Recovering a Code's Length and Synchronisation from a Noisy Intercepted Bitstream

Mathieu Cluzeau, Matthieu Finiasz

Recovering a Code's Length and Synchronisation from a Noisy Intercepted Bitstream

Mathieu Cluzeau Matthieu Finiasz

***Abstract*—We focus on the problem of recovering the length and synchronization of a linear block code from an intercepted bitstream. We place ourselves in an operational context where the intercepted bitstream contains a realistic noise level. We present two algorithms, both useful in different contexts, able to verify if a given length/synchronization is correct. Using them, we were able to practically recover the synchronization of several codes.**

I. Introduction

Most digital communications are both coded and encrypted. For this reason, in order to be able to perform a cryptanalysis, it is usually necessary to decode intercepted data. However, decoding first requires to split the intercepted bitstream into codewords: this implies that the code length and synchronization have to be recovered. In this article, we only focus on communications encoded using linear block codes. Most other articles dealing with code reconstruction [8], [4], [5], [6] (that is recovering a parity check matrix) consider this information known. They deal with the (easier) problem of finding a parity check matrix from noisy codewords. In this article, we focus on the preceding step which consists in finding the block length and the synchronization leading to words as close as possible to a vector space. It appears that the most efficient techniques to solve this problem can also be used to reconstruct the code.

This article is composed of two main sections. First, we show that looking for words in the dual code is sufficient to decide if a specific length/synchronization is correct. Then, in the second part, we present two very different techniques for searching words in the dual code. Eventually, we present some experimental results and give estimates for the maximum noise level allowing to recover a code's length and synchronization.

Previous works. This article is not the first to deal with the problem of finding the length/synchronization of a linear block code bitstream. For instance, in [1] an interesting technique based on rank computation is presented. This technique consists in computing for all length and synchronization the rank of the matrix formed with the noisy codewords. If the noise level is low enough and the length and synchronization are correct, this matrix will not be of full rank. Finding the correct length/synchronization then simply consists in finding the minimum of these ranks. When the level of noise starts to increase, it is necessary to compute the rank of sub-matrices and hope to find "low noise zones". Because of this, this techniques is limited to relatively low noise levels. Moreover, rank computation can be quite expensive on large matrices.

II. Deciding whether a Given Length and Synchronization is Correct

In this section, we consider that the received bitstream was transmitted through a binary symmetric channel with cross-over probability τ. When trying to recover the length/synchronization of a code C, the first step is to be able to decide whether a given length/synchronization is correct or not. One must thus split the input bitstream into words of the given length, starting at the given synchronization, and then decide if the words obtained are indeed noisy codewords (that is, elements of a vector space with a small amount of noise). Of course, as the target vector space (the code C we are looking for) is unknown, this problem is hard. A simpler way to look at it is to consider the dual problem: instead of looking for a vector space, we can look for elements of its orthogonal (these are, words of the dual of C). Such orthogonal words have a probability higher than $\frac{1}{2}$ to be orthogonal to a noisy codeword (if the noise level is lower than $\frac{1}{2}$ of course). In order to decide if a length/synchronization is correct, one can thus look for dual words: as we will see, if such a word can be found then the length is correct (with a probability close to one) and the synchronization is probably not far from correct.

Suppose the correct length/synchronization is (n_0, s_0) and the length/synchronization we are testing is (n, s). After splitting the input bitstream into words of length n, we build a matrix $\mathcal{G}$ such that each line of $\mathcal{G}$ is a word. This matrix is of size $M \times n$ where $M = \lfloor \frac{\ell - s}{n} \rfloor$ if ℓ is the length of the intercepted bitstream. Looking for a word of the dual consists in finding a word h of length n such that $\mathcal{G} \times h$ is of *low weight*. We distinguish three different cases.

A. Correct length/synchronization: $n = n_0$, $s = s_0$

In this case, each line of $\mathcal{G}$ is a noisy codeword. Thus, if h is a word of the dual of C of weight w the weight of the product $\mathcal{G} \times h$ strongly depends on τ and follows a binomial

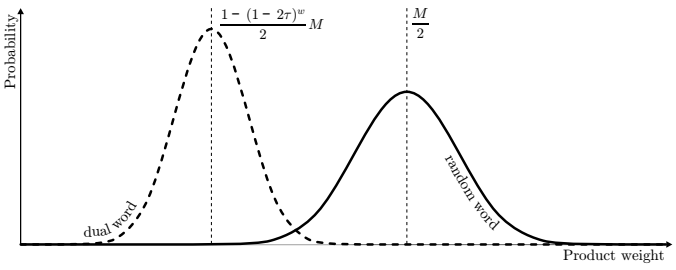

Figure 1. Distribution of the weight of the product $\mathcal{G} \times h$.

distribution, centered in $\frac{M}{2}\big(1-(1-2\tau)^w\big)$, with a variance of $\sigma^2 = \frac{M}{4}\big(1-(1-2\tau)^{2w}\big)$. This distribution is depicted in Figure 1 (dashed line). However, if h' is not a word of the dual of C, whatever its weight, the weight of the product $\mathcal{G} \times h'$ will follow a binomial distribution centered in $\frac{M}{2}$ with a variance $\frac{M}{4}$ (plain line in Figure 1).

If these two distributions have a small enough intersection, then it is possible to tell, with high probability, whether a word h is in the dual of C or not.

B. Correct length, incorrect synchronization: $n = n_0$, $s \neq s_0$

In this case, each line of $\mathcal{G}$ is composed of two different codewords: the first $s_0 - s \mod n$ bits belong to one word, the remaining bits to the next one (see Figure 2). We take a word h in the dual of C and cyclicly shift it (to the right) by $s_0 - s$ positions to obtain a word $\bar{h}$.

- If the support of $\bar{h}$ is included in $[s_0 - s, n-1]$ or $[0, s_0 - s - 1]$ then the product $\mathcal{G} \times \bar{h}$ will follow the same distribution as dual words in the previous case (the dotted line in Figure 1).
- If the support of $\bar{h}$ is split among the two intervals, then each bit of the product will be zero with probability $\frac{1}{2}$ (if we assume that the codewords are mutually independent). The weight of $\mathcal{G}\times\bar{h}$ will thus follow the same distribution as for random words.

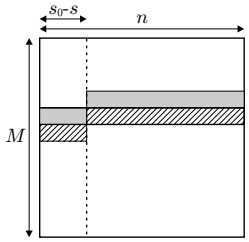

Figure 2. The matrix $\mathcal{G}$ when an incorrect synchronization was chosen.

Once again, if the two distributions are distant enough, it will be possible to decide whether a (low weight) word is in the dual of C or not. However, this will only work for words such that the support of $\bar{h}$ is not split. In practice, this will decrease the probability of finding words in the dual of C. The larger $s_0 - s$, the more this probability will decrease. If the chosen synchronization s is close to the correct synchronization s_0, the behavior of dual word finding algorithms will be nearly the same as in the first case.

C. Incorrect length: $n \neq n_0$

In this third case, as it can be seen in Figure 3, each codeword will have a different offset. There is a high probability that $\mathcal{G}$ is not close to any vectorial subspace. In practice, for any word h', the product $\mathcal{G} \times h'$ will follow the binomial distribution centered on $\frac{M}{2}$ represented by the plain line in Figure 1. The only case in which some words could follow a different distribution, is if n divides n_0 and each offset of a dual word h by n positions is also in the dual of C. If such an unlikely event occurs, a divisor of n_0 has however been found, which makes finding n_0 a much easier problem.

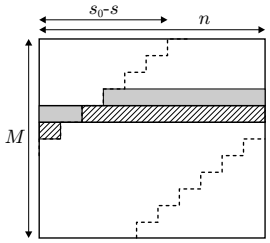

Figure 3. The matrix $\mathcal{G}$ when an incorrect length was chosen.

D. Analysis

What appears from the study of these three cases is that words following the dashed line distribution of Figure 1 can only be found when the *correct length* was chosen. If τ is known, for a given word weight w, the two distributions are known and it is thus possible to compute a threshold T. If we can find a word h such that the weight of $\mathcal{G}\times h$ is below T, then there is a high probability that the length n chosen is equal to n_0. This also means that the offset s is probably not too far from s_0. To find the exact value of s_0, it is necessary to look at the supports of dual words found. As we have seen in II-B the correct offset cannot split the support of any words. We can thus proceed by some kind of dichotomy but this requires to find several dual words for each tested offset. In the end, when the correct synchronization has been found, we usually have found enough dual words to reconstruct the complete code C.

We will now see two different algorithms to search for words in the dual of C. For each of these algorithms it is possible to estimate the number of tries needed to find one word. It is thus possible to know that a length/synchronization pair is incorrect after a certain number of unsuccessful tries.

III. Finding Words in the Dual of C

A. Exhaustive Search of Words of a Given Weight

The first thing to note when looking for words in the dual of a code is that words of (very) low weight are easier to find. First, the two distributions of Figure 1 are more distant from one another, secondly, it can be easy to exhaustively test all words of weight w. This is exactly what our first algorithm does, but in a more subtle way.

The straightforward exhaustive search technique consists in going through all words of weight w and for each of them, compute the weight of their product with $\mathcal{G}$. If one of these weights is below a threshold T, then the corresponding word likely belongs to the dual of C. In order to improve this technique, we use a birthday technique and build a list of all products $\mathcal{G} \times h_{\frac{w}{2}}$ where $h_{\frac{w}{2}}$ is a word of weight $\frac{w}{2}$. Finding a word in the dual then requires to find two words $h_{\frac{w}{2}}$ and $h'_{\frac{w}{2}}$ such that the weight of $\mathcal{G} \times (h_{\frac{w}{2}} \oplus h'_{\frac{w}{2}})$ is below T. In order to find such a pair efficiently, we select a window of size 32 which we require to fulfil $\mathcal{G} \times (h_{\frac{w}{2}} \oplus h'_{\frac{w}{2}}) = 0$: this will discard some valid $(h_{\frac{w}{2}}, h'_{\frac{w}{2}})$ pairs but we can now sort the list of $\mathcal{G} \times h_{\frac{w}{2}}$ products and simply look for exact matches. Once a match is found, we just have to check if the total weight of the product is below T.

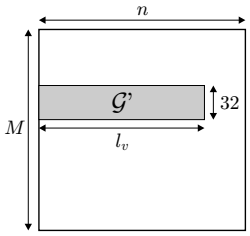

Figure 4. Selecting a submatrix $\mathcal{G}'$ for exhaustive search.

This technique makes it possible to find all words of weight w in the dual of C which vanish on the selected window with a complexity of order $O\big(n^{\frac{w}{2}}\log(n^{\frac{w}{2}})\big)$ and with a memory of $O\big(n^{\frac{w}{2}}\big)$. As this amount of memory can be very large, it is often necessary to choose a "vertical" window of size l_v and restrict our search to words of the dual with a support included in this window. In practice, we get the following algorithm:

- randomly pick a vertical window of size l_v,
- randomly pick an horizontal window of size 32 to obtain a matrix $\mathcal{G}'$ as described in Figure 4,
- compute all the xors of $\frac{w}{2}$ columns of $\mathcal{G}'$ and place them in a table,
- sort the elements of the table,
- for each collision, check if the weight of the same xor of w columns of $\mathcal{G}$ is of weight smaller than a threshold T.

Computing the threshold T. We need to select T in order to avoid all false alarms (that is, words not in the dual of C with a product by $\mathcal{G}$ below the threshold) and at the same time miss as few as possible dual words. This will be possible if the threshold can be chosen at more than 3 standard deviations from the center of each distribution. If M is large enough, this will be possible.

In order for the two "3 standard deviation" bounds to be in the correct order we need:

$$M > \left(\frac{3\sqrt{1-(1-2\tau)^{2w}}+1}{(1-2\tau)^w}\right)^2. \qquad (1)$$

If this inequality is verified, any threshold T between the two "3 standard deviations" bounds can be chosen and should give satisfactory results. In practice we choose to select T in the exact middle of this interval which, as we will see in the last section, gives very good results. This corresponds to:

$$T = \frac{M}{2}\Big(1-\frac{(1-2\tau)^w}{2}\Big)+3\frac{\sqrt{M}}{4}\big(\sqrt{1-(1-2\tau)^{2w}}-1\big).$$

Required number of tries. Suppose there are $\mathcal{N}_w$ words of weight w in the dual of C. Each run of the previous algorithm will output all words of the dual:

- of support included in the vertical window of size l_v,
- vanishing on the horizontal window of size 32.

Thus, each run will return an average number of words equal to:

$$\mathcal{N}_w \times \frac{\binom{l_v}{w}}{\binom{n}{w}} \times \Big(\frac{1+(1-2\tau)^w}{2}\Big)^{32}.$$

If $\mathcal{N}_w > 0$ (which can only be the case when the correct length was chosen), after t runs of the algorithm (each time with a different window choice), the probability of not having found any dual word is smaller than:

$$\mathcal{P} = \left(1-\frac{\binom{l_v}{w}}{\binom{n}{w}} \times \Big(\frac{1+(1-2\tau)^w}{2}\Big)^{32}\right)^t.$$

In order to recover the length of a code C we choose t such that $\mathcal{P}$ is small enough and try all possible lengths incrementally until a dual word is found. This technique will succeed as long as there exists at least one dual word of weight w.

B. Using the Canteaut-Chabaud Algorithm

As we will see in the practical experiments section, the previous algorithm can hardly be used for values of w larger than 8. However, for most codes, the minimal distance of their dual will be larger than 8. In order to deal with these codes, we propose to use another algorithm, based on the Canteaut-Chabaud information set decoding algorithm [3]. Here is how this algorithm works:

- select at random an "information set", that is, n lines among the M lines of $\mathcal{G}$
- perform a Gaussian elimination on this information set, swapping and xoring *columns* of $\mathcal{G}$ to obtain a new matrix $\mathcal{G}'$ (see Figure 5) and store the transition matrix P such that $P\mathcal{G} = \mathcal{G}'$
- choose a small window of l lines among the $M-n$ remaining lines of $\mathcal{G}$
- use the same technique as in the previous algorithm to find all combinations of $2p$ columns vanishing on the l lines of the window
- for each set of $2p$ columns, verify that the xor on the columns of $\mathcal{G}'$ is of weight lower than a threshold T
- each word h of weight $2p$ can be converted to a word of the dual $h' = h \times P$.

This can be implemented very efficiently using the Canteaut-Chabaud algorithm to select the information sets of successive iterations. In practice, this consists in only changing one position in the previous iteration information set so as to make the Gaussian elimination step less costly. Also, in order to optimize the probabilities, it is better to split the columns in two separate sets and look for collisions among words of weight p in each set. The optimal values for the two parameters l and p are chosen in the same way as in [2].

As for the previous algorithm, it would be interesting to know the probability of success of one iteration of this technique. A given word h of weight w in the dual of C will be found if the product $\mathcal{G} \times h$ is of weight:

- $2p$ on the chosen information set (in 2 sets of weight p),
- 0 on the window of size l,
- and less than $T-2p$ on the remaining positions.

For a given information set, the probability that the errors in $\mathcal{G}$ are well distributed for the previous conditions is:

$$\mathcal{P}_{\text{cor}} = \tbinom{n}{2p} q^{2p}(1-q)^{n-2p+l} \sum_{i=0}^{T-2p} \tbinom{M-n-l}{i} q^i (1-q)^{M-n-l-i}$$

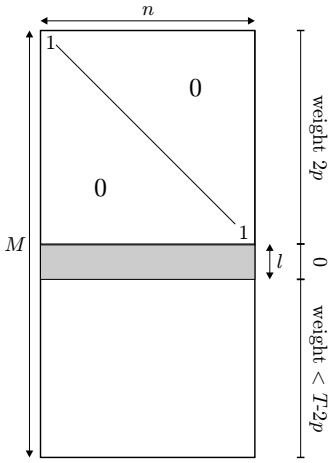

Figure 5. Using the Canteaut-Chabaud algorithm on matrix $\mathcal{G}$.

with $q = \frac{1-(1-2\tau)^w}{2}$. Thanks to this probability, it is possible to compute an estimate (neglecting the dependencies between successive iterations) of the average number of iterations required to find a given word h. However, there are many words in the dual of C and what interests us most is the average number of iterations required to find any one of these words. Unfortunately, this number is much harder to compute as it will depend on the (unknown) distribution of the words of the dual. For this reason, the threshold T will be chosen independently of this value.

We decide to choose the threshold T so as to minimize the probability of having a false alarm (that is, a word not in the dual verifying the weight conditions cited above). There are 2^n possible false candidates, and for each of them the probability of verifying the weight conditions is approximately:

$$\mathcal{P}_{\text{false}} = \frac{1}{2^M} \sum_{i=0}^{T} \binom{M}{i}.$$

If we want to avoid false alarms, we thus need $\mathcal{P}_{\text{false}} < 2^{-n}$. This is achieved by choosing:

$$T = \frac{M}{2} - \sqrt{M \frac{n \log 2}{2}}.$$

This choice makes sure that we avoid false alarms but gives us no hint about the probability of finding a word of the dual. However, what is known is that if we increase M (and thus also T), this probability of success will also increase. More details about the choice of M are given in the next section.

IV. Practical Experiments

We have previously seen how to test a length and synchronization and how to find words in the dual code of C. Our algorithms thus consist in testing all possible lengths and for each length a number of different synchronizations. In practice, for short codes, we test each length n from 1 to n_0 and $n/8$ synchronizations each time (we test all synchronizations which are a multiple of 8). For larger values of n (especially for LDPC codes), we test a fixed number of synchronizations for each length (every $\frac{n}{8}$ bits for instance). We divide our experiments in two groups which behave very differently in practice for both algorithms presented here.

A. Random Linear Codes

We first consider codes defined by a random generator matrix. For such codes, it appears that our first algorithm based on exhaustive search gives very poor results. This algorithm only works well for codes with very low weight words in their dual (weight 6 or 8 at most), which will be the case only for very short random codes (at most $n \approx 40$). Of course, such codes are seldom used in practice. However, the Canteaut-Chabaud-based algorithm makes it possible to find the length/synchronization of longer codes. Table I gives results of simulations on random codes of different lengths with different noise levels. For these lengths, it is important to note that the 10000 iterations are performed in approximately 1s. Thus, if the algorithm is able to find some words, it will also be easy to find the length/synchronization of the code. In practice, for all the length/noise combinations of Table I not containing a zero, the exact length/synchronization can be recovered in a few minutes.

Table I
Number of words found by 10000 iterations of the Canteaut-Chabaud algorithm on random codes of rate $\frac{1}{2}$. Here, $M = 5n$, and ∞ means that many words were found.

n \ τ	0.001	0.002	0.005	0.01	0.02	0.05
32	14637	27081	42570	42913	19464	210
64	∞	∞	∞	1172189	6310	0
128	∞	∞	∞	2992	0	0
256	∞	∞	0	0	0	0

For lengths longer than 256, this technique can still be successful but only if τ is very small or if particular codes with dual words of very low weight are used. This is for example the case with LDPC codes.

B. LDPC Codes

By nature, LDPC codes [7] have words of very low weight in their dual. This makes it much easier to recover their length/synchronization. Also, our first algorithm was specifically designed for such codes.

Exhaustive search. All our simulations were done on a computer with 2GB of memory for LDPC codes with parity checks of weight 6. In this setting, the value of l_v cannot be greater than 794 (with our implementation). Because of this, we managed to recover the length/synchronization of codes of length up to 1000. For longer codes, the probability of success of one iteration of the algorithm becomes too low and the number of iteration required makes a full length/synchronization recovery impractical. Table II gives the number of words of the dual found in 250 iterations of the algorithm on LDPC codes of length 1000 for different noise levels. Note that 250 such iterations take approximately 3 hours to run. One can thus see that for noise levels of 5-6% a complete length/synchronization

Table II
NUMBER OF WORDS FOUND BY 250 ITERATIONS OF THE EXHAUSTIVE SEARCH ALGORITHM ON LDPC CODES OF RATE $\frac{1}{2}$, LENGTH 1000 AND WEIGHT 6. HERE, $M = 512$. THE LAST COLUMN CORRESPONDS TO THE MINIMAL VALUE OF M GIVEN BY EQUATION (1).

τ	words found	expected words per iter.	expected total words found (including doubles)	minimal M
0.01	497	19.03	4760	56
0.02	365	3.15	787	70
0.03	117	0.57	142	88
0.04	24	0.11	28.1	110
0.05	3	0.024	6.1	140
0.06	3	0.006	1.5	180

recovery will take very long (probably a few month on a single computer). Table II also shows some interesting results:

- for $\tau \leq 0.02$, a single iteration of the algorithm is enough to verify if a length/synchronization is correct. Thus, with such noise levels, recovering the length/synchronization of the code takes less than a day. Moreover, this only requires very few intercepted words.
- for noise levels close to the correction capacity of the LDPC ($\tau = 0.06$ for instance) it is still possible to verify a length/synchronization pair. This was not obvious at first sight for such noise levels.

Soft information. LDPC codes are particularly efficient when it comes to soft information decoding. We have thus investigated the possibilities for adapting our exhaustive search algorithm to a soft information bitstream. The "vertical" window of size l_v must be chosen at random. Any bias will reduce the probability of finding some words of the dual. However, the "horizontal" window of size 32 can however be chosen so as to contain as few as possible errors. The soft information makes it possible to compute the probability that a given line of $\mathcal{G}$ contains no error. We thus sort the lines according to this probability and select lines with low probability of error. In order to randomize our selection (it has to be different for each iteration), we select each line with probability $\frac{1}{8}$ (this value is empirical but seems to give the best results) starting from the line with best probability and going down.

Experiments show that this technique is quite efficient: for an LDPC code of weight 6 with $n = 500$, $M = 1024$, $\tau = 0.01$, one iteration of the soft information algorithm returns 138 dual words in average whereas the hard information version only returns 80.

Canteaut-Chabaud algorithm. For an LDPC codes of rate $\frac{1}{2}$, the number of dual words of minimal weight w is $\frac{n}{2}$. Knowing this, it is possible to give an estimate of the maximum noise level $\tau_{\max}$ for which the Canteaut-Chabaud algorithm will find minimal weight dual words in a few seconds.

A dual word of weight w will be found if the chosen information set only contains few errors. Thus, the probability of finding one of the $\frac{n}{2}$ dual words of weight w is approximately:

$$\mathcal{P}_w = \frac{n}{2} \frac{\binom{\frac{M}{2}(1+(1-2\tau)^w)}{n}}{\binom{M}{n}}.$$

We thus define $\tau_{\max}$ by fixing this probability to 2^{-10} (meaning that 2^{10} iterations of the algorithm will be necessary):

$$\frac{n}{2} \frac{\binom{\frac{M}{2}(1+(1-2\tau_{\max})^w)}{n}}{\binom{M}{n}} = 2^{-10}.$$

This value is only an approximation of the maximum noise level that the Canteaut-Chabaud algorithm can handle. We experimented with an LDPC code of weight 6, rate $\frac{1}{2}$ and length 1000, with $M = 2048$. For such a code $\tau_{\max} = 0.0015$.

- for $\tau = \tau_{\max}$, the Canteaut-Chabaud algorithm was able to find 375 dual words in 1000 iterations (a few seconds).
- for $\tau = 0.0025$ we were still able to find a few dual words in 1000 iterations.
- for $\tau = 0.01$ we were not able to find any dual word, even in an hour of computation.

This algorithm thus only works for much lower noise levels than the exhaustive search, however, it is faster and can handle longer codes. We were able to successfully recover dual words for LDPC codes of length up to 10000, but for such lengths the maximum noise level is very low compared to the correction capacity of the LDPC.

Concerning soft information decoding with this algorithm we were not able to make any notable improvements.

V. CONCLUSION

We were able to recover the length/synchronization of various linear block codes. For random codes, our algorithms were successful only for short lengths (up to 256) with very low noise level (well below the correction capacity of the code). For LDPC codes, our experiments on codes with parity checks of weight 6 show that for length up to 1000 we can recover the length/synchronization of the code even for noise levels close to the correction capacity of the LDPC, for larger length, this will only be possible for very low noise levels.

REFERENCES

[1] J. Barbier, G. Sicot, and S. Houcke. Algebric approach of the reconstruction of linear and convolutional error correcting codes. In *CCIS 2006*.

[2] A. Canteaut. *Attaques de cryptosystèmes à mots de poids faible et construction de fonctions t-résilientes*. PhD thesis, Paris 6, 1996.

[3] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: application to primitive narrow-sense BCH codes of length 511. *IEEE Transaction on Information Theory*, 44(1):367–378, January 1998.

[4] C. Chabot. Recognition of a code in a noisy environment. In *IEEE Conference, ISIT'07*, pages 2210–2215, 2007.

[5] M. Cluzeau. Block code reconstruction using iterative decoding techniques. In *IEEE Conference, ISIT'06*, pages 2269–2273, 2006.

[6] M. Cluzeau. *Reconnaissance d'un schéma de codage*. PhD thesis, École Polytechnique, 2006.

[7] R. G. Gallager. *Low Density Parity Check Codes*. MIT Press, 1963.

[8] A. Valembois. Detection and recognition of a binary linear code. *Discrete Applied Mathematics*, 111(1-2):199–218, July 2001.