

Efficient erasure decoding of Reed-Solomon codes

Frédéric Didier

EPFL, IC - IIF - ALGO, Bâtiment BC,

Station 14, CH - 1015 Lausanne

frederic.didier@epfl.ch

Abstract—We present a practical algorithm to decode erasures of Reed-Solomon codes over the q elements binary field in $O(q \log_2^2 q)$ time where the constant implied by the O -notation is very small. Asymptotically fast algorithms based on fast polynomial arithmetic were already known, but even if their complexity is similar, they are mostly impractical. By comparison our algorithm uses only a few Walsh transforms and has been easily implemented.

I. INTRODUCTION

A linear error-correcting code of dimension k and block length n over a finite field \mathbf{F}_q is a k -dimensional linear subspace of the space \mathbf{F}_q^n . Elements of this subspace are called codewords. A linear code also comes with an encoding function that maps in a unique way an element of \mathbf{F}_q^k (the message) into a codeword. By erasure decoding, we mean the task of recovering the message knowing only a subset of the coordinates of its encoding.

Of course, if we know fewer than k coordinates of a codeword, there is always more than one possible corresponding message. Thus, a code is optimal with respect to recovering erasures if given any subset of k coordinates of a codeword, there is only one possible corresponding message. Such a code is called maximum distance separable (MDS) code. A standard and famous class of MDS codes is given by Reed-Solomon codes [4]. Such a code is obtained by evaluating polynomials over \mathbf{F}_q with degree less than k at n different points in \mathbf{F}_q . Their length n is thus bounded by the number of points in \mathbf{F}_q , that is q .

Decoding erasures of any linear code can be done by a simple Gaussian elimination with $O(k^3)$ operations. For Reed-Solomon codes, classical algorithms can decode in $O(k^2)$ and encode in $O(kn)$. Theoretically, using fast polynomial arithmetic [1], we can encode and decode them in $O(n \log^2 n \log \log n)$ [4, p. 369]. However, the algorithms involved are complex and there is a large constant hidden in this asymptotic complexity. Hence, from a practical point of view, only the quadratic time algorithms were useful, see for instance [2].

In this paper, we present a practical algorithm that can decode erasures of a Reed-Solomon codes over \mathbf{F}_q with $q = 2^m$ in $O(q \log_2^2 q)$ time. It uses $O(q \log_2 q)$ memory, but we also have a version using $O(q)$ memory with complexity $O(q \log_2^3 q)$. Here, the field operations are counted as $O(1)$ and the memory to store one field element is counted as $O(1)$ too.

This algorithm is simple (our full C implementation is less than 500 lines of code) and has a very small constant in its complexity. It uses the Walsh transform instead of the discrete Fourier transform used in previous asymptotically fast algorithms. This is possible because we actually never compute the coefficients of the involved polynomials, but just manipulate their Lagrange form at the points we received.

Notice that we can use the same algorithm to encode Reed-Solomon codes in a systematic way by choosing the first k positions of a codeword and then erasure decoding. Notice as well that the complexity does not depend on k or n , so we better use Reed-Solomon codes of length close to q and a dimension k of the same order. In the case where only a few systematic symbols are missing, we can also decode in $O(q \log_2 q)$ operations plus $O(k)$ operations per missing symbol.

Erasures codes with faster encoding and decoding complexity exist [3]. Such codes are binary codes and thus cannot be MDS, that is they require a little more than k symbols to be able to recover the message. Due to their low complexity and binary nature, these codes have many practical applications, but in some situations it may be better to use the classical Reed-Solomon codes. This is in particular the case when high rate codes are needed or when one prefers not to waste any redundancy at the price of a slightly higher complexity.

The organization of the paper is quite straightforward. We start by presenting our algorithm outline before discussing in detail its two main steps. We also recall on the way basic facts about the Walsh transform.

II. ALGORITHM OUTLINE

We start by fixing some notation. We will mainly work on the binary field \mathbf{F}_{2^m} with q elements. We will use the notation \oplus for the addition in this field to avoid confusion with the normal addition. Seeing this field as an m dimensional space over \mathbf{F}_2 , we represent its elements as binary vectors \mathbf{x} of length m .

The codewords of a Reed-Solomon code of dimension k over \mathbf{F}_{2^m} are in one-to-one correspondence with the polynomials of degree less than k over \mathbf{F}_{2^m} . Given such a polynomial P , we will take for its corresponding codeword the evaluation of P at all the points \mathbf{x} of \mathbf{F}_{2^m} , that is the image vector $(P(\mathbf{x}), \mathbf{x} \in \mathbf{F}_{2^m})$. It is of course possible to use smaller length, but our algorithm will recover the full image vector anyway.

We will always order the points $\mathbf{x} = (x_1, \dots, x_m)$ of \mathbf{F}_{2^m} by lexicographical order over the binary vector (x_1, \dots, x_m) . Moreover, for any function F defined over \mathbf{F}_{2^m} , we will write $[F]$ for its image vector over all the q points of \mathbf{F}_{2^m} ordered by this order.

Suppose now that we see only k points (or more) of a given vector $[P]$, we will show that we can then compute all the points of this vector in $O(q \log_2^2 q)$ time. This operation is sufficient to both encode and decode the code. To encode in a systematic way, we can set the first k positions of the vector $[P]$ as we want and then use the algorithm to compute the parity symbols. To decode, we will just reconstruct the vector from any k symbols and read the encoded information at the beginning of the vector.

We will write $R \subseteq \mathbf{F}_{2^m}$ for a set of k positions among the received ones. Since P is a polynomial of degree less than k , it is uniquely determined by its values at k points of the field \mathbf{F}_{2^m} . Using the well known Lagrange interpolation formula, we have:

Definition 1 (Lagrange Form): Given the values of a polynomial P of degree less than k on a subset R of k points in \mathbf{F}_{2^m} , its Lagrange form is

$$P(\mathbf{x}) = \bigoplus_{\mathbf{u} \in R} \mathbf{c}_{\mathbf{u}} \prod_{\mathbf{y} \in R, \mathbf{y} \neq \mathbf{u}} (\mathbf{x} \oplus \mathbf{y}) \quad (1)$$

where the $\mathbf{c}_{\mathbf{u}}$ are the Lagrange coefficients and belong to \mathbf{F}_{2^m} .

Our algorithm works in two steps that we will detail in the next two sections. The first step is to compute the coefficients $\mathbf{c}_{\mathbf{u}}$ and it runs in time $O(q \log_2 q)$. The second step is to evaluate the Lagrange form of P at all the points of \mathbf{F}_{2^m} and runs in time $O(q \log_2^2 q)$. Both steps rely heavily on Walsh transform computation.

III. COMPUTING LAGRANGE COEFFICIENTS

Theorem 1: Given the values at k points of a polynomial of degree less than k over \mathbf{F}_{2^m} , we can compute its Lagrange coefficients at these points in $O(q \log_2 q)$ time and $O(q)$ memory.

If we apply the formula (1) at a point $\mathbf{x} \in R$ where we know $P(\mathbf{x})$, we have

$$P(\mathbf{x}) = \mathbf{c}_{\mathbf{x}} \prod_{\mathbf{y} \in R, \mathbf{y} \neq \mathbf{x}} (\mathbf{x} \oplus \mathbf{y}) \quad \forall \mathbf{x} \in R. \quad (2)$$

The main difficulty to compute the coefficients $\mathbf{c}_{\mathbf{x}}$ is then to evaluate the product, that is

$$\Pi(\mathbf{x}) := \prod_{\mathbf{y} \in R, \mathbf{y} \neq \mathbf{x}} (\mathbf{x} \oplus \mathbf{y}). \quad (3)$$

We define a function R from \mathbf{F}_{2^m} into $\{0, 1\}$ such that $R(\mathbf{x}) = 1$ if and only if $\mathbf{x} \in R$. We can see this function as the indicator function of the received positions. With this definition, we can rewrite the product (3) as

$$\Pi(\mathbf{x}) = \prod_{\mathbf{y} \in \mathbf{F}_{2^m}, \mathbf{y} \neq \mathbf{x}} (\mathbf{x} \oplus \mathbf{y})^{R(\mathbf{y})}. \quad (4)$$

By fixing a primitive element α of the multiplicative group $\mathbf{F}_{2^m}^*$, we can also define the discrete logarithm function $L : \mathbf{F}_{2^m}^* \rightarrow [0, q-1]$ such that $L(\mathbf{x}) = i$ if and only if $\mathbf{x} = \alpha^i$. If we extend this function to \mathbf{F}_{2^m} by setting $L(\mathbf{0}) := 0$ we have

$$L(\Pi(\mathbf{x})) = \sum_{\mathbf{y} \in \mathbf{F}_{2^m}} R(\mathbf{y}) L(\mathbf{x} \oplus \mathbf{y}). \quad (5)$$

and more importantly $\alpha^{L(\Pi(\mathbf{x}))} = \Pi(\mathbf{x})$ for all \mathbf{x} in \mathbf{F}_{2^m} . This is because $\Pi(\mathbf{x})$ is never equal to zero. For someone familiar with the Walsh transform, it is now easy to notice that the value of the expression (5) can be computed for all \mathbf{x} in $O(q \log_2 q)$ operations. Of course we first have to compute the vector $[L]$ but this can be done in linear time and we can even precompute its Walsh transform.

Since the Walsh transform plays a key role in this paper, we recall how it works. The Walsh transform \widehat{R} of a function $R : \mathbf{F}_{2^m} \rightarrow \mathbf{Z}$ is a linear transform defined by

$$\widehat{R}(\mathbf{x}) = \sum_{\mathbf{y} \in \mathbf{F}_{2^m}} R(\mathbf{y}) (-1)^{\mathbf{x} \cdot \mathbf{y}}. \quad (6)$$

Moreover, we have the inverse formula

$$R(\mathbf{x}) = \frac{1}{q} \widehat{\widehat{R}}(\mathbf{x}) \quad (7)$$

and for two functions R and L from \mathbf{F}_{2^m} into \mathbf{Z} , if we define the convolution product $*$ by

$$(R * L)(\mathbf{x}) := \sum_{\mathbf{y} \in \mathbf{F}_{2^m}} R(\mathbf{y})L(\mathbf{x} \oplus \mathbf{y}) \quad (8)$$

we have

$$\widehat{R * L} = \widehat{R} \widehat{L} . \quad (9)$$

So we can compute (5) by performing three Walsh transforms. Finally, the Walsh transform can be computed efficiently in $O(q \log_2 q)$ by working on the image vector $[R]$ of a function thanks to the induction relation

$$[\widehat{R}] = [\widehat{R}_0 - \widehat{R}_1 \mid \widehat{R}_0 + \widehat{R}_1] \quad (10)$$

where both R_0 and R_1 are functions from $\mathbf{F}_{2^{m-1}}$ into \mathbf{Z} defined by $R_0(x_1, \dots, x_{m-1}) := R(0, x_1, \dots, x_{m-1})$ and $R_1(x_1, \dots, x_{m-1}) := R(1, x_1, \dots, x_{m-1})$. This algorithm is known as the fast Walsh transform.

We remark that for computing the Lagrange coefficients, we are only interested in the values modulo $q-1$. Since q is equal to 1 modulo $q-1$ we can perform all the above computation modulo $q-1$. With this tweak, the Walsh transform becomes involutive, and we do not need more than m bits per value.

IV. EVALUATING A LAGRANGE FORM

Theorem 2: Given the Lagrange form of a polynomial P over \mathbf{F}_{2^m} , we can compute its image vector $[P]$ in $O(q \log_2^2 q)$ time using $O(q \log_2 q)$ memory.

The idea to achieve this complexity start by rewriting the Lagrange form (1) of P as

$$P(\mathbf{x}) = \Pi(\mathbf{x}) \bigoplus_{\mathbf{y} \in R} \frac{\mathbf{c}_\mathbf{y}}{\mathbf{x} \oplus \mathbf{y}} \quad \forall \mathbf{x} \notin R . \quad (11)$$

To write this more conveniently, let us define an inverse function $I : \mathbf{F}_{2^m} \rightarrow \mathbf{F}_{2^m}$ that maps \mathbf{x} to \mathbf{x}^{-1} and $\mathbf{0}$ to $\mathbf{0}$. We define as well a coefficient function C from \mathbf{F}_{2^m} into \mathbf{F}_{2^m} that maps \mathbf{x} to $\mathbf{c}_\mathbf{x}$ if $\mathbf{x} \in R$ and to $\mathbf{0}$ otherwise. We then have

$$P(\mathbf{x}) = \Pi(\mathbf{x}) \bigoplus_{\mathbf{y} \in \mathbf{F}_{2^m}} C(\mathbf{y})I(\mathbf{x} \oplus \mathbf{y}) . \quad (12)$$

We already computed in the previous section the vector $[\Pi]$, so the only work left is to compute the sum on the right. This really looks like the convolution product defined in the last section except the functions now take their values in \mathbf{F}_{2^m} and not in \mathbf{Z} . To overcome this difficulty, we are going to look at the vectorial representation of the elements of \mathbf{F}_{2^m} over \mathbf{F}_2 .

We will need some more notation. For $i \in \{1, \dots, m\}$, we will write \mathbf{e}_i for the i -th elementary basis element of

the vectorial space \mathbf{F}_{2^m} over \mathbf{F}_2 . For a function C from \mathbf{F}_{2^m} into \mathbf{F}_{2^m} , we will write C_i for its i -th component, that is the function that maps \mathbf{x} into the coefficient of \mathbf{e}_i in $C(\mathbf{x})$. We can now rewrite (12) as

$$P(\mathbf{x}) = \Pi(\mathbf{x}) \bigoplus_{i=1}^m \bigoplus_{j=1}^m \mathbf{e}_i \mathbf{e}_j \bigoplus_{\mathbf{y} \in \mathbf{F}_{2^m}} C_i(\mathbf{y})I_j(\mathbf{x} \oplus \mathbf{y}) . \quad (13)$$

Now, the sum over \mathbf{y} that we have to compute m^2 times can be computed in $O(q \log_2 q)$ time. This is because we now have Boolean functions. We can thus see them as functions in \mathbf{Z} , compute

$$\left[\frac{1}{q} \widehat{C_i \widehat{I_j}} \right] \quad (14)$$

using the fast Walsh transform and take the parity of each values in the resulting vector. That is, we have

$$P(\mathbf{x}) = \Pi(\mathbf{x}) \bigoplus_{i=1}^m \bigoplus_{j=1}^m \mathbf{e}_i \mathbf{e}_j \text{parity} \left(\frac{1}{q} \widehat{C_i \widehat{I_j}}(\mathbf{x}) \right) . \quad (15)$$

Hence, we can evaluate the Lagrange form at all the points of \mathbf{F}_{2^m} in $O(q \log_2^3 q)$ operations. In order to be faster, we will choose $\mathbf{e}_i = \alpha^i$ for a primitive element α of $\mathbf{F}_{2^m}^*$. We can then rewrite the formula (15) as

$$P(\mathbf{x}) = \Pi(\mathbf{x}) \bigoplus_{s=0}^{2(m-1)} \alpha^s \bigoplus_{i+j=s} \text{parity} \left(\frac{1}{q} \widehat{C_i \widehat{I_j}}(\mathbf{x}) \right) . \quad (16)$$

Now, by using the linearity of the parity function and of the Walsh transform, we get

$$P(\mathbf{x}) = \Pi(\mathbf{x}) \bigoplus_{s=0}^{2(m-1)} \alpha^s \text{parity} \left(\frac{1}{q} \sum_{i+j=s} \widehat{C_i \widehat{I_j}}(\mathbf{x}) \right) . \quad (17)$$

If we compute and store the $[\widehat{C_i}]$ and the $[\widehat{I_j}]$ (which require $O(q \log_2 q)$ memory and $O(q \log_2^2 q)$ operations), we can evaluate this formula in $O(q \log_2^2 q)$. Indeed, we have to compute $2(m-1)$ Walsh transforms of functions which have an image vector that can be computed in $O(q \log_2 q)$.

We remark that since we only need the parity of the result, we can perform the Walsh transforms by keeping only the $m+1$ lower bits of all the values. Notice as well that we can precompute the $[\widehat{I_j}]$.

If the number of erased systematic symbols is small, an alternative decoding method is to evaluate (1) directly at these erased positions. So we still use $O(q \log_2 q)$ operations to compute the coefficients $\mathbf{c}_\mathbf{u}$ but then it is only $O(k)$ operation per erased message symbols.

Finally, an interesting fact is that k plays only a minor role in the implementation. We can just use all

the available points to decode and if we received more symbols than the degree of the polynomial sent, we will just recover the sent vector.

V. CONCLUSION

We presented a practical and fast algorithm to encode and decode Reed-Solomon codes over the erasure channel. This algorithm should be more efficient in software than the previous algorithms as soon as the size of the field is 2^{10} , or even smaller.

Moreover, it allows the use of long Reed-Solomon codes which may have some applications. For example, our implementation can encode and decode on $\mathbf{F}_{2^{16}}$ in less than a second or on $\mathbf{F}_{2^{20}}$ in a few seconds (on an Intel core 2, 1.86GHz). One may wonder where we can use such codes, but they may be worth considering, for example, to recover from failure on a storage system or to send big files over the Internet.

Finally, we considered only the case of a binary field. The same approach, namely computing Lagrange coefficients and evaluating the Lagrange form directly is certainly applicable on any field. However, generalizing the way we use Walsh transform is not straightforward and it seems unlikely that it will lead to a practical algorithm.

ACKNOWLEDGMENT

The author would like to thank Amin Shokrollahi for his help to improve this paper.

REFERENCES

- [1] Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft. *Data structures and algorithms / Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman*. Addison-Wesley, Reading, Mass. :, 1983.
- [2] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An xor-based erasure-resilient coding scheme. *Technical report, International Computer Science Institute, Berkeley, California*, 1995.
- [3] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman. Efficient erasure correcting codes. *Information Theory, IEEE Transactions on*, 47(2):569–584, Feb 2001.
- [4] F.J. MacWilliams and N.J.A. Sloane. *The theory of error-correcting codes*, volume 16. North holland mathematical library, 1977.