List decoding for binary Goppa codes

Daniel J. Bernstein

Department of Mathematics, Statistics, and Computer Science (M/C 249)
University of Illinois at Chicago, Chicago, IL 60607–7045, USA

Abstract

This paper presents a list-decoding algorithm for classical irreducible binary Goppa codes. The algorithm corrects, in polynomial time, approximately $n - \sqrt{n(n-2t-2)}$ errors in a length-n classical irreducible degree-t binary Goppa code. Compared to the best previous polynomial-time list-decoding algorithms for the same codes, the new algorithm corrects approximately $t^2/2n$ extra errors.

1. Introduction

Patterson (1975) introduced a polynomial-time decoding algorithm that corrects t errors in a classical irreducible degree-t binary Goppa code.

This paper introduces a considerably more complicated, but still polynomial-time, list-decoding algorithm for classical irreducible binary Goppa codes. The advantage of the new algorithm is that it corrects approximately

$$n - \sqrt{n(n-2t-2)} \approx t + 1 + \frac{(t+1)^2}{2(n-t-1)}$$

errors in a length-n degree-t code. Typically t is chosen in the ballpark of $n/(2 \lg n)$; in that case the new algorithm corrects approximately $t + 1 + n/(8(\lg n)^2)$ errors.

Comparison to previous list-decoding algorithms. A different strategy for decoding a degree-t classical binary Goppa code is to view it as an "alternant code," i.e., a subfield code of a degree-t generalized Reed–Solomon code over a larger field. The generalized Reed–Solomon code can be decoded by Berlekamp's algorithm, or by the famous algorithm introduced by Guruswami and Sudan (1999).

* Permanent ID of this document: `210ecf064c479a278ab2c98c379f72e0`. Date of this document: 2008.11.07. This work was carried out while the author was visiting Technische Universiteit Eindhoven. This work has been supported by the National Science Foundation under grant ITR–0716498.

Email address: `djb@cr.yp.to` (Daniel J. Bernstein).

Unfortunately, Berlekamp's algorithm is much less effective than Patterson's algorithm: it corrects only $t/2$ errors. The Guruswami–Sudan algorithm corrects approximately $n - \sqrt{n(n-t)} > t/2$ errors but still does not reach t errors. Guruswami and Sudan (1999, Section 3.1) point out this limitation of their algorithm ("performance can only be compared with the designed distance, rather than the actual distance"); they do not mention how serious this limitation is in the binary-Goppa case, where moving to a larger field chops the distance *in half*.

As far as I know, the new algorithm is the first list-decoding algorithm for a useful class of binary codes, and the first improvement in more than thirty years on the decoding power of Patterson's algorithm.

Extra errors by brute force. Another standard way to correct extra errors is to guess the positions of the extra errors. For example, one can guess e error positions, flip those e bits, and then apply Patterson's algorithm to correct t additional errors, overall correcting $t+e$ errors. The guess is correct with probability $\binom{n-e}{t}/\binom{n}{t+e}$, so after $\binom{n}{t+e}/\binom{n-e}{t}$ guesses one has a good chance of finding any particular codeword at distance $t+e$. One can bring the chance exponentially close to 1 by moderately increasing the number of guesses.

Although this algorithm involves many repetitions of Patterson's algorithm, it remains a polynomial-time algorithm if e is chosen so that $\binom{n}{t+e}/\binom{n-e}{t}$ grows polynomially. In particular, in the typical case $t \in \Theta(n/\lg n)$, one can decode $\Theta((\lg n)/(\lg \lg n))$ extra errors in polynomial time.

Similarly, one can guess e error positions, flip those e bits, and then apply this paper's new list-decoding algorithm. Compared to Patterson's original algorithm, this method decodes

- the same t errors, plus
- approximately $n - t - \sqrt{n(n-2t-2)}$ extra errors from the new algorithm, plus
- e additional errors from guessing;

and the method remains polynomial-time if e is small. In particular, for $t \approx n/(2 \lg n)$, this paper's new list-decoding algorithm adds approximately $n/(8 \lg n)^2$ extra errors, and guessing adds $\Theta((\lg n)/(\lg \lg n))$ extra errors, still in polynomial time.

This paper does not consider notions of efficiency more precise than "polynomial time"; in particular, it does not discuss the number of errors correctable in time $n^{1+o(1)}$, the number of errors correctable in time $n^{2+o(1)}$, etc. The algorithms in this paper were chosen to be as simple as possible, subject to the constraint of running in polynomial time for the desired number of errors.

An application to code-based cryptography. McEliece (1978) proposed a public-key encryption system using exactly the same codes considered in this paper. The public key is a generator matrix (or, as proposed by Niederreiter (1986), a parity-check matrix) of a code equivalent to a classical irreducible degree-t binary Goppa code chosen secretly by the receiver. The sender encodes a message and adds t errors; the receiver decodes the errors.

Adding more errors makes McEliece's system harder to break by all known attacks, but also requires the receiver to decode the additional errors, posing exactly the problem tackled in this paper: exactly how many errors can be efficiently decoded in a classical irreducible binary Goppa code? See (Bernstein, Lange, and Peters 2008) for further discussion and security analysis.

Several code-based cryptosystems have been proposed using codes over fields larger than $\mathbf{F}_2$—for example, generalized Reed–Solomon codes. Unfortunately, these variants seem considerably less secure than McEliece's original system. See, e.g., (Sidelnikov and Shestakov 1992). One could also switch to a different class of codes over $\mathbf{F}_2$, but I am not aware of codes over $\mathbf{F}_2$ that allow efficient decoding of more errors!

2. Review of classical irreducible binary Goppa codes

This section reviews three equivalent definitions of the classical irreducible binary Goppa code $\Gamma(a_1, \ldots, a_n, g)$: the "polynomial" definition, the "classical" definition, and the "evaluation" definition.

The notations $m, t, n, a_1, \ldots, a_n, g, h, \Gamma$ in this section will be reused in Sections 3, 4, and 7.

Parameters for the code. Fix an integer $m \geq 3$. Typically $m \in \{10, 11, 12\}$ in the cryptographic applications mentioned in Section 1.

Fix an integer t with $2 \leq t \leq (2^m - 1)/m$. The Goppa code will be a "degree-t code" designed to correct t errors. Extremely small and extremely large values of t are not useful, but intermediate values of t produce interesting codes; for $m = 11$ one could reasonably take (e.g.) $t = 32$, or $t = 70$, or $t = 100$.

Fix an integer n with $mt + 1 \leq n \leq 2^m$. It is common to restrict attention to the extreme case $n = 2^m$; e.g., $n = 2048$ if $m = 11$. However, a wider range of n allows a better security/efficiency tradeoff for code-based cryptography, as illustrated in (Bernstein, Lange, and Peters 2008, Section 7).

Fix a sequence $a_1, \ldots, a_n$ of distinct elements of the finite field $\mathbf{F}_{2^m}$. Typically $n = 2^m$ and $a_1, \ldots, a_n$ are chosen as all the elements of $\mathbf{F}_{2^m}$ in lexicographic order, given a standard basis for $\mathbf{F}_{2^m}$ over $\mathbf{F}_2$. For $n < 2^m$ there is more flexibility.

Finally, fix a monic degree-t irreducible polynomial $g \in \mathbf{F}_{2^m}[x]$. There are no standard choices here; in the classic study of minimum distance it is an open problem to find the best g, and in code-based cryptography it is important for g to be a randomly chosen secret.

The "polynomial" view of the code. Define $h = \prod_i (x - a_i) \in \mathbf{F}_{2^m}[x]$. In the extreme case $n = 2^m$, this polynomial h is simply $x^n - x$, with derivative $h' = nx^{n-1} - 1 = 1$, slightly simplifying some of the formulas below.

Define

$$\Gamma = \Gamma(a_1, \ldots, a_n, g) = \left\{ c \in \mathbf{F}_2^n : \sum_i c_i \frac{h}{x - a_i} \bmod g = 0 \right\}.$$

This set Γ is the kernel of the "syndrome" map $\mathbf{F}_2^n \to \mathbf{F}_{2^m}^t$ that maps c to the coefficients of $1, x, \ldots, x^{t-1}$ in $\sum_i c_i h/(x - a_i) \bmod g$; consequently Γ is an $\mathbf{F}_2$-module of dimension at least $n - mt$, i.e., an $[n, \geq n - mt]$ code over $\mathbf{F}_2$.

In other words: The polynomials $h/(x - a_1) \bmod g$, $h/(x - a_2) \bmod g$, $\ldots$, $h/(x - a_n) \bmod g$, viewed as vectors over $\mathbf{F}_2$, form a parity-check matrix for the code Γ.

The "classical" view of the code. By construction g has degree $t \geq 2$, and has none of $a_1, \ldots, a_n$ as roots. Therefore h is coprime to g.

Consequently the polynomial $\sum_i c_i h/(x - a_i)$ in $\mathbf{F}_{2^m}[x]$ is a multiple of g if and only if $\sum_i c_i/(x - a_i)$ equals 0 in the field $\mathbf{F}_{2^m}[x]/g$. The classical Goppa code associated to

$a_1, \ldots, a_n, g$ is most commonly defined as the set of $c \in \mathbf{F}_2^n$ such that $\sum_i c_i/(x-a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$; this is the same code as Γ.

Another consequence of the coprimality of h and g is that the minimum distance of C is at least $2t+1$; i.e., C is an $[n, \geq n-mt, \geq 2t+1]$ code over $\mathbf{F}_2$. Proof: If $c \in \Gamma - \{0\}$ then g divides the polynomial $\sum_i c_i h/(x-a_i) = \sum_{i:c_i=1} h/(x-a_i) = h\epsilon'/\epsilon$ where $\epsilon = \prod_{i:c_i=1}(x-a_i)$. Thus g divides ϵ'. Write ϵ as $\alpha^2 + x\beta^2$, and observe that $\beta \neq 0$, since by construction ϵ is not a square. Now $\epsilon' = \beta^2$, so g divides β^2; but g is irreducible, so g divides β, so β has degree at least t, so ϵ has degree at least $2t+1$.

The "evaluation" view of the code. Define

$$C = \left\{ c \in \mathbf{F}_{2^m}^n : \sum_i c_i \frac{h}{x-a_i} \bmod g = 0 \right\}.$$

This set C is an $[n, n-t]$ code over $\mathbf{F}_{2^m}$. The classical binary Goppa code Γ is a subfield code of C.

If f is a polynomial in $\mathbf{F}_{2^m}[x]$ with $\deg f < n-t$ then the vector

$$(f(a_1)g(a_1)/h'(a_1), f(a_2)g(a_2)/h'(a_2), ..., f(a_n)g(a_n)/h'(a_n))$$

is in C. Indeed, $\sum_i (f(a_i)g(a_i)/h'(a_i))h/(x-a_i) = fg$ by Lagrange interpolation, and $fg \bmod g = 0$. Conversely, every element of C can be written as a vector of this form: if $\sum_i c_i h/(x-a_i) \in \mathbf{F}_{2^m}[x]$ is a multiple of g, say fg, then $f(a_i)g(a_i) = c_i h'(a_i)$ so

$$c = (f(a_1)g(a_1)/h'(a_1), f(a_2)g(a_2)/h'(a_2), ..., f(a_n)g(a_n)/h'(a_n)).$$

Therefore C is a geometric Goppa code, specifically a genus-0 geometric Goppa code, specifically a geometric Goppa code over the projective line.

3. Review of Patterson's algorithm

This section reviews Patterson's algorithm for correcting t (or fewer) errors in the classical irreducible binary Goppa code $\Gamma = \Gamma(a_1, \ldots, a_n, g)$ defined in Section 2.

The algorithm. The input to the algorithm is a vector $w \in \mathbf{F}_2^n$. The output is a list of all codewords $c \in \Gamma$ such that the Hamming distance $|c-w| = \#\{i : c_i \neq w_i\}$ is at most t. There is at most one such codeword—recall that the minimum distance of Γ is at least $2t+1$.

Define the **norm** $|\varphi|$ of a polynomial $\varphi \in \mathbf{F}_{2^m}[x]$ as $2^{\deg \varphi}$ if $\varphi \neq 0$ and 0 if $\varphi = 0$. Extend the norm multiplicatively to rational functions $\varphi \in \mathbf{F}_{2^m}(x)$: the norm $|\varphi/\psi|$ is $|\varphi|/|\psi|$. For example, $|x^3/(x^5+x+1)| = |x^3|/|x^5+x+1| = 2^3/2^5 = 2^{-2}$.

Compute the square root of $(1/\sum_i w_i/(x-a_i)) - x$ in the field $\mathbf{F}_{2^m}[x]/g$. This computation fails if $\sum_i w_i/(x-a_i)$ is zero in the field; if so, output w and stop.

Lift the square root to a polynomial $s \in \mathbf{F}_{2^m}[x]$ of degree $< t$. Apply lattice-basis reduction to the lattice $L \subseteq \mathbf{F}_{2^m}[x]^2$ generated by the vectors $(s,1)$ and $(g,0)$, obtaining a minimum-length nonzero vector (α_0, β_0). Here the length $|(\alpha,\beta)|$ of a vector $(\alpha,\beta) \in \mathbf{F}_{2^m}[x]^2$ is, by definition, the norm of the polynomial $\alpha^2 + x\beta^2$.

Compute $\epsilon_0 = \alpha_0^2 + x\beta_0^2$. Use a polynomial-factorization algorithm to see whether the monic part of ϵ_0 (i.e., ϵ_0 divided by its leading coefficient) splits into distinct linear factors

of the form $x - a_i$. If it does, output the unique vector $c \in \mathbf{F}_2^n$ such that $\{i : w_i \neq c_i\} = \{i : \epsilon_0(a_i) = 0\}$.

Why the algorithm works. If the algorithm outputs w in the first step then $\sum_i w_i/(x - a_i) = 0$ in the field $\mathbf{F}_{2^m}[x]/g$ so $w \in \Gamma$. Conversely, if $w \in \Gamma$ then the algorithm correctly outputs w in the first step. Note that in this case there are no other codewords at distance $\leq 2t$.

Assume from now on that $w \notin \Gamma$. Then $\sum_i w_i/(x - a_i) \neq 0$ in $\mathbf{F}_{2^m}[x]/g$.

The specified basis $(s, 1), (g, 0)$ of L has orthogonalization $(0, 1), (g, 0)$, with lengths $|(0, 1)| = |x| = 2^1$ and $(g, 0) = |g^2| = 2^{2t}$, product 2^{2t+1}. Consequently $|(\alpha_0, \beta_0)| \leq 2^{(2t+1)/2} = 2^{t+1/2}$; i.e., $\deg \epsilon_0 \leq t + 1/2$; i.e., $\deg \epsilon_0 \leq t$.

Furthermore, the lattice L is exactly the set of vectors $(\alpha, \beta) \in \mathbf{F}_{2^m}[x]^2$ such that $\alpha - s\beta$ is a multiple of g. Consequently any $(\alpha, \beta) \in L$ satisfies $\alpha^2/\beta^2 = s^2 = (1/\sum_i w_i/(x - a_i)) - x$ in the field $\mathbf{F}_{2^m}[x]/g$, if β is not a multiple of g. The polynomial $\epsilon = \alpha^2 + x\beta^2 \in \mathbf{F}_{2^m}[x]$ satisfies $\epsilon' = \beta^2$, so $\epsilon/\epsilon' = \alpha^2/\beta^2 + x = 1/\sum_i w_i/(x - a_i)$ in the field $\mathbf{F}_{2^m}[x]/g$.

If the algorithm outputs a vector c then the monic part of ϵ_0 splits into linear factors, so ϵ_0 is not a square, so $\alpha_0^2 + x\beta_0^2$ is not a square, so $\beta_0 \neq 0$; but $\deg \beta_0 \leq (t-1)/2 < t = \deg g$, so β_0 is not a multiple of g, so $\epsilon_0/\epsilon_0' = 1/\sum_i w_i/(x - a_i)$ in the field $\mathbf{F}_{2^m}[x]/g$. Thus $\sum_i w_i/(x - a_i) = \epsilon_0'/\epsilon_0 = \sum_{i:\epsilon_0(a_i)=0} 1/(x - a_i) = \sum_{i:w_i \neq c_i} 1/(x - a_i) = \sum_i (w_i - c_i)/(x - a_i) = \sum_i w_i/(x - a_i) - \sum_i c_i/(x - a_i)$ in the field $\mathbf{F}_{2^m}[x]/g$. Subtract to see that $\sum_i c_i/(x - a_i) = 0$ in the field $\mathbf{F}_{2^m}[x]/g$, i.e., that $c \in \Gamma$. The Hamming distance $|w - c|$ is exactly $\#\{i : \epsilon_0(a_i) = 0\} = \deg \epsilon_0 \leq t$. Summary: The output of the algorithm is a codeword at distance $\leq t$ from w.

Conversely, assume that $c \in \Gamma$ has $|w - c| \leq t$. Define $\epsilon = \prod_{i:w_i \neq c_i} (x - a_i) \in \mathbf{F}_{2^m}[x]$, and write ϵ in the form $\alpha^2 + x\beta^2$. Then $\sum_i c_i/(x - a_i) = 0$ in $\mathbf{F}_{2^m}[x]/g$, so $\sum_i w_i/(x - a_i) = \sum_i (w_i - c_i)/(x - a_i) = \sum_{i:w_i \neq c_i} 1/(x - a_i) = \epsilon'/\epsilon$ in $\mathbf{F}_{2^m}[x]/g$, so $s^2 = \epsilon/\epsilon' - x = \alpha^2/\beta^2$ in $\mathbf{F}_{2^m}[x]/g$. Squaring in the field $\mathbf{F}_{2^m}[x]/g$ is injective, so $s = \alpha/\beta$ in $\mathbf{F}_{2^m}[x]/g$, so $\alpha - s\beta$ is a multiple of g in $\mathbf{F}_{2^m}[x]$; i.e., $(\alpha, \beta) \in L$. Furthermore $\deg \epsilon \leq t$ so $|(\alpha, \beta)| \leq 2^t$ so $|(\alpha, \beta)||(\alpha_0, \beta_0)| \leq 2^{2t}$. Every basis of L has product of lengths at least $|(0, 1)||(g, 0)| \geq 2^{2t+1}$, so $(\alpha, \beta), (\alpha_0, \beta_0)$ are not a basis; i.e., (α, β) is parallel to (α_0, β_0); but (α_0, β_0) has minimum length in L, so (α, β) is a multiple of (α_0, β_0), say $q(\alpha_0, \beta_0)$ where $q \in \mathbf{F}_{2^m}[x]$. Now $\epsilon = \alpha^2 + x\beta^2 = q^2(\alpha_0^2 + x\beta_0^2) = q^2\epsilon_0$. By construction ϵ is squarefree so ϵ/ϵ_0 is a constant. Hence the monic part of ϵ_0 splits into exactly the distinct linear factors $x - a_i$ that divide ϵ, and the algorithm finds exactly the codeword c.

Numerical example. Define $m = 8$, $n = 2^m = 256$, and $t = 22$. Construct $\mathbf{F}_{2^m}$ as $\mathbf{F}_2[\zeta]/(\zeta^8 + \zeta^4 + \zeta^3 + \zeta^2 + 1)$. Define $a_1 = \zeta$, $a_2 = \zeta^2$, and so on through $a_{255} = \zeta^{255} = 1$; define $a_{256} = 0$. Choose $g = x^{22} + x^{17} + x^{15} + x^{12} + x^5 + \zeta^{78} \in \mathbf{F}_{2^m}[x]$; one can easily check that g is irreducible.

Now the Goppa code Γ is a $[256, \geq 80, \geq 45]$ code over $\mathbf{F}_2$. I generated a random

element of Γ and added 22 random errors to it, obtaining the word

$$\begin{aligned} w = (&0,1,1,0,0,1,0,0,1,0,1,1,1,0,1,0,0,1,1,0,1,1,0,1,0,1,1,0,0,1,1,1,\\ &1,1,1,0,1,0,0,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,0,0,1,0,1,0,0,0,\\ &0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,0,1,1,\\ &0,1,1,1,0,0,0,1,1,0,0,1,1,1,0,0,1,0,1,1,0,0,1,1,1,1,0,1,1,0,0,0,\\ &0,0,0,0,1,1,1,0,0,1,1,1,1,0,1,0,0,0,0,0,0,1,0,0,1,0,1,1,1,1,1,1,0,\\ &0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,1,0,0,0,1,1,1,\\ &0,1,1,1,1,1,1,1,0,0,0,1,0,0,1,1,1,1,1,1,0,1,1,0,0,0,0,0,1,0,0,1,1,1,\\ &1,0,1,1,0,0,0,1,0,0,0,1,1,0,0,0,0,0,1,0,0,0,1,1,1,0,1,1,1,1,0,0,0) \end{aligned}$$

in $\mathbf{F}_2^n$. Here is what Patterson's algorithm does with this word w.

The sum $\sum_i w_i/(x-a_i)$ in the field $\mathbf{F}_{2^m}[x]/g$ is $1/(x-a_2)+1/(x-a_3)+1/(x-a_6)+\cdots = 1/(x-\zeta^2)+1/(x-\zeta^3)+1/(x-\zeta^6)+\cdots = \zeta^{64}+\zeta^{110}x+\zeta^{204}x^2+\zeta^{53}x^3+\zeta^{91}x^4+\zeta^{200}x^5+\zeta^{147}x^6+\zeta^{67}x^7+\zeta^{196}x^8+\zeta^{253}x^9+\zeta^{235}x^{10}+\zeta^{161}x^{11}+\zeta^{92}x^{12}+\zeta^{146}x^{13}+\zeta^{125}x^{14}+\zeta^{141}x^{15}+\zeta^9x^{16}+\zeta^{34}x^{17}+\zeta^{15}x^{18}+\zeta^{139}x^{19}+\zeta^{229}x^{20}+\zeta^{68}x^{21}$. Invert, subtract x, and compute a square root, namely $\zeta^{200}+\zeta^{46}x+\zeta^{51}x^2+\zeta^{91}x^3+\zeta^{232}x^4+\zeta^{12}x^5+\zeta^{179}x^6+\zeta^3x^7+\zeta^{146}x^8+\zeta^{93}x^9+\zeta^{130}x^{10}+\zeta^{92}x^{11}+\zeta^{28}x^{12}+\zeta^{219}x^{13}+\zeta^{96}x^{14}+\zeta^{114}x^{15}+\zeta^{131}x^{16}+\zeta^{61}x^{17}+\zeta^{251}x^{18}+\zeta^{76}x^{19}+\zeta^{237}x^{20}+\zeta^{40}x^{21}$. Define s as this polynomial in $\mathbf{F}_{2^m}[x]$.

The vector $(g,0)$ has degree $(22,0)$ and therefore length 2^{44}. The vector $(s,1)$ has degree $(21,0)$ and therefore length 2^{42}. The quotient $\lfloor g/s \rfloor$ is $\zeta^{-40}x-\zeta^{237-80} = \zeta^{215}x-\zeta^{157}$; the difference $(g,0)-\lfloor g/s \rfloor(s,1)$ is the vector $(g \bmod s, \zeta^{215}x-\zeta^{157})$, which has degree $(20,1)$ and therefore length 2^{40}. Continued reduction eventually produces the vector (α_0,β_0) where $\alpha_0 = \zeta^{181}+\zeta^{216}x+\zeta^{219}x^2+\zeta^{188}x^3+\zeta^{69}x^4+\zeta^{126}x^5+\zeta^{145}x^6+\zeta^{233}x^7+\zeta^{243}x^8+\zeta^{31}x^9+\zeta^{182}x^{10}+x^{11}$ and $\beta_0 = \zeta^{105}+\zeta^{50}x+\zeta^5x^2+\zeta^{116}x^3+\zeta^{150}x^4+\zeta^{123}x^5+\zeta^7x^6+\zeta^{224}x^7+\zeta^{220}x^8+\zeta^{84}x^9+\zeta^{150}x^{10}$; this vector has degree $(11,10)$ and therefore length $2^{22} \le 2^t$.

The polynomial $\epsilon_0 = \alpha_0^2 + x\beta_0^2$ splits into 22 linear factors, namely $x-\zeta^7$, $x-\zeta^{25}$, $x-\zeta^{51}$, $x-\zeta^{60}$, $x-\zeta^{68}$, $x-\zeta^{85}$, $x-\zeta^{126}$, $x-\zeta^{135}$, $x-\zeta^{136}$, $x-\zeta^{138}$, $x-\zeta^{155}$, $x-\zeta^{167}$, $x-\zeta^{168}$, $x-\zeta^{172}$, $x-\zeta^{173}$, $x-\zeta^{189}$, $x-\zeta^{191}$, $x-\zeta^{209}$, $x-\zeta^{212}$, $x-\zeta^{214}$, $x-\zeta^{234}$, $x-\zeta^{252}$. Consequently w has distance 22 from the codeword $c \in \Gamma$ obtained by correcting positions 7, 25, 51, etc.

4. Extracting more information from Patterson's algorithm

If Patterson's algorithm is given a word w at distance more than t from the closest codeword—in other words, if the error polynomial ϵ has degree larger than t—then the algorithm's output is empty. However, a closer look at the same calculations reveals more information about ϵ. This section presents an easy extension of Patterson's algorithm, identifying two polynomials ϵ_0, ϵ_1 such that ϵ is a small linear combination of ϵ_0, ϵ_1.

The algorithm. The input, as before, is a vector $w \in \mathbf{F}_2^n$. Assume that $w \notin \Gamma$.

Compute the square root of $(1/\sum_i w_i/(x-a_i))-x$ in the field $\mathbf{F}_{2^m}[x]/g$, and lift it to a polynomial $s \in \mathbf{F}_{2^m}[x]$ of degree below t.

Apply lattice-basis reduction to the lattice $L \subseteq \mathbf{F}_{2^m}[x]^2$ generated by the vectors $(s,1)$ and $(g,0)$, obtaining a minimum-length nonzero vector (α_0,β_0) *and* a minimum-length

independent vector (α_1, β_1). Here the length $|(\alpha, \beta)|$ of a vector $(\alpha, \beta) \in \mathbf{F}_{2^m}[x]^2$ is, as before, the norm of the polynomial $\alpha^2 + x\beta^2$.

Compute $\epsilon_0 = \alpha_0^2 + x\beta_0^2$ and $\epsilon_1 = \alpha_1^2 + x\beta_1^2$. Output (ϵ_0, ϵ_1).

What the algorithm accomplishes. Reduction guarantees that $|(\alpha_0, \beta_0)| \leq 2^{(2t+1)/2}$ and that $|(\alpha_0, \beta_0)||(\alpha_1, \beta_1)| = 2^{2t+1}$. Thus $\deg \epsilon_0 \leq t$, as in Section 3, and $\deg \epsilon_0 + \deg \epsilon_1 = 2t + 1$.

Fix $c \in \Gamma$. Define $\epsilon = \prod_{i:w_i \neq c_i} (x - a_i) \in \mathbf{F}_{2^m}[x]$, and write ϵ in the form $\alpha^2 + x\beta^2$. Then $(\alpha, \beta) \in L$, exactly as in Section 3, so (α, β) can be written as $q_0(\alpha_0, \beta_0) + q_1(\alpha_1, \beta_1)$ for some polynomials q_0, q_1. Consequently $\epsilon = q_0^2\epsilon_0 + q_1^2\epsilon_1$.

The explicit formulas $q_0 = (\alpha\beta_1 - \beta\alpha_1)/g$ and $q_1 = (\alpha\beta_0 - \beta\alpha_0)/g$ show that q_0 and q_1 are very small if ϵ is small. Specifically, fix an integer $u \geq 0$, and assume that $\deg \epsilon \leq t + u$; also write $t_0 = \deg \epsilon_0$, and note that $\deg \epsilon_1 = 2t + 1 - t_0$. Then $\deg \alpha_0 \leq \lfloor t_0/2 \rfloor$, $\deg \beta_0 \leq \lfloor (t_0 - 1)/2 \rfloor$, $\deg \alpha_1 \leq \lfloor (2t + 1 - t_0)/2 \rfloor$, $\deg \beta_1 \leq \lfloor (2t - t_0)/2 \rfloor$, $\deg \alpha \leq \lfloor (t + u)/2 \rfloor$, and $\deg \beta \leq \lfloor (t + u - 1)/2 \rfloor$, so $\deg q_0 \leq \lfloor (t + u + 2t - t_0)/2 \rfloor - t = \lfloor (t + u - t_0)/2 \rfloor$ and $\deg q_1 \leq \lfloor (t + u + t_0 - 1)/2 \rfloor - t = \lfloor (t_0 + u - t - 1)/2 \rfloor$.

Using the results of the algorithm. In the simplest case $u = 0$ (i.e., $\deg \epsilon \leq t$), the degree of q_1 is at most $\lfloor (t_0 - t - 1)/2 \rfloor \leq \lfloor -1/2 \rfloor < 0$, so $q_1 = 0$, so $\epsilon = q_0^2\epsilon_0$. Evidently constant multiples of ϵ_0 are the only possible squarefree choices for ϵ, and one can simply check whether the monic part of ϵ_0 splits into linear factors. This is exactly what Patterson's algorithm does.

However, for larger u, both q_0 and q_1 can be nonzero, and it is not so easy to see which choices for ϵ are possible. There are $\approx 2^{mu}$ coprime polynomial pairs (q_0, q_1) matching the degree bounds; enumerating all of those pairs is practical for a tiny fixed u, such as $u = 1$, but becomes intolerably slow as u increases.

The main point of this paper is an asymptotically much faster algorithm to pin down the possibilities for ϵ. See Section 7.

Refinement: $\gcd\{\epsilon_1, h\} = 1$. There are many choices of ϵ_1: one can adjust (α_1, β_1), without changing its length, by adding small multiples of (α_0, β_0) to it. In particular, for any $r \in \mathbf{F}_{2^m}$, one can replace (α_1, β_1) by $(\alpha_1, \beta_1) + \sqrt{r}(\alpha_0, \beta_0)$, replacing ϵ_1 by $\epsilon_1 + r\epsilon_0$.

It will be convenient later to choose ϵ_1 coprime to h. In practice it seems that, by trying several $r \in \mathbf{F}_{2^m}$, one easily finds r such that $\epsilon_1 + r\epsilon_0$ is coprime to h; consequently, replacing ϵ_1 with $\epsilon_1 + r\epsilon_0$, one obtains ϵ_1 coprime to h.

Can it be *proven* that this is always possible? Here are some remarks on this topic. I am indebted to Tanja Lange for related discussions, and for helpful comments on other parts of this paper.

If $\epsilon_1 + r_1\epsilon_0$ and $\epsilon_1 + r_2\epsilon_0$, with $r_1 \neq r_2$, have a common root s, then s is also a root of $((\epsilon_1 + r_1\epsilon_0) - (\epsilon_1 + r_2\epsilon_0))/(r_1 - r_2) = \epsilon_0$ and $(r_2(\epsilon_1 + r_1\epsilon_0) - r_1(\epsilon_1 + r_2\epsilon_0))/(r_2 - r_1) = \epsilon_1$, so s is a root of $(\epsilon_0\epsilon_1)' = g^2$, contradicting the irreducibility of g. Consequently each $s \in \mathbf{F}_{2^m}[x]$ is a root of $\epsilon_1 + r\epsilon_0$ for at most one $r \in \mathbf{F}_{2^m}[x]$.

Suppose that, for each $r \in \mathbf{F}_{2^m}[x]$, there is a root $s \in \mathbf{F}_{2^m}[x]$ of $\epsilon_1 + r\epsilon_0$. Counting then shows that each $\epsilon_1 + r\epsilon_0$ has exactly one root s, and that each s is a root of exactly one $\epsilon_1 + r\epsilon_0$. In particular, if $n < 2^m$, then there exists an $s \in \mathbf{F}_{2^m}[x]$ that is not a root of h, and the corresponding $\epsilon_1 + r\epsilon_0$ is coprime to h as desired. The only remaining case is $n = 2^m$.

Fix s, and find the unique r such that s is a root of $\epsilon_1 + r\epsilon_0$. Then $\epsilon_1(s) = r\epsilon_0(s)$. Furthermore $\epsilon_0(s) \neq 0$: otherwise $\epsilon_1(s) = 0$, contradicting the irreducibility of g as above.

Consequently $\epsilon_1(s)/\epsilon_0(s) = r$. Therefore the rational function ϵ_1/ϵ_0, applied to $\mathbf{F}_{2^m}$, is a "permutation function": it takes each value in $\mathbf{F}_{2^m}$ exactly once.

Note that a uniform random function $\mathbf{F}_{2^m} \to \mathbf{F}_{2^m}$ has probability only about $\exp(-2^m)$ of being a permutation function: for example, probability about 2^{-369} for $m = 8$. One does not expect to bump into a permutation function by chance! But this heuristic is not a proof. Some simple rational functions—including all linear functions, squares of linear functions, etc.—are permutation functions on $\mathbf{F}_{2^m}$. Is there any reason that ϵ_1/ϵ_0 *cannot* be a permutation function?

Define $\varphi = (\epsilon_0(x)\epsilon_1(y) - \epsilon_1(x)\epsilon_0(y))/(x-y) \in \mathbf{F}_{2^m}[x,y]$. If $s_1 \neq s_2$ then $\epsilon_1(s_1)/\epsilon_0(s_1) \neq \epsilon_1(s_2)/\epsilon_0(s_2)$ so $\varphi(s_1,s_2) \neq 0$. Furthermore $\varphi(x,x) = \epsilon_0\epsilon_1' - \epsilon_1\epsilon_0' = (\epsilon_0\epsilon_1)' = g^2$; therefore $\varphi(s,s) \neq 0$ for each $s \in \mathbf{F}_{2^m}[x,y]$. Thus there are no roots of φ with coordinates in $\mathbf{F}_{2^m}$. In other words, the curve φ has no points over $\mathbf{F}_{2^m}$.

The Hasse–Weil bounds imply, however, that a nonconstant curve of small degree must have points, producing a contradiction if t is small enough. Perhaps one can handle a larger range of t with refined bounds that take account of the special shape of φ; for relevant genus information see, e.g., (Avanzi 2001, Theorem 1.3.5).

To summarize: There might exist pairs (ϵ_0, ϵ_1) where ϵ_1 cannot be adjusted to be coprime to h. However, one expects that such pairs do not occur by chance. Furthermore, no such pairs exist if $n < 2^m$, and no such pairs exist if t is small.

Numerical example. As in Section 3, define $m = 8$, $n = 2^m = 256$, and $t = 22$; construct $\mathbf{F}_{2^m}$ as $\mathbf{F}_2[\zeta]/(\zeta^8+\zeta^4+\zeta^3+\zeta^2+1)$; define $a_1 = \zeta$, $a_2 = \zeta^2$, and so on through $a_{255} = \zeta^{255} = 1$; define $a_{256} = 0$; and choose $g = x^{22}+x^{17}+x^{15}+x^{12}+x^5+\zeta^{78} \in \mathbf{F}_{2^m}[x]$.

I generated a random element of the Goppa code Γ and added 24 random errors to it, obtaining the word

$$\begin{aligned} w = (&1,0,0,0,1,0,0,1,0,0,0,0,1,1,0,0,1,1,1,0,0,0,1,1,1,1,1,1,0,0,1,0,\\ &0,0,0,1,0,1,1,0,1,0,1,1,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,1,1,\\ &0,1,1,1,0,0,0,0,1,0,0,0,1,0,0,1,1,1,1,1,0,0,0,1,0,0,0,1,0,1,0,1,0,\\ &0,0,1,0,1,0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,0,1,1,1,0,1,0,0,0,0,0,1,\\ &0,1,0,0,1,1,1,0,0,1,1,1,0,1,0,1,1,0,1,0,1,0,1,0,0,1,1,0,0,0,1,0,\\ &1,1,0,0,1,0,1,1,1,0,0,0,1,1,0,1,0,1,1,1,1,1,1,0,1,1,0,1,0,0,1,0,\\ &0,0,0,1,0,1,0,0,1,1,1,1,1,1,0,1,0,0,1,1,0,1,1,1,0,0,1,1,1,1,1,1,\\ &1,1,0,1,0,1,0,1,1,0,0,1,1,1,0,1,1,1,0,0,1,0,0,0,1,0,0,0,1,1,1,0). \end{aligned}$$

Given this word, Patterson's algorithm computes $s = \zeta^{51} + \zeta^{119}x + \zeta^{64}x^2 + \zeta^{230}x^3 + \zeta^9x^4 + \zeta^{30}x^5 + \zeta^{187}x^6 + \zeta^{226}x^7 + \zeta^{55}x^8 + \zeta^{84}x^9 + \zeta^{80}x^{10} + \zeta^{72}x^{11} + \zeta^{71}x^{12} + \zeta^{152}x^{13} + \zeta^{220}x^{14} + \zeta^{221}x^{15} + \zeta^{224}x^{16} + \zeta^{154}x^{17} + \zeta^{166}x^{18} + \zeta^{130}x^{19} + \zeta^{225}x^{20} + \zeta^{11}x^{21}$. Reducing the basis $(s,1),(g,0)$ produces a minimum-length nonzero vector (α_0,β_0) and a minimum-length independent vector (α_1,β_1); here $\alpha_0 = \zeta^{52} + \zeta^{27}x + \zeta^{89}x^2 + \zeta^{58}x^3 + \zeta^{140}x^4 + \zeta^{139}x^5 + \zeta^{86}x^6 + \zeta^{247}x^7 + \zeta^{245}x^8 + \zeta^{181}x^9 + \zeta^{85}x^{10} + \zeta^{37}x^{11}$, $\beta_0 = \zeta^{26} + \zeta^{203}x + \zeta^{175}x^2 + \zeta^{130}x^3 + \zeta^{122}x^4 + \zeta^{168}x^5 + \zeta^{168}x^6 + \zeta^{95}x^7 + \zeta^{154}x^8 + \zeta^{114}x^9 + \zeta^{202}x^{10}$, $\alpha_1 = \zeta^{124} + \zeta^{115}x + \zeta^{194}x^2 + \zeta^{127}x^3 + \zeta^{175}x^4 + \zeta^{84}x^5 + \zeta^{167}x^6 + \zeta^{119}x^7 + \zeta^{55}x^8 + \zeta^{145}x^9 + \zeta^{204}x^{10}$, and $\beta_1 = \zeta^{221} + \zeta^{32}x + \zeta^{113}x^2 + \zeta^{118}x^3 + \zeta^{162}x^4 + \zeta^{93}x^5 + \zeta^{110}x^6 + \zeta^{178}x^7 + \zeta^{67}x^8 + \zeta^{140}x^9 + \zeta^{11}x^{10} + \zeta^{218}x^{11}$.

At this point Patterson's algorithm would hope for $\epsilon_0 = \alpha_0^2 + x\beta_0^2$ to divide h, but there is no such luck; there are no codewords $c \in \Gamma$ with $|w-c| \leq 22$.

The polynomial $\alpha_1^2 + x\beta_1^2$ has roots, as does the polynomial $\alpha_1^2 + x\beta_1^2 + \epsilon_0$, but the polynomial $\epsilon_1 = \alpha_1^2 + x\beta_1^2 + \zeta\epsilon_0$ has no roots; i.e., $\gcd\{\epsilon_1, h\} = 1$. This paper's extension of Patterson's algorithm outputs (ϵ_0, ϵ_1).

Out of curiosity I checked all 256 possibilities for $r \in \mathbf{F}_{2^m}$, and found that a uniform random choice of r has $\gcd\{\alpha_1^2 + x\beta_1^2 + r\epsilon_0, h\} = 1$ with probability $91/256 \approx \exp(-1)$. In retrospect it is not surprising that a few tries sufficed to find a successful value of r.

5. Review of divisors in arithmetic progressions

Consider the problem of finding all divisors of n congruent to u modulo v, where u, v, n are positive integers with $\gcd\{v, n\} = 1$. (What does this have to do with list decoding? Bear with me.)

There is no difficulty if $v \geq n^{1/2}$. Lenstra (1984) published a polynomial-time algorithm for $v \geq n^{1/3}$. Konyagin and Pomerance (1997) published a polynomial-time algorithm for $v \geq n^{3/10}$. Coppersmith, Howgrave-Graham, and Nagaraj found a polynomial-time algorithm for $v \geq n^{\alpha^2}$ for any fixed $\alpha > 1/2$; see (Howgrave-Graham 1998, Section 5.5) and (Coppersmith, Howgrave-Graham, and Nagaraj 2004). (Lenstra subsequently pointed out that one could handle $\alpha = 1/2$, but this extra refinement is not relevant here.) More generally, the Coppersmith–Howgrave-Graham–Nagaraj algorithm finds all divisors of n in an arithmetic progression $u - Hv, u - (H-1)v, \ldots, u - v, u, u + v, \ldots, u + (H-1)v, u + Hv$. The algorithm is polynomial-time if the smallest entry $u - vH$ is $n^{1/\alpha}$ and the number $2H + 1$ of entries is smaller than approximately n^{1/α^2}.

The algorithm actually does more: it finds all small integers s such that the fraction $(s + w)/n$ has small denominator. Here w is the quotient of u by v modulo n. Note that $(s + w)/n$ has denominator at most $n/(u + sv)$ if $u + sv$ divides n: indeed, $(s + w)/n = \bar{v}(u + sv)/n + (w - u\bar{v})/n + s(1 - v\bar{v})/n$, where $\bar{v}$ is the reciprocal of v modulo n.

Boneh later pointed out—see (Boneh 2000)—that the same algorithm can be viewed as a state-of-the-art list-decoding algorithm for "CRT codes" under a standard weighted distance. Take n to be a product of many small primes $p_1, p_2, \ldots$, and consider codewords $(s \bmod p_1, s \bmod p_2, \ldots)$ where $s \in \{-H, \ldots, 0, 1, \ldots, H\}$. A word $(w \bmod p_1, w \bmod p_2, \ldots)$ is close to a codeword $(s \bmod p_1, s \bmod p_2, \ldots)$ if and only if $s - w$ has a large factor in common with n, i.e., $(s - w)/n$ has small denominator.

The algorithm. Fix positive integers ℓ, k with $\ell > k$. Define $L \subset \mathbf{Q}[z]$ as the ℓ-dimensional lattice generated by the polynomials

$$1, \frac{Hz + w}{n}, \left(\frac{Hz + w}{n}\right)^2, \ldots, \left(\frac{Hz + w}{n}\right)^k,$$
$$Hz\left(\frac{Hz + w}{n}\right)^k, (Hz)^2\left(\frac{Hz + w}{n}\right)^k, \ldots, (Hz)^{\ell-k-1}\left(\frac{Hz + w}{n}\right)^k.$$

The Coppersmith–Howgrave-Graham–Nagaraj algorithm uses lattice-basis reduction to find a nonzero vector $\varphi \in L$ with small coefficients. It then finds the desired integers s by finding rational roots s/H of φ.

Specifically, L has determinant $H^{\ell(\ell-1)/2}/n^{\ell k - k(k+1)/2}$, so the well-known LLL lattice-basis-reduction algorithm finds φ with norm at most $(2H)^{(\ell-1)/2}/n^{k-k(k+1)/2\ell}$. If $|s/H| \leq 1$ then $\varphi(s/H) \leq \sqrt{\ell}(2H)^{(\ell-1)/2}/n^{k-k(k+1)/2\ell}$; but $\varphi(s/H)$ is also a multiple of $1/D^k$

where D is the denominator of $(s+w)/n$. In particular, $\varphi(s/H)$ must be 0 if $1/D^k > \sqrt{\ell}(2H)^{(\ell-1)/2}/n^{k-k(k+1)/2\ell}$.

The algorithm thus finds all integers $s \in \{-H, \ldots, -1, 0, 1, \ldots, H\}$ such that the denominator of $(s+w)/n$ is smaller than $n^{1-(k+1)/2\ell}/\ell^{1/2k}(2H)^{(\ell-1)/2k}$. By choosing a moderately large k, and choosing $\ell \approx k\sqrt{(\lg 2n)/\lg 2H}$, one can push the denominator bound up to approximately $n/2^{\sqrt{(\lg 2n)(\lg 2H)}}$, and in particular find divisors larger than approximately $2^{\sqrt{(\lg 2n)(\lg 2H)}}$.

The function-field analogue. The integers in the Coppersmith–Howgrave-Graham–Nagaraj algorithm can be replaced by polynomials over a finite field. The LLL algorithm for integer lattice-basis reduction is replaced by simpler algorithms—see, e.g., (Lenstra 1985, Section 1) and (Mulders and Storjohann 2003, Section 2)—for polynomial lattice-basis reduction.

Many of the cryptanalytic applications of the algorithm are uninteresting for polynomials, since polynomials can be factored efficiently into irreducibles. However, the list-decoding application remains interesting for polynomials—it is essentially the Guruswami–Sudan algorithm!

Section 6 extends the Coppersmith–Howgrave-Graham–Nagaraj algorithm to solve a slightly more complicated "linear combinations as divisors" problem. Section 7 presents this paper's new list-decoding method for binary Goppa codes, combining the function-field analogue of the "linear combinations as divisors" algorithm with the extension of Patterson's algorithm presented in Section 4.

6. Linear combinations as divisors

The Coppersmith–Howgrave-Graham–Nagaraj algorithm discussed in Section 5, given positive integers u, v, n with $\gcd\{v, n\} = 1$, finds all small integers s such that $u + sv$ divides n. This section explains, more generally, how to find all pairs of small coprime integers (r, s) with $r > 0$ such that $ru + sv$ divides n. The precise meaning of "small" is defined below.

The algorithm can output additional pairs (r, s). It is up to the user to check which of the pairs (r, s) is small, has $ru + sv$ dividing n, etc. However, the algorithm is guaranteed to finish quickly (and therefore to output very few pairs), and its output is guaranteed to include all of the desired pairs (r, s).

The algorithm. Compute the quotient w of u by v modulo n. This algorithm actually looks for small coprime (r, s) such that $(s + rw)/n$ has small denominator.

Fix positive integers G, H, and define $\Theta = H/G$. The algorithm focuses on pairs (r, s) such that $1 \le r \le G$ and $-H \le s \le H$.

Fix positive integers ℓ, k with $\ell > k$. Define $L \subset \mathbf{Q}[z]$ as the ℓ-dimensional lattice generated by the polynomials

$$1, \frac{\Theta z + w}{n}, \left(\frac{\Theta z + w}{n}\right)^2, \ldots, \left(\frac{\Theta z + w}{n}\right)^k,$$
$$\Theta z \left(\frac{\Theta z + w}{n}\right)^k, (\Theta z)^2 \left(\frac{\Theta z + w}{n}\right)^k, \ldots, (\Theta z)^{\ell-k-1} \left(\frac{\Theta z + w}{n}\right)^k.$$

Use lattice-basis reduction to find a nonzero vector $\varphi \in L$ with small coefficients.

For each rational root of φ: Multiply the root by Θ, write the product in the form s/r with $\gcd\{r,s\} = 1$ and $r > 0$, and output (r,s).

What the algorithm accomplishes. The determinant of L is $\Theta^{\ell(\ell-1)/2}/n^{\ell k-k(k+1)/2}$, so reduction guarantees that

$$\sqrt{\varphi_0^2+\varphi_1^2+\cdots} \le \frac{(2\Theta)^{(\ell-1)/2}}{n^{k-k(k+1)/2\ell}}.$$

Assume that $1 \le r \le G$ and $-H \le s \le H$. Then

$$\begin{aligned}\left|r^{\ell-1}\varphi\left(\frac{s}{\Theta r}\right)\right| &= \left|\varphi_0 r^{\ell-1} + \varphi_1 r^{\ell-2}\frac{s}{\Theta} + \cdots + \varphi_{\ell-1}\frac{s^{\ell-1}}{\Theta^{\ell-1}}\right| \\ &\le \sqrt{(r^{\ell-1})^2+\cdots+\left(\frac{s^{\ell-1}}{\Theta^{\ell-1}}\right)^2}\sqrt{\varphi_0^2+\varphi_1^2+\cdots} \\ &\le \sqrt{\ell}G^{\ell-1}\frac{(2\Theta)^{(\ell-1)/2}}{n^{k-k(k+1)/2\ell}} = \frac{\sqrt{\ell}(2GH)^{(\ell-1)/2}}{n^{k-k(k+1)/2\ell}}.\end{aligned}$$

Assume further that $(s+rw)/n$ has denominator D. Then $(s/r+w)/n$ is a multiple of $1/Dr$ so all of

$$1, \frac{s/r+w}{n}, \ldots, \left(\frac{(s/r+w)}{n}\right)^k, \ldots, (s/r)^{\ell-k-1}\left(\frac{(s/r+w)}{n}\right)^k$$

are multiples of $(1/r)^{\ell-k-1}(1/Dr)^k = 1/D^k r^{\ell-1}$. Thus $\varphi(s/\Theta r)$ is a multiple of $1/D^k r^{\ell-1}$; i.e., $r^{\ell-1}\varphi(s/\Theta r)$ is a multiple of $1/D^k$.

Now assume additionally that $D < n^{1-(k+1)/2\ell}/\ell^{1/2k}(2GH)^{(\ell-1)/2k}$. Then $1/D^k > \sqrt{\ell}(2GH)^{(\ell-1)/2}/n^{k-k(k+1)/2\ell}$, so $r^{\ell-1}\varphi(s/\Theta r)$ must be 0; i.e., $s/\Theta r$ is a root of φ. The algorithm finds $s/\Theta r$ if $\gcd\{r,s\} = 1$.

In particular, if $ru+sv$ is a divisor of n with $1 \le r \le G$, $-H \le s \le H$, $\gcd\{r,s\} = 1$, and $ru+sv > \ell^{1/2k}(2GH)^{(\ell-1)/2k}n^{(k+1)/2\ell}$, then the algorithm outputs (r,s).

By choosing a moderately large k, and choosing $\ell \approx k\sqrt{(\lg 2n)/\lg 2GH}$, one can push the bound $\ell^{1/2k}(2GH)^{(\ell-1)/2k}n^{(k+1)/2\ell}$ down to approximately $2^{\sqrt{(\lg 2n)(\lg 2GH)}}$.

Comparison to other "Coppersmith-type" algorithms. My survey paper (Bernstein 2008) discusses a general method that, given a polynomial f, finds all small-height rational numbers s/r such that $f(s/r)$ has small height. Here "small height" means "small numerator and small denominator." This includes finding divisors in residue classes and finding codeword errors beyond half the minimum distance, as discussed in Section 5; other standard applications are finding divisors in short intervals, finding high-power divisors, and finding modular roots.

All of these applications specify the denominator r; in other words, they find all small *integers* s such that $f(s)$ has small height. But this limitation is not inherent in the method. The method discovers small pairs (r,s) even if both r and s are allowed to vary.

In particular, one can efficiently find all small-height rational numbers s/r such that $(s/r+w)/n$ has small height—in particular, all small-height rational numbers s/r such that $ru+sv$ divides n. What I have shown in this section is that, for divisors $ru+sv \approx n^{1/\alpha}$, "small" includes all (r,s) with rs up to approximately n^{1/α^2}.

The same method generalizes to polynomials f in more variables. One can, for example, find all small integer pairs (r,s) such that $f(r,s)$ has small height. However, the bivariate

method is considerably more difficult to analyze and optimize than the univariate method. Even when the bivariate method can be proven to work, it typically searches fewer f inputs than the univariate method. What the algorithm in this section illustrates is that *homogeneous* bivariate polynomials are almost as easy to handle as univariate polynomials.

The function-field analogue. The integers in this algorithm can be replaced by polynomials over a finite field. The resulting algorithm can be used for list decoding of classical irreducible binary Goppa codes. See Section 7.

In this application one cares only about *squares* r, s. In other words, one wants to find divisors of n of the form r^2u+s^2v. One can apply lattice-basis-reduction methods directly to the polynomial $(s^2 + r^2w)/n$, but I don't see how this would allow larger rs. Perhaps I'm missing an easy factor-of-2 improvement (in general, or in the characteristic-2 case), or perhaps there's an explanation for why this type of improvement can't work.

7. List decoding via divisors

Recall that the algorithm from Section 4 finds two polynomials $\epsilon_0, \epsilon_1 \in \mathbf{F}_{2^m}[x]$ such that each desired error polynomial ϵ is a small linear combination of ϵ_0 and ϵ_1. Specifically, if $\deg \epsilon \le t + u$ and $\deg \epsilon_0 = t_0$ then $\epsilon = q_0^2\epsilon_0 + q_1^2\epsilon_1$ for some polynomials $q_0, q_1 \in \mathbf{F}_{2^m}[x]$ with $\deg q_0 \le \lfloor (t + u - t_0)/2 \rfloor$ and $\deg q_1 \le \lfloor (t_0 + u - t - 1)/2 \rfloor$.

A polynomial $\epsilon = q_0^2\epsilon_0 + q_1^2\epsilon_1$ is useful only if its monic part splits into linear factors of the form $x - a_i$; in other words, only if it divides $h = \prod_i (x - a_i)$. Note that q_0, q_1 must be coprime; otherwise ϵ would not be squarefree.

How do we search for divisors of h that are small coprime linear combinations of ϵ_0, ϵ_1? Answer: This is exactly the function-field analogue of the problem solved in the previous section!

To avoid unnecessary dependence on Sections 5 and 6, this section gives a self-contained statement of the list-decoding algorithm. Readers who have studied the algorithm in Section 6 should recognize its similarity to the algorithm in this section.

The list-decoding algorithm. Fix an integer $u \ge 0$. This algorithm will try to correct $t + u$ errors.

Compute ϵ_0, ϵ_1 by the algorithm of Section 4. Define $t_0 = \deg \epsilon_0$; $g_0 = 2\lfloor (u + t - t_0)/2 \rfloor$; $g_1 = 2\lfloor (u + t_0 - t - 1)/2 \rfloor$; and $\theta = g_1 - g_0$.

Assume for simplicity that $\gcd\{\epsilon_1, h\} = 1$. Compute a polynomial $\delta \in \mathbf{F}_{2^m}[x]$ such that $\epsilon_1\delta \bmod h = \epsilon_0$.

Fix integers $\ell > k > 0$. Define $L \subset \mathbf{F}_{2^m}(x)[z]$ as the ℓ-dimensional lattice generated by the polynomials

$$1, \frac{x^\theta z + \delta}{h}, \left(\frac{x^\theta z + \delta}{h}\right)^2, \ldots, \left(\frac{x^\theta z + \delta}{h}\right)^k,$$
$$x^\theta z \left(\frac{x^\theta z + \delta}{h}\right)^k, (x^\theta z)^2 \left(\frac{x^\theta z + \delta}{h}\right)^k, \ldots, (x^\theta z)^{\ell-k-1} \left(\frac{x^\theta z + \delta}{h}\right)^k.$$

Use lattice-basis reduction to find a minimal-length nonzero vector $\varphi \in L$. Here the length of $\varphi_0 + \varphi_1 z + \cdots$ is, by definition, $\max\{|\varphi_0|, |\varphi_1|, \ldots\}$.

Use standard polynomial-factorization algorithms to find all of φ's roots in $\mathbf{F}_{2^m}(x)$, and in particular to find roots that have the form $q_0^2/x^\theta q_1^2$ for coprime polynomials $q_0, q_1 \in \mathbf{F}_{2^m}[x]$. For each such root, compute $\epsilon = q_0^2\epsilon_0 + q_1^2\epsilon_1$, and check whether ϵ is a divisor of h; if it is, output the unique $c \in \mathbf{F}_2^n$ such that $\{i : c_i - w_i = 1\} = \{i : \epsilon(a_i) = 0\}$.

What the algorithm accomplishes. Consider any $c \in \Gamma$ with $|w - c| \le t + u$. Define $\epsilon = \prod_{i:w_i \neq c_i}(x - a_i) \in \mathbf{F}_{2^m}[x]$. Then there are polynomials $q_0, q_1 \in \mathbf{F}_{2^m}[x]$, such that $\epsilon = q_0^2\epsilon_0 + q_1^2\epsilon_1$, with $\deg q_0 \le \lfloor (t + u - t_0)/2 \rfloor = g_0/2$ and $\deg q_1 \le \lfloor (t_0 + u - t - 1)/2 \rfloor = g_1/2$; see Section 4.

If $q_0 = 0$ then $\epsilon = q_1^2\epsilon_1$, but ϵ is squarefree, so ϵ/ϵ_1 is a constant, so ϵ_1 divides h, so the algorithm outputs c. Assume from now on that $q_0 \neq 0$.

The fraction $(q_0^2\epsilon_0 + q_1^2\epsilon_1)/h$ is exactly $1/(h/\epsilon)$, so the fraction $(q_1^2 + q_0^2\delta)/h$ is a multiple of $1/(h/\epsilon)$, so the fraction $(q_1^2/q_0^2 + \delta)/h$ is a multiple of $1/(q_0^2 h/\epsilon)$. The value $\varphi(q_1^2/x^\theta q_0^2)$ is a linear combination of

$$1, \frac{q_1^2/q_0^2 + \delta}{h}, \left(\frac{q_1^2/q_0^2 + \delta}{h}\right)^2, \ldots, \left(\frac{q_1^2/q_0^2 + \delta}{h}\right)^k,$$
$$\frac{q_1^2}{q_0^2}\left(\frac{q_1^2/q_0^2 + \delta}{h}\right)^k, \ldots, \left(\frac{q_1^2}{q_0^2}\right)^{\ell-k-1}\left(\frac{q_1^2/q_0^2 + \delta}{h}\right)^k,$$

all of which are multiples of $(1/q_0^2)^{\ell-k-1}(1/(q_0^2 h/\epsilon))^k = 1/(q_0^2)^{\ell-1}(h/\epsilon)^k$. The homogenized value $(q_0^2)^{\ell-1}\varphi(q_1^2/x^\theta q_0^2)$ is therefore a multiple of $1/(h/\epsilon)^k$, which has degree $-k(n - \deg\epsilon)$.

The specified basis elements of L have z-degrees $0, 1, 2, \ldots, k, k+1, k+2, \ldots, \ell-1$ respectively, with leading coefficients

$$1, \frac{x^\theta}{h}, \left(\frac{x^\theta}{h}\right)^2, \ldots, \left(\frac{x^\theta}{h}\right)^k, x^\theta\left(\frac{x^\theta}{h}\right)^k, (x^\theta)^2\left(\frac{x^\theta}{h}\right)^k, \ldots, (x^\theta)^{\ell-k-1}\left(\frac{x^\theta}{h}\right)^k.$$

Thus L is a lattice of dimension ℓ. Furthermore, the product of these leading coefficients is $x^{\theta(\ell-1)\ell/2}/h^{k\ell-k(k+1)/2}$, with degree $\theta(\ell-1)\ell/2 + n(k(k+1)/2 - k\ell)$. Thus each coefficient of φ has degree at most $\theta(\ell-1)/2 + n(k(k+1)/2\ell - k)$.

The degree of q_0^2 is at most g_0, and the degree of q_1^2/x^θ is at most $g_1 - \theta = g_0$, so the homogenized value $(q_0^2)^{\ell-1}\varphi(q_1^2/x^\theta q_0^2) = \varphi_0(q_0^2)^{\ell-1} + \varphi_1(q_0^2)^{\ell-2}q_1^2/x^\theta + \cdots + \varphi_{\ell-1}(q_1^2/x^\theta)^{\ell-1}$ has degree at most $\theta(\ell-1)/2 + n(k(k+1)/2\ell - k) + (\ell-1)g_0 = (g_0+g_1)(\ell-1)/2 + n(k(k+1)/2\ell - k)$.

If $\deg\epsilon > (g_0+g_1)(\ell-1)/2k + n(k+1)/2\ell$ then $-k(n - \deg\epsilon) > (g_0+g_1)(\ell-1)/2 + n(k(k+1)/2\ell - k)$ so $(q_0^2)^{\ell-1}\varphi(q_1^2/x^\theta q_0^2)$ must be 0. The algorithm finds $q_1^2/x^\theta q_0^2$ as a root of φ, finds (q_0, q_1) since $\gcd\{q_0, q_1\} = 1$, finds ϵ, sees that ϵ divides h, and outputs c.

By choosing a moderately large k, and choosing $\ell \approx k\sqrt{n/(g_0+g_1)}$, one can push the degree bound $(g_0+g_1)(\ell-1)/2k + n(k+1)/2\ell$ to approximately $\sqrt{n(g_0+g_1)} \approx \sqrt{2(u-1)n}$. If the degree bound is below $t+u$ then the algorithm will find every codeword at distance $t+u$ from w; if the degree bound is below $t+u-1$ then the algorithm will find every codeword at distance $t+u$ or $t+u-1$ from w; etc. One can cover smaller distances by running the algorithm several times with different choices of u. (See (Bernstein 2008, Section 6) for discussion of an analogous loop in the Coppersmith–Howgrave-Graham–Nagaraj algorithm.)

This decoding guarantee breaks down at approximately $n - \sqrt{n(n-2t-2)}$ errors: the degree bound $\sqrt{2(u-1)n}$ grows past $t+u$ as $t+u$ grows past $n - \sqrt{n(n-2t-2)}$.

Handling non-coprimality. This algorithm assumes that $\gcd\{\epsilon_1, h\} = 1$; see Section 4 for further discussion of this condition. A straightforward extension of the algorithm would allow a larger $\gcd\{\epsilon_1, h\}$ but would be correspondingly less effective. I don't know whether there is a polynomial-time algorithm that provably handles arbitrary $\gcd\{\epsilon_1, h\}$ without losing effectiveness.

Numerical example. This example is a continuation of the example in Section 4. Recall that the extension of Patterson's algorithm produced two polynomials $\epsilon_0 = \zeta^{74}x^{22} + \cdots$ and $\epsilon_1 = \zeta^{181}x^{23} + \cdots$ with $\gcd\{\epsilon_1, h\} = 1$. The goal of the algorithm in this section is to find a small linear combination $\epsilon = q_0^2\epsilon_0 + q_1^2\epsilon_1$ that divides $h = x^{256} - x$.

Choose $u = 2$. Then $t_0 = 22$, $g_0 = 2$, $g_1 = 0$, and $\theta = -2$. The algorithm will search for ϵ of degree $t + u = 24$; equivalently, for q_0 of degree $\leq g_0/2 = 1$ and q_0 of degree $\leq g_1/2 = 0$.

Choose $k = 8$ and $\ell = 87$. Note that $(g_0+g_1)(\ell-1)/2k+n(k+1)/2\ell = 2783/116 < 24$. This example requires a moderately large k, since $t + u = 24$ is quite close to $n - \sqrt{n(n-2t-2)} \approx 24.1$.

Divide ϵ_0 by ϵ_1 modulo h to obtain $\delta = \zeta^{200}x^{255} + \zeta^{62}x^{254} + \cdots + \zeta^{85}x + \zeta^{104}$. Define L as the $\mathbf{F}_{2^m}[x]$-submodule of $\mathbf{F}_{2^m}(x)[z]$ generated by

$$1, \frac{z/x^2+\delta}{h}, \ldots, \frac{(z/x^2+\delta)^8}{h^8}, \left(\frac{z}{x^2}\right)\frac{(z/x^2+\delta)^8}{h^8}, \ldots, \left(\frac{z}{x^2}\right)^{78}\frac{(z/x^2+\delta)^8}{h^8}.$$

Then L is an 87-dimensional lattice. The coefficients of $1, z, z^2, \ldots, z^{86}$ in the generators are the columns of the following 87×87 matrix:

$$\begin{pmatrix}
1 & \delta/h & \delta^2/h^2 & \delta^3/h^3 & \delta^4/h^4 & \delta^5/h^5 & \delta^6/h^6 & \delta^7/h^7 & \delta^8/h^8 & 0 & \cdots & 0 \\
0 & 1/x^2h & 0 & \delta^2/x^2h^3 & 0 & \delta^4/x^2h^5 & 0 & \delta^6/x^2h^7 & 0 & \delta^8/x^2h^8 & \cdots & 0 \\
0 & 0 & 1/x^4h^2 & \delta/x^4h^3 & 0 & 0 & \delta^4/x^4h^6 & \delta^5/x^4h^7 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 1/x^6h^3 & 0 & 0 & 0 & \delta^4/x^6h^7 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 1/x^8h^4 & \delta/x^8h^5 & \delta^2/x^8h^6 & \delta^3/x^8h^7 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 1/x^{10}h^5 & 0 & \delta^2/x^{10}h^7 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1/x^{12}h^6 & \delta/x^{12}h^7 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/x^{14}h^7 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/x^{16}h^8 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/x^{18}h^8 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1/x^{172}h^8
\end{pmatrix}$$

It is convenient for computation to scale the entire matrix by $x^{172}h^8 = x^{2220} + x^{180}$, to avoid working with fractions. The determinant of the scaled matrix is $x^{7482}h^{36}$, with degree $16698 < 192 \cdot 87$, so lattice-basis reduction is guaranteed to find a nonzero vector $\varphi \in x^{172}h^8L$ where each component has degree < 192.

I reduced the lattice basis and, unsurprisingly, found such a vector, namely $\varphi = \varphi_0 + \varphi_1 z + \cdots + \varphi_{86}z^{86}$ where $\varphi_0 = \zeta^{232}x^{191} + \zeta^{42}x^{190} + \cdots + \zeta^{244}x^{172}$, $\varphi_1 = \zeta^{232}x^{191} + \zeta^{226}x^{190} + \cdots + \zeta^{132}x^{170}$, and so on through $\varphi_{86} = \zeta^{145}x^{191} + \zeta^{10}x^{190} + \cdots + \zeta^{36}x^0$. It turned out that the first 6 successive minima of the lattice all have degree < 192, so there were actually $256^6 - 1$ possibilities for φ.

I then computed the roots of φ in $\mathbf{F}_{2^m}[x]$ and found exactly one root of the desired form: namely, $\varphi(q_1^2/x^\theta q_0^2) = 0$ for $q_1 = \zeta^{153}$ and $q_0 = x - \zeta^{175}$. This calculation was particularly straightforward since the irreducible factorization $\varphi_{86} = \zeta^{145}(x^{170} + \cdots)(x^{13} + \cdots)(x^3 + \cdots)(x^3 + \cdots)(x - \zeta^{175})^2$ had only one square factor. A greatest-common-divisor calculation between leading terms of two independent short vectors would have revealed the same denominator even more easily.

Finally, the sum $\epsilon = q_0^2\epsilon_0 + q_1^2\epsilon_1 = \zeta^{74}x^{24} + \cdots$ has 24 distinct roots, namely $\zeta^2, \zeta^6, \zeta^7, \zeta^{15}, \zeta^{23}, \zeta^{38}, \zeta^{46}, \zeta^{59}, \zeta^{71}, \zeta^{73}, \zeta^{86}, \zeta^{88}, \zeta^{131}, \zeta^{138}, \zeta^{142}, \zeta^{150}, \zeta^{153}, \zeta^{159}, \zeta^{163}, \zeta^{165}, \zeta^{171}, \zeta^{172}, \zeta^{206}, \zeta^{214}$. Correcting the corresponding positions in w produces the unique $c \in \Gamma$ with $|w - c| \leq 24$.

References

(Avanzi 2001) Roberto M. Avanzi, *A study on polynomials in separated variables with low genus factors*, Ph.D. thesis, Universität Essen. URL: `http://caccioppoli.mac.rub.de/website/papers/phdthesis.pdf`. Citations in this document: §4.

(Bernstein 2008) Daniel J. Bernstein, "Reducing lattice bases to find small-height values of univariate polynomials", pp. 421–446 in (Buhler and Stevenhagen 2008). URL: `http://cr.yp.to/papers.html#smallheight`. Citations in this document: §6, §7.

(Bernstein, Lange, and Peters 2008) Daniel J. Bernstein, Tanja Lange, and Christiane Peters, "Attacking and defending the McEliece cryptosystem", pp. 31–46 in (Buchmann and Ding 2008). URL: `http://cr.yp.to/papers.html#mceliece`. Citations in this document: §1, §2.

(Boneh 2000) Dan Boneh, "Finding smooth integers in short intervals using CRT decoding", pp. 265–272 in (STOC 2000); see also newer version (Boneh 2002). Citations in this document: §5.

(Boneh 2002) Dan Boneh, "Finding smooth integers in short intervals using CRT decoding", *Journal of Computer and System Sciences* **64**, 768–784; see also older version (Boneh 2000). ISSN 0022–0000. MR 1 912 302. URL: `http://crypto.stanford.edu/~dabo/abstracts/CRTdecode.html`.

(Buchmann and Ding 2008) Johannes Buchmann and 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. ISBN 978–3–540–88402–6. See (Bernstein, Lange, and Peters 2008).

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

(Coppersmith, Howgrave-Graham, and Nagaraj 2004) Don Coppersmith, Nick Howgrave-Graham, and S. V. Nagaraj, "Divisors in residue classes, constructively". URL: `http://eprint.iacr.org/2004/339`. Citations in this document: §5.

(Graham and Nešetřil 1997) Ronald L. Graham and Jaroslav Nešetřil (editors), *The mathematics of Paul Erdős. I*, Algorithms and Combinatorics **13**, Springer-Verlag, Berlin. ISBN 3–540–61032–4. MR 97f:00032. See (Konyagin and Pomerance 1997).

(Guruswami and Sudan 1999) Venkatesan Guruswami and Madhu Sudan, "Improved decoding of Reed-Solomon and algebraic-geometry codes", *IEEE Transactions on Information Theory* **45**, 1757–1767. ISSN 0018–9448. MR 2000j:94033. URL: `http://theory.lcs.mit.edu/~madhu/bib.html`. Citations in this document: §1, §1.

(Howgrave-Graham 1998) Nicholas Howgrave-Graham, *Computational mathematics inspired by RSA*, Ph.D. thesis. URL: `http://cr.yp.to/bib/entries.html#1998/howgrave-graham`. Citations in this document: §5.

(Konyagin and Pomerance 1997) Sergei Konyagin and Carl Pomerance, "On primes recognizable in deterministic polynomial time", pp. 176–198 in (Graham and Nešetřil 1997). MR 98a:11184. URL: http://cr.yp.to/bib/entries.html#1997/konyagin. Citations in this document: §5.

(Lenstra 1984) Hendrik W. Lenstra, Jr., "Divisors in residue classes", *Mathematics of Computation* **42**, 331–340. ISSN 0025–5718. MR 85b:11118. URL: http://www.jstor.org/sici?sici=0025-5718(198401)42:165<331:DIRC>2.0.CO;2-6. Citations in this document: §5.

(Lenstra 1985) Arjen K. Lenstra, "Factoring multivariate polynomials over finite fields", *Journal of Computer and System Sciences* **30**, 235–248. MR 87a:11124. Citations in this document: §5.

(McEliece 1978) Robert J. McEliece, "A public-key cryptosystem based on algebraic coding theory", JPL DSN Progress Report, 114–116. URL: http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF. Citations in this document: §1.

(Mulders and Storjohann 2003) Thom Mulders and Arne Storjohann, "On lattice reduction for polynomial matrices", *Journal of Symbolic Computation* **35**, 377–401. Citations in this document: §5.

(Niederreiter 1986) Harald Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory", *Problems of Control and Information Theory* **15**, 159–166. Citations in this document: §1.

(Patterson 1975) Nicholas J. Patterson, "The algebraic decoding of Goppa codes", *IEEE Transactions on Information Theory* **21**, 203–207. Citations in this document: §1.

(Sidelnikov and Shestakov 1992) Vladimir M. Sidelnikov and Sergey O. Shestakov, "On an encoding system constructed on the basis of generalized Reed-Solomon codes", *Discrete Mathematics and Applications* **2**, 439–444. MR 94f:94009. Citations in this document: §1.

(STOC 2000) — (no editor), *Proceedings of the 32nd annual ACM symposium on theory of computing*, Association for Computing Machinery, New York. ISBN 1–58113–184–4. See (Boneh 2000).