kappa n, (1-\kappa)n]_q$ *Reed Solomon code that decodes up to* $t \leq (1-\sqrt{\kappa})n$ *errors.*

The motivation for this particular version is that in order to solve the bounded distance decoding problem, one needs to ensure that the number of outputs (i.e., the number codewords within the given bound t) is polynomial in n. Such a bound does exist for the value of t as given above [6, 12], thus raising the hope that this problem may be solvable in polynomial time also.

Similar questions may also be raised about decoding multivariate polynomials. In particular, we don't have polynomial time algorithms matching the bounded distance decoding algorithm from [16], even for the case of bivariate polynomials. This we feel may be the most tractable problem here.

Open Problem 5 *Find a bounded distance decoding algorithm for the bivariate polynomial code* $\mathcal{C}_{\mathrm{POLY},2,\kappa n,n}$ *that decodes up to* $t \leq (1-\sqrt{2\kappa})n^2$ *errors.*

References

1. S. Ar, R. Lipton, R. Rubinfeld and M. Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing*, to appear. Preliminary version in *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 503–512, 1992.
2. E. R. Berlekamp. *Algebraic Coding Theory*. McGraw Hill, New York, 1968.
3. E. R. Berlekamp. Bounded Distance +1 Soft-Decision Reed-Solomon Decoding. In *IEEE Transactions on Information Theory*, pages 704-720, vol. 42, no. 3, May 1996.
4. E. R. Berlekamp, R. J. McEliece and H. C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24:384–386, 1978.
5. R. DeMillo and R. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, June 1978.
6. O. Goldreich, R. Rubinfeld and M. Sudan. Learning polynomials with queries: The highly noisy case. *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pp. 294–303, 1995.

7. D. Grigoriev. Factorization of Polynomials over a Finite Field and the Solution of Systems of Algebraic Equations. Translated from Zapiski Nauchnykh Seminarov Lenningradskogo Otdeleniya Matematicheskogo Instituta im. V. A. Steklova AN SSSR, Vol. 137, pp. 20-79, 1984.
8. E. Kaltofen. A Polynomial-Time Reduction from Bivariate to Univariate Integral Polynomial Factorization. In *23rd Annual Symposium on Foundations of Computer Science*, pages 57-64, 1982.
9. E. Kaltofen. Polynomial factorization 1987–1991. *LATIN '92*, I. Simon (Ed.) Springer LNCS, v. 583:294–313, 1992.
10. R. Lidl and H. Niederreiter. *Introduction to Finite Fields and their Applications.* Cambridge University Press, 1986
11. F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, Amsterdam, 1981.
12. J. Radhakrishnan. *Personal communication*, January, 1996.
13. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.
14. M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.
15. D. A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.
16. M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180-193, March 1997. See also `http://theory.lcs.mit.edu/~ madhu/papers.html` for a more recent version.
17. J. H. van Lint. *Introduction to Coding Theory.* Springer-Verlag, New York, 1982.
18. A. Vardy. Algorithmic complexity in coding theory and the minimum distance problem. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pp. 92–109, 1997.
19. B. L. van der Waerden. Algebra, Volume 1. Frederick Ungar Publishing Co., Inc., page 82.
20. L. Welch and E. R. Berlekamp. Error correction of algebraic block codes. *US Patent* Number 4,633,470, issued December 1986.
21. R. E. Zippel. Probabilistic algorithms for sparse polynomials. *EUROSAM '79, Lecture Notes in Computer Science*, 72:216–226, 1979.