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 9020690, CCR 9625344 and CCR 9732206 and 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 operations in a fixed field. 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 $2 \le c \le n$; the algorithms use Las Vegas type randomization in the case of $2 < c \le n$.

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]). The problem is equivalent to the problem $\mathrm{I} \cdot \mathrm{POWER} \cdot \mathrm{SUMS}(min)$ 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 theory and practice of algebraic coding. The solution of the key equation is one of the main stages of decoding, and the improvement of the known algorithms for this stage is a theoretical (as well as practical) research challenge.

In [Be68], E. R. Berlekamp proposed a solution algorithm for the key equation (a $\mathrm{KEY} \cdot \mathrm{SOLVER}$) that used order of n^2 operations in a fixed field $\mathbf{F}$ of constants. (Hereafter, we will refer to such as *ops*). Another group of $\mathrm{KEY} \cdot \mathrm{SOLVER}$s relies on the reduction to the extended Euclidean algorithm for polynomials (hereafter we will referr to such an algorithm as *eEa*). Such $\mathrm{KEY} \cdot \mathrm{SOLVER}$s 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 $\mathrm{KEY} \cdot \mathrm{SOLVER}$s 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 $\mathrm{KEY} \cdot \mathrm{SOLVER}$s supporting the latter nearly linear asymptotic cost bound rely on application of the fast version of the eEa; consequently, they share its major deficiency: these $\mathrm{KEY} \cdot \mathrm{SOLVER}$s are not well-structured and imply a considerable overhead constant hidden in the above "O" notation of $O(\mu(n) \log n)$. This is a too high price for the asymptotic 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 $\mathrm{KEY} \cdot \mathrm{SOLVER}$s, using order of n^2 ops.

Over the fields of constants that have characteristics $c = 0$ or $c > n$, Brent's modification of Newton's iteration [Br75] solves the key equation asymptotically faster, by using $O(\mu(n)) = O((n \log n) \log\log n)$ ops. Furthermore, in this case the resulting KEY·SOLVER is much better structured than the fast eEa, and the overhead constant is much smaller, that is, we yield substantial and uniform advantage over the fast eEa approach. In section 3 and the appendix, we reduce the entire solution to 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 by means of FFT or binary segmentation (cf. [BP94], pages 276–279). Even where such a KEY·SOLVER relies on the straightforward algorithm, it involves a total of at most $(8/3)n^2 + O(n)$ ops provided that $c = 0$ or $c > n$, and $n + 1$ is a power of 2. Assuming the FFT based fast polynomial arithmetic and the 2^{i+3}-th roots of 1 available, 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 bound of $66 n \log_2 n + O(n)$ ops (see appendix A). Furthermore, since the algorithm is reduced essentially to a rather short sequence of polynomial multiplications, it can be effectively parallelized, which is an advantage over both Berlekamp's algorithm and the fast eEa approach.

It is interesting that the Read-Solomon codes, which are the most popular BCH codes, allow their natural construction over a field of characteristic $n+1$, in which case our algorithm of section 3 applies. The transition to such a field, however, would require to change the current practice of building the codes in the fields of characteristic 2. For this reason and even more so because, as mentioned, n is rather small in practice of decoding, our approach seems to be of purely theoretical interest so far. We believe, however, that our section 3 does provide some new insight into the solution of the key equation.

The same can be said about our algorithms that handle the cases where $2 \leq c \leq n$. In these cases, the 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 of performing $O(\mu(n)) = O((n \log n) \log\log n)$ 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 $\sqrt{n^2/\mu(n)}$, 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, on $c-1$ processors; such a solution requires no data exchange among the processors.

Even without using parallelism, however, our deterministic reduction to $c-1$ Padé problems means acceleration by roughly 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, 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. [P99]), which is a fundamental computation in linear algebra.

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$\cdot$POWER$\cdot$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 this study to devise and analyze new asymptotically faster KEY $\cdot$ 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 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 improve the original draft of the paper.

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

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

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

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 I $\cdot$ POWER $\cdot$ 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. $\square$

Next, we will restate the problem I $\cdot$ POWER $\cdot$ SUMS(min) in the form closely related 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 $s_1, \dots, s_{2n}$, as the problem KEY $\cdot$ SOLVE(n), which is clearly equivalent to I $\cdot$ POWER $\cdot$ SUMS(min).

Remark 2.1. Assuming that the zeros $z_1, \dots, z_n$ are indeterminates in (2.1), one may prove that there are exactly n algebraically independent power sums s_k of (2.2) among $s_1, \dots, s_{m+n}$, 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.

3 A KEY·SOLVER over the fields of characteristics 0 or greater than n

Let us write $-T(x) = (\ln P(x))' = \frac{P'(x)}{P(x)}$, $S(x) = \sum_{k=1}^{\infty}(s_k/k)x^k$ and deduce from (2.3) that $S'(x) = T(x)$ and $(\log P(x))' = -S'(x)$. Over the fields of characteristic $c = 0$, we may integrate the latter equation and obtain that

$$F_P(x) = \ln P(x) + S(x) = 0 \tag{3.1}$$

(cf. [BP94], Eq-n (1.4.10), page 34). (2.6) gives us a starting point for the solution of equation (3.1) by Newton's iteration, which recursively computes all the remaining n coefficients of $P(x)$ in $O(\mu(n))$ ops for $\mu(n) = O((n \log n) \log\log n)$ of (1.1) [Br75]. For $c > 0$, we may define $1/k$ for $0 < k < c$ and $S(x) \bmod x^c$. Then we may apply Newton's iteration in order to compute $P(x) \bmod x^h$ at the cost $O(\mu(h))$ for any $h \leq c$. In particular, if $c > n$, we choose $h = n+1$ and obtain $P(x) \bmod x^{n+1}$ in $O(\mu(n))$ ops.

We will next specify Newton's iteration in some detail, to prepare also its subsequent extension to the case where $0 < c \leq n$. We write $\gamma = \lceil \log_2(n+1) \rceil$,

$$R_h(x) = P(x) \bmod x^h \ , \quad R'_h(x) = P'(x) \bmod x^h \ , \quad h = 2, 4, \ldots, 2^\gamma, \tag{3.2}$$

and observe that $2^\gamma \geq n+1$ and $R_{2^\gamma}(x) = P(x) \bmod x^{2^\gamma} = P(x)$. Then we obtain from (2.3) that

$$\frac{R'_{2h}(x)}{R_{2h}(x)} = -\sum_{k=0}^{2h-1} s_{k+1}x^k \bmod x^{2h} \ . \tag{3.3}$$

We also observe that

$$R_2(x) = P(x) \bmod x^2 = 1 - s_1 x \ . \tag{3.4}$$

Now, by following [Br75] (cf. also [Sc82], [P90], [BP94], pages 34–35; and [P96]), we recursively compute $R_4(x), R_8(x), \ldots, R_{2^\gamma}(x)$. Namely, by applying Newton's iteration to the equation $F_p(x) = 0$ of (3.1) and observing that $F'_P(x) = 1/P(x)$, we obtain that

$$R_{2h}(x) = R_h(x) - F_{R_h}(x)/F'_{R_h}(x) = R_h(x)(1 - \ln R_h(x) - S(x)) \bmod x^{2h}$$

(cf. [Br75]). Rewrite the latter equation as follows:

$$R_{2h}(x) = R_h(x)M_h(x) \bmod x^{2h} \ , \tag{3.5}$$

where $M_h(x) = 1 - \ln R_h(x) - S(x) \bmod x^{2h}$. Deduce that

$$M_h(x) = 1 - \sum_{i=h}^{2h-1} m_{h,i} x^i \ , \tag{3.6}$$

$$-M_h'(x) = \frac{R_h'(x)}{R_h(x)} + S'(x) \bmod x^{2h-1} \ ,$$

that is,

$$-M_h'(x) = \frac{R_h'(x)}{R_h(x)} + \sum_{k=0}^{2h-2} s_{k+1} x^k \bmod x^{2h-1} \ . \tag{3.7}$$

Note that the relations (3.5)–(3.7) depend on $T(x) = S'(x)$ (but not on $S(x)$) and hold for any c, even where $S(x)$ is not defined. (One may ignore (3.1), introduce (3.5) and (3.6) for the unknown parameters $m_{h,i}$, $i = h, \ldots, 2h-1$, and obtain (3.7) via the division modulo x^{2h-1} of the two polynomials, that is, of $R_{2h}'(x) = R_h'(x)M_h(x) + R_h(x)M_h'(x) \bmod x^{2h-1}$ by $R_{2h}(x) = R_h(x)M_h(x) \bmod x^{2h}$ (cf. e.g. [BP94], pages 34–35).)

Now, suppose again that $c = 0$ or $c > 2^\gamma$. Combine (3.4) and (3.7) for $h = 2$ in order to compute $M_2'(x)$. Integrate $M_2'(x)$ and obtain $M_2(x)$. Substitute $R_2(x)$ and $M_2(x)$ into (3.5) for $h = 2$ and obtain $R_4(x)$. This completes the first step of a recursive process. At its i-th recursive step, for $i = 1, 2, \ldots, \gamma - 1$, we assume the input $R_h(x)$, $s_1, \ldots, s_{2h-1}$, for $h = 2^i$, and successively compute $M_h'(x)$ of (3.7), $M_h(x)$ (by means of the integration of $M_h'(x)$), and $R_{2h}(x)$ of (3.5) (by means of polynomial multiplication modulo x^{2h}). We arrive at $P(x) = R_{2^\gamma}(x)$ for $i = \gamma - 1$.

For each h, $h = 2, 4, \ldots, 2^{\gamma-1}$, the computation is reduced essentially to two polynomial multiplications modulo x^{2h-1} and x^{2h}, respectively, and to computing the reciprocal of a polynomial modulo x^{2h-1}. Furthermore, since $\frac{1}{R_{2g}(x)} = \frac{1}{R_g(x)} \bmod x^g$, we may partly reuse each computed reciprocal at the next recursive step, thus performing only two steps of Sieveking-Kung's version of Newton's iteration per reciprocal. Summarizing, we only need $O(\mu(h)) = O((h \log h) \log\log h)$ ops at the h-th step and a total of $O(\mu(c)) = O((c \log c) \log\log c)$ ops, at all steps (see appendix A for further details and more specific complexity estimates).

Now assume that $2^{\gamma-1} \le n < c$ and let us try to apply the same process in order to compute $R_{2h}(x)$ for $h = 2^{\gamma-1}$. Then, if $2h > c$, the algorithm will stumble at the integration stage. Indeed, if c divides i and if $2h > i \ge h$, then we cannot recover the coefficient $m_{h,i}$ from $M_h'(x)$ because $(x^{gc})' = 0 \bmod c$ for all integers g. The algorithm, however, recovers $M_h(x) \bmod x^c$ and, consequently, $R_c(x) = R_{2h}(x) \bmod x^c$, remaining within the same asymptotic cost bound. Formally, we will proceed with the above recursive computation of $R_h(x)$

until we arrive at $R_h(x)$ for $h \le n < 2h$; then we will perform modulo x^{n+1} its single step, to obtain $M_h(x) \bmod x^{n+1}$ and, finally, $R_{2h}(x) \bmod x^{n+1} = R_{n+1}(x) = P(x)$, by using a total of $O(\mu(n)) = O((n \log n) \log\log n)$ ops.

Summarizing, we obtain the following theorem:

Theorem 3.1. *The problems KEY · SOLVE(n) and I · POWER · SUMS(min) can be solved by using $O(\mu(n)) = O((n \log n) \log\log n)$ ops over any field of characteristic $c = 0$ or $c > n$.*

4 Some definitions and auxiliary results for the study over any field

Fact 2.1 shows that over a field of a positive characteristic we have an infinite set of polynomials $P(x)$ satisfying (2.3) for a fixed sequence $s_1, s_2, \ldots$. The approach of the previous section does not give us any means for selecting a polynomial of the minimum degree in this set. To extend the solution of the key equation in the case of a field of an arbitrary characteristic, we need some additional techniques. We will start with recalling some definitions and auxiliary results.

We will write $\gcd(U_i(x), i = 1, \ldots, k)$ and $\mathrm{lcm}(U_i(x), i = 1, \ldots, k)$ to denote the *greatest common divisor (gcd)* and the *least common multiple (lcm)* of k polynomials $U_1(x), \ldots, U_k(x)$.

We write $\deg U(x)$ to denote the degree of a polynomial $U(x)$.

For a formal power series

$$A(x) = \sum_{i=0}^{\infty} a_i x^i \ , \tag{4.1}$$

its *Padé approximation table* (to which we will referred to as its *Padé table*) is defined as follows: for two nonnegative integers g and h, the (g, h)-th entry of such a table is represented by a nonzero pair of polynomials $U(x)$ and $V(x)$ satisfying the following relations:

$$U(x) - V(x)A(x) = 0 \bmod x^{g+h+1} \ , \tag{4.2}$$

$$\deg U(x) \le g, \quad \deg V(x) \le h \ . \tag{4.3}$$

Theorem 4.1 [Gr72]. *Over any field of constants, for any pair of nonnegative integers g and h and for any formal power series $A(x)$ of (4.1), there exist pairs of polynomials $U(x)$ and $V(x)$ satisfying (4.2) and (4.3); furthermore, the ratio $U(x)/V(x)$ is defined uniquely.*

Due to theorem 4.1, we may ensure uniqueness of the pair of $U(x)$ and $V(x)$ satisfying (4.2) and (4.3) if we require that the polynomials $U(x)$ and $V(x)$ have the *minimal degrees* and if we scale these polynomials to make the nonvanishing trailing coefficient of $V(x)$ equal to 1. We will cite the problem of computing the polynomials $U(x)$ and $V(x)$ satisfying the latter requirements as *PADÉ(min) of size* (g, h).

We have the following simple observation:

Fact 4.2. *If it is known that*

$$V(0) = 1 \tag{4.4}$$

for a solution $U(x)$, $V(x)$ to PADÉ(min), then such a solution is unique and satisfies the equation

$$\gcd(U(x), V(x)) = 1 \ . \tag{4.5}$$

Remark 4.1 (cf. [G86]). Generally, it is not possible to satisfy (4.5) without assuming (4.4).

Hereafter, we will only deal with the entries of Padé tables normalized so as to satisfy (4.4).

Theorem 4.3 (cf. [BGY80], [CK91]). *For two polynomials of degrees at most n, their gcd and lcm can be computed over any field of constants by using $O(\mu(n)\log n) = O((\log n)^2 n \log\log n)$ ops. Furthermore, $O(\mu(n)\log n) = O((\log n)^2 n \log\log n)$ ops suffice in order to compute, over any field of constants, a solution to PADÉ(min) of size (g,h) provided that $g + h = O(n)$.*

Now, we observe that the solution $(W(x),\ P(x))$ to the problem $\mathrm{KEY} \cdot \mathrm{SOLVE}(n)$ is also the solution $(U(x),\ V(x))$ to the associated problem PADÉ(min) of size (g, h), for $g = h = n$ and for the input power series $A(x) = -\sum_k s_k x^k$ (cf. (2.4)). (Note that both problems share the requirement for the minimization of the degrees of the output polynomial.)

corollary *Over any field of constants, the problem $KEY \cdot SOLVE(n)$ has a unique solution shared with a problem PADÉ(min) of size (n, n). Such a solution can be computed in $O(\mu(n)\log n)$ ops; furthermore, it satisfies the equation*

$$\gcd(W(x),\ P(x)) = 1. \tag{4.6}$$

We will conclude this section with the following result, which we will use in section 8.

Fact 4.4. *If (over a field $\mathbf{F}$ of constants having a positive characteristic c) a polynomial $T(x^c)$ divides $P(x)$, the solution polynomial to I·POWER·SUMS(min), then $T(y)$ is a constant.*

Proof. We recall that $U(x^c) = (U(x))^c$ for any polynomial $U(x)$ over a field of a positive characteristic c. Therefore, $T(x^c) = (T(x))^c$, and all the zeros of $T(x^c)$ are the c-fold zeros of $P(x)$. Such zeros make no impact on the power sums s_k of (2.2) computed over $\mathbf{F}$. Due to the assumption that the degree of $P(x)$ is minimum, we conclude that $T(y) = T(x^c)$ is a constant. $\Box$

5 A KEY · SOLVER over the fields of characteristic 2

Our next goal is to decrease by about 50% the size of the Padé entry involved in the solution of the problems I · POWER · SUMS(min) and KEY · SOLVE(n) in the important case of the computations over a field $\mathbf{F}$ of characteristic $c = 2$. Similar effect was achieved by a distinct technique in [Be68], ch. 7.6; our approach is conceptually simpler and also gives us a chance to introduce some useful techniques for our KEY · SOLVERs over any field. In the case where $c = 2$, we have

$$P'(x) = \sum_{i=0}^{n_1} p_{2i+1} y^i = P_1(y), \tag{5.1}$$

$$W(x) = P(x) + xP'(x) = \sum_{i=0}^{n_0} p_{2i} y^i = P_0(y), \tag{5.2}$$

$$P(x) = P_0(y) + xP_1(y), \tag{5.3}$$

where $y = x^2$,

$$\deg P_0(y) = n_0 = \lfloor n/2 \rfloor, \quad \deg P_1(y) = n_1 = \lfloor (n-1)/2 \rfloor, \quad n_0 + n_1 = n - 1. \tag{5.4}$$

Now, we recall that $-s_k = s_k$ for all k and that $-1 = 1$ since we work over a field $\mathbf{F}$ of characteristic 2, and we apply (5.1)–(5.3) in order to rewrite (2.4) as follows:

$$\frac{P_0(y)}{P_0(y) + xP_1(y)} = \sum_{k=0}^{\infty} s_k x^k,$$

$$1 + \frac{xP_1(y)}{P_0(y)} = 1 / \sum_{k=0}^{\infty} s_k x^k,$$

$$\frac{P_1(y)}{P_0(y)} = (1 + 1/\sum_{k=0}^{\infty} s_k x^k)/x = \sum_{i=0}^{\infty} b_i y^i. \tag{5.5}$$

The following algorithm (over fields of characteristic 2) relies on the equation (5.5) and yields the desired size decrease.

Algorithm 5.1, a KEY · SOLVER for $c = 2$.

Input: the first $2n - 1$ power sums, $s_1, \ldots, s_{2n-1}$ of (2.2).

Output: the coefficients of the polynomials $P(x)$ of (2.1) and $W(x)$ of (2.5) and (5.2).

Computations:

1. Compute the coefficients of the polynomial

$$B(y) = (x^{-1}((\sum_{k=0}^{2n-1} s_k x^k)^{-1} + 1)) \bmod x^{2n-1} = \sum_{i=0}^{n-1} b_i y^i, \quad \text{for} \quad y = x^2.$$

2. Compute the $(n_1,\ n_0)$-th entry, $(P_{1,1}(y),\ P_{0,1}(y))$, of the Padé table for $B(y)$.

3. Compute and output the polynomial $P(x) = P_{0,1}(x^2) + xP_{1,1}(x^2)$.

4. Output the polynomial $W(x) = P_{0,1}(x^2)$.

Correctness of the algorithm immediately follows from the equations (5.1)–(5.5) and theorem 4.1.

Stage 1 amounts to computing modulo x^{2n-1} the reciprocal of a polynomial; this only requires $O(\mu(n))$ ops (see e.g. [BP94]). Stage 2 can be performed in $O(\mu(n)\log n)$ ops, due to theorem 4.3. Note that we only need to compute the (n_1, n_0)-th Padé entry for n_0 and n_1 of (5.4), where $n_1 \le n_0 \le n/2$, versus the computation of the $(n-1, n)$-th or (n, n)-th Padé entry, required in the solution of [BGY80], cited in section 4.

We will also show an alternative derivation of (5.5), which may be of some technical interest. Represent the formal power series $T(x) = -P'(x)/P(x)$ as

$$T(x) = T_0(y) + xT_1(y) \ , \tag{5.6}$$

$$y = x^2,\ T_0(y) = \sum_{k=0}^{\infty} s_{2k+1} y^k, \quad T_1(y) = \sum_{k=0}^{\infty} s_{2k+2} y^k. \tag{5.7}$$

Recall that

$$P'(x) = -P(x)T(x). \tag{5.8}$$

Differentiate both sides, observe that $P''(x) = 0$ over $\mathbf{F}$, and obtain that

$$P'(x)T(x) + P(x)T'(x) = 0.$$

Substitute (5.1), (5.3) and (5.6) into the latter equation, observe that $T'(x) = T_1(y)$, and obtain that $P_1(y)(T_0(y) + xT_1(y)) + (P_0(y) + xP_1(y))T_1(y) = 0$. Since $2P_1(y)T_1(y) = 0$ over $\mathbf{F}$, it follows that

$$P_1(y)T_0(y) + P_0(y)T_1(y) = 0.$$

Since $1 = -1$ in $\mathbf{F}$, we rewrite this equation as follows [cf. (5.5)]:

$$\frac{P_1(y)}{P_0(y)} = \frac{T_1(y)}{T_0(y)} = \sum_{i=0}^{\infty} b_i y^i. \tag{5.9}$$

Remark 5.1. Over a field of characteristic $c > 0$, we may differentiate (5.8) $c-1$ times and obtain $c-1$ equations in $P(x)$, $T(x)$ and its derivatives and higher order derivatives, which would enable us to express $P^{(k)}(x)/P^{(k-1)}(x)$ via $T(x), T'(x), \ldots, T^{(k)}(x)$. For instance, for $c = 3$, we deduce that $P''(x)/P'(x) = 2T'(x)/T(x) + T''(x)/T'(x)$.

6 Towards the reduction of KEY · SOLVE(n) to $c-1$ problems PADÉ(min) of sizes at most $(n/c, n/c)$ (preliminaries and an outline)

Extension of algorithm 5.1 to the case of a field $\mathbf{F}$ of any positive characteristic c is not straightforward. The idea is to reduce KEY · SOLVE(n) to computing the (g, h)-th entry of a Padé table of some formal power series $Q(x)$ (associated with $P(x)$ and c) for smaller g and h, not exceeding n/c. In this section, we will make the first step into this direction, by preparing the reduction to $c-1$ problems PADÉ(min), each of size at most $(n/c, n/c)$. We will start with the respective extension of the equations (5.1)–(5.9).

Represent $P(x)$ of (2.1) in the form

$$P(x) = \sum_{k=0}^{c-1} P_k(x^c)x^k \ , \tag{6.1}$$

where

$$P_k(y) = \sum_{i=0}^{n_k} p_{k+ic}y^i \ , \quad k = 0, 1, \ldots, c-1 \ , \tag{6.2}$$

$$n_0 = \lfloor n/c \rfloor \ , \quad n_0 - 1 \le n_{k+1} \le n_k \ , \quad k = 0, 1, \ldots, c-2 \ , \quad n = \sum_{k=0}^{c-1} n_k \ . \tag{6.3}$$

Observe that

$$P_0(0) = P(0) = p_0 = 1 \tag{6.4}$$

(cf. (2.1)). Define the formal power series,

$$Q(x) = \frac{P(x)}{P_0(x^c)} = \sum_{i=0}^{\infty} q_i x^i \ , \tag{6.5}$$

$$Q_k(y) = \frac{P_k(y)}{P_0(y)} = \sum_{i=0}^{\infty} q_{k+ic}y^i \ , \quad k = 1, 2, \ldots, c-1 \ , \tag{6.6}$$

and observe that

$$Q(x) = 1 + \sum_{k=1}^{c-1} x^k Q_k(x^c) \ . \tag{6.7}$$

(6.7) implies that

$$q_0 = 1 \ , \quad q_{jc} = 0 \ , \quad j = 1, 2, \ldots , \tag{6.8}$$

$$q_i = p_i \ , \quad i = 1, 2, \ldots , c-1 \ .$$

We will need some further properties of $P_k(x^c)$ and $Q(x)$. In particular, we note that

$$[P_k(x^c)]'_x = 0 \ \text{ for all } \ k, \quad P'(x) = \sum_{k=1}^{c-1} k x^{k-1} P_k(x^c) \tag{6.9}$$

over the field $\mathbf{F}$ (since $(x^c)'_x = 0 \bmod c$) and deduce that

$$Q'(x) = (P(x)/P_0(x^c))' = P'(x)/P_0(x^c) \ .$$

Consequently, $\frac{Q'(x)}{Q(x)} = \frac{P'(x)}{P(x)}$. Substitute (2.3) and obtain that

$$\frac{Q'(x)}{Q(x)} = -\sum_{k=0}^{\infty} s_{k+1} x^k \ . \tag{6.10}$$

Hereafter, we will write

$$\ell = (n_0 + n_1)c \leq 2c\lfloor n/c \rfloor \leq 2n \ . \tag{6.11}$$

Based on equations (6.1)-(6.11), we will reduce the solution of KEY·SOLVE(n) to performing the following stages:

a) Compute the coefficients of $Q(x) \bmod x^l$.

b) Solve $c-1$ problems PADÉ (min) of sizes (n_k, n_0) with inputs $Q_k(y)$, where $n_k \leq n_0 \leq n/c$, $k = 1, \ldots, c-1$.

c) Recover $P_0(y)$ from these solutions, then immediately recover $P_k(y)$ for $k = 1, \ldots, c-1$ from (6.6) and $P(x)$ from (6.1).

The above algorithm extends the solution from the easy case of $c = 2$, covered in the preceding section. For a general c we will cover stage a) in section 7, stages b) and c) in section 8. The main computational advantage of this approach versus the reduction of KEY·SOLVE(n) to a single problem PADÉ(min) of size (n, n) is the option of solving concurrently all the $c-1$ problems PADÉ(min) of smaller sizes. In section 9, we will simplify the computations even further, by applying randomization (or heuristics) to replace the bottleneck stage b) by the solution of a single problem PADÉ(min) of small size (n_1, n_0).

7 Efficient computation of the auxiliary polynomial $Q(x) \bmod x^{\ell+1}$

We are going to describe stage a) outlined in the previous section. The transition from $P(x)$ to $P(x)(1-xz)^c = P(x)(1-x^cz^c)$ for $c > 0$ changes $P_0(x)$ into $P_0(x)(1-x^cz^c)$ but leaves invariant the power sums $s_1, s_2, \dots$ (cf. the proof of fact 2.1) and the ratio $Q(x) = P(x)/P_0(x)$ of (6.5). Therefore, the power sums $s_1, s_2, \dots$ cannot uniquely define $P(x)$, but we will next show that they uniquely define $Q(x)$; moreover, we will extend the algorithm of section 3 to compute (over any field of a positive characteristic c) the unique ratio $Q(x)$ of (6.5) as soon as we are given the values $s_1, s_2, \dots$. To yield such an extension, we redefine $R_h(x)$ of (3.2) by writing $R_h(x) = Q(x) \bmod x^h$ for all h. Then (6.10) will enable us to apply Newton's process of section 3 to compute the first c coefficients of $Q(x)$ at a low computational cost. We will next extend this process, by using equations (6.8), to compute all other coefficients of $Q(x)$ recursively, at a low computational cost.

In [Sc93] this was done by means of the reduction to solving a triangular linear system of $\lfloor n/c \rfloor$ equations, which requires order of $(n/c)^2$ ops for n coefficients of $Q(x)$, in addition to $O(\mu(n)) = O((n \log n) \log\log n)$ ops needed for the reduction. We will follow [P96], where a total of $O(\mu(n))$ ops were used. At first, from equations (3.5) and (3.6) for $h \le c < 2h$, we deduce that

$$q_c = -\sum_{i=0}^{c-h} q_i m_{h,c-i} \ . \tag{7.1}$$

Now, substitute the equations $q_0 = 1$, $q_c = 0$ of (6.8) into (7.1) and deduce that

$$m_{h,c} = -\sum_{i=1}^{c-h} q_i m_{h,c-i} \ .$$

Similarly, from (3.5) and (3.6) for $h \le jc < 2h$, deduce the equations

$$q_{jc} = -\sum_{i=0}^{jc-h} q_i m_{h,jc-i}$$

(which turn into (7.1) for $j = 1$). Substitute $q_0 = 1$ and $q_{jc} = 0$ of (6.8) and arrive at the equations

$$m_{h,jc} = -\sum_{i=1}^{jc-h} q_i m_{h,jc-i} \ . \tag{7.2}$$

Observe that the coefficients $m_{h,jc-i}$ on the right-hand side are computed in the integration stage of the algorithm of section 3 (cf. (3.7)), except for those coefficients $m_{h,jc-i}$ for which c divides i. We will replace the latter coefficients by 0. This will not change equations (7.2) since $q_i = 0$ if $i > 0$ and if c divides i, due to (6.8).

Such a replacement turns the vector $\mathbf{m}_h = \{m_{h,jc}, j = 1,2,\ldots,g\}$, $g = \lfloor(2h-1)/c\rfloor$, into a subvector of the convolution vector for two known vectors, that is, $-(q_1,\ldots,q_{gc-h})$ and $(m^*_{h,h},\ldots,m^*_{h,gc-1})$, where $m^*_{h,i} = 0$ if c divides i, $m^*_{h,i} = m_{h,i}$ otherwise. $O(\mu(h)) = O((h\log h)\log\log h))$ ops suffice in order to compute such a convolution vector $\mathbf{m}_h$ [CK91]. By repeating the latter process recursively, for all h, $n \geq h = 2^\alpha \geq c/2$, we obtain the remaining coefficients of the polynomials $M_h(x)$ and $R_{2h}(x)$ for all h, at the cost of performing $O(\mu(n)) = O((n\log n)\log\log n)$ additional ops, and then we extend theorem 3.1 as follows:

Theorem 7.1. *Given the power sums $s_1,\ldots,s_\ell$, where $\ell = (n_0+n_1)c \leq 2n$ (cf. (2.1), (2.2), (6.11)), the coefficients of the polynomial $Q(x) \bmod x^{\ell+1}$ (cf. (6.1) – (6.5)) can be computed over any field of any positive characteristic c by using $O(\mu(n)) = O((n\log n)\log\log n)$ ops.*

8 Recovery of the output polynomial $P(x)$ from the auxiliary polynomial $Q(x) \bmod x^{\ell+1}$

The relations (6.1)–(6.11) imply that all the coefficients of the polynomials

$$Q_k(y) \bmod y^{n_k+n_0+1} = Q_k(x^c) \bmod x^{(n_k+n_0)c+1}\ ,\quad k = 1,2,\ldots,c-1\ , \tag{8.1}$$

are among the coefficients of the polynomial $Q(x) \bmod x^{\ell+1}$, computed by the algorithm of the previous section. The next algorithm enables us to recover the coefficients of the polynomials $P_k(y) = P_k(x^c)$ from the entries of Padé tables for $Q_k(y)$, for $k = 0,1,\ldots,c-1$, which will immediately define $P(x)$, due to (6.1).

Algorithm 8.1, a KEY $\cdot$ SOLVER for $c > 0$.

Input: integers $c \geq 2$, $n \geq 1$, $n_0, n_1, \ldots, n_{c-1}$, satisfying (6.3), a field $\mathbf{F}$ of characteristic c, the values $s_1,\ldots,s_\ell$, and the coefficients of the polynomial $Q_\ell(x) = Q(x) \bmod x^{\ell+1}$, for $Q(x)$ of (6.5)–(6.7) and for ℓ of (6.11), where $Q(x) \not\equiv 0$, $Q(x)$ is a formal power series satisfying (6.10) modulo $x^{\ell+1}$ for a given set $\{s_1,\ldots,s_\ell\}$.

Output: the coefficients of the solution polynomial $P(x)$ for KEY $\cdot$ SOLVE(n) and I $\cdot$ POWER $\cdot$ SUMS(min).

Computations:

1. Express the coefficients of all polynomials $Q_k(y) \bmod y^{n_k+n_0+1} = Q_k(x^c) \bmod x^{(n_k+n_0)c+1}$, for $k = 1,2,\ldots,c-1$, as the appropriate coefficients of $Q(x) \bmod x^{\ell+1}$ (cf. equations (6.5)–(6.7), (8.1)). Successively or concurrently, for $k = 1,2,\ldots,c-1$, compute the normalized

(n_k, n_0)-th entries $(P_{k,k}(y),\ P_{0,k}(y))$ of the Padé tables for $Q_k(y)$, such that $P_{0,k}(0) = 1$ (cf. (4.4) and (6.4)).

2. Compute

$$L_0(y) = \mathrm{lcm}(P_{0,k}(y)\ ,\quad k = 1, 2, \ldots, c-1)\ , \tag{8.2}$$

such that $L_0(0) = 1$.

3. Write

$$P_0(y) = L_0(y)\ . \tag{8.3}$$

Recall (6.6) and compute the polynomials

$$P_k(y) = Q_k(y)P_0(y) \mod y^{n_k+1}\ ,\quad k = 1, 2, \ldots, c-1\ . \tag{8.4}$$

Output the coefficients of the polynomial $P(x)$ of (6.1).

Let us prove *correctness* of algorithm 8.1, that is, let us deduce (8.3). The first step of the proof is to show that $L_0(y)$ divides $P_0(y)$. By the definition of Padé entries, we have $P_{k,k}(y)/P_{0,k}(y) = Q_k(y) \bmod y^{n_k+n_0+1}$, for all k. The latter equation, equation (6.6), and theorem 4.1 together imply that

$$P_{k,k}(y)/P_{0,k}(y) = P_k(y)/P_0(y) \quad \text{for all} \quad k$$

or, equivalently,

$$P_k(y) = P_{k,k}(y)P_0(y)/P_{0,k}(y) \quad \text{for all} \quad k. \tag{8.5}$$

Due to the normalization assumption that $P_{0,k}(0) = 1$, we have

$$\gcd(P_{k,k}(y), P_{0,k}(y)) = 1 \quad \text{for all} \quad k \tag{8.6}$$

(cf. fact 4.2). (8.5) and (8.6) together imply that $P_{0,k}(y)$ divides $P_0(y)$ for all k. Consequently, $L_0(y)$ divides $P_0(y)$ (cf. (8.2)).

Now, let $T(y)$ denote the quotient polynomial $P_0(y)/L_0(y)$. It remains to prove that $T(y)$ is identically 1. (8.2) implies that $T(y)$ divides $P_0(y)/P_{0,k}(y)$, for all k. Therefore, due to (8.5), $T(y)$ divides $P_k(y)$, for all k. Consequently, (6.1) implies that $T(x^c)$ divides $P(x)$. By the virtue of fact 4.4, $T(y)$ is a constant. Moreover, due to the normalization assumptions that $P_0(0) = L_0(0) = 1$, we have $T(0) = 1$ and, consequently, $T(y) \equiv 1$. Therefore, we arrive at (8.3) and (8.4). This completes the proof of correctness of algorithm 8.1. $\square$

9 Simplification of algorithm 8.1 by means of heuristics and randomization. Derandomization technique

In this section we will propose some heuristics and randomization to decrease (ideally to 1) the number $c-1$ of the problems PADÉ(min) whose solution was required at stage b) of the algorithm outlined in section 6. At the end, we will also comment on derandomization.

For $c = 2$, stage 1 of algorithm 8.1 is reduced to computing the single (n_1, n_0)-th entry of the Padé table for $Q_1(y)$, whereas stages 2 and 3 are not needed at all, since in this case, $L_0(y) = P_{0,1}(y)$ (cf. (8.2) for $c = 2$ and $k = 1$) and, consequently, $P_1(y) = P_{1,1}(y)$ (cf. (8.4) for $c = 2$, $k = 1$). The next observations will enable us to obtain a similar simplification for any $c \leq n$ by using randomization.

Fact 9.1.

$$P_0(y) = P_{0,k}(y) \ , \quad P_k(y) = P_{k,k}(y) \tag{9.1}$$

if

$$\deg P_{0,k}(y) = n_0 \quad and/or \quad \deg P_{k,k}(y) = n_k \ , \tag{9.2}$$

for some k, $1 \leq k < c$.

Proof. The implication of (9.1) by (9.2) immediately follows from (8.5)–(8.7) and the normalization assumption for the entries of Padé tables. □

For any fixed k, relations (9.2) hold for a large class of inputs to I · POWER · SUMS(min). This suggests the following *heuristic practical simplification* of algorithm 8.1: compute the pairs $(P_{0,k}(y), P_{k,k}(y))$ successively, for $k = 1, 2, \dots, c-1$, each time checking if (9.2) holds; if this is the case for a current value of k, skip stage 2, write $P_0(y) = P_{0,k}(y)$, and compute $P_k(y)$ for all k by using (8.4).

If (9.2) holds for some k, the computations are substantially simplified. In the worst case, (9.2) is never achieved, and we proceed as in the original algorithm 8.1, completing its both stages 1 and 2. The next (Las Vegas randomized) algorithm enables us to skip stage 2 and go to stage 3 already after computing the (n_1, n_0)-th entry of a single Padé table, for the formal power series $\sum_{k=1}^{c-1} a_k Q_k(y)$ where a_k are random scalars. This enables us to produce the pair $(\sum_{k=1}^{c-1} a_k P_k(y), P_0(y))$ with a high probability (cf. (8.4)). The algorithm includes verification that (unless FAILURE is output) the output polynomial $P(x)$ is indeed the solution of KEY · SOLVE(n) and I · POWER · SUMS(min).

Algorithm 9.1, a randomized KEY $\cdot$ SOLVER.

Input: the same as in algorithm 8.1 and, in addition, a finite set S of elements of the field $\mathbf{F}$ or of its algebraic extension $\mathbf{E}$.

Output: either the coefficients of the solution polynomial $P(x)$ for KEY $\cdot$ SOLVE(n) and I $\cdot$ POWER $\cdot$ SUMS(min) or FAILURE, with a probability at most $(n_0+1)/|S|$, where $|S|$ denotes the cardinality of the set S.

Computations:

0. Choose $c-1$ random values $a_1,\ldots,a_{c-1}$ from the set S, independently of each other and assuming the uniform probability distribution on S.

1. Compute the normalized (n_1,n_0)-th entry, $(\widetilde{P}_{\mathbf{a}}(y),\widetilde{P}_0(y))$, of the Padé table for the formal power series $Q_{\mathbf{a}}(y)=\sum_{k=1}^{c-1}a_kQ_k(y)$, such that $\widetilde{P}_0(0)=1$ (cf. (4.4) and (6.4)).

2. Compute $\widetilde{P}_k(y)$ for $k=1,\ldots,c-1$, by using (8.4), where $P_k(y)$ and $P_0(y)$ are replaced by $\widetilde{P}_k(y)$ and $\widetilde{P}_0(y)$, respectively. If $\deg\widetilde{P}_0(y)<n_0=\lfloor n/c\rfloor$ and if $\deg\widetilde{P}_k(y)>\deg\widetilde{P}_{\mathbf{a}}(y)$ for some $k\geq 1$, output FAILURE. Otherwise compute $\widetilde{P}(x)=\sum_{k=1}^{c-1}a_k\widetilde{P}_k(y)$ (cf. (6.1)).

3. If $\deg\widetilde{P}_0(y)=n_0$, output $P(x)=\widetilde{P}(x)$. Otherwise check if

$$\widetilde{P}'(x)+\widetilde{P}(x)T(x)=0 \bmod x^{2n+1} \tag{9.3}$$

for $T(x)=\sum_{k=0}^{\infty}s_{k+1}x^k$ (cf. (2.3)). If so, output $P(x)=\widetilde{P}(x)$. Otherwise, output FAILURE.

Let us prove *correctness* of the algorithm. Let $P(x)$ denote the solution to I $\cdot$ POWER $\cdot$ SUMS(min) for the input values $s_1,\ldots,s_l$ of algorithms 8.1 and 9.1 and let us show that $\widetilde{P}(x)=P(x)$ unless the algorithm outputs FAILURE. Indeed, unless the algorithm outputs FAILURE, the polynomial $\widetilde{P}(x)$ satisfies equation (9.3), and since $P(0)=\widetilde{P}(0)=1$, it remains to show that

$$\deg\widetilde{P}(x)\leq\deg P(x)\ . \tag{9.4}$$

We have

$$Q_{\mathbf{a}}(y)=P_{\mathbf{a}}(y)/P_0(y)=\widetilde{P}_{\mathbf{a}}(y)/\widetilde{P}_0(y) \bmod y^{n_0+n_1+1}\ ,$$

where $P_0(x),P_1(x),\ldots,P_{c-1}(x)$ denote the c polynomials satisfying equations (6.1)–(6.4), (8.4) and $P_{\mathbf{a}}(y)=\sum_{k=1}^{c-1}a_kP_k(y)$. Now, by theorem 4.1 and the definition of $\widetilde{P}_{\mathbf{a}}(y)$ and $\widetilde{P}_0(y)$ (at stage 1 of algorithm 9.1), we have

$$P_{\mathbf{a}}(y)/P_0(y)=\widetilde{P}_{\mathbf{a}}(y)/\widetilde{P}_0(y).$$

Fact 4.2 and the normalization assumptions $\widetilde{P}_0(0) = P_0(0) = 1$ together imply that $\widetilde{P}_0(y)$ divides $P_0(y)$ and $\widetilde{P}_{\mathbf{a}}(y)$ divides $P_{\mathbf{a}}(y)$; furthermore, it follows that

$$\widetilde{P}_0(y) = P_0(y) \ , \qquad \widetilde{P}_{\mathbf{a}}(y) = P_{\mathbf{a}}(y) \tag{9.5}$$

unless we have simultaneously

$$\deg P_0(y) > \deg \widetilde{P}_0(y) \ , \qquad \deg P_{\mathbf{a}}(y) > \deg \ \widetilde{P}_{\mathbf{a}}(y) \ . \tag{9.6}$$

If (9.5) holds, then (8.4) implies that $\widetilde{P}_k(y) = P_k(y)$ for all k, and, therefore, $\widetilde{P}(x) = P(x)$ (cf. (6.1)).

Now, suppose that (9.5) does not hold. Then (9.6) holds, and it follows that

$$\max_{k\geq 0} \ \deg \ P_k(y) > \max_{k\geq 0} \ \deg \ \widetilde{P}_k(y)$$

unless FAILURE has been output at stage 2 of algorithm 9.1. The latter inequality, the definition of $\widetilde{P}(x)$ as $\sum_{k=1}^{c-1} a_k \widetilde{P}_k(y)$ at stage 2 of algorithm 9.1, and equation (6.1) combined imply (9.4) also in this case.

To complete the correctness proof, let us estimate the failure probability for algorithm 9.1. Due to the definition of $\widetilde{P}_k(x)$ and $\widetilde{P}(x)$, theorem 4.1, and fact 4.2, we arrive at the equations

$$P(x) = \widetilde{P}(x) \tag{9.7}$$

and

$$P_k(x) = \widetilde{P}_k(x) \quad \text{for } , k = 0, \ldots, c-1, \tag{9.8}$$

provided that

$$\gcd(P_{\mathbf{a}}(y), P_0(y)) = 1 \ . \tag{9.9}$$

Surely, under (9.7) we cannot have failure at stage 3 of algorithm 9.1. On the other hand, (9.9) holds unless the resultant R of the two polynomials $P_{\mathbf{a}}(y)$ and $P_0(y)$ vanishes [vdW53]. Therefore, failure at stage 3 implies that R vanishes.

Let us estimate the probablitity that R vanishes for a fixed vector $\mathbf{a}$. We will rely on the following fundamental result, due to [DL78] though more frequently cited from [S80], [Z79].

Theorem 9.1 *[DL78], [S80], [Z79]. Let $p(X) = p(x_1, \ldots, x_m)$ be a nonzero m-variate polynomial of a total degree d. Let S be a finite subset of cardinality $|S|$ in the domain of the definition of $p(X)$ and let $X^* = \{x_1^*, \ldots, x_m^*\}$ be a point in S^m, whose coordinates $x_1^*, \ldots, x_m^*$*

have been chosen at random in S, independently of each other and under the uniform probability distribution on S. Then

$$\text{Probability}\{p(X^*) = 0\} \leq d/|S| \ .$$

We recall that the resultant R is a polynomial of a degree at most n_0 in the coefficients of $P_{\mathbf{a}}(y)$ (cf. e.g. [BP94]). Therefore, R is a polynomial of a degree at most n_0 in $a_1, \ldots, a_{c-1}$. This polynomial does not vanish identically in $a_1, \ldots, a_{c-1}$. Indeed, we surely have

$$\gcd(P_0(y), P_1(y), \ldots, P_{c-1}(y)) = 1 \ ,$$

for otherwise $P(x)$ would not be the solution to I$\cdot$POWER$\cdot$SUMS(min), that is, the minimum degree solution of (2.3) (cf. (6.1) and (6.9)).

Now, the upper bound $n_0/|S|$ on the probability of failure at stage 3 follows from theorem 9.1. Also, under (9.8), failure at stage 2 would have implied a nondegenerating linear equation in $a_1, \ldots, a_{c-1}$. Therefore, theorem 9.1 immediately implies the upper bound $1/|S|$ on the probability of failure at stage 2. This gives us the desired estimate for the overall probability of failure of algorithm 9.1 and, consequently, completes the entire proof of its correctness. $\square$

If algorithm 9.1 outputs FAILURE (and this occurs with a probability of at most $\delta = (n_0 + 1)/|S|$, which is small for large $|S|$), we may generate $(c-1)j$ random parameters and apply the same algoirthm j times, to dercease the FAILURE probability to or below δ^j. Alternatively, we may reuse the same $c-1$ parameters. For instance, we may adopt the following policy: at the i-th application of algorithm 9.1, for $i = 1, \ldots, c-1$, replace $Q_{\mathbf{a}}(y) = Q_{\mathbf{a}(1)}(y)$ by $Q_{\mathbf{a}(i)}(y) = \sum_{k=1}^{c-1} a_{k+i-1} Q_k(y)$, $i = 1, \ldots, c-1$, $a_h = 0$ for $h \geq c$. With a high probability, the algorithm outputs $P(x)$ for some $i \leq c-1$ (rather than FAILURE for all i), but even if FAILURE has been output for all i, we may compute the polynomial $P_0(y)$ as $\mathrm{lcm}(\widetilde{P}_{0,1}(y), \ldots, \widetilde{P}_{0,c-1}(y))$, where $(\widetilde{P}_{\mathbf{a}(i)}(y), \widetilde{P}_{0,i}(y))$ is the (n_1, n_0)-th entry of the Padé table for $Q_{\mathbf{a}(i)}(y)$; then we may yield *derandomization*, that is, obtain $P(x)$ deterministically, by using (8.4) and (6.1). The correctness proof for the latter *deterministic algorithm* (to which we refer as **algorithm 9.2**) is similar to one for algorithm 8.1, since for every k, $k = 1, \ldots, c-1$, one may express $P_1(y), \ldots, P_k(y)$ linearly via $P_{\mathbf{a}(1)}(y), \ldots, P_{\mathbf{a}(c-1)}(y)$, and vice versa.

10 Deterministic and randomized computational cost estimates

Let us next deduce deterministic complexity estimates for the problems I$\cdot$POWER$\cdot$SUMS(min) and KEY$\cdot$SOLVE(n). The case where $c = 0$ or $c > n$ is covered by theorem 3.1. Let $0 < c \leq n$.

Then the computational cost of performing stages 1 and 2 of algorithm 8.1 is immediately estimated based on theorem 4.3. Stage 3 amounts to $c-1$ multiplications of polynomials having degrees at most $\lfloor n/c \rfloor$. Such multiplications require a total of $O(c\mu(n/c)) = O(\mu(n)) = O((n\log n)\log\log n)$ ops for $0 < c \leq n$ [CK91]. Summarizing and taking into account the equivalence of the computational problems KEY $\cdot$ SOLVE(n) and I $\cdot$ POWER $\cdot$ SUMS(min) to each other, theorems 3.1 and 7.1, and simplifications implied by (9.2), we arrive at the next result:

Theorem 10.1. *The problems I $\cdot$ POWER $\cdot$ SUMS(min) and KEY $\cdot$ SOLVE(n) can be solved over a field of a characteristic c by using $O(\mu(n)) = O((n\log n)\log\log n)$ ops if $c = 0$ or $c > n$ and by using $O(\mu(n) + c\mu(n/c)\log(n/c)) = O((n\log n)\log\log n + (\log(n/c))^2 n\log(1+\log(n/c))))$ ops for $0 < c \leq n$. The latter bound decreases to $O(\mu(n) + k\mu(n/c)\log(n/c)) = O((n\log n)\log\log n + k(\log(n/c))^2(n/c)\log(1+\log(n/c)))$ ops in the case where (9.2) holds for some k, $1 \leq k < c$.*

Similar analysis of algorithm 9.1 leads us to the following result:

Theorem 10.2. *The problems I$\cdot$POWER$\cdot$SUMS(min) and KEY$\cdot$SOLVE(n) can be solved over a field of a characteristic c, $0 < c \leq n$, by means of a randomized Las Vegas algorithm, which generates $c-1$ parameters chosen at random (independently of each other) from a fixed finite set S (under the uniform probability distribution on S) and in addition involves*

$$O(\mu(n) + \mu(n/c)\log(n/c)) = O((n\log n)\log\log n + (\log(n/c))^2(n/c)\log(1+\log(n/c)))$$

ops. The algorithm fails with a probability at most $(1 + \lfloor n/c \rfloor)/|S|$, where $|S|$ denotes the cardinality of the set S.

Remark 10.1. Based on algorithm 9.2, one may combine the results of theorems 10.1 and 10.2, to yield deterministic solution of the key equation (at the price of performing some additional computations) in the unlikely event of the failure of the probabilistic algorithm.

REFERENCES

Be68 E. R. Berlekamp, *Algebraic Coding Theory,* McGraw-Hill, New York, 1968.

BGY80 R. P. Brent, F. G. Gustavson, D. Y. Y.Yun, Fast Solution of Toeplitz Systems of Equations and Computation of Padé Approximations, *J. of Algorithms*, 1, 259-295, 1980.

BP94 D. Bini, V.Y. Pan, *Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms*, Birkhäuser, Boston, 1994.

Br75 Br75 R. P. Brent, Multiple-precision Zero-finding Methods and the Complexity of Elementary Function Evaluation, *Analytic Computational Complexity*, J. F. Traub, Ed., Academic Press, New York, 151–176, 1975.

CK91 D. G. Cantor, E. Kaltofen, On Fast Multiplication of Polynomials over Arbitrary Rings, *Acta Inf.*, 28, 7, 697–701, 1991.

DL78 R.A. Demillo, R.J. Lipton, A Probabilistic Remark on Algebraic Program Testing, *Information Process. Letters*, 7, 4, 193–195, 1978.

Gat86 J. von zur Gathen, Representation and Parallel Computations for Rational Functions, *SIAM J. on Computing,* 15, 432-452, 1986.

Gr72 W. B. Gragg, The Padé Table and Its Relation to Certain Algorithms of Numerical Analysis, *SIAM Review*, 14, 1, 1–62, 1972.

K25 S. Kakeya, On Fundamental Systems of Symmetric Functions, *Japan J. of Math.*, 2, 69–80, 1925.

K27 S. Kakeya, On Fundamental Systems of Symmetric Functions, *Japan J. of Math.*, 4, 77–85, 1927.

K-G W. Keller-Gehrig, Fast Algorithms for Characteristic Polynomial, *Theoretical Computer Science*, **36**, 309-317, 1985.

Man77 D. M. Mandelbaum, A Method for Decoding of Generalized Goppa Codes, *IEEE Trans. on Information Theory,* 23, 137-140, 1977.

Mas69 J. L. Massey, Shift -Register Synthesis and BCH Decoding, *IEEE Trans. on Information Theory,* 15, 122-127, 1969.

Mi75 W. H. Mills, Continued Fractions and Linear Recurrences, *Math. of Computation,* 29,173-180, 1975.

Mo73 R. Moenck, Fast Computation of GCDs, *Proc. 5th Ann. ACM Symp. on Theory of Computing* , 142–151, ACM Press, New York, 1973.

N27 K. Nakamura, On the Representation of Symmetric Functions by Power-Sums Which Form the Fundamental System, *Japan J. Math.*, 4, 87–92, 1927.

P90 V.Y. Pan, Parallel Least-Squares Solution of General and Toeplitz-like Linear Syatems, *Proc. 2nd Ann. ACM Symp. on Parallel Algorithms and Architectures*, 244–253, ACM Press, New York, 1990.

P96 V. Y. Pan, Parallel Computation of Polynomial GCD and Some Related Parallel Computations over Abstract Fields, *Theoretical Computer Science*, 162, 2, 173-223, 1996.

P97 V. Y. Pan, Fast Solution of the Key Equation for Decoding the BCH Error-Correcting Codes, *Proc. 27th Ann. ACM Symp. on Theory of Computing*, 168-175 ACM Press, New York, 1997.

P99 V. Y. Pan, New Deterministic Parallel Algorithms for the Characteristic Polynomial of a Matrix over Abstract Fields, preprint, 1999.

Pat75 N. J. Patterson, The Algebraic Decoding of Goppa Codes, *IEEE Trans. on Information Theory,* 21, 203-207, 1975.

S75 M. K. Sain, Minimum Torsion Spaces and the Practical Input/Output Problem, *Information and Control,* 29, 103-124, 1975.

Sa77 D. V. Sarwate, On the Complexity of Decoding Goppa Codes, *IEEE Trans. on Information Theory,* 23, 515–516, 1977.

S80 J. T. Schwartz, Fast Probabilistic Algorithms for Verification of Polynomial Identities, *J. of ACM*, 27, 4, 701-717, 1980.

Sc82 A. Schönhage, The Fundamental Theorem of Algebra in Terms of Computational Complexity, manuscript, *Dept. of Math., University of Tübingen*, Tübingen, Germany, 1982.

Sc93 A. Schönhage, Fast Parallel Computation of Characteristic Polynomials by Leverrier's Power Sum Method Adapted to Fields of Finite Characteristic, *Proc. 20th ICALP, Lecture Notes in Computer Science*, 700, 410-417, Springer, Berlin, 1993.

SKHN75 Y. Sugiyama, M. Kasahara. S. Hirawawa, T. Namekawa, A Method for Solving Key Equation for Decoding Goppa Codes, *Information and Control,* 27, 87-99, 1975.

V900 K. T. Vahlen, Über Fundamentalsysteme für symmetrisch Functionen, *Acta Mathematica*, 23, 91–120, 1900.

vdW53 B. L. van der Waerden, *Modern Algebra*, Frederick Ungar Publishing Co., New York, 1953.

Y76 D. Y. Y. Yun, On Square-free Decomposition Algorithms, *Proc. ACM Symp. on Symb. and Algebraic Comp.*, 26–35, ACM Press, New York, 1976.

Z79 R. E. Zippel, Probabilistic Algorithms for Sparse Polynomials, *Proc. EUROSAM 79, Lecture Notes in Computer Science*, 72, 216-226, Springer, Berlin, 1979.

A Application and extension of the two-step Newton's iteration for polynomial reciprocals

The h-th recursive step of the algorithm of section 3, $h = 1, \dots, \lfloor \log_2(n+1) \rfloor - 1$, is reduced essentially to two polynomial multiplications modulo x^{2g-1}, for the transition from $(1/R_{2g}(x))$ mod x^{g-1} to $(1/R_{2g}(x))$ mod x^{2g-1}, and to two other polynomial multiplications modulo x^{2g-1} and x^{2g}, respectively, for $g = 2^h$. This means that $(8/3)n^2 + O(n)$ ops suffice at all steps, provided that $n+1$ is a power of 2 and that all the polynomial products are computed by the classical method. We will next estimate the computational cost of a recursive step of the same algorithim of section 3 assuming fast (FFT based) polynomial arithmetic.

We will reuse the results of the preceding computations, in particular, when we apply Sieveking-Kung's algorithm, which specifies Newton's iteration to computing polynomial reciprocals (cf. [BP94], page 22). The next subalgorithm incorporates the relevant steps of this iterative algorithm into the i-th recursive step of our algorithm of section 3. In the description of this subalgorithm, we let Ω_k denote the set of the k-th roots of 1, let $g = 2^{i-1}$, so that $2g = 2^i$, and assume that the set Ω_{8g} is available.

Subalgorithm A.1.

Input: the values $s_1, \ldots, s_{4g-1}$, the coefficients of the polynomial $R_{2g}(x)$ and $T_g(x) = -(1/R_g(x)) \bmod x^g$, the set Ω_{8g}, and the values of $T_g(x)$ on the subset Ω_{4g} of Ω_{8g}.

Output: the coefficients of the polynomials $R_{4g}(x)$ and $T_{2g}(x) = -(1/R_{2g}(x)) \bmod x^{2g}$ and the values of the polynomial $T_{2g}(x)$ on the set Ω_{8g}.

Computations:

1. Evaluate $R_{2g}(x)$ and $R'_{2g}(x)$ on the set Ω_{8g}, which includes the evaluation on the subset Ω_{4g} of Ω_{8g}.

2. Evaluate $\bar{T}_{2g}(x) = T_g(x)(2 + R_{2g}(x)T_g(x))$ on the set Ω_{4g}.

3. Compute the coefficients of the polynomials $\bar{T}_{2g}(x)$ (interpolation) and $T_{2g}(x) = \bar{T}_{2g}(x) \bmod x^{2g}$.

4. Evaluate $T_{2g}(x)$ on the set Ω_{8g}.

5. Evaluate $\bar{M}'_{2g}(x) = R'_{2g}(x)T_{2g}(x)(2 + R_{2g}(x)T_{2g}(x))$ on the set Ω_{8g} (by using the results of steps 1 and 4).

6. Compute the coefficients of the polynomials $\bar{M}'_{2g}(x)$ (interpolation) and $\bar{M}'_{2g}(x) \bmod x^{4g-1}$.

7. Compute the coefficients of the polynomials $M'_{2g}(x) = \bar{M}'_{2g}(x) \bmod x^{4g-1} + \sum_{k=0}^{4g-2} s_{k+1}x^k$ and $M_{2g}(x) = 1 + x^{2g}M^*_{2g}(x)$.

8. Evaluate $M^*_{2g}(x)$ on the set Ω_{4g}.

9. Evaluate $R_{2g}(x)M^*_{2g}(x)$ on the set Ω_{4g} (by using the results of steps 1 and 8).

10. Compute the coefficients of the polynomial $R_{2g}(x)M^*_{2g}(x)$ (interpolation).

11. Compute the coefficients of the polynomial $\bar{R}_{4g}(x) = R_{2g}(x) + x^{2g}R_{2g}(x)M^*_{2g}(x) = R_{2g}(x)M_{2g}(x)$ and $R_{4g}(x) = \bar{R}_{4g}(x) \bmod x^{4g}$.

Correctness of this algorithm follows from equations (3.4)–(3.7) and $T_h(x)(2+R_{2h}(x)T_h(x)) = -(1/R_{2h}(x)) \bmod x^{2h}$, for $h = g$, $h = 2g$ (the latter equation defines Newton's iteration for the reciprocal (cf. e.g. [BP94], page 22)).

To estimate the computational cost, let $C_{F(k)}$ denote the computational cost of the discrete Fourier transform on the set Ω_k. We have

$$C_{F(k)} \leq 1.5k \log_2 k = 3i2^{i-1}, \quad for\ k = 2^i. \tag{A.1}$$

The overall computational cost bound is dominated by the sum of the cost bounds $3C_{F(8g)} + C_{F(4g)}$, for performing four forward FFTs on stages 1, 4 and 8 (for polynomial evaluation on Ω_{8g} and Ω_{4g}), and $2C_{F(4g)} + C_{F(8g)}$, for performing three inverse FFTs (for polynomial interpolation on Ω_{4g} and Ω_{8g}). The other stages only require $O(g)$ ops. The overall cost of performing subalgorithm A.1 is, therefore, bounded by

$$3C_{F(4g)} + 4C_{F(8g)} + O(g) = 66g \log_2 g + O(g)$$

(cf. (A.1)), which only slightly exceeds the cost of multiplying two pairs of polynomials modulo x^{2g} and x^{4g}, respectively.

Summarizing all applications of the subalgorithm during the computations of section 3 and assuming that $n + 1$ is a power of 2, we arrive at the overall cost bound of $66n \log_2 n + O(n)$ ops.