A Displacement Approach to Efficient Decoding of Algebraic-Geometric Codes

Vadim Olshevsky
Department of Mathematics
Georgia State University

M. Amin Shokrollahi
Department of Fundamental Mathematics
Bell Labs

Abstract

Using methods originating in numerical analysis, we will develop a unified framework for derivation of efficient list decoding algorithms for algebraic-geometric codes. We will demonstrate our method by accelerating Sudan's list decoding algorithm for Reed-Solomon codes [22], its generalization to algebraic-geometric codes by Shokrollahi and Wasserman [21], and the recent improvement of Guruswami and Sudan [8] in the case of Reed-Solomon codes.

The basic problem we attack in this paper is that of efficiently finding nonzero elements in the kernel of a *structured matrix*. The structure of such an $n \times n$- matrix allows it to be "compressed" to αn parameters for some α which is usually a constant in applications. The concept of structure is formalized using the displacement operator. The displacement operator allows to perform matrix operations on the compressed version of the matrix. In particular, we can find a PLU-decomposition of the original matrix in time $O(\alpha n^2)$, which is quadratic in n for constant α.

We will derive appropriate displacement operators for matrices that occur in the context of list decoding, and apply our general algorithm to them. For example, we will obtain algorithms that use $O(n^2\ell)$ and $O(n^{7/3}\ell)$ operations over the base field for list decoding of Reed-Solomon codes and algebraic-geometric codes from certain plane curves, respectively, where ℓ is the length of the list. Assuming that ℓ is constant, this gives algorithms of running time $O(n^2)$ and $O(n^{7/3})$, which is the same as the running time of conventional decoding algorithms. We will also sketch methods to parallelize our algorithms.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
STOC '99 Atlanta GA USA
Copyright ACM 1999 1-58113-067-8/99/05...$5.00

1 Introduction

Matrices with different patterns of structure are often encountered in the context of coding theory. Examples include Hankel, Vandermonde, and Cauchy matrices which arise in the Berlekamp-Massey algorithm [2], Reed-Solomon codes, and classical Goppa codes [14], respectively. In most of these applications one is interested in a certain nonzero element in the (right-)kernel of the matrix. This problem has been solved efficiently for each of the above cases. Although it is obvious that these algorithms make use of the structure of the underlying matrices, this exploitation is often rather implicit, and limited to the particular pattern of structure.

In this paper, we apply an alternative general method, called the method of displacement, for efficiently computing a PLU-decomposition of a structured matrix. Though this method has been successfully used in other contexts such as image processing, system theory, or interpolation (see the surveys in [9, 13, 17]), its use in coding theory is novel and quite powerful, as we will see below. Its power stems from the fact that it enables one to perform matrix operations on "compressed" versions of a structured matrix, rather than on the matrix itself. One of the main characteristics of a structured $n \times n$-matrix is that its n^2 entries are

functions of only $O(n)$ parameters. In many situations, these functions are rather simple, so that it makes sense to say that the matrix can be compressed to $O(n)$ parameters. Now note that basic operations on matrices such as Gaussian elimination use $O(n^3)$ operations because they update the entries of the matrix $O(n)$ times. Here and in the sequel, an "operation" denotes one the four fundamental arithmetic operations in the base field. The "running time" of an algorithm is to be understood as the number of operations it performs. The question is thus whether it is possible to perform Gaussian elimination on the "compressed" version of the matrix, thereby reducing the running time of the algorithm to something close to $O(n^2)$. As was realized by Morf [15, 16] and independently by Bitmead-Anderson [4], the displacement idea allows for a concise derivation of exactly such an algorithm (though limited at that time to a special "Toeplitz-like" structure). The details of the general algorithm that applies to all the above mentioned special structured matrices (Hankel, Vandermonde, Cauchy, etc.) will be carried out in the next section. Using the general method of displacement, we show how to compute a PLU-decomposition of various structured matrices occurring in coding theory in time $O(n^2)$. Noting that an element in the kernel of U is then obtained in time $O(n^2)$, this gives an algorithm of overall running time $O(n^2)$.

To demonstrate the power of our method, we will derive solutions to various decoding problems concerning Reed-Solomon- and algebraic-geometric codes. These codes arguably form one of the most powerful known classes of linear codes. They are constructed by evaluation of certain functions on an irreducible curve at some points of the curve. The simplest case is provided by Reed-Solomon (RS-) codes where one evaluates polynomials of some bounded degree at n distinct elements of the base field: these polynomials and the elements of the field can be regarded as functions and points on the projective line, respectively. The power of AG-codes comes from the enormous amount of freedom one has in constructing them. In particular, using sophisticated sequences of curves over a finite field related to modular curves, one can obtain explicit sequences of codes that surpass the Gilbert-Varshamov bound.

The last ten years have witnessed major developments with respect to the decoding problem for AG-codes [11]. As far as conventional decoding goes, one of the best algorithms is that of Feng-Rao [5] which decodes AG-codes up to the designed error-correction bound (and sometimes beyond). A more practical version of this algorithm is given in [20].

A basic shortcoming of all conventional decoding algorithms is that their outcome is unknown if the number of errors exceeds the error-correction bound $(d-1)/2$ of the code, where d is the minimum distance of the code. Building on a sequence of previous results [23, 3, 1], Sudan [22] was the first to invent an efficient "list-decoding" algorithm for RS-codes. Given a received word and an integer ℓ, his algorithm returns a list of size at most ℓ of codewords which have distance at most e from the received word, where e is a parameter depending on ℓ and the code. This algorithm, its subsequent generalizations by Shokrollahi and Wasserman [21] to algebraic-geometric codes, and the recent extension by Guruswami and Sudan [8] are among the best decoding algorithms known in terms of the number of errors they can correct.

The list decoding process for AG-codes consists in the first step of computing a nonzero element in the kernel of a certain matrix. The second step then involves a root finding method. The latter step is a subject of investigation of its own and can be solved very efficiently in many cases [6], so we will concentrate in this paper on the first step only. This will be done by applying our general algorithm in in Sections 4 and 5. Specifically, we will for instance easily prove that decoding RS-codes of block length n with lists of length ℓ can be accomplished in time $O(n^2\ell)$. This result matches that of Roth and Ruckenstein [19], though the latter has been obtained using completely different methods. Furthermore, we will design a novel $O(n^{7/3}\ell)$ algorithm for list decoding of certain AG-codes from plane curves of block length n with lists of length ℓ. We remark that, using other means, Høholdt and Refslund Nielsen [10] have obtained an algorithm for list decoding on Hermitian curves which is based on [8], but is more efficient. However, they do not give a rigorous analysis of their algorithm and their methods differ substantially from ours.

Our methodology also applies to erasure decod-

ing of AG-codes from plane curves to yield a new algorithm with running time $O(n^{7/3})$. Furthermore, our approach allows us to parallelize all the algorithms discussed: they can be modified to run in time $O(n)$ on $O(n)$ processors. We will sketch this later in Section 3. Because simple processors are becoming cheaper, this result seems to be of particular practical relevance.

The specific decoding problems highlighted in this paper only serve as examples of the power of the displacement method and are by no means a complete list. The full paper will include further coding theoretic problems that can be successfully attacked using our method.

2 The Displacement Structure

Let m and n be positive integers, K be a field, and $D \in K^{m\times m}$, $A \in K^{n\times n}$. We define the *displacement operator* $\nabla = \nabla_{D,A}: K^{m\times n} \to K^{m\times n}$ by $\nabla(V) := DV - VA$. The ∇-*displacement rank* of V is defined as the rank of the matrix $\nabla(V)$. If this rank is α, then $\nabla(V)$ can be written as GB with $G \in K^{m\times\alpha}$ and $B \in K^{\alpha\times n}$. The pair (G, B) is then called a ∇-*generator* for V. If α is small and D and A are sufficiently simple, then the ∇-operator allows to "compress" the matrix V to matrices with a total of $\alpha(m+n)$ entries. Furthermore, one can efficiently compute with the compressed form as the following lemma suggests.

Lemma 2.1 *Let the matrices in $DV - VA = GB$ be partitioned as*

$$G = [g_{11} \mid G_{21}],\ B = [b_{11} \mid B_{12}],$$
$$D = \begin{bmatrix} d_1 & \mathbf{0}, \\ \star & D_2 \end{bmatrix}, A = \begin{bmatrix} a_1 & \star \\ \mathbf{0} & A_2 \end{bmatrix},$$
$$V = \begin{bmatrix} v_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}.$$

Suppose that v_{11} is nonzero. Then the Schur complement $V_2 := V_{22} - v_{11}^{-1}V_{21}V_{12}$ of V satisfies the equation

$$D_2V_2 - V_2A_2 = G_2B_2,$$

where $G_2 = G_{21} - v_{11}^{-1}g_{11}V_{12}$ and $B_2 = B_{12} - v_{11}^{-1}b_{11}V_{12}$.

A proof can be found in [18, Lemma 3.1], see also [7]. The lemma shows how to obtain an LU-factorization for V (if it exists), by operating only on the matrices G and B. For this, one only needs to know the following well-known facts: the first column of L below the upper right entry is given by the first column of the Schur-complement V_2, and the first row of U is given by the first row of V_2.

The lemma suggests an algorithm for computing an LU-decomposition of V which may not work if the upper right entry of V of any of its successive Schur complements are zero. In this situation partial pivoting can be used. This leads to a modification of the above lemma in which we have to assume that the matrix D is diagonal. We briefly sketch the modification; details can be found in [17, Sect. 3.5]. If $v_{11} = 0$ and the first column of V contains a nonzero element, say at position $(k, 1)$, then we can consider PV instead of P, where P is the matrix corresponding to interchanging rows 1 and k. The displacement equation for PV is then given by $(PDP^{\mathsf{T}})(PV) - PVA = PGB$. Since P can be an arbitrary transposition, for PDP^{T} to be lower triangular we have to assume that D is diagonal. This explains the assumptions in the algorithm below (see [17, Sect. 3.5]).

Algorithm 2.2 *On input a diagonal matrix $D \in K^{m\times m}$, an upper triangular matrix $A \in K^{n\times n}$, and a $\nabla_{D,A}$-generator (G, B) for $V \in K^{m\times n}$ in $DV - VA = GB$, the algorithm outputs a permutation matrix P, a lower triangular matrix $L \in K^{m\times m}$, and an upper triangular matrix $U \in K^{m\times n}$, such that $V = PLU$.*

(1) *Recover from the generator the first column of V.*

(2) *Determine the position, say $(k, 1)$, of a nonzero entry of V. If it does not exist, then set the first column of L equal to $[1, \mathbf{0}]^{\mathsf{T}}$ and the first row of U equal to the first row of V, and go to Step (4). Otherwise interchange the first and the k-th diagonal entries of A and the first and the k-th rows of G and call P_1 the permutation matrix corresponding to this transposition.*

(3) *Recover from the generator the first row of $P_1V =: \left(\begin{smallmatrix} v_{11} & V_{12} \\ V_{21} & V_{22} \end{smallmatrix}\right)$. Store $[1, V_{12}/v_{11}]^{\mathsf{T}}$ as the first column of L and $[v_{11}, V_{12}]$ as the first row of U.*

(4) *If $v_{11} \neq 0$, compute by Lemma 2.1 a generator of the Schur complement V_2 of P_1V. If $v_{11} = 0$, then set $V_2 := V_{22}$, $G_2 := G_{21}$, and $B_2 := B_{12}$.*

(5) *Proceed recursively with V_2 which is now represented by its generator (G_2, B_2) to finally obtain the factorization $V = PLU$, where $P = P_1 \cdots P_\mu$ with P_k being the permutation used at the k-th step of the recursion and $\mu = \min\{m, n\}$.*

The correctness of the above algorithm and its running time depend on steps (1) and (3). Note that it may not be possible to recover the first row and column of V from the matrices D, A, G, B. In fact, recovery from these data alone is only possible if $\nabla_{D,A}$ is an isomorphism. For simplicity we assume in the following that this is the case. In the general case one has to augment the (D, A, G, B) by more data corresponding to the kernel of ∇, see [17, Sect. 5].

Lemma 2.3 *Suppose that steps (1) and (3) of Algorithm 2.2 run in time $O(m)$ and $O(n)$, respectively. Then the total running time of that algorithm is $O(\alpha mn)$, where α is the displacement rank of V with respect to $\nabla_{D,A}$.*

PROOF. The proof is obvious once one realizes that Step (4) runs in time $O(\alpha(m+n))$, and that the algorithm is performed recursively at most $\min\{m, n\}$ times. □

In this paper we are mainly concerned with finding a nonzero element in the kernel of V. Once a PLU-decomposition for V is known, such an element can be found in time $O(\min\{m, n\}^2)$ by the straightforward backward substitution algorithm for solving a homogeneous upper triangular system of equations.

Corollary 2.4 *Suppose that the matrix $V \in K^{m \times n}$ has displacement rank α with respect to the isomorphism $\nabla_{D,A}$ and that V has rank less than n. Then one can compute a nonzero element in the kernel of V with $O(\alpha mn)$ operations.*

3 Parallel Algorithms

The algorithm given in the last section can be customized to run in parallel if certain additional conditions are satisfied. We will sketch the approach here.

To begin with, assume that the matrix A of Algorithm 2.2 is equal to D. This will enable performing steps (1) and (3) of that algorithm in parallel, as we will see below. In this case, however, the operator $\nabla_{D,D}$ is not an isomorphism anymore, so that additional work is necessary to recover the matrix V from the data D, G, B.

The additional data consists of the diagonal entries $v_{11}, v_{22}, \ldots, v_{nn}$ of the matrix V. During the course of the algorithm these values are updated to the diagonal entries of the Schur-complements.

The first column of V is obtained in the following way: first, compute the first column $(\gamma_1, \ldots, \gamma_m)^T$ of GB. Using m processors, this takes $O(\alpha)$ time, where α is the displacement rank of V with respect to $\nabla_{D,D}$. This vector equals the first column of $DV - VD$, i.e., it equals $(0, (d_2 - d_1)v_{21}, \ldots, (d_m - d_1)v_{m1})^T$, where we have denoted the diagonal entries of D by $d_1, \ldots, d_m$. From this, one can compute the first column of V with m processors in constant time. If the entry v_{11} is nonzero, then one can in the same way as above compute the first row of V and proceed with the algorithm. Additional care has to be taken, however, when $v_{11} = 0$. The somewhat lengthy, but straightforward details will be presented in the final version of the paper.

The algorithm then proceeds exactly in the same way as Algorithm 2.2 to obtain a PLU-decomposition. Since solving a homogeneous upper triangular system of linear equations can be easily customized to run in parallel linear time, this gives an algorithm for computing a nontrivial element in the kernel of V in time $O(\alpha n)$ on $O(n)$ processors.

4 List Decoding of Reed-Solomon Codes

In the following we will be dealing with matrices that have a repetitive pattern. The following notation will help us to concisely describe them. Let $\varphi, \varphi_1, \ldots, \varphi_t \colon M \to K$ be functions from a set M into a field K, and let $m_1, \ldots, m_n$ be elements in M. We define

$$[\varphi_1(m), \ldots, \varphi_t(m)]_{(m_1, \ldots, m_n)} :=$$

$$\begin{pmatrix} \varphi_1(m_1) & \varphi_2(m_1) & \cdots & \varphi_t(m_1) \\ \varphi_1(m_2) & \varphi_2(m_2) & \cdots & \varphi_t(m_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(m_n) & \varphi_2(m_n) & \cdots & \varphi_t(m_n) \end{pmatrix}.$$

Whenever $m_1, \ldots, m_n$ are clear from the context, we will replace the subscript $(m_1, \ldots, m_n)$ by $1, n$. Further, we denote the diagonal matrix with diagonal entries $\varphi(m_1), \ldots, \varphi(m_n)$ by $\mathrm{diag}[\varphi]_{1,n}$.

In [22] Sudan describes an algorithm for list decoding of RS-codes which we will briefly describe here. Let $\mathbb{F}_q[x]_{<k}$ denote the space of polynomials over $\mathbb{F}_q$ of degree less than k, and let $x_1, \ldots, x_n$ denote distinct elements of $\mathbb{F}_q$, where $k \leq n$. The image of the morphism $\gamma: \mathbb{F}_q[x]_{<k} \to \mathbb{F}_q^n$ mapping a polynomial f to the vector $v := (f(x_1), \ldots, f(x_n))$ is a linear code over $\mathbb{F}_q$ of dimension k and minimum distance $n-k+1$. Suppose the vector v is sent over a communication channel and the vector $u := (y_1, \ldots, y_n)$ is received. If the Hamming distance between u and v is at most $(n-k)/2$, then conventional decoding algorithms like the Berlekamp-Massey algorithm can decode u to the correct codeword v. If the number e of errors is larger than $(n-k)/2$, then Sudan's algorithm compiles a list of at most $\ell = \ell(e)$ codewords which contains v. The algorithm consists of two steps. Let $B := \lceil (n+1)/(\ell+1) + \ell(k-1)/2 - 1 \rceil$. The first step uses Hermite interpolation to compute a bivariate polynomial $F = \sum_{i=0}^{\ell} F_i(x) y^i$ with $\deg F_i \leq B - ik$ and such that $F(x_t, y_t) = 0$ for all $t = 1, \ldots, n$. The second step computes the roots of F which are of the form $y - g(x)$, $g \in \mathbb{F}_q[x]_{<k}$, and outputs those g such that $\gamma(g)$ and u have distance at most e. The relationship between ℓ and e is given by $e \leq n - B$. There are efficient algorithms for solving the second step using Hensel lifting [6, 19]. In the following we will concentrate on the problem of efficiently computing the polynomial F.

The polynomial F corresponds to a nonzero element in the kernel of a matrix V with a repetitive structure which we will describe below. Let $d_0, \ldots, d_\ell \geq 0$ be defined by $d_i := B - ik$. For each $i = 0, \ldots, b$ let V_i be the matrix

$$V_i := \left[y^i x^{d_i}, \cdots, y^i x, y^i \right]_{1,n}$$

and define the matrix $V \in K^{n \times (n+1)}$ with the block decomposition

$$V = (V_\ell \mid V_{\ell-1} \mid \cdots \mid V_1 \mid V_0). \tag{1}$$

(Here and for the rest of this section the subscript "$1, n$" will always refer to the points $(x_1, y_1), \ldots, (x_n, y_n)$.) Let $\mathfrak{x}$ be a nonzero vector such that $V\mathfrak{x} = 0$. Then, interpreting the entries of $\mathfrak{x}$ as coefficients of the F_j (starting from F_ℓ down to F_0 and reading the coefficients from high powers to low powers of x), this gives a polynomial F as desired. Using the displacement structure approach, we can easily develop an algorithm with running time $O(\ell n^2)$ for computing F. Assuming that ℓ is a constant (a reasonable assumption in applications), this gives an algorithm with a running time that is quadratic in n.

For computing F we will first prove that V has displacement rank at most $\ell + 1$. Let $D := \mathrm{diag}[x]_{1,n}$ (i.e., D is the diagonal matrix with diagonal entries $x_1, \ldots, x_n$). Further, let $A \in K^{(n+1)\times(n+1)}$ be the *upper shift matrix* of format $n+1$, i.e.,

$$A := \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

Let

$$G = \left[y^b x^{d_\ell}, y^{\ell-1}(x^{d_{\ell-1}} - y), \cdots, y(x^{d_1} - y), x^{d_0} - y \right]_{1,n} \in K^{n \times (\ell+1)} \tag{2}$$

and

$$B = (B_\ell \mid B_{\ell-1} \mid \cdots \mid B_1 \mid B_0) \in K^{(\ell+1)\times(n+1)}, \tag{3}$$

where each B_i is an $(\ell+1) \times (d_i+1)$-matrix whose only nonzero position is at $(\ell+1-i, 1)$. Then, one easily proves that

$$DV - VA = GB. \tag{4}$$

Hence, V has displacement rank at most $\ell+1$ with respect to $\nabla_{D,A}$. We can now invoke the general algorithm 2.2 to compute a PLU-decomposition of V using only the matrices D, G, B. (In fact, since

we are only interested in nonzero elements in the kernel of V, it suffices to compute the matrix U only.) To apply Algorithm 2.2 we need to recover the first row and the first column of V from the first row and the first column of GB. The next algorithm shows how to do this in the case where all the x_i are nonzero (which is equivalent to $\nabla_{D,A}$ being an isomorphism).

Algorithm 4.1 *On input a diagonal matrix D with diagonal entries $x_1, \ldots, x_m$, all of them nonzero, a matrix $G \in K^{m \times \alpha}$ as in (2), and a matrix $B \in K^{\alpha \times (m+1)}$ as in (3), this algorithm computes the first row and the first column of the matrix V given by $DV - VA = GB$, where A is the upper shift matrix of format $m+1$.*

(1) *Compute the first row $(r_1, \ldots, r_{m+1})$ and the first column $(\gamma_1, \ldots, \gamma_n)^\top$ of GB.*

(2) *For $i = 1, \ldots, m$ set $v_i := \gamma_i / x_i$.*

(3) *Set $c_1 := v_1$. For $i = 2, \ldots, m+1$ set $c_i := (r_i + c_{i-1})/x_i$.*

(4) *Output row $(c_1, \ldots, c_{m+1})$ and column $(v_1, \ldots, v_m)^\top$.*

Proposition 4.2 *The above algorithm correctly computes its output with $O(dm)$ operations in the field K.*

PROOF. Correctness of the algorithm follows immediately from the following observation: let $(c_1, \ldots, c_{m+1})$ and $(c_1, v_2, \ldots, v_m)^\top$ be the first row and the first column of V, respectively. Then

$$DV - VA = \begin{pmatrix} x_1 c_1 & x_1 c_2 - c_1 & \cdots & x_1 c_{m+1} - c_m \\ x_2 v_2 & \star & \cdots & \star \\ \vdots & \vdots & \cdots & \vdots \\ x_m v_m & \star & \cdots & \star \end{pmatrix},$$

where the $\star$'s denote elements in K. As for the running time, observe first that computing the first row and first column of GB requires at most $2dm$ operations over K. Further, Step (2) and (3) require at most m and $2m$ operations, respectively. □

Using the above result we obtain the following general algorithm for list decoding of RS-codes for the case where none of the x_i is zero.

Algorithm 4.3 *On input $(x_1, y_1), \ldots, (x_n, y_n) \in K^2$ and integers $d_0 \geq d_1, \ldots, d_\ell$ such that $\sum_{i=0}^{\ell}(d_i + 1) = n + 1$ and such that none of the x_i is zero, this algorithm computes a nonzero solution of the homogeneous system of linear equations given by $V\mathfrak{x} = 0$, where V is given in (1).*

(1) *Set $G_1 := G$, $B_1 := B$, $U := 0$, where G and B are given in (2) and (3) respectively.*

(2) *For $i = 1, \ldots, n$ do*

(a) *Let D_i be the diagonal matrix with diagonal entries $x_i, \ldots, x_n$. Use Algorithm 4.1 with input D_i, G_i, and B_i to compute column $\mathfrak{c} = (c_1, \ldots, c_{n-i+1})^\top$.*

(b) *If $\mathfrak{c}$ is nonzero, then find location k such that $c_k \neq 0$. Interchange rows k and 1 of $\mathfrak{c}$ and of G_i. If $\mathfrak{c}$ is zero, then perform the next step, increment i by 1 and go to Step* (a).

(c) *Use Algorithm 4.1 with input D_i, G_i and B_i to compute row $\mathfrak{r} = (r_1, \ldots, r_{n+2-i})$. Write $\mathfrak{r}$ in i-th row of matrix U starting at position i.*

(d) *For $j = 2, \ldots, n-i+1$ replace row j of G_i by $-c_j/c_1$ times the first row plus the j-th row. Set G_{i+1} as the matrix formed from G_i by deleting the first row.*

(e) *For $j = 2, \ldots, n-i+2$ replace the j-th column of B_i by $-r_j/c_1$ times the first column plus the j-th column. Set B_{i+1} as the matrix formed from B_i by deleting the first column.*

(3) *Find nonzero solution of $U\mathfrak{x} = 0$ and output $\mathfrak{x}$.*

Theorem 4.4 *The above algorithm correctly computes its output with $O(\ell n^2)$ operations over the field K.*

PROOF. Follows at once from Corollary 2.4 and Proposition 4.2. □

The performance of the algorithm can be further enhanced by the following observation: since we are only interested in a nonzero element in the kernel of V, we can stop the algorithm, as soon as we find a zero element in the diagonal of the matrix U. A zero element on the main diagonal of U is obtained after at most $\mathrm{rk}(V)$ steps. Hence,

the running time of the algorithm with this "early abort" strategy is $O(n\ell \mathrm{rk}(V))$.

We further remark that the algorithm can be modified to deal with the case that one x_i is zero as well. One way to carry this out is described below. We may w.l.o.g. assume that $x_1 = 0$. In this case, deletion of the first row of the matrix V yields another matrix $\tilde{V}$ of the form given in (1) in which none of the x_i is zero. Algorithm 4.3 computes for $\tilde{V}$ an upper triangular matrix $\tilde{U}$ such that $\tilde{V} = \tilde{P}\tilde{L}\tilde{U}$ for some permutation matrix $\tilde{P}$ and some nonsingular lower triangular matrix $\tilde{L}$. Let U be the matrix whose first row is that of V, and whose remaining rows are those of $\tilde{U}$. It is then obvious that there is a permutation matrix P and a nonsingular lower triangular matrix L such that $V = PLU$. Hence, a vector $\mathfrak{x}$ satisfies $V\mathfrak{x} = 0$ iff it satisfies $U\mathfrak{x} = 0$. Because of the structure of U, one can find such a nonzero vector $\mathfrak{x}$ in time $O(n^2)$.

5 List Decoding of Algebraic-Geometric Codes

In [21] the authors generalize Sudan's algorithm [22] to algebraic-geometric (AG-) codes. We briefly discuss this generalization. Let $\mathcal{X}$ be an irreducible algebraic curve over the finite field $\mathbb{F}_q$, let $Q, P_1, \ldots, P_n$ be distinct $\mathbb{F}_q$-rational points of $\mathcal{X}$, and let $L(\alpha Q)$ denote the linear space of the divisor αQ, i.e., the space of all functions in the function field of $\mathcal{X}$ that have only a pole of order at most α at Q. The (one-point) AG-code associated to these data is then defined as the image of the $\mathbb{F}_q$-linear map $L(\alpha Q) \to \mathbb{F}_q^n$ mapping a function f to the vector $(f(P_1), \ldots, f(P_n))$. It is well known that this linear code has dimension $k \geq \alpha - g + 1$ and minimum distance $d \geq n - \alpha$, where g denotes the genus of the curve. Suppose that we want to decode this code with a list of length at most ℓ. Let $\beta := \lceil (n+1)/(\ell+1) + \ell\alpha/2 + g - 1 \rceil$. Let $\varphi_1, \ldots, \varphi_t$, $t = \beta - g + 1$, be elements of $L(\beta Q)$ with strictly increasing pole orders at Q. Let $(y_1, \ldots, y_n)$ be the received word. The algorithm in [21] first finds functions $u_0, \ldots, u_\ell$ with $u_i \in L((\beta - i\alpha)Q)$ such that $\sum_i u_i(P_j) y_j^i = 0$ for all $1 \leq j \leq n$. This step is accomplished by computing a nonzero element in the kernel of the matrix

$$V := \left[y^\ell \varphi_{s_\ell} \cdots y^\ell \varphi_1 \;\middle|\; \cdots \;\middle|\; y\varphi_{s_1} \cdots y\varphi_1 \;\middle|\; \varphi_{s_0} \cdots \varphi_1 \right]_{1,n}, \tag{5}$$

where $s_j = \beta - j\alpha - g + 1$. To simplify the discussions, we assume that there are two functions $\varphi, \psi \in L(\beta Q)$ such that all the φ_i are of the form $\varphi^a \psi^b$ and such that the order of poles at Q of φ is smaller than that of ψ. Further, we assume that φ does not vanish at any of the points $P_1, \ldots, P_n$. We remark that the method described below can be modified to deal with a situation in which any of the above assumptions is not valid.

Let d be the order of poles of φ at Q. Then it is easily seen that any element of $L(\beta Q)$ is of the form $\varphi^a \psi^b$, where $0 \leq b < d$. Now we divide each of the blocks of the matrix V into subblocks in the following way: by changing the order of the columns, we write the t-th block of V in the form

$$\left[y^t \psi^s \varphi^{a_s} \cdots y^t \psi^s \varphi^0 \mid \cdots \mid y^t \psi^0 \varphi^{a_0} \cdots y^t \psi^0 \varphi^0 \right]_{1,n}.$$

By the above remarks, $s < d$. Let now D be the diagonal matrix $\mathrm{diag}[\varphi]_{1,n}$, and let A be the upper shift matrix of format $n+1$. Then in a similar way as in the last section one sees that there is a matrix $G \in \mathbb{F}_q^{n \times d\ell}$ and a matrix $B \in \mathbb{F}_q^{d\ell \times m}$ such that $DV - VA = GB$, hence is of rank at most $d\ell$. Furthermore, Algorithm 4.1 shows how to recover the first row and the first column of V using the matrices D, A, G, B. Now we can use Algorithm 4.3 to prove the following theorem.

Theorem 5.1 *Algorithm 4.3 applied to the matrices D, V, A, G, B above computes a nonzero element in the kernel of V with $O(n^2 d\ell)$ operations in $\mathbb{F}_q$.*

The same remarks at the end of the last section also apply here. In particular, an early abort strategy reduces the running time to $O(n\mathrm{rk}(V)d\ell)$.

To give a final estimate of the result, we need to relate d and n. This depends on the specific structure of the curve and the divisor used in the definition. For instance, for Hermitian function fields defined over $\mathbb{F}_{q^2}$ by the equations $X^{q+1} = Y^q + Y$ and for Q defined as the common pole of the functions X and Y, the parameter d equals $q+1$, while the genus g is $O(q^2)$ and $n = O(q^3)$. As a result,

the algorithm uses $O(n^{7/3}\ell)$ $\mathbb{F}_q$-operations. Note that for $\ell = 1$, i.e., for unique decoding, this is exactly the running time of the algorithm presented in [12]. A similar assertion can be obtained for certain codes from general plane algebraic curves with many points.

We remark that the same method as above can be applied to efficient erasure decoding of AG-codes. Suppose that the received word u is erased in some positions and denote by $Q_1, \ldots, Q_s$ the points of the curve which correspond to the non-erased positions of u. Let $(y_1, \ldots, y_s)$ denote the vector consisting of the received coordinates of y ordered in correspondence to the ordering $Q_1, \ldots, Q_s$. We assume that s is at least $n - d + 1$, where d is the minimum distance of the code. This guarantees that there is exactly one codeword whose coordinates at the points $Q_1, \ldots, Q_s$ are equal to those of u at these points. As above suppose that the code is constructed by evaluation of functions from $L(\alpha Q)$ and that $\varphi_1, \ldots, \varphi_t$ form a basis for $L(\alpha Q)$. Define the matrix

$$V = [\varphi_1 \cdots \varphi_t]_{Q_1,\ldots,Q_s}.$$

Let $(\lambda_1, \ldots, \lambda_t)^\top$ be the solution to the system of linear equations

$$V(\lambda_1, \ldots, \lambda_t)^\top = (y_1, \ldots, y_s)^\top.$$

Then the decoded codeword is the image of $\sum_{i=1}^{t} \lambda_i \varphi_i$ under the evaluation map. Our aim is thus to find $\lambda_1, \ldots, \lambda_t$. Our method above computes a PLU-decomposition for V in time $O(std)$. From this, we can compute the λ_i in time $O(\max\{s^2, t^2\})$. The desired codeword is then computed using $O(tn)$ operations in $\mathbb{F}_q$.

Theorem 5.2 *Let C be a one-point AG-code of dimension k and block length n built from the divisor Q. Assume that there are two functions φ and ψ such that for any m any function in $L(mQ)$ can be written as a polynomial in φ and ψ. Assume further that the order of poles of φ at Q is d. Then for any δ less than the minimum distance of the code any pattern of δ erasures can be decoded with $O(kd(n-\delta))$ operations.*

6 The Improved Algorithm

In [8] the authors describe an extension of algorithms presented in [22] and [21] in which they use a variant of Hermite interpolation rather than a Lagrange interpolation. In the case of RS-codes, the input to the algorithm is a set of n points (x_i, y_i) in the affine plane over $\mathbb{F}_q$, and parameters r, k, and ℓ. Let β be the smallest integer such that $(\ell+1)\beta > \binom{\ell+1}{2}k + \binom{r+1}{2}n$. The output of the algorithm is a nonzero bivariate polynomial $G = \sum_{i=0}^{\ell} G_i(x)y^i$ such that $\deg G_i < \beta - ik$, $G(x_i, y_i) = 0$ for all $i \le n$, and $G(x + x_i, y + y_i)$ does not contain any monomial of degree less than r. Such a polynomial G corresponds to a nonzero element in the kernel of a certain matrix V which we will describe below. Let $t \le n$ be a fixed positive integer. For $0 \le i < r$ and $0 \le j \le \ell$ let $V_{ij}^t \in \mathbb{F}_q^{(r-i)\times(\beta-jk)}$ be the matrix having (μ,ν)-entry equal to $\binom{\nu}{\mu} x_t^{\nu-\mu}$, where $0 \le \mu < r - i$ and $0 \le \nu \le \beta - jk$. Now define the matrix V_t as a block matrix with r block rows and $\ell + 1$ block columns, with block entry (i, j) equal to $\binom{j}{i} y_t^{j-i} V_{ij}^t$. The matrix V then equals

$$V = \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_n \end{pmatrix}. \tag{6}$$

V has $m := \binom{r+1}{2}n$ rows and $s := (\ell+1)\beta - \binom{\ell+1}{2}k$ columns (and $s > m$). Let C be the lower shift matrix of format s, i.e., C is the transpose of the upper shift matrix of format s introduced in Section 4. Further, let J_i^t denote the $i \times i$-Jordan block having x_t in its main diagonal and 1's on its lower sub-diagonal. Let J^t be the block diagonal matrix with block diagonal entries $J_r^t, \ldots, J_1^t$. Let J be the block diagonal matrix with block diagonal entries $J^1, \ldots, J^n$. Then, a quick calculation shows that $JV - VC$ has rank at most $\ell + 1$. It is now tempting to apply the general results of Section 2 to this situation. However, J is not a diagonal matrix. To remedy this situation, we need the following simple result.

Proposition 6.1 *Let the matrices $V \in K^{m\times n}$ and $W \in K^{n\times\ell}$ have displacement ranks r_1 and r_2 with respect to $\nabla_{F,A}$ and $\nabla_{A,R}$, respectively. Then VW*

has displacement rank at most $r_1 + r_2$ with respect to $\nabla_{F,R}$. Moreover, a generator for VW can be obtained from the generators of V and W with $O((r_1 + r_2)n^2)$ sequential time, and in time $O((r_1+r_2)n)$ on $O(n)$ processors.

The proof of this result follows trivially from the definitions. The assertion on the running time follows from the sequential and parallel running times of the trivial matrix multiplication algorithms.

Let $\mathbb{F}$ be a suitable extension of $\mathbb{F}_q$ having at least s elements, and denote by W the $(s \times s)$-Vandermonde matrix whose rows consist of powers of these elements. Let Δ denote the diagonal matrix having these elements as its diagonal entries. To avoid tedious arguments, we assume that none of the diagonal entries of Δ are zero. Then W has displacement rank one with respect to $\nabla_{\Delta,C^\top}$. Thus, the previous proposition shows that $WV^\top$ has displacement rank $\leq \ell + 2$ with respect to $\nabla_{\Delta,J^\top}$. We can now apply Algorithm 4.3 to obtain a PLU-decomposition of $WV^\top$, where $P, L \in \mathbb{F}^{s\times s}$ and $U \in \mathbb{F}^{s\times m}$. By Corollary 2.4 and Proposition 4.2 this takes $O(\ell sm)$ operations over the field $\mathbb{F}$. Further, we obtain $V = U^\top P^\top L^\top (W^{-1})^\top$. To find a nontrivial element v in the kernel of V, we first compute a nontrivial element u in the kernel of $U^\top$; this can be achieved with $O(m^2)$ operations over the field $\mathbb{F}$. Next we solve the system of linear equations $L^\top w = Pu$. Since $L^\top$ is upper triangular and of full rank m, this takes $O(m^2)$ operations. The desired element v is then obtained as $v = W^\top w$, and its computation takes $O(s^2)$ operations. In total, this gives a sequential algorithm with running time $O(s^2\ell)$ over the field $\mathbb{F}$. Each operation in $\mathbb{F}$ uses $O(\log_q^2(s))$ operations over the base field $\mathbb{F}_q$. Hence, we obtain an algorithm with running time $O(s^2 \log_q^2(s)\ell)$. In the algorithm of Guruswami and Sudan [8] s equals $O(r^2 n)$. Furthermore n and q have the same order of magnitude. As a result, we obtain an algorithm with running time $O(n^2 r^4 \log_q(r)\ell)$. In many practical situations r and ℓ are constant; hence this gives an algorithm with running time $O(n^2)$.

We remark that the above algorithm can be modified to possibly avoid computations in the extension field $\mathbb{F}$. This is done by using a block diagonal matrix for W whose blocks are Vandermonde matrices of sizes given by the blocks of the matrix V, i.e., given by the blocks of lengths $\beta - jk$, $0 \leq j \leq \ell$. If none of these sizes exceeds the size q of the base field, then there is no need for switching to an extension field.

The same methodology as above can be applied to obtain a parallel algorithm for computing a nontrivial element in the kernel of the matrix V given in (1). Choose $n+1$ distinct elements from $\mathbb{F}_q$ (or an extension thereof) and denote by W the Vandermonde matrix corresponding to these elements and by Δ the diagonal matrix having these elements as its diagonal entries. Further, let C denote the upper shift matrix of format $n+1$, and let D denote the diagonal matrix having entries $x_1, \ldots, x_n$, see Section 4. As usual, we assume that D and Δ are invertible. Since V has displacement rank $\leq \ell + 1$ with respect to $\nabla_{D,C}$ (see (4)), and W has displacement rank one with respect to $\nabla_{\Delta,C^\top}$, Proposition 6.1 proves that $WV^\top$ has displacement rank $\leq \ell + 2$ with respect to $\nabla_{\Delta,D}$, and that generators of this operator can be calculated in time $O(\ell n)$ on $O(n)$ processors. Using results of Section 3, we see that we can compute a PLU-decomposition of $WV^\top$ in time $O(\ell n)$ on $O(n)$ processors. It is now easy to see that from this we can compute a nontrivial element in the kernel of V in time $O(n)$ on $O(n)$ processors. The final algorithm is a parallel algorithm that computes a nontrivial element in the kernel of the matrix V in time $O(\ell n)$ on $O(n)$ processors.

7 Open Questions and Future Work

In this paper we have introduced a general method originating from numerical analysis for efficient list decoding of AG-codes. Our algorithm computes a PLU-decomposition of a given dense structured $(n \times m)$-matrix in time close to $O(n^2)$, where closeness depends on the so-called displacement rank of the matrix. The paper discussed three applications, that of efficient list decoding of RS-codes, of AG-codes, and efficient erasure decoding of AG-codes. There are many more applications of this method to coding theoretic problems, like the the improved algorithm of [8] for AG-codes, and parallel algorithms for improved list decoding of RS-codes, to name a few. These and other applications are in preparation and some of them will be included in the final version of the paper.

References

[1] S. Ar, R. Lipton, R. Rubinfeld, and M. Sudan. Reconstructing algebraic functions from mixed data. In *Proc. 33rd FOCS*, pages 503–512, 1992.

[2] E.R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, New York, 1968.

[3] E.R. Berlekamp. Bounded distance + 1 soft decision Reed-Solomon decoding. *IEEE Trans. Inform. Theory*, 42:704–720, 1996.

[4] R. Bitmead and B. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *Linear Algebra and its Applications*, 34:103–116, 1980.

[5] G.L. Feng and T.R.N. Rao. Decoding algebraic-geometric codes up to the designed minimum distance. *IEEE Trans. Inform. Theory*, 39:37–45, 1993.

[6] S. Gao and M.A. Shokrollahi. Computing roots of polynomials over function fields of curves. Preprint, 1998.

[7] I. Gohberg and V. Olshevsky. Fast state-space algorithms for matrix Nehari and Nehari-Takagi interpolation problems. *Integral Equations and Operator Theory*, 20:44–83, 1994.

[8] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, 1998.

[9] G. Heinig and K. Rost. *Algebraic Methods for Toeplitz-like matrices and operators*, volume 13 of *Operator Theory.* Birkhäuser, Boston, 1984.

[10] T. Høholdt and R. Refslund Nielsen. Decoding Hermitian codes with Sudan's algorithm. Preprint, Denmark Technical University, 1999.

[11] T. Høholdt and R. Pellikaan. On the decoding of algebraic-geometric codes. *IEEE Trans. Inform. Theory*, 41:1589–1614, 1995.

[12] J. Justesen, K.J. Larsen, H.E. Jensen, and T. Høholdt. Fast decoding of codes from algebraic plane curves. *IEEE Trans. Inform. Theory*, 38:111–119, 1992.

[13] T. Kailath and A.H. Sayed. Displacement structure: Theory and applications. *SIAM Review*, 37:297–386, 1995.

[14] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, 1988.

[15] M. Morf. *Fast algorithms for multivariable systems.* PhD thesis, Stanford University, 1974.

[16] M. Morf. Doubling algorithms for Toeplitz and related equations. In *Prceedings of IEEE Conference on Acoustics, Speech, and Signal Processing, Denver*, pages 954–959, 1980.

[17] V. Olshevsky. Pivoting for structured matrices with applications. http://www.cs.gsu.edu/~matvro, 1997.

[18] V. Olshevsky and V. Pan. A superfast state-space algorithm for tangential Nevanlinna-Pick interpolation problem. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 192–201, 1998.

[19] R. Roth and G. Ruckenstein. Efficient decoding of reed-solomon codes beyond half the minimum distance. In *Prceedings of 1998 EIII International Symposium on Information Theory*, pages 56–56, 1998.

[20] S. Sakata, J. Justesen, Y. Madelung, H.E. Jensen, and T. Høholdt. Fast decoding of algebraic-geometric codes up to the designed minimum distance. *IEEE Trans. Inform. Theory*, 41:1672–1677, 1995.

[21] M.A. Shokrollahi and H. Wasserman. Decoding algebraic-geometric codes beyond the error-correction bound. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 241–248, 1998.

[22] M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *J. Compl.*, 13:180–193, 1997.

[23] L.R. Welch and E.R. Berlekamp. Error correction for algebraic block codes. U.S. Patent 4,633,470, issued Dec. 30, 1986.