Simplified high-speed high-distance list decoding for alternant codes

Daniel J. Bernstein

Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to

Abstract. This paper presents a simplified list-decoding algorithm to correct any number w of errors in any alternant code of any length n with any designed distance $t+1$ over any finite field $\mathbf{F}_q$; in particular, in the classical Goppa codes used in the McEliece and Niederreiter public-key cryptosystems. The algorithm is efficient for w close to, and in many cases slightly beyond, the $\mathbf{F}_q$ Johnson bound $J' = n' - \sqrt{n'(n'-t-1)}$ where $n' = n(q-1)/q$, assuming $t+1 \le n'$. In the typical case that $qn/t \in (\lg n)^{O(1)}$ and that the parent field has $(\lg n)^{O(1)}$ bits, the algorithm uses $n(\lg n)^{O(1)}$ bit operations for $w \le J' - n/(\lg n)^{O(1)}$; $O(n^{4.5})$ bit operations for $w \le J' + o((\lg n)/\lg\lg n)$; and $n^{O(1)}$ bit operations for $w \le J' + O((\lg n)/\lg\lg n)$.

1 Introduction

Take any prime power q; integer $m \ge 1$; integer $n \ge m$ with $n \le q^m$; integer $t \ge 1$ with $t \le n/m$; distinct $\alpha_1, \ldots, \alpha_n \in \mathbf{F}_{q^m}$; and nonzero $\beta_1, \ldots, \beta_n \in \mathbf{F}_{q^m}$. Define

$$C = \big\{(\beta_1 f(\alpha_1), \ldots, \beta_n f(\alpha_n)) : \\ f \in \mathbf{F}_{q^m}[x]; \quad \deg f < n-t; \quad \beta_i f(\alpha_i) \in \mathbf{F}_q \text{ for each } i\big\}.$$

This set C is an $[n, \ge n-mt, \ge t+1]$ linear code over $\mathbf{F}_q$. In other words: it is a subspace of the $\mathbf{F}_q$-vector space $\mathbf{F}_q^n$; it has dimension at least $n-mt$, i.e., at least q^{n-mt} elements; and any two distinct elements of it have Hamming distance at least $t+1$, i.e., differ in at least $t+1$ coordinates.

Any code C defined as above is called an **alternant code**. This class of codes was introduced by Helgert in [38], independently by Chien and Choy in [22], and independently by Delsarte in [28]. The class includes binary Reed–Solomon codes, which had been introduced by Reed and Solomon in [47]; BCH codes, which had been introduced by Hocquenghem in [39] and independently by Bose and Ray-Chaudhuri in [16]; various odd-characteristic generalizations

This work was supported in part by NIST grant 60NANB10D263 and in part by the Cisco University Research Program. Permanent ID of this document: f529c2ab14c4ec22244b0a3f0190089b. Date: 2011.09.15.

introduced by Gorenstein and Zierler in [33]; and classical Goppa codes, which had been introduced by Goppa in [31] and [32].

The w-error-correction problem for C is the problem of finding $c \in C$, given a vector at distance w from c. For $w \le \lfloor t/2 \rfloor$ the vector dictates a unique possibility for c, but this does not mean that c is easy to find. There are at least q^{n-mt} codewords, and the cost of enumerating them all is exponential in n, except in the (rarely used) case that t is very close to n/m. Fortunately, early research produced much better algorithms for the $\lfloor t/2 \rfloor$-error-correction problem:

- Peterson in [46] introduced an algorithm using $n^{O(1)}$ arithmetic operations in $\mathbf{F}_{q^m}$. Each of those operations uses a polynomial number of bit operations, under the extremely weak assumption that q^m has $n^{O(1)}$ bits. Applications typically choose q^m to have only $O(\lg n)$ bits.
- Berlekamp in [7] introduced an algorithm using only $O(n^2)$ operations in $\mathbf{F}_{q^m}$. If q^m has $(\lg n)^{O(1)}$ bits then each operation in $\mathbf{F}_{q^m}$ uses $(\lg n)^{O(1)}$ bit operations, so Berlekamp's algorithm uses $n^2(\lg n)^{O(1)}$ bit operations.
- Justesen in [42], and independently Sarwate as reported in [48], introduced an algorithm using only $n(\lg n)^{2+o(1)}$ operations in $\mathbf{F}_{q^m}$. If q^m has only $(\lg n)^{O(1)}$ bits then this algorithm uses only $n(\lg n)^{O(1)}$ bit operations.

What about $w > \lfloor t/2 \rfloor$? The big-field Johnson bound states that there are only polynomially many possibilities for c if $w < n - \sqrt{n(n-t-1)}$, assuming $t+1 \le n$. Guruswami and Sudan, in a famous 1998 paper [35], introduced a polynomial-time algorithm to compute the list of possibilities for c if $w < n - \sqrt{n(n-t-1)}$. An intermediate range of w was already covered by an algorithm of Sudan in [50], but [35] introduced "multiplicities" to push w much higher.

Even better, the $\mathbf{F}_q$ Johnson bound states that there are only polynomially many possibilities for c if $w < n' - \sqrt{n'(n'-t-1)}$ where $n' = n(q-1)/q$, assuming $t+1 \le n'$ and $q \in n^{O(1)}$. In 2000 Koetter and Vardy introduced a polynomial-time algorithm to compute the list of possibilities; see [34, Section 6.3.8]. Compared to the unique-decoding case $w = \lfloor t/2 \rfloor$, the big-field Johnson bound extends the distance by approximately $t^2/8n$, and the $\mathbf{F}_q$ Johnson bound further extends the distance by approximately $t^2/8n(q-1)$; this improvement is particularly impressive for $q = 2$.

Unfortunately, "polynomial time" does not mean fast. Several subsequent papers have improved the complexity of list decoding, but each paper fails at least one, if not all, of the following desiderata:

- Speed. For example, the recent paper [4] reports list-decoding cost "quadratic in the blocklength n" (counting the number of operations in $\mathbf{F}_{q^m}$); but this is asymptotically much larger than the $n(\lg n)^{2+o(1)}$ that had been achieved decades earlier for $w = \lfloor t/2 \rfloor$.
- Effectiveness (how many errors are decoded). For example, the recent paper [52] is limited to the big-field Johnson distance, significantly below the $\mathbf{F}_q$ Johnson distance if q is small.
- Simplicity. For example, [3]—one of the few papers reporting essentially-linear-time list decoding—is sufficiently general to handle arbitrary weights,

such as the optimized Koetter–Vardy weights; but the user is required to trace the desired weights (after scaling and rounding to integers) through a thicket of degree computations.

It seems that every implementation of code-based cryptography avoids list decoding, even though [11, Section 7] pointed out years ago that list decoding improves the tradeoff between key size and security level against all known attacks. One can blame the non-use of list decoding on the lack of simple high-speed high-distance decoding algorithms. Some list-decoding papers try to compensate by adding generality, for example studying higher-genus algebraic-geometry codes, but if list decoding is not usable even for the most basic constructions of alternant codes then obviously it will also not be usable for higher-genus codes!

This paper presents a list-decoding algorithm that is simultaneously (1) fast, (2) effective, and (3) simple. The algorithm continues to work for arbitrarily large values of w, although its speed degrades as w approaches and passes the $\mathbf{F}_q$ Johnson bound. Specifically, in the typical case that n/t, q, and $\lg q^m$ are all in $(\lg n)^{O(1)}$, the algorithm uses

- $n(\lg n)^{O(1)}$ bit operations for $w \le n' - \sqrt{n'(n'-t-1)} - n/(\lg n)^{O(1)}$;
- $O(n^{4.5})$ bit operations for $w \le n' - \sqrt{n'(n'-t-1)} + o((\lg n)/\lg\lg n)$; and
- $n^{O(1)}$ bit operations for $w \le n' - \sqrt{n'(n'-t-1)} + O((\lg n)/\lg\lg n)$.

Note that the $n(\lg n)^{O(1)}$ bound does not imply competitive speed with other $n(\lg n)^{O(1)}$ algorithms; it merely implies that the speed ratio is bounded by $(\lg n)^{O(1)}$. However, the $O(n^{4.5})$ bound allows easy comparisons to, e.g., the $n^{7+o(1)}$ achieved in [4, Corollary 5.8] for w slightly below $n' - \sqrt{n'(n'-t-1)}$, or the $n^{6+o(1)}$ achieved in [6] for w slightly below $n - \sqrt{n(n-t-1)}$.

The word "simplified" in the title might suggest that I obtained this algorithm by starting from an existing acceleration of the Koetter–Vardy algorithm and simplifying it. I actually obtained the algorithm in a completely different way. I started with a very simple algorithm by Howgrave-Graham that was published in 1997 and that was subsequently understood to have the same decoding capability as the Guruswami–Sudan algorithm. I then tweaked the Howgrave-Graham algorithm to match the Koetter–Vardy results. The Howgrave-Graham algorithm does not seem to be widely known among coding theorists, including those working on code-based cryptography; see Section 3 and [9] for further discussion of the history.

2 Review of fast arithmetic

This section reviews several standard subroutines for fast multiplication, fast lattice-basis reduction, etc.

All of the algorithms here are $\mathbf{F}_{q^m}$-algebraic algorithms, i.e., sequences of additions, subtractions, multiplications, divisions, and comparisons of elements of $\mathbf{F}_{q^m}$. For a formal definition of this model of computation see, e.g., [18]. **Cost** here refers to total algebraic complexity over $\mathbf{F}_{q^m}$, i.e., the number of arithmetic operations performed in $\mathbf{F}_{q^m}$.

The weak assumption $\lg q^m \in (\lg n)^{O(1)}$ implies that each of these operations in $\mathbf{F}_{q^m}$ can be carried out using $(\lg n)^{O(1)}$ bit operations. The weaker assumption $\lg q^m \in n^{O(1)}$ implies that each of these operations in $\mathbf{F}_{q^m}$ can be carried out using $n^{O(1)}$ bit operations.

Fast multiplication. Multiplying two d-coefficient polynomials in $\mathbf{F}_{q^m}[x]$—i.e., two polynomials of degree below d—costs $d(\lg d)^{1+o(1)}$. See, e.g., my online survey paper [8, Section 4] for algorithmic details and credits.

Fast multiplication of many inputs. Computing a product of d linear polynomials costs $d(\lg d)^{2+o(1)}$. See, e.g., [8, Section 12].

Fast evaluation. Computing $Y(x_1), Y(x_2), \ldots, Y(x_d)$, given $x_1, \ldots, x_d \in \mathbf{F}_{q^m}$ and a d-coefficient polynomial $Y \in \mathbf{F}_{q^m}[x]$, costs $d(\lg d)^{2+o(1)}$. See, e.g., [8, Section 18].

Fast interpolation. For any distinct $x_1, \ldots, x_d \in \mathbf{F}_{q^m}$ and any $y_1, \ldots, y_d \in \mathbf{F}_{q^m}$ there is a unique polynomial $Y \in \mathbf{F}_{q^m}[x]$ of degree below d having $Y(x_1) = y_1$, $Y(x_2) = y_2$, and so on through $Y(x_d) = y_d$. Computing this polynomial Y from $x_1, \ldots, x_d, y_1, \ldots, y_d$ costs $d(\lg d)^{2+o(1)}$. See, e.g., [8, Section 23].

Fast lattice-basis reduction. If an $\ell \times \ell$ matrix over $\mathbf{F}_{q^m}[x]$ has nonzero determinant D then there is a nonzero linear combination Q of the matrix columns such that $\deg Q \le (\deg D)/\ell$. Here $\deg Q$ means the maximum degree of the entries of Q.

If each of the matrix entries is a d-coefficient polynomial then computing such a Q costs $\ell^{\Omega} d(\lg \ell d)^{O(1)}$ by [30, Theorem 3.8]. Here Ω is any positive real number such that $\ell \times \ell$ matrix multiplication costs $O(\ell^{\Omega})$. One can trivially take $\Omega = 3$, but state-of-the-art matrix-multiplication techniques have pushed Ω below 2.5.

There is an error in the proof of [30, Theorem 3.8]: the authors assume, without justification, that they can quickly find $x_0 \in \mathbf{F}_{q^m}$ such that $D(x_0) \ne 0$. Unfortunately, it is entirely possible that *every* $x_0 \in \mathbf{F}_{q^m}$ will have $D(x_0) = 0$; in such cases, the algorithm stated in [30, Section 3] will fail. The simplest workaround is to replace $\mathbf{F}_{q^m}$ by an extension having significantly more than $\deg D$ elements; extension degree $(\lg \ell d)^{O(1)}$ always suffices, leaving the cost bound $\ell^{\Omega} d(\lg \ell d)^{O(1)}$ unaffected. (Extension degree 2 suffices for the matrix shape used later in this paper, since D visibly splits into linear factors in $\mathbf{F}_{q^m}[x]$.)

A closer look at the algorithm in [30] shows that the cost is $d(\lg d)^{2+o(1)}$ if ℓ and the required extension degree are bounded by $(\lg d)^{o(1)}$. The same complexity also appeared later in [3]. As ℓ increases, the algorithm in [3] scales as $\ell^{3+o(1)}$ rather than $\ell^{\Omega+o(1)}$.

Fast root-finding. The traditional factorization method for a polynomial in $\mathbf{Q}[y]$, introduced by Zassenhaus in [55] four decades ago, begins with a factorization of the polynomial modulo a small prime number p, and then uses Newton iteration ("Hensel's lemma") to lift the factorization to factorizations modulo p^2,

p^4, etc. A few Newton steps produce enough p-adic precision to determine the factorization in $\mathbf{Q}[y]$; see, e.g., [29, Theorem 15.20]. This procedure relies on a preliminary "squarefree factorization" of the polynomial, but that factorization has essentially linear cost; see [29, Theorem 14.23].

In the case of linear factors (i.e., roots) the entire factorization procedure uses $\ell^{2+o(1)}d(\lg d)^{2+o(1)}$ bit operations for ℓ-coefficient polynomials with d-bit integer coefficients; see [29, Theorem 15.21]. There has been a tremendous amount of research on algorithms for the first step, factoring in $(\mathbf{Z}/p)[y]$, but rather naive algorithms are adequate if ℓ is much smaller than d and if one allows randomization. There has also been a tremendous amount of research on algorithms to handle higher-degree factors, but for this paper linear factors are adequate.

One can obtain essentially the same speed by computing approximate roots in $\mathbf{R}$ with an analogous Newton iteration, but working with the p-adic numbers $\mathbf{Q}_p$ is simpler because it avoids roundoff error. There are still a few technical details that require attention: one must avoid primes p that divide denominators of the original coefficients; one must also avoid primes p that create new squared factors. There are not many bad choices of p; see [29, Lemma 15.1].

Zassenhaus's method is not limited to the rational number field $\mathbf{Q}$. Replacing $\mathbf{Q}$ by the rational function field $\mathbf{F}_{q^m}(x)$, and replacing the small prime p of $\mathbf{Z}$ by a small irreducible element p of $\mathbf{F}_{q^m}[x]$, produces a factorization method for $\mathbf{F}_{q^m}(x)[y]$; see, e.g., [29, Theorem 15.23]. Squarefree factorization becomes slightly more complicated, as discussed in [29, page 447], but is still fast. The cost for the initial factorization modulo p is $\ell^{2+o(1)}(\lg q^m)^{1+o(1)}$ by [29, Theorem 14.14]. There are subquadratic factorization algorithms in the literature, but this refinement is not necessary for this paper.

The root-finding conclusion that matters for this paper—the polynomial analogue of [29, Theorem 15.21]—is the following. There is a standard algorithm that, given a nonzero polynomial $Q \in \mathbf{F}_{q^m}(x)[y]$, finds all y-roots of Q. If Q is an ℓ-coefficient polynomial (in y), each coefficient in turn being a d-coefficient polynomial (in x), then the entire procedure costs $\ell^{2+o(1)}((\lg q^m)^{1+o(1)}+d(\lg d)^{2+o(1)})$. Note that this cost bound is influenced by $\lg q^m$, the number of bits of the parent field $\mathbf{F}_{q^m}$; one needs to put limits on q^m not merely to control the translation from cost into bit operations, but also to control the cost of factorization.

3 Correcting nearly $n - \sqrt{n(n-t-1)}$ errors

This section states a simple high-speed list-decoding algorithm that corrects errors up to the big-field Johnson bound. The algorithm in the next section is more general and more powerful, correcting more errors; but the algorithm in this section is slightly simpler, and the reader is encouraged to read it first.

Parameters. This algorithm has three parameters: a positive integer $w \le n$, the number of errors to be corrected; an integer $k \ge 0$; and an integer $\ell \ge k$. The algorithm assumes that $t+1 \le n$ and that these parameters satisfy

$$n\frac{k(k+1)}{2} + (n-t-1)\frac{\ell(\ell-1)}{2} < \ell k(n-w),$$

i.e., $(1-(t+1)/n)(1-1/\ell) < (1-w/n)^2 - (1-w/n-k/\ell)^2 - k/\ell^2$.

One can take ℓ in $O(n^2)$ for any w smaller than the big-field Johnson bound. My main interest is in the case $\ell \in (\lg n)^{O(1)}$, achievable when there is a noticeable gap between w and the big-field Johnson bound. Further notes on parameter selection appear below. The total cost of the algorithm will turn out to be bounded by

- $n(\lg n)^{O(1)}$ if $\ell \in (\lg n)^{O(1)}$ and $\lg q^m \in (\lg n)^{O(1)}$; and by
- $n^{\Omega+2+o(1)}$ if $\ell \in O(n)$ and $\lg q^m \in O(n^{\Omega})$; and by
- $n^{2\Omega+3+o(1)}$ if $\ell \in O(n^2)$ and $\lg q^m \in O(n^{2\Omega-1})$; and by
- $n^{O(1)}$ if $\ell \in O(n^2)$ and $\lg q^m \in n^{O(1)}$.

For example, Step 2 below costs $\ell^3 n(\lg \ell n)^{1+o(1)}$, which is visibly within each of these bounds. I will state the cost of each step as a function of ℓ, n, and (when relevant) q^m.

Input and output. The algorithm input is a vector $v \in \mathbf{F}_q^n$. The algorithm output is the set of $c \in C$ of Hamming distance at most w from v.

Step 1: initial interpolation. Compute the polynomial $A = (x-\alpha_1)(x-\alpha_2)\cdots(x-\alpha_n) \in \mathbf{F}_{q^m}[x]$. Also compute the unique polynomial $V \in \mathbf{F}_{q^m}[x]$ with $\deg V < n$ satisfying $V(\alpha_1) = v_1/\beta_1$, $V(\alpha_2) = v_2/\beta_2$, and so on through $V(\alpha_n) = v_n/\beta_n$. This costs $n(\lg n)^{2+o(1)}$.

Step 2: lattice-basis construction. Define $X = x^{n-t-1}$ and $F = Xy - V \in \mathbf{F}_{q^m}[x,y]$. Compute the ℓ polynomials

$$\begin{aligned}
M_0 &= A^k;\\
M_1 &= A^{k-1}F = A^{k-1}Xy - A^{k-1}V;\\
M_2 &= A^{k-2}F^2 = A^{k-2}X^2y^2 - 2A^{k-2}XVy + A^{k-2}V^2;\\
&\vdots\\
M_{k-1} &= AF^{k-1} = AX^{k-1}y^{k-1} - \cdots;\\
M_k &= F^k = X^k y^k - \cdots;\\
M_{k+1} &= F^{k+1} = X^{k+1}y^{k+1} - \cdots;\\
&\vdots\\
M_{\ell-1} &= F^{\ell-1} = X^{\ell-1}y^{\ell-1} - \cdots
\end{aligned}$$

in $\mathbf{F}_{q^m}[x,y]$. If $\ell = k$ then $M_{\ell-1}$ is defined as AF^{k-1}, not $F^{\ell-1}$. (One can save time by replacing $F^k, F^{k+1}, \ldots, F^{\ell-1}$ with $F^k, XyF^k, \ldots, (Xy)^{\ell-1-k}F^k$, but the speedup is not visible at the level of detail of the analysis below.)

The coefficients of powers of y here form an $\ell \times \ell$ triangular matrix. There are several straightforward ways to compute all of the matrix entries with a total of $O(\ell^2)$ multiplications in $\mathbf{F}_{q^m}[x]$, each multiplication involving polynomials of degree $O(\ell n)$. The total cost is just $\ell^3 n(\lg \ell n)^{1+o(1)}$.

Step 3: lattice-basis reduction. The determinant of the aforementioned $\ell \times \ell$ matrix of coefficients of $M_0, \ldots, M_{\ell-1}$ is the product of the diagonal entries of the matrix (since the matrix is triangular), i.e., the product of the leading coefficients of $M_0, \ldots, M_{\ell-1}$, namely

$$A^k \cdot A^{k-1}X \cdot A^{k-2}X^2 \cdots X^k \cdot X^{k+1} \cdots X^{\ell-1} = A^{k(k+1)/2} X^{\ell(\ell-1)/2},$$

of degree $nk(k+1)/2 + (n-t-1)\ell(\ell-1)/2$. Inside the lattice $\mathbf{F}_{q^m}[x]M_0 + \cdots + \mathbf{F}_{q^m}[x]M_{\ell-1} \subseteq \mathbf{F}_{q^m}[x,y]$ find a nonzero polynomial Q having x-degree at most $(nk(k+1)/2 + (n-t-1)\ell(\ell-1)/2)/\ell$, and therefore x-degree below $k(n-w)$. This costs $\ell^{\Omega} n\ell(\lg \ell^2 n)^{O(1)} = \ell^{\Omega+1} n(\lg \ell n)^{O(1)}$.

Step 4: factorization. Compute all $f \in \mathbf{F}_{q^m}[x]$ such that $Q(x, f/X) = 0$; i.e., compute all factors of Q having the form $y - f/X$ with $f \in \mathbf{F}_{q^m}[x]$. Note that there are at most $\ell - 1$ such factors, since Q has y-degree at most $\ell - 1$. This costs $\ell^{2+o(1)}((\lg q^m)^{1+o(1)} + n\ell(\lg \ell n)^{2+o(1)})$.

For each polynomial $f \in \mathbf{F}_{q^m}[x]$ such that $Q(x, f/X) = 0$ and $\deg f < n - t$: Compute $c = (\beta_1 f(\alpha_1), \ldots, \beta_n f(\alpha_n)) \in \mathbf{F}_{q^m}^n$. Output c if $c \in \mathbf{F}_q^n$ and $|c - v| \leq w$, where $|c - v|$ means the Hamming weight of $c - v$. This costs $n(\lg n)^{2+o(1)}$.

Why the algorithm works. Each output c from the algorithm is checked, in Step 4, to be an element of C with $|c - v| \leq w$.

Conversely, consider any $c \in C$ with $|c - v| \leq w$. There is a polynomial $f \in \mathbf{F}_{q^m}[x]$ with $\deg f < n - t$ such that $c = (\beta_1 f(\alpha_1), \ldots, \beta_n f(\alpha_n))$. The goal is to show that the algorithm outputs c; equivalently, that f is found in Step 4 of the algorithm.

The hypothesis $|c - v| \leq w$ means that there are at least $n - w$ indices i for which $c_i = v_i$; i.e., for which $\beta_i f(\alpha_i) = \beta_i V(\alpha_i)$; i.e., for which α_i is a root of $f - V$. In other words, $\gcd\{A, f - V\}$ has degree at least $n - w$.

Consider the map $y \mapsto f/X$ from $\mathbf{F}_{q^m}[x, Xy]$ to $\mathbf{F}_{q^m}[x]$. The image of $F = Xy - V$ is $f - V$, so the images of $M_0, M_1, \ldots, M_{\ell-1}$ are $A^k, A^{k-1}(f-V), \ldots, (f-V)^k, \ldots, (f-V)^\ell$. Each of these polynomials is divisible by $\gcd\{A, f-V\}^k$. The image of Q, namely $Q(x, f/X)$, is therefore also divisible by $\gcd\{A, f - V\}^k$.

Write Q as $Q_0 + Q_1 y + \cdots + Q_{\ell-1} y^{\ell-1}$. Then $Q(x, f/X) = Q_0 + Q_1(f/X) + \cdots + Q_{\ell-1}(f/X)^{\ell-1}$. Each Q_i has degree below $k(n-w)$, and f/X has degree at most 0, so $Q(x, f/X)$ has degree below $k(n-w)$; but $Q(x, f/X)$ is divisible by $\gcd\{A, f - V\}^k$, which has degree at least $k(n-w)$. Consequently $Q(x, f/X) = 0$ as claimed.

Notes on parameter selection. Suitable k, ℓ exist with $\ell \in O(nt)$ whenever w is smaller than the big-field Johnson bound. For example, the integers $k = (n-w)(t+1) \geq 0$ and $\ell = n(t+1) > k$ have $(1 - (t+1)/n)(1 - 1/\ell) = 1 - (t+1)/n - 1/\ell + 1/n^2$ and $(1 - w/n - k/\ell)^2 + k/\ell^2 = (1 - w/n)/\ell < 1/\ell$; so w, k, ℓ are in the parameter space if $(1 - w/n)^2 \geq 1 - (t+1)/n + 1/n^2$, i.e., if $(n-w)^2 \geq n(n-t-1) + 1$. Both $(n-w)^2$ and $n(n-t-1)$ are integers, so this condition is equivalent to $(n-w)^2 > n(n-t-1)$, i.e., $w < n - \sqrt{n(n-t-1)}$.

This choice of ℓ is simpler and smaller than the choice made in [36, Lemma 7 and Proposition 9]. Here is an absurdly large numerical example to illustrate the worst-case asymptotics: for $n = 1000007$ and $t = 67774$ and $w = 34482$, one can take $k = 65438456875$ and $\ell = 67775474425$, while [36, Lemma 7] chooses $k = 932238525625$.

My main interest is in much smaller values of ℓ. Choosing k as $\lfloor(1 - w/n)\ell\rfloor$ guarantees $0 \le k < \ell$ since $w > 0$, and guarantees $(1 - w/n - k/\ell)^2 + k/\ell^2 < 1/\ell^2 + 1/\ell$, so w, k, ℓ are in the parameter space if $(1 - w/n)^2 \ge (1 - (t + 1)/n)(1 - 1/\ell) + 1/\ell^2 + 1/\ell$; i.e., $(1 - w/n)^2 \ge 1 - (t + 1)/n + (t + 1)/n\ell + 1/\ell^2$; i.e., $(1 - w/n)^2 - (1 - J/n)^2 \ge (t + 1)/n\ell + 1/\ell^2$ where J is the big-field Johnson bound; i.e., $J - w \ge ((t + 1)/\ell + n/\ell^2)/(2 - w/n - J/n)$. One can achieve this with $\ell \in (\lg n)^{O(1)}$ if $J - w$ is at least $n/(\lg n)^{O(1)}$.

There are limits to how far this idea can be pushed. For example, it is tempting to take k, ℓ as constants, so that cost factors such as ℓ^2 can be replaced by $O(1)$. The same replacement was used to justify, e.g., the statement "quadratic in the blocklength n" in [4, Abstract]. Apparently it is not instantly obvious that—at least for small q, such as the case $q = 2$ highlighted in [4]—this replacement is fundamentally flawed!

The difficulty is the following. If q is constant, or more generally $n^{o(1)}$, then $t \in o(n)$, so $J - t/2 \in o(t)$. Choosing $k, \ell \in O(1)$ then forces w to be smaller than $\lfloor t/2\rfloor$ for all sufficiently large n: in other words, the algorithm cannot correct more errors than Berlekamp's algorithm once n is sufficiently large. For the same reason, the "quadratic" claim in [4] is content-free: it might be true that taking constants k, ℓ limits the algorithm in [4] to cost $O(n^2)$, but then the algorithm cannot correct more errors than a trivial combination of brute-force list decoding for small n and Berlekamp's algorithm for large n, which also costs $O(n^2)$.

Of course, this criticism does not apply to bounds that treat ϵ, k, ℓ as variables, such as the bound $O(n^2/\epsilon^5)$ in [4, Corollary 5.7]. Furthermore, the "rational" algorithms of [54] and [10] allow a better tradeoff between k, ℓ, w and can meaningfully take $k, \ell \in O(1)$.

History. Håstad showed in 1988 that one could find all small roots of a polynomial modulo a large integer N by applying the famous LLL lattice-basis reduction algorithm. The same result was found independently by Vallée, Girault, and Toffin in 1989. See [37] and [53].

Coppersmith, in a famous 1996 paper, incorporated multiplicities into the Vallée–Girault–Toffin algorithm, drastically increasing the range of roots that could be found. Coppersmith also showed that similar lattices could be used to find not merely polynomial values that are *multiples* of N but also polynomial values that are *divisors* of N. See [24] and [25].

The next year Howgrave-Graham in [40] introduced a critical simplification in Coppersmith's algorithm. Coppersmith had identified the relevant lattice by linear constraints; Howgrave-Graham directly wrote down generators for the lattice. For example, for the problem of finding a divisor of N within X of V, Howgrave-Graham chose parameters k, ℓ, wrote down the lattice generated by $N^k, N^{k-1}(Xy + V), \ldots, (Xy + V)^k, \ldots, (Xy + V)^k(Xy)^{\ell-k-1}$, found a short vector Q in the lattice, and found small roots of Q. See [41, page 101] (with "p_0" for V, "u" for k, "h" for ℓ, "b_1" for Q, "N" for N, and "X" for X).

The same algorithm finds any integer within X of V that has a sufficiently large common divisor with N. One does not need the integer to *be* the divisor. This generalized perspective did not appear in [**24**], [**25**], [**40**], or [**41**], but did appear in papers a few years later, as discussed below.

For comparison, the problem of decoding Reed–Solomon codes is the problem of finding a polynomial f no larger than $X = x^{n-t-1}$ sharing many values with a received polynomial V (interpolated from the received word); i.e., the problem of finding a polynomial (namely $V - f$) that is within X of V and that has a large common divisor with $(x - \alpha_1) \cdots (x - \alpha_n)$. Except for a trivial replacement of integers with polynomials, this problem is a special case of the problem stated in the previous paragraph, and the decoding algorithm displayed in this section—correcting approximately $n - \sqrt{n(n-t-1)}$ errors—is a special case of the Howgrave-Graham algorithm.

The first announcement of this decoding effectiveness was by Guruswami and Sudan in [**35**] in 1998. With hindsight it is easy to see that [**35**] constructs the same lattice as Howgrave-Graham, finds the same short vector Q in the lattice, and finds the same roots of Q. Like Coppersmith, and unlike Howgrave-Graham, [**35**] identifies the lattice through linear constraints. Unlike Coppersmith, [**35**] states these constraints locally: the lattice is exactly the set of polynomials of degree below ℓ that vanish to multiplicity at least k at various points. This local perspective allowed Guruswami and Sudan to generalize, varying multiplicities separately at each point; but this generalization is not necessary for any of the decoding problems that I am considering, and it makes the algorithm very slow. [**35**] uses linear algebra to solve a large two-dimensional interpolation problem, finding a short vector Q in the specified lattice; it is much more efficient to first solve a simpler one-dimensional interpolation problem (computing V), and then write down basis vectors for the same lattice (namely A^k, $A^{k-1}F$, etc.).

Boneh in [**14**], motivated by the Guruswami–Sudan results, stated a CRT list-decoding algorithm with quantitatively analogous error-correcting capabilities. Boneh also stated an algorithm for the more general problem of finding any polynomial value having a large gcd with N; this obviously includes the multiple-of-N problems and the divisor-of-N problems. The algorithm in [**14**] constructs the same lattice as the Howgrave-Graham algorithm (in the same way), finds the same Q, and finds the same roots; the only difference is that the Howgrave-Graham algorithm throws away more of the outputs. The very large overlap between the algorithms was not pointed out in [**14**].

In 2003 I posted the first draft of a survey paper [**9**] giving a unified algorithm statement for univariate polynomials over $\mathbf{Q}$. I showed that a unified parameter optimization produced, as special cases, the quantitative results that had been obtained by Coppersmith, Howgrave-Graham, Boneh, et al. for various applications. I took a slightly broader perspective, allowing a large gcd for polynomial values on *rational* inputs, although at the time I did not see any way to use this extra generality; subsequent applications include [**54**], [**10**], and [**20**].

I discussed CRT decoding in [**9**, Section 7], and said that replacing $\mathbf{Q}$ with a rational function field in the same algorithm would decode Reed–Solomon codes as effectively as the Guruswami–Sudan algorithm. I had not actually read the Guruswami–Sudan paper at that point, and I did not realize that Guruswami and Sudan were missing the Howgrave-Graham simplification. I also had no idea that Koetter and Vardy had quantitatively improved the Guruswami–Sudan results, moving from the big-field Johnson bound to the $\mathbf{F}_q$ Johnson bound; I learned this much later when Augot kindly sent me a copy of [4]. I do not see any way to use the algorithm stated in [9] to obtain the Koetter–Vardy results: an extra tweak is required, and is the main content of Section 4 of this paper. The advantages of this tweaked algorithm over the Koetter–Vardy algorithm are analogous to the advantages of the Howgrave-Graham algorithm over the Guruswami–Sudan algorithm: most importantly, the local specification of the lattice is eliminated in favor of directly writing down lattice generators starting from V.

Cohn and Heninger in [23] presented an explicit function-field version of the Howgrave-Graham algorithm, including a generalization from the rational function field $\mathbf{F}_{q^m}(x)$ to arbitrary function fields; this generalization includes list decoding for algebraic-geometry codes. In the case of Reed–Solomon codes, [23, Section 6] reaches the big-field Johnson bound with cost only $n^{2\Omega+3+o(1)}$. Cost bounds for other choices of ℓ can also be extracted straightforwardly from the analysis in [23] and match the cost bounds shown in this section. However, this generalization still does not cover the Koetter–Vardy results.

4 Correcting nearly $n' - \sqrt{n'(n'-t-1)}$ errors

This section states a simple high-speed list-decoding algorithm that corrects errors up to the $\mathbf{F}_q$ Johnson bound.

Parameters. The algorithm has four parameters: a positive integer $w \le n$, the number of errors to be corrected; an integer $j \ge 0$; an integer $k \ge j$; and an integer $\ell \ge (q-1)j + k$. The algorithm assumes that $t + 1 \le n$ and that these parameters satisfy

$$n\frac{k(k+1)}{2} + n(q-1)\frac{j(j+1)}{2} + (n-t-1)\frac{\ell(\ell-1)}{2} < \ell(k(n-w) + jw),$$

i.e., $(1-(t+1)/n)(1-1/\ell) < (1-w/n)^2 + (w/n)^2/(q-1) - (1-w/n-k/\ell)^2 - (w/n-(q-1)j/\ell)^2/(q-1) - k/\ell^2 - (q-1)j/\ell^2$.

Suitable j, k, ℓ exist whenever w is smaller than the $\mathbf{F}_q$ Johnson bound, as discussed below. The special case $j = 0$ of this algorithm (with the computations of B and E straightforwardly eliminated) is exactly the algorithm of the previous section, and is usable only when w is smaller than the big-field Johnson bound.

The asymptotic cost bounds for this algorithm, as functions of n, ℓ, q^m, are exactly as in the previous section: for example, the cost is bounded by $n(\lg n)^{O(1)}$ if $\ell \in (\lg n)^{O(1)}$ and $\lg q^m \in (\lg n)^{O(1)}$, and is bounded by $n^{\Omega+2+o(1)}$ if $\ell \in O(n)$ and $\lg q^m \in O(n^{\Omega})$.

Input and output. The algorithm input is a vector $v \in \mathbf{F}_q^n$. The algorithm output is the set of $c \in C$ of Hamming distance at most w from v.

Step 1: initial interpolation. Compute the polynomial $A = (x - \alpha_1)(x - \alpha_2)\cdots(x - \alpha_n) \in \mathbf{F}_{q^m}[x]$; the unique polynomial $V \in \mathbf{F}_{q^m}[x]$ with $\deg V < n$ satisfying $V(\alpha_1) = v_1/\beta_1$, $V(\alpha_2) = v_2/\beta_2$, and so on through $V(\alpha_n) = v_n/\beta_n$; and the unique polynomial $B \in \mathbf{F}_{q^m}[x]$ with $\deg B < n$ satisfying $B(\alpha_1) = 1/\beta_1^{q-1}$, $B(\alpha_2) = 1/\beta_2^{q-1}$, and so on through $B(\alpha_n) = 1/\beta_n^{q-1}$.

Step 2: lattice-basis construction. Define $X = x^{n-t-1}$; $F = Xy - V \in \mathbf{F}_{q^m}[x, y]$; and $E = F^q - FB$. Compute the ℓ polynomials $M_0, M_1, \dots, M_{\ell-1} \in \mathbf{F}_{q^m}[x, y]$ shown in Figure 4.1. Observe that each of $M_0, M_1, \dots, M_{\ell-1}$ includes A, E, and F to a total power of at least k; that each of $M_0, M_1, \dots, M_{\ell-1}$ includes A and E to a total power of at least j; and that M_i has y-degree i.

The simplest strategy is to begin by computing $E, E^2, \dots, E^j$; $A, A^2, \dots, A^k$; and $F, F^2, \dots, F^{\max\{k-j+q-1,\ell-qj-1\}}$. Each M_i is then a product of three known polynomials. Overall this procedure uses $O(\ell)$ polynomial products in $\mathbf{F}_{q^m}[x, y]$, each of product degree $\le \ell - 1$ in y and $O(\ell n)$ in x. Kronecker substitution $x \mapsto y^\ell$ reduces these products to $O(\ell^2 n)$-coefficient products in $\mathbf{F}_{q^m}[y]$, each of which costs $\ell^2 n(\lg \ell^2 n)^{1+o(1)}$, for a total cost of $\ell^3 n(\lg \ell n)^{1+o(1)}$.

Step 3: lattice-basis reduction. The matrix of coefficients of $M_0, \dots, M_{\ell-1}$ has determinant

$$A^{(k-j)(k+j+1)/2+qj(j+1)/2} X^{\ell(\ell-1)/2} = A^{k(k+1)/2+(q-1)j(j+1)/2} X^{\ell(\ell-1)/2}$$

of degree $nk(k+1)/2 + n(q-1)j(j+1)/2 + (n-t-1)\ell(\ell-1)/2$. Inside the lattice $\mathbf{F}_{q^m}[x]M_0 + \cdots + \mathbf{F}_{q^m}[x]M_{\ell-1} \subseteq \mathbf{F}_{q^m}[x, y]$ find a nonzero polynomial Q having x-degree at most $(nk(k+1)/2 + n(q-1)j(j+1)/2 + (n-t-1)\ell(\ell-1)/2)/\ell$, and therefore x-degree below $k(n-w) + jw$.

Step 4: factorization. Compute all $f \in \mathbf{F}_{q^m}[x]$ such that $Q(x, f/X) = 0$; i.e., compute all factors of Q having the form $y - f/X$ with $f \in \mathbf{F}_{q^m}[x]$. For each polynomial $f \in \mathbf{F}_{q^m}[x]$ such that $Q(x, f/X) = 0$ and $\deg f < n - t$: Compute $c = (\beta_1 f(\alpha_1), \dots, \beta_n f(\alpha_n)) \in \mathbf{F}_{q^m}^n$. Output c if $c \in \mathbf{F}_q^n$ and $|c - v| \le w$.

Why the algorithm works. Consider any $c \in C$ with $|c - v| \le w$. There is a polynomial $f \in \mathbf{F}_{q^m}[x]$ with $\deg f < n-t$ such that $c = (\beta_1 f(\alpha_1), \dots, \beta_n f(\alpha_n))$. The goal, as in the previous section, is to show that the algorithm finds f in Step 4.

As before consider the map $y \mapsto f/X$ from $\mathbf{F}_{q^m}[x, Xy]$ to $\mathbf{F}_{q^m}[x]$. This map takes A, F, E to $A, f - V, (f - V)^q - (f - V)B$ respectively.

There are exactly $n - |c-v|$ indices i for which $c_i = v_i$, i.e., for which $f(\alpha_i) = V(\alpha_i)$. Each of these indices has $x - \alpha_i$ dividing $f - V$, A, and $(f-V)^q - (f-V)B$, so $(x - \alpha_i)^k$ divides the images of $M_0, M_1, \dots, M_{\ell-1}$.

There are also exactly $|c - v|$ indices i for which $c_i \ne v_i$, i.e., for which $\beta_i f(\alpha_i) \ne \beta_i V(\alpha_i)$. Both $\beta_i f(\alpha_i)$ and $\beta_i V(\alpha_i)$ are in $\mathbf{F}_q$, so the difference $\beta_i f(\alpha_i) - \beta_i V(\alpha_i)$ is a nonzero element of $\mathbf{F}_q$; i.e., $\beta_i^{q-1}(f(\alpha_i) - V(\alpha_i))^{q-1} = 1$; i.e., $(f(\alpha_i) - V(\alpha_i))^{q-1} = B(\alpha_i)$. Each of these indices has $x - \alpha_i$ dividing both $(f-V)^q - (f-V)B$ and A, so $(x - \alpha_i)^j$ divides the images of $M_0, M_1, \dots, M_{\ell-1}$.

The image of Q is thus divisible by $\prod_{i:c_i=v_i}(x-\alpha_i)^k \cdot \prod_{i:c_i\ne v_i}(x-\alpha_i)^j$, which has degree $k(n - |c-v|) + j|c-v| = kn - (k-j)|c-v| \ge kn - (k-j)w =$

$$\begin{array}{rll}
M_0 &= A^k F^0; & \text{(start of initial batch)}\\
M_1 &= A^{k-1} F^1; & \\
&\vdots & \\
M_{k-j-1} &= A^{j+1} F^{k-j-1}; & \\
M_{k-j} &= A^j F^{k-j}; & \text{(start of intermediate batch 0)}\\
M_{k-j+1} &= A^j F^{k-j+1}; & \\
&\vdots & \\
M_{k-j+q-1} &= A^j F^{k-j+q-1}; & \\
M_{k-j+q} &= A^{j-1} E F^{k-j}; & \text{(start of intermediate batch 1)}\\
M_{k-j+q+1} &= A^{j-1} E F^{k-j+1}; & \\
&\vdots & \\
M_{k-j+2q-1} &= A^{j-1} E F^{k-j+q-1}; & \\
&\vdots & \vdots\\
M_{k-j+(j-1)q} &= A E^{j-1} F^{k-j}; & \text{(start of intermediate batch } j-1\text{)}\\
M_{k-j+(j-1)q+1} &= A E^{j-1} F^{k-j+1}; & \\
&\vdots & \\
M_{k-j+jq-1} &= A E^{j-1} F^{k-j+q-1}; & \\
M_{k-j+jq} &= E^j F^{k-j}; & \text{(start of final batch)}\\
M_{k-j+jq+1} &= E^j F^{k-j+1}; & \\
&\vdots & \\
M_{\ell-1} &= E^j F^{\ell-qj-1} &
\end{array}$$

Fig. 4.1. Polynomials constructed in the new algorithm. There is an initial batch of length $k-j$; j intermediate batches, each of length q; and a final batch of length $\ell-(q-1)j-k$. If $\ell=(q-1)j+k$ and $j>0$ then the last polynomial is $AE^{j-1}F^{k-j+q-1}$; if $\ell=(q-1)j+k$ and $j=0$ then the last polynomial is AF^{k-1}.

$k(n-w)+jw$; but the image of Q has degree below $k(n-w)+jw$, so it must be 0 as desired.

Notes on parameter selection. Assume that $t+1 \le n'$ where $n' = n(q-1)/q$. As before write $J' = n' - \sqrt{n'(n'-t-1)}$.

Suitable j, k, ℓ exist with $\ell \in O(qnt)$ for each positive integer $w < J'$. For example, the integers $j = 2w(t+1)$, $k = 2(q-1)(n-w)(t+1)$, and $\ell = 2(q-1)n(t+1)$ have $(1-(t+1)/n)(1-1/\ell) = 1-(t+1)/n-1/\ell+1/2(q-1)n^2$

and $(1-w/n-k/\ell)^2+(w/n-(q-1)j/\ell)^2/(q-1)+k/\ell^2+(q-1)j/\ell^2=1/\ell$; so w,j,k,ℓ are in the parameter space if $1-(t+1)/n+1/2(q-1)n^2<(1-w/n)^2+(w/n)^2/(q-1)$, i.e., $(q-1)n(n-t-1)+1/2<(q-1)(n-w)^2+w^2$. Both $(q-1)n(n-t-1)$ and $(q-1)(n-w)^2+w^2$ are integers, so this inequality holds if and only if $(q-1)n(n-t-1)<(q-1)(n-w)^2+w^2$, which is equivalent to $(n'-w)^2>n'(n'-t-1)$, i.e., $w<n'-\sqrt{n'(n'-t-1)}$.

These parameters have $\ell\in O(n^2)$ if $q\in O(1)$; and $\ell\le n^2(\lg n)^{O(1)}$ if $q\in(\lg n)^{O(1)}$; and $\ell\in n^{O(1)}$ if $q\in n^{O(1)}$. If q grows superpolynomially with n then this algorithm obviously cannot run in polynomial time, except in the special case $j=0$ covered in the previous section. Such a large q would also force the $\mathbf{F}_q$ Johnson bound to be extremely close to the big-field Johnson bound; if there is an integer w between the two bounds then correcting w errors in polynomial time is, as far as I know, an open problem.

My main interest is in small q and, as in the previous section, small ℓ. It seems reasonable, although not always exactly optimal, to choose k as $\lfloor(1-w/n)\ell\rfloor$ and j as $\lfloor(w/n)\ell/(q-1)\rfloor$. Then $0\le j\le k$ since $(w/n)/(q-1)\le 1-w/n$, and $\ell\ge(q-1)j+k$. These choices also guarantee that $(1-w/n-k/\ell)^2<1/\ell^2$, that $k/\ell^2\le(1-w/n)/\ell$, that $(w/n-(q-1)j/\ell)^2/(q-1)<(q-1)/\ell^2$, and that $(q-1)j/\ell^2\le(w/n)/\ell$, so w,j,k,ℓ are in the parameter space if $(1-(t+1)/n)(1-1/\ell)\le(1-w/n)^2+(w/n)^2/(q-1)-1/\ell-q/\ell^2$; i.e., $1-(t+1)/n+(t+1)/n\ell\le(1-w/n)^2+(w/n)^2/(q-1)-q/\ell^2$; i.e., $(1-J'/n)^2+(J'/n)^2/(q-1)+(t+1)/n\ell+q/\ell^2\le(1-w/n)^2+(w/n)^2/(q-1)$; i.e.,

$$J'-w\ge\frac{(t+1)/\ell+qn/\ell^2}{2-(w+J')/n'}.$$

Assume from now on that $q\in(\lg n)^{O(1)}$. Then $t\le n/m\le(n\lg q)/\lg n\in O((n\lg\lg n)/\lg n)$, so w and J' are both bounded by $O((n\lg\lg n)/\lg n)$, so $2-(w+J')/n'$ is bounded below by 1 for all sufficiently large n. If the gap $J'-w$ is at least 1 then one can push $(t+1)/\ell+qn/\ell^2$ below $J'-w$ by taking ℓ larger than both $2(t+1)$ and $\sqrt{2qn}$; this is achievable with $\ell\in O(n)$. If the gap $J'-w$ is at least $n/(\lg n)^{O(1)}$ then one can take $\ell\in(\lg n)^{O(1)}$.

5 Correcting more errors

One can trivially build a w-error-correcting algorithm from a $(w-1)$-error-correcting algorithm as follows: guess an error position (probability w/n); guess the error value (probability $1/(q-1)$); correct the error; apply the $(w-1)$-error-correcting algorithm. If the guess does not find the desired $c\in C$, try again.

This procedure takes $(q-1)n/w$ repetitions on average. With more repetitions one can confidently list *all* $c\in C$ at distance w; but I will focus on the effort required to find a *particular* $c\in C$ at distance w. Note that in the previous sections there was no reason to distinguish between these problems: the algorithms in the previous sections find all answers at almost exactly the same cost as finding the first answer.

A consequence of this reduction is that, for small q, there is no point in pushing the algorithms of the previous sections very close to their limits: instead of correcting $J' - 0.001$ errors one can much more cheaply correct $J' - 1.001$ errors and guess the remaining error.

More generally, one can build a w-error-correcting algorithm as follows: guess e distinct error positions (probability $w(w-1)\cdots(w-e+1)/n(n-1)\cdots(n-e+1)$); guess the error values (probability $1/(q-1)^e$); correct the errors; apply a $(w-e)$-error-correcting algorithm. This takes $(q-1)^e n(n-1)\cdots(n-e+1)/w(w-1)\cdots(w-e+1)$ repetitions on average.

Assume that $q \in (\lg n)^{O(1)}$, that $n/t \in (\lg n)^{O(1)}$, and that $w-e \ge \lfloor t/2 \rfloor$. The average number of repetitions is then bounded by $(2(q-1)n/t)^e \in (\lg n)^{O(e)}$; i.e., by $n^{O(1)}$ if $e \in O((\lg n)/\lg\lg n)$, and by $n^{o(1)}$ if $e \in o((\lg n)/\lg\lg n)$. In particular, this algorithm corrects $J' + o((\lg n)/\lg\lg n)$ errors using $n^{\Omega+2+o(1)}$ bit operations, and corrects $J' + O((\lg n)/\lg\lg n)$ errors using $n^{O(1)}$ bit operations.

6 Application to classical Goppa codes

The code C is called a **classical Goppa code** if there is a monic degree-t polynomial $g \in \mathbf{F}_{q^m}[x]$ such that each β_i can be expressed as $g(\alpha_i)/A'(\alpha_i)$. Here $A = \prod_i (x - \alpha_i) \in \mathbf{F}_{q^m}[x]$ as in Sections 3 and 4. In this case C is denoted $\Gamma_q(\alpha_1, \ldots, \alpha_n, g)$.

Sugiyama, Kasahara, Hirasawa, and Namekawa showed in [51] that

$$\Gamma_q(\alpha_1, \ldots, \alpha_n, \prod_i g_i^{e_i}) = \Gamma_q(\alpha_1, \ldots, \alpha_n, \prod_i g_i^{e_i + [e_i \bmod q = q-1]})$$

when the g_i's are distinct monic irreducible polynomials. Here $[e_i \bmod q = q-1]$ means 1 if $e_i \in \{q-1, 2q-1, \ldots\}$, otherwise 0. For example, $\Gamma_2(\ldots, g) = \Gamma_2(\ldots, g^2)$ if g is squarefree; this had been proven earlier by Goppa in [31] using a different technique.

Write $g = \prod_i g_i^{e_i}$ and $\overline{g} = \prod_i g_i^{e_i + [e_i \bmod q = q-1]}$. The Sugiyama–Kasahara–Hirasawa–Namekawa identity $\Gamma_q(\ldots, g) = \Gamma_q(\ldots, \overline{g})$ implies that one can correct w errors in $\Gamma_q(\ldots, g)$ by using any w-error-correcting algorithm for $\Gamma_q(\ldots, \overline{g})$. If some $e_i \bmod q = q-1$ then $\overline{g}$ has larger degree than g, making all of these error-correcting algorithms more effective for $\overline{g}$ than for g.

In particular, combining the SKHN identity with Berlekamp's algorithm corrects $\lfloor qt/2 \rfloor$ errors in "wild Goppa codes" $\Gamma_q(\ldots, g^{q-1})$ with squarefree g. Combining the SKHN identity with the Guruswami–Sudan algorithm corrects nearly $n - \sqrt{n(n-qt-1)}$ errors in the same codes in polynomial time, as discussed in [12, Section 5]. Combining the SKHN identity with the Koetter–Vardy algorithm corrects nearly $n' - \sqrt{n'(n'-qt-1)}$ errors in polynomial time, as pointed out in [4]. Combining the SKHN identity with the algorithm in this paper corrects even more errors in polynomial time.

See also [5] for a different approach that decodes more errors in some cases, particularly for $q = 3$.

References

[1] — (no editor), *39th annual symposium on foundations of computer science, FOCS '98, November 8–11, 1998, Palo Alto, California, USA*, IEEE Computer Society, 1998. See [35].

[2] — (no editor), *Proceedings of the 32nd annual ACM symposium on theory of computing*, Association for Computing Machinery, New York, 2000. See [14].

[3] Michael Alekhnovich, *Linear diophantine equations over polynomials and soft decoding of Reed-Solomon codes*, IEEE Transactions on Information Theory **51** (2005), 2257–2265. Cited from §1, §2, §2.

[4] Daniel Augot, Morgan Barbier, Alain Couvreur, *List-decoding of binary Goppa codes up to the binary Johnson bound* (2010). Cited from §1, §1, §3, §3, §3, §3, §3, §3, §6.

[5] Paulo S. L. M. Barreto, Richard Lindner, Rafael Misoczki, *Decoding square-free Goppa codes over $\mathbf{F}_p$* (2010). Cited from §6.

[6] Peter Beelen, Kristian Brander, *Key equations for list decoding of Reed–Solomon codes and how to solve them*, Journal of Symbolic Computation **45** (2010), 773–786. Cited from §1.

[7] Elwyn R. Berlekamp, *Algebraic coding theory*, McGraw-Hill, New York, 1968. Cited from §1.

[8] Daniel J. Bernstein, *Fast multiplication and its applications*, in [**19**] (2008), 325–384. Cited from §2, §2, §2, §2.

[9] Daniel J. Bernstein, *Reducing lattice bases to find small-height values of univariate polynomials*, in [**19**] (2008), 421–446. Cited from §1, §3, §3, §3.

[10] Daniel J. Bernstein, *List decoding for binary Goppa codes*, in IWCC 2011 [**21**] (2011), 62–80. Cited from §3, §3.

[11] Daniel J. Bernstein, Tanja Lange, Christiane Peters, *Attacking and defending the McEliece cryptosystem*, in PQCrypto 2008 [**17**] (2008), 31–46. Cited from §1.

[12] Daniel J. Bernstein, Tanja Lange, Christiane Peters, *Wild McEliece*, in SAC 2010 [**13**] (2011), 143–158. Cited from §6.

[13] Alex Biryukov, Guang Gong, Douglas R. Stinson (editors), *Selected areas in cryptography—17th international workshop, SAC 2010, Waterloo, Ontario, Canada, August 12–13, 2010, revised selected papers*, Lecture Notes in Computer Science, 6544, Springer, 2011. See [12].

[14] Dan Boneh, *Finding smooth integers in short intervals using CRT decoding*, in STOC 2000 [**2**] (2000), 265–272; see also newer version [**15**]. Cited from §3, §3, §3.

[15] Dan Boneh, *Finding smooth integers in short intervals using CRT decoding*, Journal of Computer and System Sciences **64** (2002), 768–784; see also older version [**14**].

[16] Raj C. Bose, Dijen K. Ray-Chaudhuri, *On a class of error correcting binary group codes*, Information and Control **3** (1960), 68–79. Cited from §1.

[17] Johannes Buchmann, Jintai Ding (editors), *Post-quantum cryptography, second international workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, proceedings*, Lecture Notes in Computer Science, 5299, Springer, 2008. See [11].

[18] Peter Bürgisser, Michael Clausen, Mohammed Amin Shokrollahi, *Algebraic complexity theory*, Springer-Verlag, Berlin, 1997. Cited from §2.

[19] Joe P. Buhler, Peter Stevenhagen (editors), *Surveys in algorithmic number theory*, Mathematical Sciences Research Institute Publications, 44, Cambridge University Press, New York, 2008. See [8], [9].

[20] Guilhem Castagnos, Antoine Joux, Fabien Laguillaumie, Phong Q. Nguyen, *Factoring pq^2 with quadratic forms: nice cryptanalyses*, in Asiacrypt 2009 [**43**] (2009), 469–486. Cited from §3.

[21] Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, Chaoping Xing (editors), *Coding and cryptology—third international workshop, IWCC 2011, Qingdao, China, May 30–June 3, 2011, proceedings*, Lecture Notes in Computer Science, 6639, Springer, 2011. See [10].

[22] Robert T. Chien, David M. Choy, *Algebraic generalization of BCH-Goppa-Helgert codes*, IEEE Transactions on Information Theory **21**, 70–79. Cited from §1.

[23] Henry Cohn, Nadia Heninger, *Ideal forms of Coppersmith's theorem and Guruswami-Sudan list decoding* (2010). Cited from §3, §3, §3.

[24] Don Coppersmith, *Finding a small root of a univariate modular equation*, in Eurocrypt 1996 [**44**] (1996), 155–165; see also newer version [**26**]. Cited from §3, §3.

[25] Don Coppersmith, *Finding a small root of a bivariate integer equation; factoring with high bits known*, in Eurocrypt 1996 [**44**] (1996), 178–189; see also newer version [**26**]. Cited from §3, §3.

[26] Don Coppersmith, *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*, Journal of Cryptology **10** (1997), 233–260; see also older version [**24**] and [**25**].

[27] Michael Darnell (editor), *Cryptography and coding: proceedings of the 6th IMA International Conference held at the Royal Agricultural College, Cirencester, December 17–19, 1997*, Lecture Notes in Computer Science, 1355, Springer-Verlag, 1997. See [40].

[28] Philippe Delsarte, *On subfield subcodes of modified Reed-Solomon codes*, IEEE Transactions on Information Theory **21** (1975), 575–576. Cited from §1.

[29] Joachim von zur Gathen, Jürgen Gerhard, *Modern computer algebra, second edition*, Cambridge University Press, Cambridge, 2003. Cited from §2, §2, §2, §2, §2, §2, §2, §2.

[30] Pascal Giorgi, Claude-Pierre Jeannerod, Gilles Villard, *On the complexity of polynomial matrix computations*, in ISSAC 2003 [**49**] (2003), 135–142. Cited from §2, §2, §2, §2.

[31] Valery D. Goppa, *A new class of linear error correcting codes*, Problemy Peredachi Informatsii **6** (1970), 24–30. Cited from §1, §6.

[32] Valery D. Goppa, *Rational representation of codes and (L, g)-codes*, Problemy Peredachi Informatsii **7** (1971), 41–49. Cited from §1.

[33] Daniel Gorenstein, Neal Zierler, *A class of error-correcting codes in p^m symbols*, Journal of the Society for Industrial and Applied Mathematics **9** (1961), 207–214. Cited from §1.

[34] Venkatesan Guruswami, *List decoding of error-correcting codes*, Ph.D. thesis, Massachusetts Institute of Technology, 2001. Cited from §1.

[35] Venkatesan Guruswami, Madhu Sudan, *Improved decoding of Reed-Solomon and algebraic-geometry codes*, in FOCS 1998 [**1**] (1998), 28–39; see also newer version [**36**]. Cited from §1, §1, §3, §3, §3, §3, §3.

[36] Venkatesan Guruswami, Madhu Sudan, *Improved decoding of Reed-Solomon and algebraic-geometry codes*, IEEE Transactions on Information Theory **45** (1999), 1757–1767; see also older version [**36**]. Cited from §3, §3.

[37] Johan Håstad, *Solving simultaneous modular equations of low degree*, SIAM Journal on Computing **17** (1988), 336–341. Cited from §3.

[38] Hermann J. Helgert, *Alternant codes*, Information and Control **26** (1974), 369–380. Cited from §1.

[39] Alexis Hocquenghem, *Codes correcteurs d'erreurs*, Chiffres **2** (1959), 147–156. Cited from §1.

[40] Nicholas Howgrave-Graham, *Finding small roots of univariate modular equations revisited*, in Cirencester 1997 [**27**] (1997), 131–142. Cited from §3, §3.

[41] Nicholas Howgrave-Graham, *Computational mathematics inspired by RSA*, Ph.D. thesis, 1998. Cited from §3, §3.

[42] Jorn Justesen, *On the complexity of decoding Reed–Solomon codes*, IEEE Transactions on Information Theory **22** (1976), 237–238. Cited from §1.

[43] Mitsuru Matsui (editor), *Advances in cryptology—ASIACRYPT 2009, 15th international conference on the theory and application of cryptology and information security, Tokyo, Japan, December 6–10, 2009, proceedings*, Lecture Notes in Computer Science, 5912, Springer, 2009. See [20].

[44] Ueli M. Maurer (editor), *Advances in cryptology—EUROCRYPT '96: proceedings of the fifteenth international conference on the theory and application of cryptographic techniques held in Saragossa, May 12–16, 1996*, Lecture Notes in Computer Science, 1070, Springer-Verlag, Berlin, 1996. See [24], [25].

[45] Teo Mora (editor), *Applied algebra, algebraic algorithms and error-correcting codes: proceedings of the sixth international conference (AAECC-6) held in Rome, July 4–8, 1988*, Lecture Notes in Computer Science, 357, Springer-Verlag, Berlin, 1989. See [53].

[46] W. Wesley Peterson, *Encoding and error-correction procedures for the Bose-Chaudhuri codes*, Transactions of the Institute of Radio Engineers **6** (1960), 459–470. Cited from §1.

[47] Irving S. Reed, Gustave Solomon, *Polynomial codes over certain finite fields*, Journal of the Society for Industrial and Applied Mathematics **8** (1960), 300–304. Cited from §1.

[48] Dilip V. Sarwate, *On the complexity of decoding Goppa codes*, IEEE Transactions on Information Theory **23** (1977), 515–516. Cited from §1.

[49] J. Rafael Sendra (editor), *Symbolic and algebraic computation, international symposium ISSAC 2003, Drexel University, Philadelphia, Pennsylvania, USA, August 3–6, 2003, proceedings*, Association for Computing Machinery, 2003. See [30].

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

[51] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, Toshihiko Namekawa, *Further results on Goppa codes and their applications to constructing efficient binary codes*, IEEE Transactions on Information Theory **22** (1976), 518–526. Cited from §6.

[52] Peter Trifonov, *Efficient interpolation in the Guruswami–Sudan algorithm*, IEEE Transactions on Information Theory **56** (2010), 4341–4349. Cited from §1.

[53] Brigitte Vallée, Marc Girault, Philippe Toffin, *How to guess ℓth roots modulo n by reducing lattice bases*, in AAECC 1989 [**45**] (1989), 427–442. Cited from §3.

[54] Yingquan Wu, *New list decoding algorithms for Reed–Solomon and BCH codes*, IEEE Transactions On Information Theory **54** (2008). Cited from §3, §3.

[55] Hans Zassenhaus, *On Hensel factorization. I*, Journal of Number Theory **1** (1969), 291–311. Cited from §2.