Re-encoding reformulation and application to
Welch-Berlekamp algorithm

Morgan Barbier
University of Caen – GREYC
morgan.barbier@unicaen.fr
2012-12-21

Abstract

The main decoding algorithms for Reed-Solomon codes are based on a bivariate interpolation step, which is expensive in time complexity. Lot of interpolation methods were proposed in order to decrease the complexity of this procedure but stay expensive. Then Koetter, Ma and Vardy proposed in 2010 a technique, called re-encoding, which permits to reduce the running time. However this trick is devoted for the Koetter interpolation algorithm. We propose a reformulation of the re-encoding for any interpolation methods. Finally, we apply it to the Welch-Berlekamp algorithm.

Keywords: Reed-Solomon codes, Welch-Berlekamp algorithm, Re-encoding.

1 Introduction

The algebraic decoding algorithms for the Reed-Solomon codes were very studied from the last decades, especially their decoding algorithms. The Welch-Berlekamp decoding method provides a simple approach to decode the Reed-Solomon codes up to the correction capacity of the code [WB86]. Then in 1997, Sudan generalizes this approach to decode beyond this bound, which supplies the first list decoding method for this family [Sud97]. Two years latter, Guruswami and Sudan introduced another generalization of the last method to correct even more errors, that is up to the Johnson bound [GS99].

In these three previous methods, a bivariate interpolation step is needed, moreover their time complexities are given by this procedure which can be expensive. Thus a lot of algorithms were proposed to solve the bivariate interpolation as efficient as possible [Koe96, GR06, AZ08, Tri10]. Even with these computation improvements, the bivariate interpolation step is always expensive.

In this way, Koetter, Ma, and Vardy introduced the notion of re-encoding [KMV11]. This trick does not decrease the asymptotic complexity in general,
but permits a considerable gain in practice. The re-encoding can be split
into three phases as following : start to perform a translation by a codeword
on the received word such that \( k \) positions become null, then modify the
intern statement of the interpolation algorithm to have benefits of the null
positions, finally remove after the interpolation the translation did at the first
step. This technique implies to modify the intern state of the interpolation
algorithm in relation with the null positions to speed up the running time of
the interpolation step. This adjustement of the intern state is the main, and
maybe the only one, drawback of the re-encoding.

In this article, we propose a new reformulation of the re-encoding. This
reformulation permits to use the re-encoding trick with any bivariate inter-
polation algorithm without preliminary modification. However to be generic
is under an assumption between the multiplicity and the \( Y \)-degree of the
interpolated polynomial. We apply this reformulation to the Welch-Berlekamp
algorithm and we observe that the gain is huge.

The organization of the paper is as follows. In the Section 2 we recall the
main decoding algorithms based on interpolation as Welch-Berlekamp, Sudan
and Guruswami-Sudan. The Section 3 is devoted to recall the principle of the
original re-encoding and to introduce our reformulation. Finally, in the Section 4
we apply our revisited re-encoding to Welsh-Berlekamp algorithm and present
the performances.

2 Interpolation based decoding algorithms

2.1 Bivariate interpolation for the decoding

Different decoding algorithms are based on the bivariate interpolation. This step
is the most expensive and the asymptotic complexity is given by this bivariate
interpolation. For example, Welch-Berlekamp, Sudan and Guruswami-Sudan
algorithms are based on this procedure. Since the list decoding algorithm for
alternant codes \cite{ABC11}, is also based on interpolation step, we can also apply
the re-encoding on it. In this paper we propose to deal only with the decoding
algorithms for Reed-Solomon codes, it is why we propose firstly to recall the
definition of this class of codes.

Definition 1 (Reed-Solomon codes). Let \( \alpha_1, \ldots, \alpha_n \in \mathbb{F}_q \) be \( n \) distinct elements
of \( \mathbb{F}_q \). The Reed-Solomon code of dimension \( k \) and support \( \{ \alpha_i \} \) is given by

\[
\text{RS}(\alpha, k) = \{(P(\alpha_1), \ldots, P(\alpha_n)) : P \in \mathbb{F}_q[X]_{\leq k}\}.
\]

The three following algorithms are based on the same principle:

1. Compute a bivariate polynomial by interpolation of the received word \( y \)
and the support of the Reed-Solomon code \( \alpha \).
2. Compute the univariate polynomial(s) \( P \) which generated the code-word(s), as \( Y \)-root(s) of the bivariate polynomial.

The differences between the three following algorithm are the parameters of the bivariate interpolation and represent the most expensive cost in the complexity of these methods. In the following, we present quickly the main decoding algorithms for Reed-Solomon codes, the interested reader can find more information in [Bar11].

### 2.1.1 Welch-Berlekamp

The Welch-Berlekamp algorithm is an unambiguous decoding algorithm devoted to the Reed-Solomon codes [WB86]. However, the most famous list decoding algorithms are based on this method. This is why, we propose to recall the main step of this algorithm.

This method is based on the computation of the bivariate polynomial by interpolation satisfying

\[
\begin{align*}
\left( \text{IP}_{WB} \right) & \triangleq \\
0 \neq Q(X, Y) & \triangleq Q_0(X) + YQ_1(X), \\
Q(\alpha_i, y_i) & = 0, \ \forall i \in \{1, \ldots, n\}, \\
\deg Q_0 & \leq n - t - 1, \\
\deg Q_1 & \leq n - t - k,
\end{align*}
\]

where \( t = \left\lfloor \frac{n-k}{2} \right\rfloor \) is the correction capacity of the Reed-Solomon code. Thus we obtain the following pseudo-code:

**Algorithm 1:** Welch-Berlekamp

**Input:** The received word \( y \in F_q^n \) and the Reed-Solomon code \( C \).

**Output:** The codeword \( c \in C \) if it exists such that \( d(c, y) \leq t = \left\lfloor \frac{n-k}{2} \right\rfloor \), under the polynomial form.

begin

\[
Q(X, Y) \leftarrow \text{Interpolation}(\text{IP}_{WB}, C)
\]

return \(-\frac{Q_0(X)}{Q_1(X)}\)

### 2.1.2 Sudan

Sudan remarked that if we wish to correct more errors, with the Welch-Berlekamp algorithm, it could happen that there exist different \( Y \)-roots of the bivariate polynomial satisfying the condition [Sud97]. So he proposed to modify the interpolation problem in this way:

\[
\begin{align*}
\left( \text{IP}_S \right) & \triangleq \\
0 \neq Q(X, Y) & \triangleq \sum_{i=0}^{\ell} Q_i(X)Y^i, \\
Q(\alpha_i, y_i) & = 0, \ \forall i \in \{1, \ldots, n\}, \\
\deg Q_j & \leq n - T - 1 - j(k - 1), \ \forall j \in \{0, \ldots, \ell\}.
\end{align*}
\]
We can summarize this method:

**Algorithm 2: Sudan**

**Input:** The received word \( y \in F_q^n \) and the Reed-Solomon code \( C \).

**Output:** A list of codewords \( c_i \) of \( C \), such that \( \forall i, \, d(v, c_i) \leq T \).

begin
\[
Q(X, Y) \leftarrow \text{Interpolation}(IP, C)
\]
\[
(P_1, \ldots, P_\ell) \leftarrow Y\text{-Roots}(Q(X, Y))
\]
\[
\text{Candidate} \leftarrow \{\}
\]
for \( i \in \{1, \ldots, \ell\} \) do
\[
\text{if } d(P_i(\alpha), y) \leq T \text{ then }
\]
\[
\text{Candidate} \leftarrow \text{Candidate} \cup \{P_i(\alpha)\}
\]
return Candidate.

2.1.3 Guruswami-Sudan

Since the Guruswami-Sudan algorithm introduces the notion of root with multiplicity from Sudan algorithm [GS99]. Let us to recall the definition of the Hasse derivative.

**Definition 2 (Hasse derivative).** Let \( Q(X, Y) \in F_q[X, Y] \) be a bivariate polynomial and \( a, b \) be two positive integers. The \((a, b)\)-th Hasse derivative of \( Q \) is
\[
Q^{[a,b]}(X, Y) \triangleq \sum_{i=a}^{\deg_X(Q)} \sum_{j=b}^{\deg_Y(Q)} \binom{i}{a} \binom{j}{b} q_{i,j} X^{i-a} Y^{j-b}.
\]

Thanks to the Hasse derivative, we can give the definition of the root with multiplicity higher than one.

**Definition 3 (Root with multiplicity).** Let \( Q(X, Y) \in F_q[X, Y] \) be a bivariate polynomial and \( (\alpha, \beta) \in (F_q)^2 \) be a point. The point \((\alpha, \beta)\) is a root with multiplicity \( s \in \mathbb{N} \) if and only if \( s \) is the largest integer such that for all \( i + j < s \)
\[
Q^{[i,j]}(\alpha, \beta) = 0.
\]

Guruswami and Sudan noticed that it could happen that for some two polynomials \( P_{i_0}, P_{j_0} \), we have \( y_{i_0} = P_{i_0}(\alpha_{k_0}) = P_{j_0}(\alpha_{k_0}) \) and so the point \((\alpha_{k_0}, y_{i_0})\) is a root of \( Q \) with multiplicity at least 2. So they proposed to add multiplicity constraint during the bivariate interpolation step.

\[
(\mathbb{IP}_{GS}) \triangleq \left\{ \begin{array}{ll}
0 \neq Q(X, Y) \triangleq \sum_{i=0}^\ell Q_i(X)Y^i, \\
Q(\alpha_i, y_i) = 0, \text{ with multiplicity } s, \forall i \in \{1, \ldots, n\}, \\
\deg(Q_j) \leq s(n - T) - 1 - j(k - 1), \forall j \in \{0, \ldots, \ell\}.
\end{array} \right.
\]

thus the pseudo-code of this method is given by:
Algorithm 3: Guruswami-Sudan

Input: The received word \( y \in F_q^n \) and the Reed-Solomon code \( C \).
Output: A list of codewords \( c_i \) of \( C \), such that \( \forall i, d(v, c_i) \leq T \).

begin
\[ Q(X,Y) \leftarrow \text{Interpolation}(\mathbb{IP}_{GS}, C) \]
\( (P_1, \ldots, P_\ell) \leftarrow \text{Y-Roots}(Q(X,Y)) \)
\( \text{Candidate} \leftarrow \{\} \)
for \( i \in \{1, \ldots, \ell\} \) do
\[ \text{if } d(P_i(\alpha), y) \leq T \text{ then } \]
\( \text{Candidate} \leftarrow \text{Candidate} \cup \{P_i(\alpha)\} \)
return \( \text{Candidate} \).

2.2 Re-encoding

Definition 4 (Interpolation problem). Let \( \mathcal{P} \triangleq \{(\alpha_1, y_1), \ldots, (\alpha_n, y_n)\} \subset (F_q \times F_q)^n \). The interpolation problem with multiplicity \( s \) associated to \( \mathcal{P} \), \( \mathbb{IP}(\mathcal{P}, s) \), consists in finding \( Q(X, Y) \) such that the points \((\alpha_i, y_i)\) are a root of \( Q(X, Y) \) with multiplicity at least \( s \).

Lemma 1. Let \( s \) be an integer, \( \alpha, \beta \in F_q \) and \( Q(X, Y) \in F_q[X, Y] \) a bivariate polynomial such that the point \((\alpha, \beta)\) is a root of \( Q \) with multiplicity \( s \). Then for all univariate polynomial \( P \) such that \( P(\alpha) = \beta \), we have
\[ (X - \alpha)^s \mid Q(X, P(X)). \]

Proof. See [Gur05, Lemma 6.6, p. 103]. \( \square \)

We can generalize the previous lemma for all interpolation points, taking care the multiplicity.

Proposition 1. Let \( \mathcal{P} \subset (F_q \times F_q)^n \) and \( s \) be a positive integer. The polynomial \( Q(X, Y) \) is a solution of \( \mathbb{IP}(\mathcal{P}, s) \) if and only if
\[ \forall b \in \{0, \ldots, s-1\}, \prod_{i=1}^{n}(X - \alpha_i)^{s-b} \mid Q^{[b]}(X, L(X)), \]
where \( Q^{[b]}(X, Y) = Q^{[0,b]}(X, Y) \) is the \( b \)-th Hasse derivative in \( Y \), and \( L(X) \) is the Lagrange polynomial of \( \mathcal{P} \), that is for all \( i \in \{1, \ldots, n\} \), \( L(\alpha_i) = y_i \).

Proof. See [AZ08, Proposition 1]. \( \square \)

Let \( \mathcal{P} \subset (F_q \times F_q)^n \) and \( L_k(X) \) be the Lagrange polynomial on \( k \) elements of \( \mathcal{P} \), without lost in generality, assuming the \( k \) first positions. Let
\[ \mathcal{P}_n = \left\{ \left( \alpha_i, y_i - L_k(\alpha_i) \right) : \forall i \in \{1, \ldots, n\} \right\}. \]
Then for all \( i \in \{1, \ldots, k\} \), \( r_i = 0 \).
Proposition 2. Let $\mathcal{P} \subset (F_q \times F_q)^n$ and $s$ be a positive integer. The polynomial $Q(X,Y)$ is a solution of $\mathbb{IP}((\mathcal{P},s)$ if and only if $Q(X,Y + L_k(X))$ is a solution of $\mathbb{IP}((\mathcal{P},s)$.

Proof. See [KMV11, Theorem 3]. \hfill \Box

3 Revisited re-encoding

3.1 Re-encoding and interpolation algorithm

A problem occurs with the re-encoding process: we have to modify the interpolation algorithm in order to take care of the $k$ first interpolation points in order to speed up the computation. So for each interpolation algorithm we have to adapt the initialization step to have the total benefits of the re-encoding step. As far we know, only the Koetter interpolation algorithm was modified to perform it. Although lot of interpolation algorithms were proposed, we can for the moment, use the re-encoding trick only using the Koetter interpolation algorithm.

3.2 Revisited re-encoding

Let $L_n(X)$ be the interpolation Lagrange polynomial of the set $\mathcal{P}_n$. Thus $\forall i \in \{1, \ldots, n\}, L_n(\alpha_i) = r_i$, and $\deg(L_n) \leq n-1$. Since for all $i \in \{1, \ldots, k\}, r_i = 0$, it exists the polynomial $L_{n-k}(X)$ such that

$$L_{n-k}(X) = \frac{L_n(X)}{\prod_{i=1}^{k}(X - \alpha_i)}.$$

Thanks to the previous remark on the Lagrange polynomials, we deduce the following proposition which is the key ingredient of our reformulation.

Proposition 3. Let $\mathcal{P}_{n-k} \triangleq \{(\alpha_{k+1}, L_{n-k}(\alpha_{k+1})), \ldots, (\alpha_n, L_{n-k}(\alpha_n))\} \subset (F_q \times F_q)^{n-k}$,

$$R(X,Y) = \sum_{j=0}^{\deg_Y(R)} R_j(X)Y^j,$$

be a bivariate polynomial over $F_q$ and $s$ be a positive integer such that $s \leq \deg_Y R$. The polynomial $R$ is a solution of $\mathbb{IP}(\mathcal{P}_{n-k},s)$ if and only if

$$Q(X,Y) = \sum_{j=0}^{\deg_Y(R)} \left(R_j(X) \prod_{i=1}^{k}(X - \alpha_i)^{s-j}\right)Y^j,$$

is a solution of $\mathbb{IP}(\mathcal{P}_n,s)$.
Proof. Since \( R \) is a solution of \( \mathbb{P}(\Psi_{n-k}, s) \), then for all \( b \in \{0, \ldots, s-1\} \)
\[
\prod_{i=k+1}^{n} (X - \alpha_i)^{s-b} \mid R^{[b]}(X, L_{n-k}(X))
\]
\[
\prod_{i=k+1}^{n} (X - \alpha_i)^{s-b} \mid \sum_{j=b}^{\deg_{Y}(R)} \binom{j}{b} R_j(X)(L_{n-k}(X))^j - b
\]
\[
\prod_{i=1}^{n} (X - \alpha_i)^{s-b} \mid \sum_{j=b}^{\deg_{Y}(R)} \binom{j}{b} R_j(X) \prod_{i=1}^{k}(X - \alpha_i)^{s-j} (L_n(X))^j - b
\]
\[
\prod_{i=1}^{n} (X - \alpha_i)^{s-b} \mid Q(X, L_n(X)).
\]

Since \( s \geq \deg_{Y} R, s - j \geq 0 \), the statement is hold. \( \square \)

Thanks to the Proposition we can compute a solution of the interpolation problem on \( \Psi = \{(\alpha_1, y_1), \ldots, (\alpha_n, y_n)\} \) from \( \Psi_n = \{(\alpha_1, 0), \ldots, (\alpha_k, 0), (\alpha_{k+1}, y_{k+1} - L_k(\alpha_{k+1})), \ldots, (\alpha_n, y_n - L_k(\alpha_n))\} \). Our revisited re-encoding could be seen as a decoding on the puncturing code. Since the Reed-Solomon code are MDS, the punctured code has the same dimension and is also a Reed-Solomon code. We could imagine to reiterate the re-encoding process taking \( \Psi_{n-k} = \Psi' \), then the decoding will make on the multi puncturing code and the correction radius will increase.

The Proposition is under the assumption that the multiplicity \( s \) is greater or equal than the \( Y \)-degree of \( R \), a solution of \( \mathbb{P}(\Psi_{n-k}, s) \). Which is not a problem, because a solution of \( \mathbb{P}(\Psi_{n-k}, s + k) \), for all positive integer \( k \), is also a solution of \( \mathbb{P}(\Psi_{n-k}, s) \). However, this artificial augmentation of the multiplicity could increase also the \( X \)-degree of the solution, and so introduces some issue for the interpolation problem related to the decoding. This is why we deal only with the Welch-Berlekamp algorithm in Section 4.

4 Application to the Welch-Berlekamp algorithm

4.1 Straightforward application

In the Welch-Berlekamp decoding context, use the principle of the revisited re-encoding, is straightforward. Indeed, the only one condition in order to make practical our re-encoding is the multiplicity \( s \) is greater or equal than the \( Y \)-degree of the bivariate polynomial to compute. In the Welch-Berlekamp context the multiplicity \( s \) is exactly equal to the \( Y \)-degree, that is 1. Let \( S(X, Y) = S_0(X) + Y S_1(X) \in \mathbb{F}_q[X, Y] \) be a solution of \( \mathbb{P}(\Psi_{n-k}, 1) \), then \( R \) given by the
Proposition 3

\[ R(X, Y) = S_0(X) \prod_{i=1}^{k} (X - \alpha_i) + Y S_1(X), \]

is a solution of the \( \mathbb{P}(\mathcal{P}_n, 1) \). Keeping the same notations and using the Proposition 2, we deduce directly a solution of the interpolation problem \( \mathbb{P}(\mathcal{P}, 1) \) from the simpler one \( \mathbb{P}(\mathcal{P}_{n-k}, 1) \). Let \( Q(X, Y) \in \mathbb{F}_q[X, Y] \) such that

\[ Q(X, Y) = R(X, Y + L_k(X)) = \left( S_0(X) \prod_{i=1}^{k} (X - \alpha_i) + L_k(X) S_1(X) \right) + Y S_1. \]

In order to satisfy the interpolation conditions of the Welch-Berlekamp algorithm, we must have:
\[ \deg S_1 \leq n - t - k \text{ and } \deg S_0 \leq n - t - 1 - k. \]

It can be rewritten as
\[ \forall j \in \{0, 1\}, \ \deg S_j \leq n - t - k - 1 - j(-1). \]

We deduce that the weighted-degree changes during the bivariate interpolation. Using the example describes below, we have to interpolate \( n - k \) points with the weighted-degree equal to -1, instead of to interplolate \( n \) points with the weighted-degree \( k - 1 \), without modify the intern state of the interpolation algorithm. Let us to illustrate our claim by a toy example.

**Example 1.** Let \( \mathbb{F}_8, \alpha \) be a 7-th primitive root of the unity such that \( \alpha^3 + \alpha + 1 = 0 \), \( C \) be the Reed-Solomon code \( \text{RS}(\alpha^i)_{i=0,\ldots,6,2} \) over \( \mathbb{F}_8 \). Hence the Welch-Berlekamp method can correct up to \( \left\lfloor \frac{d-1}{2} \right\rfloor = 2 \) errors. Let \( P(X) = \alpha^6 X + \alpha^5 \in \mathbb{F}_8[X] \) be the message under its polynomial form. The associated codeword is then \( (\alpha, \alpha^4, \alpha^6, \alpha^3, \alpha^2, 1, 0) \). Assume there are 2 errors occur during the transmission in the first and 5-th positions, the received word is \( (\alpha^5, \alpha^4, \alpha^6, \alpha^3, \alpha^2, 1, 0) \).

Now let us to perform the revisited re-encoding. Using the previous notations, the Lagrange interpolation polynomial of the original interpolation points set \( \mathcal{P}_n = \{(1, \alpha^5), (\alpha, \alpha^4), (\alpha^2, \alpha^6), (\alpha^3, \alpha^3), (\alpha^4, \alpha^3), (\alpha^5, 1), (\alpha^6, 0)\} \) is

\[ L_n = X^6 + \alpha^4 X^4 + \alpha^2 X^3 + \alpha^3 X^2 + \alpha^2 X + \alpha^2. \]

Assume that we want to vanish the 2 first points, then the Lagrange interpolation polynomial on these points is \( L_k = \alpha^4 X + 1 \), and the quotient

\[ L_{n-k} = \frac{L_n(X)}{(X-1)(X-\alpha)} = X^4 + \alpha^3 X^3 + X^2 + X + \alpha. \]

Then the new interpolation points set is

\[ \mathcal{P}_{n-k} = \{(\alpha^2, \alpha^4), (\alpha^3, \alpha^2), (\alpha^4, 0), (\alpha^5, \alpha^6), (\alpha^6, \alpha)\}. \]
Hence the bivariate polynomial which interpolates $\mathfrak{P}_{n-k}$ with multiplicity $s = 1$ and weighted-degree -1 is

\[
S(X, Y) = Y(\alpha^6 X^2 + \alpha^4 X + \alpha^3) + \alpha^2 X + \alpha^6. = S_1 = S_0
\]

We deduce the polynomial which interpolates the $\mathfrak{P}_n = \{(1, 0), (\alpha, 0), (\alpha^2, \alpha^4), (\alpha^3, \alpha^2), (\alpha^4, 0), (\alpha^5, \alpha^6), (\alpha^6, \alpha)\}$, is

\[
R(X, Y) = Y S_1(X) + (X - 1)(X - \alpha)S_0(X) = Y(\alpha^6 X^2 + \alpha^4 X + \alpha^3) + \alpha^2 X^3 + \alpha X^2 + \alpha^5 X + 1.
\]

To finish the reconstruction step of the interpolation, we compute

\[
Q(X, Y) = R(X, Y + L_n(X)) = Y(\alpha^6 X^2 + \alpha^4 X + \alpha^3) + \alpha^5 X^3 + \alpha^6 X^2 + \alpha = Q_1 = Q_0.
\]

In the Welch-Berlekamp algorithm the $Y$-root search is trivial. Indeed, it consists only in the division of the $Q_0$ by $Q_1$

\[
P = -\frac{Q_0}{Q_1} = \alpha^6 X + \alpha^5,
\]

which is exactly the sent message under the polynomial form.

### 4.2 Performance

Since it is one of the main goals of this article, we assume that we cannot modify the intern state of the interpolation algorithm. The asymptotic complexity of Koetter algorithm is $O(LN^2)$ where $L$ is the $Y$-degree of the bivariate polynomial $Q$ and $N$ the number of the linear constraints given by the interpolation conditions. Then the complexity of the standard Welch-Berlekamp algorithm is $O(n^3)$. While the complexity of the original re-encoding is also $O(n^3)$, there is a non trivial speed up, since the first coordinates are 0, our revisited re-encoding exhibits an asymptotic complexity of $O((n-k)^2)$. We perform some timing tests with a naive implementation of the Welch-Berlekamp algorithm in the high level computer algebra system: MAGMA [BCP97]. We propose to compare 3 decoding methods: Welch-Berlekamp algorithm without re-encoding, Welch-Berlekamp algorithm with the original re-encoding, and finally the Welch-Berlekamp with our revisited re-encoding. These 3 decoding methods were implemented with the same interpolation function, without modification or particular parameterization. The experimentations were done on a 2.13GHz Intel(R) Xeon(R). The timings presented in the Table provided are in seconds unit for 100 iterations for each set of parameters.
Table 1: Comparison between the Welch-Berlekamp without re-encoding, with original re-encoding and revisited re-encoding. The shown timings are in second unit for 100 computations.

<table>
<thead>
<tr>
<th>$m$</th>
<th>$\mathcal{C}$</th>
<th>usual</th>
<th>original re-encoding</th>
<th>revisited re-encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>RS[15, 8]</td>
<td>0.270</td>
<td>0.230</td>
<td>0.090</td>
</tr>
<tr>
<td></td>
<td>RS[15, 10]</td>
<td>0.290</td>
<td>0.210</td>
<td>0.080</td>
</tr>
<tr>
<td></td>
<td>RS[15, 12]</td>
<td>0.250</td>
<td>0.180</td>
<td>0.050</td>
</tr>
<tr>
<td></td>
<td>RS[15, 14]</td>
<td>0.230</td>
<td>0.180</td>
<td>0.050</td>
</tr>
<tr>
<td>5</td>
<td>RS[31, 16]</td>
<td>0.930</td>
<td>0.760</td>
<td>0.250</td>
</tr>
<tr>
<td></td>
<td>RS[31, 20]</td>
<td>0.820</td>
<td>0.710</td>
<td>0.220</td>
</tr>
<tr>
<td></td>
<td>RS[31, 24]</td>
<td>0.820</td>
<td>0.660</td>
<td>0.100</td>
</tr>
<tr>
<td></td>
<td>RS[31, 28]</td>
<td>0.840</td>
<td>0.550</td>
<td>0.080</td>
</tr>
<tr>
<td>6</td>
<td>RS[63, 32]</td>
<td>3.440</td>
<td>3.130</td>
<td>1.070</td>
</tr>
<tr>
<td></td>
<td>RS[63, 40]</td>
<td>3.480</td>
<td>2.890</td>
<td>0.650</td>
</tr>
<tr>
<td></td>
<td>RS[63, 48]</td>
<td>3.460</td>
<td>2.580</td>
<td>0.390</td>
</tr>
<tr>
<td></td>
<td>RS[63, 56]</td>
<td>3.350</td>
<td>2.300</td>
<td>0.260</td>
</tr>
<tr>
<td>7</td>
<td>RS[127, 64]</td>
<td>16.760</td>
<td>15.220</td>
<td>4.440</td>
</tr>
<tr>
<td></td>
<td>RS[127, 96]</td>
<td>17.400</td>
<td>13.160</td>
<td>1.260</td>
</tr>
<tr>
<td></td>
<td>RS[127, 112]</td>
<td>17.780</td>
<td>11.600</td>
<td>0.560</td>
</tr>
<tr>
<td>8</td>
<td>RS[255, 128]</td>
<td>100.780</td>
<td>92.070</td>
<td>21.150</td>
</tr>
<tr>
<td></td>
<td>RS[255, 160]</td>
<td>104.100</td>
<td>88.200</td>
<td>11.300</td>
</tr>
<tr>
<td></td>
<td>RS[255, 192]</td>
<td>109.840</td>
<td>83.910</td>
<td>5.110</td>
</tr>
<tr>
<td></td>
<td>RS[255, 224]</td>
<td>113.550</td>
<td>74.440</td>
<td>1.950</td>
</tr>
</tbody>
</table>

5 Conclusion

We introduce a new reformulation of the re-encoding process which permits to make it usable with any interpolation algorithm. However the assumption that the multiplicity $s$ is smaller than the $Y$-degree is the price to be generic. We perform different tests with the Welch-Berlekamp algorithm showing that our reformulation provides an very important gain.

References


