A NEW ALGORITHM FOR DECODING REED-SOLOMON CODES

SHUHONG GAO

Abstract. A new algorithm is developed for decoding Reed-Solomon codes. It uses fast Fourier transforms and computes the message symbols directly without explicitly finding error locations or error magnitudes. In the decoding radius (up to half of the minimum distance), the new method is easily adapted for error and erasure decoding. It can also detect all errors outside the decoding radius. Compared with the Berlekamp-Massey algorithm, discovered in the late 1960's, the new method seems simpler and more natural yet it has a similar time complexity.

1. Introduction

Reed-Solomon codes are the most popular codes in practical use today with applications ranging from CD players in our living rooms to spacecrafts in deep space exploration. Their main advantage lies in two facts: high capability of correcting both random and burst errors; and existence of efficient decoding algorithm for them, namely the Berlekamp-Massey algorithm, discovered in the late 1960's [1, 9]. The Berlekamp-Massey algorithm first finds the syndromes and then the error locations and error magnitudes. It has a quadratic running time and is efficient in practice. Possibly due to this reason, the Berlekamp-Massey approach has since become a prototype for decoding many other linear codes.

In this paper we present a new approach for decoding Reed-Solomon codes. Our method deviates from the Berlekamp-Massey approach in the sense that it computes message symbols directly without explicitly finding error locations or error magnitudes. The message polynomial is in some sense "visible" in the whole decoding process and our algorithm goes straight after it. The main operations used are interpolation, partial gcd, and long division of polynomials. They can all be implemented by fast algorithms based on FFTs. It is hoped that our approach provides an alternative way to study the decoding problem of many other codes including algebraic geometry codes.

We fix the following notations throughout the paper: q a prime power, $\mathbb{F}_q$ the finite field of q elements, n, k, d integers with $1 \leq k < n \leq q$ and $d = n - k + 1$.

Date: January 31, 2002.

The author was supported in part by National Science Foundation (NSF) under Grant DMS9970637, National Security Agency (NSA) under Grant MDA904-00-1-0048, Office of Naval Research (ONR) under Grant N00014-00-1-0565 in the DoD Multidisciplinary University Research Initiative (MURI) program, and South Carolina Commission on Higher Education under a Research Initiative Grant.

2. Encoding Reed-Solomon codes

We fix any n elements of $\mathbb{F}_q$, say $a_1, a_2, \ldots, a_n \in \mathbb{F}_q$. To encode a packet of k information symbols

$$(m_1, m_2, \ldots, m_k),$$

where each symbol m_i is an element in $\mathbb{F}_q$, first form the *message polynomial*

$$f(x) = m_1 + m_2 x + \cdots + m_k x^{k-1}, \tag{1}$$

then compute

$$c_i = f(a_i) \in \mathbb{F}_q, \quad i = 1, 2, \ldots, n. \tag{2}$$

The corresponding codeword is

$$\mathbf{c} = (c_1, c_2, \ldots, c_n).$$

When the information symbols take all the values in $\mathbb{F}_q$, we get q^k codewords of length n. It is straightforward to see that these codewords form a linear code over $\mathbb{F}_q$ with minimum distance $d = n - k + 1$ (which is best possible for any (n, k) linear code). This code was discovered by Reed and Solomon [11], so called an (n, k, d) Reed-Solomon code.

The encoding (2) was the original approach of Reed and Solomon, and it is not systematic in the sense that the information symbols do not appear directly as part of the codeword $\mathbf{c}$. This approach fell out favor after Gorenstein and Zierler [8] discovered that Reed-Solomon codes (for $n = q - 1$) are special cyclic codes and can be encoded via generator polynomials. Note that (2) is in fact a discrete Fourier transform in finite fields. We shall see below that our decoding algorithm uses the inverse Fourier transform. In many cases fast algorithms are available for computing a Fourier transform and its inverse, as described in Section 5. Hence, the nonsystematic encoding in (2) is advantageous for our approach.

3. Decoding Reed-Solomon codes

Let $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{F}_q^n$ be a received word which comes from a codeword $\mathbf{c}$ with t errors where $t \leq (d-1)/2$. Our goal is to find the message polynomial $f(x)$ in (1) that defined the original codeword $\mathbf{c}$. We precompute the polynomial

$$g_0 = \prod_{i=1}^{n} (x - a_i) \in \mathbb{F}_q[x].$$

Note that g_0 is known for many cases. For example, $g_0 = x^q - x$ when $n = q$, and $g_0 = x^n - 1$ when $n \mid q - 1$ and $a_1, a_2, \ldots, a_n$ form a multiplicative group in $\mathbb{F}_q$. To decode $\mathbf{b}$, we proceed as follows.

Algorithm 1: Decoding Reed-Solomon codes

Input: A received vector $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{F}_q^n$.
Output: A message polynomial $m_1 + m_2 x + \cdots m_k x^{k-1}$, or "Decoding failure."

Step 1: (**Interpolation**) Find the unique polynomial $g_1(x) \in \mathbb{F}_q[x]$ of degree $\leq n-1$ such that

$$g_1(a_i) = b_i, \quad 1 \leq i \leq n.$$

Step 2: (**Partial gcd**) Apply the extended Euclidean algorithm to $g_0(x)$ and $g_1(x)$. Stop when the remainder, say $g(x)$, has degree $< \frac{1}{2}(n+k)$. Suppose we have at this time

$$u(x)g_0(x) + v(x)g_1(x) = g(x).$$

Step 3: (**Long division**) Divide $g(x)$ by $v(x)$, say

$$g(x) = f_1(x)v(x) + r(x),$$

where $\deg r(x) < \deg v(x)$. If $r(x) = 0$ and $f_1(x)$ has degree $< k$ then output $f_1(x)$, otherwise output "Decoding failure" (which means that more than $(d-1)/2$ errors have occurred).

We shall see later that the polynomial $v(x)$ is in fact the error locator polynomial whose roots contain all the positions a_i where errors have occurred. Our decoding algorithm does not need to know the actual error positions a_i or the error magnitudes.

We now explain why the algorithm works. Recall the extended Euclidean algorithm (EEA) when applied to two nonzero polynomials r_0, r_1 over $\mathbb{F}_q$. Let

$$u_0 = 1, u_1 = 0, \quad v_0 = 0, v_1 = 1.$$

The extended Euclidean algorithm performs a sequence of long divisions:

$$r_{i-1} = q_i r_i + r_{i+1}, \quad \deg(r_{i+1}) < \deg(r_i), \ i = 1, 2, \ldots m,$$

where $r_i \neq 0$, $1 \leq i \leq m$, and $r_{m+1} = 0$, and simultaneously computes

$$u_{i+1} = u_{i-1} - q_i u_i, \quad v_{i+1} = v_{i-1} - q_i v_i, \quad 1 \leq i \leq m. \tag{3}$$

It is easy to prove by induction that $\gcd(r_0, r_1) = r_m$ (made monic if necessary) and that

$$r_i = u_i r_0 + v_i r_1, \quad 0 \leq i \leq m+1. \tag{4}$$

Lemma 3.1. *Let r_0 and r_1 be two nonzero polynomials over $\mathbb{F}_q$. Suppose the extended Euclidean algorithm performs the computation as described above. Then*

$$u_{m+1} = (-1)^{m+1}\frac{r_1}{r_m}, \quad v_{m+1} = (-1)^m \frac{r_0}{r_m}.$$

Proof. We have from (3) that

$$\begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \begin{bmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{bmatrix}.$$

Iterating the above matrix equation i times yields

$$\begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -q_{i-1} \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 \\ 1 & -q_1 \end{bmatrix} \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \end{bmatrix}.$$

Recall that

$$\begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Hence,

$$u_i v_{i+1} - u_{i+1} v_i = \det \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \prod_{j=1}^{i} \det \begin{bmatrix} 0 & 1 \\ 1 & -q_j \end{bmatrix} = (-1)^i.$$

Now we have from (4) that

$$\begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \end{bmatrix}, \quad 0 \le i \le m.$$

Therefore,

$$\begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix}^{-1} \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix} = (-1)^i \begin{bmatrix} v_{i+1} & -v_i \\ -u_{i+1} & u_i \end{bmatrix} \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix}, \quad 0 \le i \le m.$$

Since $r_{m+1} = 0$, the theorem follows from the above equation when $i = m$. □

The next lemma is the key to our decoding algorithm.

Lemma 3.2. *Let* $g_0(x) = w_0(x) \cdot r_0(x) + \epsilon_0(x)$ *and* $g_1(x) = w_0(x) \cdot r_1(x) + \epsilon_1(x)$, *with* $\gcd(r_0(x), r_1(x)) = 1$ *and*

$$\deg r_i(x) \le t, \quad \deg \epsilon_i(x) \le \ell, \quad i = 1, 2.$$

Suppose d_0 *satisfies*

$$\deg w_0(x) \ge d_0 > \ell + t.$$

Apply the extended Euclidean algorithm (EEA) to $g_0(x)$ *and* $g_1(x)$, *and stop whenever the remainder* $g(x)$ *has degree* $< d_0$. *Suppose that at termination we have*

$$u(x)g_0(x) + v(x)g_1(x) = g(x).$$

Then

$$u(x) = -\alpha\, r_1(x), \quad v(x) = \alpha\, r_0(x)$$

for some nonzero $\alpha \in \mathbb{F}_q$.

Proof. We prove that EEA computes the same quotient sequence for the pair r_0 and r_1 as for g_0 and g_1. More precisely, suppose

$$r_{i-1} = q_i r_i + r_{i+1}, \quad \deg r_{i+1} < \deg r_i, \quad i = 1, 2, \ldots, m,$$

where $r_{m+1} = 0$ and $r_m \in \mathbb{F}_q \setminus \{0\}$ (as $\gcd(r_0, r_1) = 1$). Let

$$\begin{array}{lll} u_0 = 1 & u_1 = 0 & u_{i+1} = u_{i-1} - q_i u_i, \quad 1 \le i \le m, \\ v_0 = 0 & v_1 = 1 & v_{i+1} = v_{i-1} - q_i v_i, \quad 1 \le i \le m. \end{array}$$

Then

$$r_i = u_i r_0 + v_i r_1, \quad 0 \le i \le m+1,$$

and

$$\deg u_i \le \deg r_1 \le t, \quad \deg v_i \le \deg r_0 \le t.$$

By Lemma 3.1,

$$u_{m+1} = (-1)^{m+1} r_1 / r_m, \quad v_{m+1} = (-1)^m r_0 / r_m. \tag{5}$$

Now define

$$g_i = u_i g_0 + v_i g_1, \quad 2 \le i \le m+1.$$

Certainly, we have

$$g_{i-1} = q_i g_i + g_{i+1}, \quad 1 \le i \le m.$$

We need to show that the degrees of $g_1, \ldots, g_{m+1}$ decrease strictly. Indeed, for $0 \le i \le m+1$,

$$g_i = u_i(w_0 r_0 + \epsilon_0) + v_i(w_0 r_1 + \epsilon_1) = w_0(u_i r_0 + v_i r_1) + (u_i\epsilon_0 + v_i\epsilon_1) = w_0 r_i + (u_i\epsilon_0 + v_i\epsilon_1).$$

Since $\deg(u_i\epsilon_0 + v_i\epsilon_1) \le \ell + t < d_0 \le \deg w_0$ for $0 \le i \le m+1$, we have

$$\deg g_i = \deg w_0 + \deg r_i \ge \deg w_0 \ge d_0 > \ell + t, \quad 0 \le i \le m,$$

and

$$\deg g_{m+1} = \deg(u_{m+1}\epsilon_0 + v_{m+1}\epsilon_1) \le \ell + t.$$

By our assumption above, the degrees of $r_1, \ldots, r_m$ decrease strictly, so do the degrees of $g_1, \ldots, g_{m+1}$. This means that the quotient sequence of g_0 and g_1, up to step m, is exactly $q_1, \ldots, q_m$, the same as that of r_0 and r_1. This in turn implies that the u and v sequences for g_0 and g_1 are also the same. Also, the step m is the first time that the remainder g_{m+1} has degree $< d_0$ and at this step

$$u_{m+1} g_0 + v_{m+1} g_1 = g_{m+1},$$

where u_{m+1} and v_{m+1} are as in (5). This proves the lemma. □

Now back to the correctness of our algorithm.

Theorem 3.3. *If the received vector* $\mathbf{b}$ *has distance at most* $(d-1)/2$ *from a codeword* $\mathbf{c}$ *defined by a message polynomial* $f(x)$ *in (1) then Algorithm 1 returns* $f(x)$*, otherwise "Decoding failure" is returned.*

Proof. Suppose the received vector $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ has distance $t \le (d-1)/2$ from a (unique) codeword $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ defined by $f(x)$ as in (1) and (2). Define the error locator polynomial to be

$$w(x) = \prod_{\substack{1 \le i \le n, \\ c_i \ne b_i}} (x - a_i)$$

so $\deg(w) = t$. Let $w_0(x)$ be the cofactor of $w(x)$ in $g_0(x)$, namely

$$g_0(x) = w_0(x) w(x).$$

Define $\bar{w}(x) \in \mathbb{F}_q[x]$ to be the unique polynomial with $\deg(\bar{w}) < t$ such that

$$\bar{w}(a_i) = (b_i - c_i)/w_0(a_i), \quad \text{for all } 1 \le i \le n \text{ with } b_i \ne c_i.$$

Then $\gcd(w(x), \bar{w}(x)) = 1$, and

$$g_1(x) = w_0(x) \cdot \bar{w}(x) + f(x),$$

as both sides have degrees less than n and have the same value b_i when evaluated at a_i for $1 \le i \le n$.

Let $d_0 = (n+k)/2$. Note that

$$\deg(w_0(x)) = n - t \ge d_0 > k - 1 + t \ge \deg(f(x)) + \deg(w(x)).$$

Then by Lemma 3.2, we have $u(x) = -\alpha\bar{w}(x)$ and $v(x) = \alpha w(x)$ for some $\alpha \in \mathbb{F}_q \setminus \{0\}$. Hence, $g(x) = u(x)g_0(x) + v(x)g_1(x) = v(x)f(x)$. This means that in Step 3 of Algorithm 1 the remainder should be zero and the quotient $f_1(x)$ is equal to $f(x)$ (which has degree $< k$) as expected.

On the other hand, suppose the algorithm returns a polynomial $f_1(x)$ in Step 3. Certainly, $f_1(x)$ defines a codeword (as it has degree $< k$). The identity in Step 2 implies that

$$u(x)g_0(x) = v(x)(f_1(x) - g_1(x)),$$

hence

$$v(a_i)(f_1(a_i) - g_1(a_i)) = 0, \quad 1 \le i \le n.$$

But $v(x)$ has degree $t \le (d-1)/2$, we see that $f_1(a_i) = g_1(a_i)$ for at least $n-t \ge n-(d-1)/2$ values of i. Hence $\mathbf{b}$ is within distance $(d-1)/2$ from the codeword defined by $f_1(x)$. Therefore, if $\mathbf{b}$ has distance greater than $(d-1)/2$ to every codeword, then the algorithm must return "Decoding failure" in Step 3. □

Theorem 3.3 shows that the only possible decoding error occurs when there are too many errors in the transmission moving original codeword into the ball of radius $(d-1)/2$ of another codeword. This is a case that is impossible to detect by any means from the received word alone.

The above proof shows more about the decoding algorithm. Note that

$$g_0 = w_0(x)w(x) \tag{6}$$
$$g_1 = w_0(x)\bar{w}(x) + f(x), \tag{7}$$

where $w(x)$ is the error locator polynomial. The polynomials $u(x)$ and $v(x)$ computed in Step 2 are actually $-\alpha\bar{w}(x)$ and $\alpha w(x)$, respectively, for some nonzero $\alpha \in \mathbb{F}_q$. Hence,

$$f(x) = g_1(x) + \frac{g_0(x)}{v(x)}u(x). \tag{8}$$

The message polynomial $f(x)$ is hidden in $g_1(x)$, yet "visible" to our algorithm, and we can go straight after it.

Alternatively, we observe from Lemma 3.2 that $u(x)$ and $v(x)$ can be computed from only the higher parts of the coefficients of g_0 and g_1. In fact, the polynomials $\epsilon_0(x)$ and $\epsilon_1(x)$ (the lower parts) can be arbitrary, so not affecting the computation leading to $u(x)$ and $v(x)$. Hence, we may simply ignore the coefficients at the lower parts. We need the degree ℓ (of $\epsilon_i(x)$) to satisfy

$$n - \ell > \ell + t,$$

implying that

$$\ell \le n - 2t - 1.$$

As $2t + 1 \le d$, we can take $\ell = n - d$. At the start of Step 2, we can rewrite $g_0(x)$ and $g_1(x)$ as

$$g_0(x) = (\text{terms of degree} \le n-d) + x^{n-d+1}s_0(x), \tag{9}$$
$$g_1(x) = (\text{terms of degree} \le n-d) + x^{n-d+1}s_1(x). \tag{10}$$

That is, $s_0(x)$ and $s_1(x)$ are nothing but the higher part of the coefficients $g_0(x)$ and $g_1(x)$, respectively. For example, $s_0(x) = x^{d-1}$ when $g_0 = x^q - x$ or $x^n - 1$. By Lemma 3.2, we can apply the Euclidean algorithm directly to $s_0(x)$ and $s_1(x)$ to get $u(x)$ and $v(x)$. Here $s_1(x)$ has degree at most $d-2$ (the $d-1$ higher coefficients of g_1) and serves as the "syndrome" of the received word. Algorithm 1 can thus be modified as follows.

Algorithm 1a: Decoding Reed-Solomon codes (modified)

Input: A received vector $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{F}_q^n$.

Output: A message polynomial $m_1 + m_2 x + \cdots m_k x^{k-1}$, or "Decoding failure."

Step 1: (**Interpolation**) Find the unique polynomial $g_1(x) \in \mathbb{F}_q[x]$ of degree $\leq n-1$ such that

$$g_1(a_i) = b_i, \quad 1 \leq i \leq n.$$

Step 2: (**Partial gcd**) Form the polynomials $s_0(x)$ and $s_1(x)$ as in (9) and (10) from $g_0(x)$ and $g_1(x)$. Apply the extended Euclidean algorithm to $s_0(x)$ and $s_1(x)$. Stop when the remainder, say $g(x)$, has degree $< (d+1)/2$. Suppose we have at this time

$$u(x)s_0(x) + v(x)s_1(x) = g(x).$$

Step 3: (**Division and multiplication**) Divide $g_0(x)$ by $v(x)$, say

$$g_0(x) = h_1(x)v(x) + r(x),$$

where $\deg r(x) < \deg v(x)$. If $r(x) \neq 0$ then output "Decoding failure"; otherwise, compute

$$f_1(x) := g_1(x) + h_1(x)u(x).$$

If $f_1(x)$ has degree $< k$ then output $f_1(x)$, otherwise output "Decoding failure."

With this modified algorithm, the gcd step is cheaper but the last step is more expensive. Theorem 3.3 above still holds for this modified version. We should remark that the step on partial gcd looks very similar to the approach of Sugiyama et al. [12].

4. Decoding with errors and erasures

It is easy to adapt our algorithm to decoding with erasures. This is desirable in practice as it is often possible to have information about some of the error locations and making use of this information will greatly enhance the performance of codes [10].

Let $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{F}_q^n$ be a received word with s erasures. Then it is possible to correct t further errors, where

$$2t + s < d = n - k + 1.$$

Let S be the set of erasure positions (that is, positions where errors have been detected), and the erasure polynomial

$$s(x) = \prod_{i \in S} (x - a_i).$$

If we ignore the positions in S in the original RS codewords, then we obtain another RS code with length $n-s$, dimension k and distance $d = n-s-k+1$. To decode $\mathbf{b}$, we could simply ignore the positions in S and apply Algorithm 1 or 1a to the truncated word. More precisely, in Algorithm 1 or 1a, one only needs to replace n by $n-s$, d by $d-s$, $g_0(x)$ by $g_0(x)/s(x)$, and to require in the interpolation step that $g_1(x)$, degree $\leq n-s-1$, be such that $g_1(a_i) = b_i$ for all $i \notin S$. This works since all the above arguments still apply for the truncated code (an RS code). The detailed modification of the algorithms is left to the reader.

5. Fast Fourier transforms

Our decoding algorithm needs efficient algorithms for evaluation, interpolation, gcd, multiplication and division of polynomials. Fortunately, all these operations can be computed via fast Fourier transforms (FFT) (see for instance [7]). In this section, we give a brief survey of the best algorithms and their time complexities available, including some recent work of the author.

For any fixed distinct points $a_1, a_2, \ldots, a_n \in \mathbb{F}_q$, the transform from a polynomial $f \in \mathbb{F}_q[x]$ to its values $f(a_1), f(a_2), \ldots, f(a_n)$ is called a *discrete Fourier transform*, denoted by $\mathrm{DFT}(f)$. We write

$$\mathrm{DFT}(f_0, f_1, \ldots, f_{n-1}) = (f(a_1), f(a_2), \ldots, f(a_n)),$$

where and whereafter we identify a polynomial $f = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1} \in \mathbb{F}[x]$ of degree $< n$ with its coefficient vector $(f_0, f_1, \ldots, f_{n-1})$. Hence, DFT is a bijection on $\mathbb{F}_q^n$. The interpolation is the inverse transform, called the *inverse discrete Fourier transform*, denoted by DFT^{-1}. By a fast Fourier transform (FFT), we mean *any fast algorithm* that computes DFT of length n with time complexity $\mathcal{O}(n(\log n)^\ell)$ for some small constant ℓ.

Over an arbitrary finite field $\mathbb{F}_q$, multiplication and division of polynomials of degree $< n$ can be computed using $\mathcal{O}(n \log n \operatorname{loglog} n)$ operations in $\mathbb{F}_q$, and gcd using $\mathcal{O}(n \log^2 n \operatorname{loglog} n)$ operations. For arbitrary points $a_1, a_2, \ldots, a_n$, DFT and DFT^{-1} can be computed using $\mathcal{O}(n \log^2 n \operatorname{loglog} n)$ operations in $\mathbb{F}_q$. The details of these algorithms can be found in Chapters 8–11 in [7]. The FFTs work well for large n, but may not be practical for small n (say $n < 1000$).

More efficient FFTs are possible if n is a product of small factors and the points a_i's have special structures, say form a group of order n. The most useful case is n being a power of 2 or a power of 3. The usual FFT requires that the n points a_i's form a multiplicative group of order n, that is, the polynomial $x^n - 1$ has n distinct roots in the underlying field. In this case we say that the field supports FFT, and we call such an FFT multiplicative. A multiplicative FFT has time complexity $\mathcal{O}(n \log n)$ and can be implemented in parallel time $\mathcal{O}(\log n)$, where the implicit constants are small.

Finite fields supporting multiplicative FFT. In the following, we list several families of finite fields $\mathbb{F}_q$ that support multiplicative FFTs.

(a) Fermat primes $F_i = 2^{2^i} + 1$, for $i = 0, 1, 2, 3$ or 4. That is,

$$p = 3, 5, 17, 257, 65537.$$

For each of these primes p, $\omega = 3$ is a primitive element in $\mathbb{F}_p$. Fermat conjectured that F_i is prime for all i, but he was completely wrong, since it is now known to be composite for all $5 \le i \le 32$. In fact, the next number $2^{2^5} + 1 = 641 \cdot 6700417$, and complete factorization of F_i is known for all $i \le 11$. Currently $i = 33$ is the smallest for which the compositeness of F_i is unknown (by August 29, 2001; see http://www.prothsearch.net/fermat.html).

(b) $p = 3 \cdot 2^k + 1$. All values of $k \le 2000$ that give primes of this form are

$$k = 1, 2, 5, 6, 8, 12, 18, 30, 36, 41, 66, 189, 201, 209, 276, 353, 408, 438, 534.$$

(c) $p = 5 \cdot 2^k + 1$. All values of $k \le 2000$ that give primes of this form are

$$k = 1, 3, 13, 15, 25, 39, 55, 75, 85, 127, 1947.$$

(d) $p = 7 \cdot 2^k + 1$. All values of $k \le 2000$ that give primes of this form are

$$k = 2, 4, 6, 14, 20, 26, 26, 50, 52, 92, 120, 174, 180, 190, 290, 320, 390, 432, 616, 830, 1804.$$

(e) $p = 9 \cdot 2^k + 1$. All values of $k \le 2000$ that give primes of this form are

$$\begin{aligned} k \;=\; & 1, 2, 3, 6, 7, 11, 14, 17, 33, 42, 43, 63, 65, 67, 81, 134, 162, 206, \\ & 211, 366, 663, 782, 1305, 1411, 1494. \end{aligned}$$

(f) Mersenne primes: $p = 2^k - 1$. All known Mersenne primes come from

$$\begin{aligned} k \;=\; & 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, \\ & 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503, \\ & 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593. \end{aligned}$$

For a Mersenne prime $p = 2^k - 1$,

$$p^2 - 1 = (p + 1)(p - 1) = 2^{k+1}(2^{k-1} - 1)$$

is divisible by $n = 2^{k+1}$. So one can apply FFT over $\mathbb{F}_{p^2}$. Since $p \equiv 3 \pmod 4$, we know $x^2 + 1$ is irreducible over $\mathbb{F}_p$ and

$$\mathbb{F}_{p^2} = \mathbb{F}_p[i] = \{a + bi : a, b \in \mathbb{F}_p\},$$

where $i^2 = -1$. For explicit construction of elements of order $n = 2^{k+1}$ in $\mathbb{F}_{p^2}$, see Blake, et al. [3]. It should be remarked that these fields $\mathbb{F}_{p^2}$ also support 4-adic FFTs which are more efficient than the usual binary FFTs.

For all of the above fields, we can take n to be a power of 2 with $n \mid (q - 1)$ and a_i all the roots of $x^n - 1$. Then an FFT and its inverse at these points can be computed using $\mathcal{O}(n \log n)$ operations in $\mathbb{F}_q$. By using FFTs, polynomial multiplication and division can also be computed using $\mathcal{O}(n \log n)$ operations, and gcd using $\mathcal{O}(n \log^2 n)$ operations. The implicit constants in all these running times are very small, so these algorithms are practical for $n \ge 256$.

Additive FFT. Unfortunately multiplicative FFTs are not supported by many finite fields, especially fields of characteristic two which are preferred in practical implementation of error-correcting codes. Cantor (1989) finds a way to use the additive structure of the underlying field to perform an FFT over a finite field of order p^m where m is a power of p. This method is generalized by von zur Gathen and Gerhard (1996) to arbitrary m. Their additive FFTs (for $p = 2$) use $\mathcal{O}(n \log^2 n)$ additions and $\mathcal{O}(n \log^2 n)$ multiplications in $\mathbb{F}_q$.

For fields of characteristic two and for $n = 2^m$, Gao [5] recently improves on Cantor's method. When m is a power of 2, the above time complexity can be improved to $\mathcal{O}(n \log n \operatorname{loglog} n)$. For arbitrary m, there is an additive FFT using $\mathcal{O}(n \log^2 n)$ additions and $\mathcal{O}(n \log n)$ multiplications in $\mathbb{F}_q$. These FFTs are highly parallel and can be implemented in parallel time $\mathcal{O}(\log^2 n)$. Since all the implicit constants are very small, these additive FFTs are suitable for practical implementation of RS codes.

Among all the majors operations, it seems that computing gcd is most troublesome to be implemented by fast algorithms. If the number t of errors is small, the fast algorithms mentioned above may not provide much advantage over the usual method via long division. But if t is large, one should definitely consider using the fast methods.

6. Conclusions

We have presented a new method for decoding Reed-Solomon codes. Our method is easily adapted to decode with errors and erasures. Additionally, all the errors outside the decoding radius can be detected, a desirable feature in many applications. Unlike the Berlekamp-Massey algorithm, our algorithm does not require the codes to be cyclic. It should be possible to adapt our method to BCH codes and to generalize it to decode algebraic geometry codes via Grobner bases. Indeed our forthcoming work will show how to generalize the current method to decode Hermitian codes via Grobner bases. We have also given a brief overview of the FFT-based fast algorithms available for the major operations needed in our algorithm, namely, polynomial evaluation, interpolation, multiplication, division and gcd. These fast algorithms should be considered in practical implementation of our decoding algorithm for long codes.

Acknowledgment. The author thanks Jeff Farr for useful comments on an earlier draft of the paper.

References

[1] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968. (Revised edition, Laguna Hills: Aegean Park Press, 1984.)

[2] R. E. Blahut, "Decoding of cyclic codes and codes on curves," in *Handbook of Coding Theory*, Vol. II (Eds. V. S. Pless, W. C. Huffman and R. A. Brualdi), Elsevier, 1998, 1569–1633.

[3] I.F. Blake, S. Gao and R.C. Mullin, "Explicit factorization of $x^{2^k} + 1$ over F_p with prime $p \equiv 3 \pmod 4$," *App. Alg. in Eng., Comm. and Comp.* **4** (1993), 89–94.

[4] D.G. Cantor, "On arithmetical algorithms over finite fields," *J. of Combinatorial Theory* **A 56** (1989), 285-300.

[5] S. Gao, "Additive fast Fourier transforms over finite fields," in preparation.
[6] J. von zur Gathen and J. Gerhard, "Arithmetic and factorization of polynomials over $\mathbb{F}_2$," in *Proc. ISSAC'96*, Zürich, Switzerland, 1996, ACM press, 1–9.
[7] J. von zur Gathen and J. Gerhard, *Modern computer algebra*, Cambridge University Press, New York, 1999.
[8] D. Gorenstein and N. Zierler, "A class of error-correcting codes in p^m symbols," *J. Soc. Indust. Appl. Math.* **9** (1961), 207–214.
[9] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Information Theory IT*-15 1969 122–127.
[10] M. B. Pursley, "Reed-Solomon codes in frequency-hop communications," in *Reed-Solomon Codes and Their Applications* (Eds. S. B. Wicker and V. K. Bhargava), IEEE Press, 1994, 150–174.
[11] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM J. Appl. Math.* **8** (1960), 300–304.
[12] Y. Sugiyama, M. Kasahara, S. Hirasawa, and Toshihiko Namekawa, "A method for solving key equation for decoding Goppa codes," *Information and Control* **27** (1975), 87–99.

Department of Mathematical Sciences, Clemson University, Clemson, SC 29634-0975 USA
E-mail address: sgao@math.clemson.edu