A quadratic Welch-Berlekamp algorithm to decode generalized Gabidulin codes, and some variants

Gwezheneg Robert

To cite this version:


HAL Id: hal-01320286
https://hal.archives-ouvertes.fr/hal-01320286
Submitted on 23 May 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives| 4.0 International License
A quadratic Welch-Berlekamp algorithm to decode generalized Gabidulin codes, and some variants

Gwezheneg ROBERT
IRMAR, Université de Rennes 1
Rennes, France
Email: gwezheneg.robert@univ-rennes1.fr

Abstract—Gabidulin codes are Maximum Rank Distance (MRD) codes. They have been recently generalized to cyclic Galois extension fields. The unique decoding problem is equivalent to the linear reconstruction problem.

The aim of this article is the study of an algorithm to solve this reconstruction problem. We prove that the output of our algorithm is a solution of the reconstruction problem. Then we give some variants.

We also establish that (one of the variant of) the algorithm is quadratic.

Index Terms—Gabidulin codes, rank metric, skew polynomials, decoding algorithm.

I. MOTIVATION

Gabidulin codes have been discovered in 1978 [4], in 1985 [5] then in 1991 [12]. In Gabidulin’s article, the codes are defined similarly to Reed-Solomon codes. The difference between these two kinds of codes is the metric and the evaluated polynomials. Thus, they have some common properties, like their optimality concerning the Singleton bound. This optimality explains the various uses of Gabidulin codes.

For example, in network coding, interesting codes are designed from Gabidulin codes with the lifting construction. Such codes are also used for distributed storage [10].

Binary Gabidulin codes have been used to design spacetime code [3] [7]. The main difficulty was to get codewords in the complex field. For this purpose, Gabidulin codes have been generalized to cyclic Galois extensions of the rational field [1] [11]. Nevertheless, a large error-correcting capability is useless without an efficient decoding algorithm. Several algorithms have been proposed, most of them coming from Reed-Solomon decoders. Among all these algorithms, we choose the Welch-Berlekamp-like one [6] for several reasons. First of all, the algorithm has a quadratic complexity. Then, it is easy to implement and requires few memory. Last reasons come from the study of the algorithm and confirm this choice. A division-free variant enables the use of this algorithm for codes over integer rings [11]. Then, there is no conditions on the support of the Gabidulin code, which enables the adaptation of the algorithm for decoding erasures [11].

II. CONTRIBUTION

The algorithm we study is inspired by the Welch-Berlekamp algorithm [2], adapted to Gabidulin codes [6]. We focus on the reconstruction problem, which is the main part of the decoder. There are some cases for which the algorithm of [6] does not work. Namely, the algorithm fails if a discrepancy $(u_0 \text{ or } u_1)$ is zero. In this article, we fill the remaining gaps and prove that the output is a solution of the reconstruction problem. Moreover, we give three variants. We also establish the complexity of the algorithm.

Section III gives definition and properties of generalized Gabidulin codes. Section IV introduces a decoding method, based on the reconstruction problem. Proofs are omitted due to brevity (see [1], [11] and [6]). Contribution concerns the reconstruction algorithm (Algorithm 2). In Section V, we prove that the output of the algorithm is a solution of the reconstruction problem. We also give the algebraic complexity of this algorithm. Section VI is dedicated to the three variants of the algorithm.

III. GENERALISED GABIDULIN CODES

In this section, we generalise the definition of Gabidulin codes to cyclic Galois extensions. We first introduce $\theta$-polynomials and rank metric.

In the whole article, we consider a cyclic Galois extension $K \hookrightarrow L$ of degree $m$, and a generator $\theta$ of the Galois group of the extension. (We directly choose convenient extensions, which fill the conditions required for Theorems 1 and 4.) Let $B = (b_1, \ldots, b_m)$ be a $K$-basis of $L$ (as a $K$-vector space). The following mapping enables to turn an element of $L$ into a vector of $K^m$.

$$\text{ext}_B : L \ni x = x_1 b_1 + \cdots + x_m b_m \mapsto (x_1, \ldots, x_m)$$

A. $\theta$-polynomials

$\theta$-polynomials are a particular case of skew polynomials [9]. They are the generalisation of $q$-polynomials [8] used for original Gabidulin codes.

Definition 1 ($\theta$-polynomials).

• A $\theta$-polynomial with coefficients in $L$ is an element of the form

$$\sum_{i \geq 0} a_i X^i : a_i \in L,$$

with a finite number of non-zero $a_i$.

• The degree of a non-zero $\theta$-polynomial $\sum_{i \geq 0} a_i X^i$ is $\deg(A) = \max \{i : a_i \neq 0\}$. (By convention, the degree of the zero $\theta$-polynomial is $-\infty$.)
The set of \( \theta \)-polynomials with coefficients in \( L \) is denoted by \( L[X; \theta] \).

We can define some operations over \( \theta \)-polynomials.

**Definition 2** (Operations over \( \theta \)-polynomials). Let \( A = \sum_{i=0}^{n} a_i X^i \in L[X; \theta] \), \( B = \sum_{i=0}^{m} b_i X^i \in L[X; \theta] \), and \( c \in L \).

- The addition is: \( A + B = \sum_{i=0}^{\max(n,m)} (a_i + b_i) X^i \).
- The \( (\text{symbolic}) \) product is defined by: \( X \cdot c = \theta(c) \cdot X \). Thus the product is well-defined: \( A \cdot B = \sum_{i,j} a_i \theta^i(b_j) X^{i+j} \).
- The evaluation is: \( A(c) = \sum_{i} a_i \theta^i(c) \).

The ring of \( \theta \)-polynomials is a non-commutative ring. There is a left and a right Euclidean division. Notice that in this ring, there are several ways to define an evaluation. We choose the operator evaluation since it generalizes the evaluation used in Gabidulin codes over finite fields.

**Remark 1.** When we consider finite fields \( \mathbb{F}_q \hookrightarrow \mathbb{F}_{q^m} \), provided with the Frobenius automorphism \( x \mapsto x^q \), we get the \( q \)-polynomials used in original Gabidulin codes.

The algorithm relies on the three following properties.

**Theorem 1.**
- Let \( A \) be a \( \theta \)-polynomial of degree \( d \). Then the set of roots of \( A \) (i.e. elements \( x \) such that \( A(x) = 0 \)) is a \( K \)-vector space of dimension at most \( d \).
- Let \( A \) be a \( \theta \)-polynomial such that \( A(v_i) = 0 : 1 \leq i \leq s \), where \( v_1, \ldots, v_s \) are \( K \)-linearly independent elements of \( L \). Then either \( \deg(A) \geq s \) or \( A = 0 \).

**Definition 3** (Annihilator \( \theta \)-polynomial).
- Let \( V \) be a \( K \)-subspace of \( L \) (as a vector space). An annihilator \( \theta \)-polynomial of \( V \) is a monic non-zero \( \theta \)-polynomial \( A \) of minimal degree such that:
  \[
  \forall v \in V, \quad A(v) = 0.
  \]
- Let \( (v_1, \ldots, v_s) \in L^s \). An annihilator \( \theta \)-polynomial of \( (v_1, \ldots, v_s) \) is an annihilator \( \theta \)-polynomial of \( \text{Vect}(v_1, \ldots, v_s) \).

**Proposition 2.** Given a \( K \)-subspace \( V \) of \( L \), there is a unique annihilator \( \theta \)-polynomial of \( V \). Moreover, its degree is exactly \( \dim(V) \).

**Definition 4** (Interpolating \( \theta \)-polynomial). Let \( (x_1, \ldots, x_s) \in L^s \) and \( (y_1, \ldots, y_s) \in L^s \). An interpolating \( \theta \)-polynomial \( I \) is a \( \theta \)-polynomial of minimal degree such that:
  \[
  \forall 1 \leq i \leq s, \quad I(x_i) = y_i.
  \]

**Proposition 3.** If \( x_1, \ldots, x_s \) are \( K \)-linearly independent, then there exists a unique interpolating \( \theta \)-polynomial at degree at most \( s - 1 \).

**B. Rank metric**

**Definition 5** (Rank metric). Let \( x = (x_1, \ldots, x_s) \in L^s \). The rank weight of \( x \) is the rank of the matrix whose columns are the column vectors \( \text{ext}_B(x_i) \):
  \[
  w_r(x) = \text{rank} \left( \text{ext}_B(x_1), \ldots, \text{ext}_B(x_s) \right).
  \]

The rank distance between \( x \) and \( y \) is \( d_r(x, y) = w_r(x - y) \).

There is another way to define this metric.

**Definition 6** (Alternative rank metric). Let \( x = (x_1, \ldots, x_s) \in L^s \). The rank weight \( w_{r^*}(x) \) of \( x \) is the degree of the annihilator \( \theta \)-polynomial of \( \text{Vect}(x_1, \ldots, x_s) \).

The rank distance between \( x \) and \( y \) is \( d_{r^*}(x, y) = w_{r^*}(x - y) \).

**Theorem 4.**
  \[
  \forall x \in L^s, \quad w_r(x) = w_{r^*}(x)
  \]

Thus, the two metrics are identical and called the rank metric.

**Proposition 5** (Singleton bound). Let \( C \) be an error correcting code of length \( n \), dimension \( k \). Then its minimal distance \( d \) (in rank metric), satisfies: \( d \leq n - k + 1 \).

**Definition 7** (MRD codes). Let \( C \) be an error correcting code of parameters \([n, k, d] \). The code is MRD (Maximal Rank Distance) if \( d = n - k + 1 \).

**C. Gabidulin codes**

**Definition 8** (Generalised Gabidulin code). Let \( k \leq n \leq m \) be integers and \( g = (g_1, \ldots, g_n) \) be \( K \)-linearly independent elements of \( L \). The generalised Gabidulin code \( \text{Gab}_{\theta,k}(g) \) of length \( n \), dimension \( k \) and support \( g \) is:
  \[
  \text{Gab}_{\theta,k}(g) = \{ f(g_1), \ldots, f(g_n) \} : \begin{cases}
  f \in L[X; \theta], & \deg(f) < k \\
  f \in L[X; \theta], \deg(f) < k \}
  \end{cases}.
  \]

The codeword obtained with the information word \( f \) is denoted by \( c(f) \).

**Proposition 6.** A generalised Gabidulin code is a linear code, whose generator matrix is:
  \[
  \begin{pmatrix}
  \theta^0(g_1) & \cdots & \theta^0(g_n) \\
  \vdots & \ddots & \vdots \\
  \theta^{k-1}(g_1) & \cdots & \theta^{k-1}(g_n)
  \end{pmatrix}.
  \]

**Proposition 7.** A generalised Gabidulin code of length \( n \) and dimension \( k \) has minimal distance (in rank metric) \( d = n - k + 1 \). Thus, generalised Gabidulin codes are MRD codes, and we can correct an error of rank \( t \) if:
  \[
  t \leq \left\lfloor \frac{n - k}{2} \right\rfloor.
  \]

**Example 1.** Consider the extension \( \mathbb{F}_2 \hookrightarrow \mathbb{F}_{2^4} = \mathbb{F}_2[Y]/(Y^4 + Y + 1) \), provided with the Frobenius automorphism \( x \mapsto x^2 \). Let \( \mathcal{G} \) be the Gabidulin code of length \( n = 4 \), dimension \( k = 2 \) and support \( g = (1, \alpha, \alpha^2, \alpha^3) \). We consider the information word:
  \[
  f = \alpha^5 + \alpha^7 X.
  \]

Then, the corresponding codeword is:
  \[
  c(f) = (\alpha^{13}, \alpha^5, \alpha^8, \alpha^3) \in \mathbb{F}_{16}^4.
  \]
IV. Decoding by Reconstruction

The reconstruction problem and its adaptation to the decoding of Gabidulin codes is described in [6]. The method is also valid for Gabidulin codes over infinite fields.

Let \( G_{N,n,k}(g) \) be a Gabidulin code. Let \( f \) be an information word, and \( e(f) \) the corresponding codeword. During the transmission, an error is added, and the receiver gets \( y = e(f) + e \).

**Definition 9 (Decoding Problem).** Given a Gabidulin code of length \( n \), dimension \( k \) and support \( g \), and a received word \( y \), solving the Decoding Problem \( \text{Dec}(n,k,g,y) \) consists of finding

- a \( \theta \)-polynomial \( f \in L[X;\theta] \) of degree \( \deg f < k \),
- a vector \( e \in L^n \) of rank \( \text{rank}(e) \leq \left\lfloor \frac{n-k}{2} \right\rfloor \),

such that

\[ y_i = f(g_i) + e_i. \]

We can reformulate this problem with only \( \theta \)-polynomials. This is the linear reconstruction.

**Definition 10 (Linear Reconstruction Problem).** Given integers \( k \leq n \) and \( t \leq n \), a vector \( g \in L^n \) of \( K \)-linearly independent values, and a vector \( y \in L^n \), solving the Linear Reconstruction problem \( \text{LR}(n,k,t,g,y) \) consists of finding

- a \( \theta \)-polynomial \( N \in L[X;\theta] \) of degree \( \deg N < k + t \),
- a non-zero \( \theta \)-polynomial \( W \in L[X;\theta] \) of degree \( \deg W \leq t \),

such that

\[ W(y_i) = N(g_i). \]

Any solution \((f,e)\) of \( \text{Dec}(n,k,g,y) \) gives a solution \((N,W)\) of \( \text{LR}(n,k,\left\lfloor \frac{n-k}{2} \right\rfloor, g,y) \), where \( W \) is the annihilator polynomial of \( e \), and \( N = W \cdot f \). The interesting fact is the reverse property.

**Theorem 8.** If \( t \leq \left\lfloor \frac{n-k}{2} \right\rfloor \) and if there exists a solution of \( \text{Dec}(n,k,g,y) \), then any solution of \( \text{LR}(n,k,t,g,y) \) gives the solution of \( \text{Dec}(n,k,g,y) \).

The decoding algorithm is given in Algorithm 1.

---

### Algorithm 1: decoding algorithm

**Input:** dimension \( k \) and length \( n \) of the code \((g_1,\ldots,g_n)\), support of the code \((y_1,\ldots,y_n)\), received word

**Output:** \( f \), the information word

1: Compute \((N,W)\), solution of \( \text{LR}(n,k,t,g,y) \)
2: Compute \( Q \) and \( R \) such that \( N = W \cdot Q + R \) (Euclidean division)
3: If \( R = 0 \)
   4: \( \text{return } Q \)
5: \( \text{else} \)
6: \( \text{return } \text{"Failure: too much errors"} \)
7: \( \text{end if} \)

---

**Algorithm 2: reconstruction algorithm**

**Input:** dimension \( k \) and length \( n \) of the code error-correcting capability \( t \) of the code \((g_1,\ldots,g_n)\), support of the code \((y_1,\ldots,y_n)\), received word

**Output:** \( N \) and \( W \) solutions of the LR \((n,k,t,g,y)\)

0: \# Initialisation
1: \( N_0 \leftarrow 1 \)
2: \( N_1 \leftarrow 0 \)
3: \( W_0(X) \leftarrow 0 \)
4: \( W_1(X) \leftarrow 1 \)
5: for \( i \) from 1 to \( k \)
6: \( N_i \leftarrow N_1 + \frac{y_i - N_0(g_i)}{N_0(g_i)} \cdot N_0 \)
7: \( N_0 \leftarrow (X - \frac{g_i}{N_0(g_i)}) \cdot N_0 \)
8: \# Main Loop
9: for \( i \) from \( k + 1 \) to \( n \)
10: \# Computation of discrepancies
11: \( u_0 \leftarrow N_0(g_i) - W_0(g_i) \)
12: \( u_1 \leftarrow N_1(g_i) - W_1(g_i) \)
13: \# Secondary Loop
14: if \( u_0 \neq 0 \) and \( u_1 = 0 \)
15: \( j \leftarrow i + 1 \)
16: while \( u_0 \neq 0 \) and \( u_1 = 0 \) and \( j < n \)
17: \( u_0 \leftarrow N_0(g_j) - W_0(g_j) \)
18: \( u_1 \leftarrow N_1(g_j) - W_1(g_j) \)
19: \( j \leftarrow j + 1 \)
20: end while
21: if \( j = n + 1 \)
22: \( g_i \leftarrow g_j \)
23: \( y_i \leftarrow y_j \)
24: \# end if
25: \# update of \( \theta \)-polynomials, according to discrepancies
26: if \( u_0 = 0 \) and \( u_1 = 0 \)
27: \( N_0' \leftarrow (X - \frac{u_0}{u_1}) \cdot N_1 \)
28: \( W_0' \leftarrow (X - \frac{u_0}{u_1}) \cdot W_1 \)
29: \( N_1' \leftarrow N_0 - \frac{\theta}{\theta_0} N_1 \)
30: \( W_1' \leftarrow W_0 - \frac{\theta}{\theta_0} W_1 \)
31: \# end if
32: if \( u_0 = 0 \) and \( u_1 \neq 0 \)
33: \( N_0' \leftarrow (X - \frac{u_0}{u_1}) \cdot N_1 \)
34: \( W_0' \leftarrow (X - \frac{u_0}{u_1}) \cdot W_1 \)
35: \( N_1' \leftarrow N_0 - \frac{\theta}{\theta_0} N_1 \)
36: \( W_1' \leftarrow W_0 - \frac{\theta}{\theta_0} W_1 \)
37: \# end if
38: if \( u_0 \neq 0 \) and \( u_1 = 0 \)
39: \( N_0' \leftarrow N_0 \)
40: \( W_0' \leftarrow W_0 \)
41: \# end if
42: \# end if
43: \( N_0 \leftarrow N_0' \)
44: \( W_0 \leftarrow W_0' \)
45: \( N_1 \leftarrow N_1' \)
46: \( W_1 \leftarrow W_1' \)
47: for \( i \) from 1 to \( k \)
48: \( N_0 \leftarrow N_1 \)
49: \( W_0 \leftarrow W_1 \)
50: \# end for
51: if \( n = k \) is even
52: \( \text{return } N_1, W_1 \)
53: \# else
54: \( \text{return } N_0, W_0 \)
55: \# end if

---

V. Proof of Correctness

We now focus on the Reconstruction algorithm, given above (Algorithm 2).
Thus, we have the following upper bounds:

1. $N(g_i) = W(y_i), 1 \leq i \leq n$
2. $\deg(N) < k + t$
3. $\deg(W) \leq t$
4. $W \neq 0$

The proof is cut in three parts.

**Proposition 10.** With the notations of Theorem 9, we have $N(g_i) = W(y_i), 1 \leq i \leq n$.

**Proof:** We just have to check, by induction, that after step $i$, (during initialisation loop or main loop) $N$ and $W$ verify the induction hypothesis:

$$\mathcal{H}(i) : \begin{cases} N_0(g_j) = W_0(y_j), 1 \leq j \leq i \\ N_1(g_j) = W_1(y_j), 1 \leq j \leq i \end{cases}$$

At the end of the algorithm (normal output), we have $\mathcal{H}(n)$. In case of premature input (line 39), we have $N_0(g_j) = W_0(y_j), 1 \leq j \leq i$ by induction hypothesis, and we have $N_0(g_j) = W_0(y_j), i \leq j$, which is part of the conditions of premature input.

**Proposition 11.** With the notations of Theorem 9, we have $\deg(N) < k + t$ and $\deg(W) \leq t$.

**Proof:** During the initialisation loop, degrees of $N_1$ and $N_0$ increase by 1 during each step. During the main loop, they increase as follows:

$$\begin{align*}
\deg(N_0) &= \deg(N_1), \\
\deg(W_0) &= \deg(W_1), \\
\deg(N_0) &\leq \max(\deg(N_0), \deg(N_1)), \\
\deg(W_0) &\leq \max(\deg(W_0), \deg(W_1)).
\end{align*}$$

Thus, we have the following upper bounds:

$$\begin{align*}
\deg(N_0) &\leq k + \left\lfloor \frac{-k}{2} \right\rfloor, \\
\deg(W_0) &\leq \left\lfloor \frac{-k+1}{2} \right\rfloor, \\
\deg(N_1) &\leq k - 1 + \left\lfloor \frac{k+1}{2} \right\rfloor, \\
\deg(W_1) &\leq \left\lfloor \frac{k}{2} \right\rfloor.
\end{align*}$$

At the end of the algorithm (normal output), one of the pairs $(N_0, W_0)$ or $(N_1, W_1)$ verifies the degree conditions. Since this upper bound is increasing along the steps, in case of premature output, we get a pair which verifies degree conditions.

**Proposition 12.** With the notations of Theorem 9, we have $W \neq 0$.

**Proof:** First, remark that $W = 0$ implies $N = 0$. Indeed, when $W = 0$, the interpolation conditions become $N(g_i) = 0$, thus $N$ has $n$ roots linearly independent, so $N = 0$. Then, we can list the degree possibilities according to the parity of the step and according to which polynomial reaches the upper bounds. The 8 cases are the following.

- For even steps:
  - all polynomials reach the upper bound
  - only $N_1$ has degree lower than the upper bound
  - only $W_0$ has degree lower than the upper bound
  - both $N_1$ and $W_0$ have degree lower than the upper bound

- For odd steps:
  - all polynomials reach the upper bound
  - only $N_0$ has degree lower than the upper bound
  - only $W_1$ has degree lower than the upper bound
  - both $N_0$ and $W_1$ have degree lower than the upper bound

Then, we never have the pair $(0, 0)$ since two polynomials of the same pair never have lower degree than the upper bound simultaneously.

**Proposition 13.** The main part of the algorithm (meaning all except the secondary loop) requires $2n^3 + O(n)$ multiplications and $2n$ divisions.

During the secondary loop, some operations are added, due to computation of discrepancies (lines 34 and 35). In the worst case (meaning that we compute all possible discrepancies at each step), we add $\frac{1}{4}(n-k)(n^2 + 5nk + 2k^2) + O(n^2)$ multiplications. The third variant of section VI decreases this complexity to 0, making the algorithm quadratic.

**Remark 2.** The secondary loop enables to stop the algorithm as soon as we have computed a solution. (When the weight of the error is $\tau$, the algorithm stops after $2\tau$ steps.)

**VI. Variants**

The goal of this section is to describe some variants of the algorithm. For each of them, we see the interest and the effect on the complexity.

A. A division-free algorithm

This variant has interest for codes in number fields. It enables, when the support of the code, the information word and the error have coefficients in the integer ring, to get polynomials whose coefficients are also in the integer ring. We only have to modify updates, which become: (lines that do NOT appear in variants haven’t changed)

**Modification for ’division-free’ variant**

5 : $\lambda \leftarrow 1$
6 : for $i$ from 1 to $k$
7 : $N_1 \leftarrow N_0(x_i)N_1 + (\lambda y_i - N_1(x_i)) \cdot N_0$
8 : $\lambda \leftarrow N_0(x_i)\lambda$
9 : $N_0 \leftarrow N_0(x_i)(X - \theta(N_0(x_i))) \cdot N_0$
10 : end for
Proposition 14. The algorithm requires $3n^2 + O(n)$ multiplications and 0 divisions.

B. Lower degree polynomials

We can remark that $N_0$ and $N_1$ are updated by additions and left multiplications. Thus, their value along the main loop is a polynomial combination of their value after the initialisation, namely the annihilator $A$ of $(g_1, \ldots, g_k)$ and the interpolating polynomial $I$ of $(y_1, \ldots, y_k)$. Thus, we have

$$N_t = P_t A + W_t I.$$

$P_t$ is updated by the same formulas as $N_t$ and $W_t$.

We have to modify the computation of the discrepancies.

Modification for 'lower-degree' variant

$$23, 34 : \quad u_0 = P_0(A(g_1)) + W_0 I(g_1) - y_1$$
$$24, 35 : \quad u_1 = P_1(A(g_1)) + W_1 I(g_1) - y_1$$

The last modification is part of the decoding algorithm but not of the reconstruction algorithm, since it concerns the final division $W \backslash N$ of $N$ by $W$. This division becomes $W \backslash (P A) + I$.

Proposition 15. The number of multiplication is decreased by $1.5k(n-k)$ for the basic algorithm and by $3.5k(n-k)$ for the division-free variant.

C. Updating discrepancies

Instead of computing two discrepancies at each step, we can initialise them after the initialisation of polynomials, and update them at each step. The discrepancies $u_0$ and $u_1$, now denoted $u_{0,j}$ and $u_{1,j}$, since they depend on the pair $(g_j, y_j)$, are updated by the same formulas than $N_0$, $W_0$, $N_1$ and $W_1$. Thus, the following lines are added at the initialisation

Modification for 'updating discrepancies' variant

$$11 : \quad \text{for } j \text{ from } k+1 \text{ to } n$$
$$12 : \quad u_{0,j} \leftarrow N_0(g_j) - W_0(y_j)$$
$$13 : \quad u_{1,j} \leftarrow N_1(g_j) - W_1(y_j)$$
$$14 : \quad \text{end for}$$

Computations of discrepancies (lines 23, 24, 34 and 35) are removed.

Then, updates become

Modification for 'updating discr.' variant

$$54, 64 : \quad u_{0,j} \leftarrow (X - \frac{y_1}{u_1}) \cdot u_{1,j}$$
$$57 : \quad u_{1,j} \leftarrow u_{0,j} - \frac{g_0}{u_1} v_{1,j}$$
$$74 : \quad u_{0,j} \leftarrow X \cdot v_{1,j}$$
$$67, 77 : \quad u_{1,j} \leftarrow u_{0,j}$$

The advantage of this variant is that the secondary loop doesn’t require computation. Thus, the algorithm is always quadratic.

Proposition 16. The main part of the algorithm (meaning all except additional computations from the secondary loop) has the same complexity for the basic algorithm. In the division-free variant, there are $O(n^2 - k^2)$ additional multiplications. In both cases, the number of computation in the secondary loop is decreased to 0, thus the algorithm is always quadratic.

VII. Conclusion

In this paper, we have described an algorithm solving the reconstruction problem, which is the main step of the decoding of generalised Gabidulin codes. We have proved that the output of the algorithm is a solution of the reconstruction problem. We have adapted the algorithm to integer rings with the ‘division-free’ variant.

We have also established that the algorithm is quadratic. Indeed, the main part of the algorithm (lines 1 to 29 and 50 to 95) requires a quadratic number of operations, in all variants. In the basic algorithm (meaning Algorithm 2 without variant), the secondary loop (lines 30 to 49) is cubic in the worst case. The ‘updating discrepancies’ variant enables to decrease the number of these additional computations to 0, making the whole algorithm always quadratic.

Another improvement concerning the complexity is given. The ‘lower degree’ variant enables to decrease the constants hidden in the $O$-notation.

REFERENCES