New Techniques for Decoding the BCH Error-Correcting Codes [1]

Victor Y. Pan

Department of Mathematics and Computer Science

Lehman College, City University of New York

Bronx, NY 10468

Internet: VPAN@LCVAX.LEHMAN.CUNY.EDU

(Supported by NSF Grants CCR 020690, CCR 9625344, and CCR 973206

and by PSC CUNY Awards Nos. 666327, 667340, 668365 and 669363,)

Abstract

The coefficients of a polynomial of a degree n can be expressed via the power sums of its zeros by means of a polynomial equation known as the key equation for decoding the BCH error-correcting codes. Berlekamp's algorithm of 1968 solves this equation by using order of n^2 field operations in a fixed. Several algorithms of 1975-1980 rely on the extended Euclidean algorithm and computing Padé approximation, which yields solution in $O(n(\log n)^2 \log\log n)$ operations, though a considerable overhead constant is hidden in the "O" notation. We show algorithms (depending on the characteristic c of the ground field of the allowed constants) that simplify the solution and lead to further improvements of the latter bound, by factors ranging from order of $\log n$, for $c = 0$ and $c > n$ (in which case the overhead constant drops dramatically), to order of $\min(c, \log n)$, for $2 \leq c \leq n$; the algorithms use Las Vegas type randomization in the case of $2 < c \leq n$.

Key Words: algebraic codes, decoding, BCH error-correcting codes, Reed-Solomon codes, key equation, Padé approximation.

1991 Math. Subject Classification: 11T71, 12Y05, 94B35, 68Q40.

[1] The results of this paper have been presented at the 27th Annual ACM Symposium on Theory of Computing, 1997. The author is greatful to the ACM for permission to reuse them.

1 Introduction.

The main topic of this paper is the solution of the *key equation* for decoding the BCH error-correcting codes (cf. [Be68], pp. 178–188, and [M69]). This problem is equivalent to the problem I · POWER · SUMS(min) of the recovery of the coefficients of the minimum degree polynomial from a given sequence of $2n$ power sums of its zeros and is a major problem of algebraic coding theory and practice. The solution of the key equation is one of the main stages of decoding, and here the improvement of the known solution algorithms is theoretiate (as well as practical) research chalenge. Besides, the I. POWER. SUM (point) as some applications to matrix computations (see [**?**]).

In [Be68], E. R. Berlekamp has proposed a solution algorithm for the key equation (a KEY · SOLVER) that uses order of n^2 operations in a fixed field $\mathbf{F}$ of constants, hereafter referred to as *ops*. Various KEY · SOLVERs based on the reduction to the extended Euclidean algorithm for polynomials (hereafter referred to as *eEa*) have been proposed in [SKHN75], [Mi75], [Man77] and [BGY80] (see also [Pat74], [S75] and [Sa77]). By using a fast version of the eEa (based on the construction of [Mo73] and completed in [BGY80]), one may decrease the asymptotic cost bound n^2 dramatically, almost by factor n. The resulting KEY · SOLVERs require $O(\mu(n)\log n)$ ops provided that $\mu(n)$ ops suffice in order to multiply modulo x^n a pair of polynomials in x. Since

$$\mu(n) = O((n\log n)\log\log n) \tag{1.1}$$

over any field of constants [CK91], the bound $O(\mu(n)\log n)$ means $O(n(\log n)^2\log\log n)$. (Over the fields of constants supporting FFT, we have $\mu(n) = O(n\log n)$, and then the factor $\log\log n$ can be dropped.) Such an asymptotic improvement, however, is purely theoretical. In particular, the cited KEY · SOLVERs supporting the latter nearly linear asymptotic cost bound rely on application of the fast version of the eEa; consequently, they share its major deficiencies: these KEY · SOLVERs are not well-structured and imply a considerable overhead constant hidden in the above "O" notation of $O(\mu(n)\log n)$, which is a too high price for the decrease of the ops bound, taking into account that in practice n is not large. The alternative reduction to fast Toeplitz matrix computations yields the same asymptotic cost bound for the key equation [BGY80], [BP94], but similar problems arise here too. As a result, the users are not happy about the known asymptotically fast solution algorithms, and the present day hardware for solving the key equation relies on Berlekamp-like KEY · SOLVERs, using order of n^2 ops.

In section 3, we will show that over the fields of constants of characteristics $c = 0$ and $c > n$, Newton's iteration solves the key equation asymptotically faster, in $O(\mu(n)) = O((n \log n) \log\log n)$ ops, and in this case the overhead constant is much smaller. The algorithm is a simple chain of $\gamma - 1$ recursive steps, $\gamma = \lceil \log_2(n+1) \rceil$; the computational cost of performing the i-th step, $i = 1, \ldots, \gamma - 1$, essentially amounts to a few polynomial multiplications modulo x^h for $h \leq 2^{i+1}$, which can be performed fast the straightforward classical algorithm, by means of FFT or binary segmentation (cf. [BP94], pages 276–279). In particular, our KEY. SOLVER based on the straightforward algorithm in values a total of at most $(8/3)n^2 + O(n)$ ops where $c = 0$ or $c > n$, and $n+1$ is a power of 2. Assuming fast polynomial arithmetic where 2^{i+3}-th roots of 1 are available and FFT is applied, the overall cost of performing the i-th step of our algorithim of section 3 is roughly the cost of performing FFT three times on the 2^{i+2}-th roots of 1 and four times on the 2^{i+3}-th roots of 1 which gives us the overall cost bounf of $66n \log_2 n + O(n)$ ops (see appendix A). Our algorithm is reduced essentially to a rather short sequence of polynomial multiplications, which allows its simple and effective parallelization yielding here advantage versus both Berlekamp's and eEa's approaches. It is interesting that the Read-Solomon codes, which are the most popolar BCH codes, allow their natural construction over a field of characteristic $n+1$, in which case our algorithm of section 3 applies. The transitim to such a field, however, would require to change the current practical routine of builoing the codes in the field of characteristic 2. Clearly our present progress is not strong enough to motivate such a dramatic change. Moreover, practically n is rather small, and Berlekamp's approach should remain the method of choice in practic, whreas our approach seems to be of purely theoretical interest so far. We hope that our algorithm of section 3 does give new insight into the solution of the key equation. The some can be said about our other algorithims, which handle the cases where $2 \leq c \leq n$. In these cases, our algorithm of section 3 does not apply directly. More refined techniques of sections 4–10 enable us to extend Newton's iteration (at the cost $O(\mu(n)) = O((n \log n) \log\log n)$ ops) in order to reduce the key equation to Padé approximation problem. Our progress versus the known reduction of this kind [BGY80] is in the decrease by factor c of the size of the resulting Padé problem and, therefore, of the cost of its fastest known solution. The asymptotically slower solutions, using order of n^2 ops with small overhead constants, are accelerated by factor c^2. For c of the order of $\sqrt{n^2/\mu(n)}$, say, this means the cost order decrease from n^2 to $\mu(n)$; for larger c the computational cost of solving the Padé problem becomes negligible versus the reduction cost

of $O(\mu(n))$.

More precisely, we present two KEY $\cdot$ SOLVERs in the case where $0 < c \leq n$. One of them, that is, our deterministic algorithm 8.1, reduces KEY $\cdot$ SOLVE(n) to at most $c-1$ Padé problems of smaller sizes, of at most $(n/c, n/c)$ each. All these $c-1$ problems can be effectively solved concurrently; such a solution requires no data exchange among the processors. Even without using parallelism, this means acceleration by factor c Compared to the reduction of the KEY $\cdot$ SOLVE(n) to a single Padé problem of the size (n, n) shown in [BGY80], provided that one uses Padé solvers that have quadratic cost; the new reduction supports the same asymptotic cost bound as [BGY80] does if the Padé solvers rely on the fast eEa. For a large class of inputs, we only need to solve a single Padé problem of the smaller size, which means further acceleration by factor $c-1$. Moreover, we may achieve this effect for any input by means of randomization of Las Vegas type. Namely, our algorithm 9.1 uses $c-1$ random parameters and either solves the key equation correctly by reducing it, at the cost of performing $O(\mu(n))$ ops, to a single Padé problem of a size at most $(n/c, n/c)$ or fails with a small probability, but never outputs wrong answer. Furthermore, in the unlikely case of the failure, we may reuse the results of the computations and the random parameters, so as to yield an extension to deterministic solution (cf. our algorithm 9.2 and remark 10.1).

In addition to application to decoding the error-correcting codes, our algorithms can be applied to parallel computation of the characteristic polynomial of a matrix over finite fields (cf.[**?**]), which is a fundamental linear algebra computation.

Remark 1.1. Technically, the present paper extends [P96], whose technical origin can be further traced back to [V900], [K25], [K27], [N27], [Sc93], and [BP94]. It should be noted, however, that even the upper estimates of [P96] for the computational complexity of I·POWER·SUMS, which improve ones of [Sc93] and [BP94], still exceed the order of n^2 ops, that is, the upper estimates of the present paper are smaller by order of magnitude than ones of [Sc93], [BP94] and [P96].

We will organize our paper as follows. In section 2, we define the inverse power sums problem and the key equation for the BCH decoding. In section 3, we show how to compute the coefficients of $P(x)$ rapidly, provided that $c > n$ or $c = 0$. In section 4, we state some auxiliary facts and definitions. In section 5, we consider the case of the computations over the fields of characteristic 2. In sections 7-10, we extend these considerations to devise and analyze now asymptotically faster KEY. SOLVERS over any field.

In the appendix, we estimate the cost of computations by the algorithim of section 3 (in the case where $c = 0 or c > n$) assuming at first classical polynomial arithmetic and then fast FFT based polynomial arithmetic.

Acknowledgements. I am grateful to Igor Shparlinski and Dan Spielman for valuable comments and information on algebraic codes, to and Eric Rains for the clarification of the role of Newton's iteration for the algorithm of section 3, and to the editor and both referees for helping me to improv the origonal drft of hte paper.

2 The problem $\mathrm{I} \cdot \mathrm{POWER} \cdot \mathrm{SUMS}(min)$ and the key equation for BCH decoding

The inverse power sum problem (which we will denote by $\mathrm{I} \cdot \mathrm{POWER} \cdot \mathrm{SUMS}(n)$) is the problem of computing the coefficients $p_1, \ldots, p_n$ of a polynomial of a degree at most n,

$$P(x) = \sum_{i=0}^{n} p_i x^i = \prod_{j=1}^{n} (1 - x z_j), \quad p_0 = 1 \ , \tag{2.1}$$

where we are given the power sums of the zeros of this polynomial,

$$s_k = \sum_{j=1}^{n} z_j^k, \quad k = 1, 2, \ldots, 2n. \tag{2.2}$$

The converse problem of computing the first $2n$ power sums from given coefficients of $p(x)$ is simpler: it amounts to computing modulo x^{2n} at first the reciprocal $1/P(x)$ and then its product by $P'(x)$ (cf. equation (2.3) below).

We will assume computations over a field $\mathbf{F}$ of a characteristic c; to ensure uniqueness of the solution, we will additionally require that the output polynomial has the minimum degree (not exceeding n). In the above form, the problem will be referred to as $\mathrm{I} \cdot \mathrm{POWER} \cdot \mathrm{SUMS}(min)$. Without the minimization requirement, the problem would not be well defined for positive c, as can be seen from the following simple observation:

Fact 2.1. *If equations (2.1) and (2.2) hold over a field $\mathbf{F}$ of a characteristic c for some polynomial $P(x)$, then they also hold for any scalar z and for $P(x)$ replaced by the polynomial $P(x)(1 - xz)^c$.*

Proof. The impact of the c-fold zero $x = 1/z$ of $P(x)(1 - xz)^c$ on the power sum s_k amounts to adding cz^k, which is 0 mod c, that is, such an impact is nil over $\mathbf{F}$, for all k. □

Next, we will restate the problem $\mathrm{I}\cdot\mathrm{POWER}\cdot\mathrm{SUMS}(min)$ in the form closely reduced to decoding the BCH error-correcting codes. At first, recall that

$$\frac{P'(x)}{P(x)} = -\sum_{j=1}^{n} \frac{z_j}{1-xz_j} \ .$$

Substitute $\frac{1}{1-xz_j} = \sum_{i=0}^{\infty}(xz_j)^i$ and obtain that

$$-\frac{P'(x)}{P(x)} = \sum_{j=1}^{n} z_j \sum_{k=0}^{\infty}(xz_j)^k = \sum_{k=0}^{\infty} x^k \sum_{j=1}^{n} z_j^{k+1} \ .$$

Substitute (2.2) on the right-hand side and express the above ratio via generating function for the power sums as follows:

$$\frac{P'(x)}{P(x)} = -\sum_{k=0}^{\infty} s_{k+1}x^k \ . \tag{2.3}$$

Multiply both sides by $xP(x)$, then add $P(x)$ on both sides, reduce the right-hand side modulo x^{2n+1}, and obtain that

$$W(x) = -P(x)\sum_{k=0}^{2n} s_k x^k \bmod \ x^{2n+1}, \tag{2.4}$$

where we write

$$W(x) = P(x) + xP'(x), \quad s_0 = -1. \tag{2.5}$$

Note that

$$P(0) = W(0) = 1, \tag{2.6}$$

due to (2.1). After Berlekamp [Be68], (2.4) is called the *key equation* for decoding the BCH error-correcting codes. We will cite the problem of computing the pair of polynomials $P(x)$ and $W(x)$ that have the minimum degrees (not exceeding n) and satisfy the equations (2.4)–(2.6), for some fixed $s_1, \ldots, s_{2n}$, as the problem $\mathrm{KEY}\cdot\mathrm{SOLVE}(n)$, which is clearly equivalent to $\mathrm{I}\cdot\mathrm{POWER}\cdot\mathrm{SUMS}(min)$.

Remark 2.1. Assuming that the zeros $z_1, \ldots, z_n$ are indeterminates in (2.1), one may prove that there are exactly n algebraically independent power sums s_k of (2.2) among $s_1, \ldots, s_{m+n}$, where $m = 0$ if $c = 0$, $m = \lfloor (n-1)/(c-1) \rfloor$ otherwise [Ka25], [Ka27], [Sc93]. Note that $s_{ci} = s_i^c$ for all i and c.