Decoding Beyond Half the Minimal Distance in Small Binary Linear Codes

Julien du Crest
supervised by Gilles Zémor

May 2020

Internship course

My internship followed an unusual course and I ended up doing research in a subject somehow far from what I originally expected .
Initially my supervisor and I planned to work on the applications of codes to perfectly secure multiparty computation so I started the internship studying the book on Multiparty Computation by Cramer et al.[1]. Although this work was mostly bibliographic, we did try to propose a code based view of the Byzantine Generals problem but this work didn't lead anywhere so I won't expand on it in this report. During that period, I took an interest in decoding techniques, especially the Reed-Solomon decoding algorithm from Gao[3] based on polynomial GCD and Fast Fourier Transform. Upon discussing the subject with my supervisor, it turned out this decoding technique was not too far from another decoding technique he was interested in, and he introduced me to a problem he had started to tackle with some students earlier. This problem interested me because it was at the intersection between decoding techniques and small product codes[1], two key topics in Multiparty Computation . It turned out I never came back to the Multiparty setting hence the peculiar course of this internship I was referring to.

To make this report easier to follow, I chose to focus only on the second part of my internship and to take it as a research subject in itself because I felt there was enough material in it to fill a whole report.

[1] see the next section for definition

Introduction

Note: In the following, *code* will always stand for *linear code*. I will make no distinction between the *linear code* and the associated *vector space* and use indistinctively *codeword* and *vector*. Also the norm |.| will always be the Hamming weight.

Finding good codes is sometimes easier than finding efficient (or even polynomial time) algorithms to decode them. The most striking example of this might be the famous Reed-Solomon codes that the authors published without even a polynomial-time decoding algorithm, and it took a few more years to find polynomial algorithms with decent complexity. The idea was to do the opposite and create efficient *small* codes with a built in decoding technique. My work is the followup of a Master's thesis by two of my supervisor's students [2].
I will first present the decoding technique we focused on, and then explain our general technique to build codes accepting this decoding. I will then proceed to explain the improvement I made to this technique, and finally show a few of my most promising results and compare them to the state of the art.

State of the Art

Today the best codes of small length are the BCH codes [4].My work does not compare directly with theses because BCH codes can only decode up to half the minimal distance with no decoding errors, whereas the decoding algorithm I used can decode beyond half of the minimal distance, the price being some decoding errors. Indeed there is a small probability for the erroneous codeword to be closer to a different codeword than the original, hence being wrongly decoded. I am not aware of any previous work in the same setting as mine, hence I used tables containing all the best known BCH codes up to size 256 to benchmark the codes I created[5].

Acknowledgement

I want to thank my supervisor Gilles Zemor for the great time I had doing research with him and for the great amount of his time he shared with me introducing me to new concepts and thinking together about problems on the blackboard. I also want to thank the 12 a.m lunch crew (they will recognize themselves if they ever read this) for their warm welcome, and because they had (or they found) an answer to all of my questions whether it be Number Theory, Algebraic Geometry or Combinatorics. Its been a pleasure spending these few months in Bordeaux with all of you.

Pellikaan Decoding

In the following I will explain a decoding method first analyzed by Pellikaan[6] that will be at the center of our attention.

Preliminaries

I first need to define some basic operations:

Schur Product : Given two vectors $u = (u_1, ..., u_n), v = (v_1, ..., v_n)$, their Schur product (denoted uv in the following) is $uv = (u_1v_1, ...u_nv_n)$

Product Code : Given two linear codes C, D, the product code (denoted CD) is $CD = \{cd, \quad c \in C, d \in D\}$

It immediately follows that if C and D are linear codes, the product code CD is also a linear code, and if $\mathcal{B}_C, \mathcal{B}_D$ are some of their basis, then $\{ef, \quad e \in \mathcal{B}_C, f \in \mathcal{B}_D\}$ is a generator of CD. Note that it is most often not a basis. This gives us a very crude upper bound on the dimension of the product state: $\dim CL \leqslant \dim C \times \dim L$.

vC-Product Code and v-Punctured Code: Given a linear code C and vector v of the same length, let's call vC-Product Code the space $vC = \{vc, c \in C\}$ and v-Punctured Code of C the space $C_{|v} = \{c_{|supp(v)}, c \in C\}$. Although the two notions are different in general, notice that they describe homomorphic spaces when $v \in {\mathbf{F}_2}^n$, hence I will use them indiscriminately later when we focus of binary codes.

Pellikaan decoding algorithm

Let C a code of length n on K, and p the number of errors to correct. Suppose there is a code L together with $\Pi = CL$, both of length n such that:

(i) $\dim L > p$

(ii) $\dim L + \dim \Pi \leqslant n$

(iii) $\text{dmin } \Pi > pn$

(iv) $\text{dmin } L + \text{dmin } C > n$

Given a codeword $c \in C$ and an error vector $e \in K^n$, let's denote $y = c + e$ the erroneous codeword.
The algorithm is the following:

Find a couple $(l, \pi) \in L \times \Pi$ such that $ly = \pi$
Find $c^* \in C$ such that $c^*l = \pi$
Return c^*

Algorithm 1: Pellikaan Decoding

Let's now show that the requirements on the codes C, L, Π are enough to ensure the proper decoding.

Lemma 1. *There exist a couple* (l, π) *such that* $ly = \pi$.

Proof. $|e| = p$, hence (i) implies that $\dim L_{|e} < \dim L$. This means $\ker L_{|e} \neq \varnothing$ and there is an $l \in L$ such that $le = (0, ..., 0)$ Such an l verifies $ly = l(c+e) = lc+le = lc+(0, ..., 0) = lc \in CL = \Pi$ □

Lemma 2. *A couple* (y, π) *can be found in polynomial time.*

Proof. The equation $yl = \pi, y \in Y, \pi \in \Pi$ has n equations and $\dim L + \dim \Pi$ variables. By (ii), there are less variables than the number of equations so this set of equations can be solved to find a couple (l, π).
Provided these equations are independent [2], then this equation admit a unique solution. □

Lemma 3. $yl = \pi \implies el = (0, ..., 0)$ *hence* c *agrees with* y *on all the non-null positions of* l

Proof. By (iii), the smallest vector in Π has a Hamming weight bigger than p. Because $yl \in \Pi$ and $cl \in \Pi$, it follows that $el \in \Pi$. The fact that the support of el is included in the support of e implies that $el = (0, ..., 0)$. Finally, $yl = (c + e)l = cl$ so y and c agree on all the non-null positions of l □

Lemma 4. *The solution* c^* *of* $cl = \pi$ *is unique and* $c^* = c$.

Proof. Suppose there are two codewords $c_1, c_2 \in C$ that are solution to the equation, then $c_1 l = \pi = c_2 l$, hence $(c_1 - c_2)l = (0, ..., 0)$. By (iv), the dimensions of C and L add up to more than n hence there can be no two words in C, L such that their product is null because they cannot have a disjoint support. This gives the unicity of c^*. To conclude, it was shown in lemma 3 that c was solution to $cl = \pi$ hence $c = c^*$. □

The Pellikaan decoding algorithm only uses very basic linear algebra primitives. This is both an advantage and an inconvenient, as those primitives are already well studied, and whereas their complexity is optimal or near optimal, in any case there is not much room for improvement on this side. A positive note is that all those operations can be parallelized efficiently.

Relaxing the requirements

Unfortunately this method is very difficult to implement. Apart for the Reed-Solomon codes, it is very difficult to find non-trivial (C, L, Π) triples with the required properties, especially in the case of binary codes[9].
So we will get rid of the conditions (iii) and (iv). This has immediates consequences:
First, the locators we find might be such that $le \neq (0, ..., 0)$, we will call those *parasite locators*. On a first look, those parasite locators are very bad because they mislead us on the support of e, but I will show a way to work around this later. Secondly, the non-unicity of c^* gives rise to a space of *potential codewords* during the decryption. For a specific l, the space of potential codewords is the kernel of the vector space spanned by $\{lv, v \in \mathcal{B}_C\}$ and is hence a vector space. Let K_l denote that space, we are then left to explore it and look for the closest vector to y in it. This is equivalent to looking for the smallest vector in $\{v - y, v \in K_l\}$ and it is well known that there is no efficient method to find the smallest

[2] This depends on the codes used, here I'm not interested in the specifics and I only present the general idea

vector in a vector space, so every times such a space comes up we will need to exhaustively look all of its vectors to find the one closest to y. Because the size of the vector space grows exponantially with the dimension of K_l, ensuring that it stays small will be one of our main goals.

Creating codes that accept Pellikan Decoding

Given a code C, and apart from some very specific examples like the Reed-Solomon codes, it is unknown how to come up with a non trivial code L such that $\dim CL < n$. To bypass this problem, the method we used to generate the code C is to start from the locator space L.

The method is as follow :

1. Generate L
2. Compute $L^2 = LL$
3. Set $C = L^{2^\perp}$ and $\Pi = CL$

Let's first show that the codes generated in that way allow for the Pellikaan decoding (the relaxed version with only conditions (i) and (ii)).
In order to abide to (i), p must be set to less or equal than $\dim L - 1$.
The condition (ii) is in fact implied by the construction (which was designed for this) :

$$\begin{aligned} & C = L^{2^\perp} \\ \implies & < c, l_1 l_2 >= 0 \quad \forall c \in C, l_1, l_2 \in L \quad \text{but since } < c, l_1 l_2 >=< c l_1, l_2 > \\ \implies & < c l_1, l_2 >= 0 \quad \forall c \in C, l_1, l_2 \in L \\ \implies & \Pi = CL \subset L^\perp \end{aligned}$$

Hence $\dim L + \dim \Pi \leqslant n$

With this method, the way to create good codes is to find L such that L^2 is small. Indeed once the dimension of L is fixed, the maximal number of errors that can be corrected is capped so our only method to improve the code is to try to increase the dimension of C, to increase the code rate $\frac{p}{\dim C}$. Finding non-trivial codes that have a small square is hard in general and although its a very interesting subject (and one studied by my supervisor), I didn't have the time to focus deeply on this problematic so I won't expand on it in the report.

Note that the orthogonal complement operation doesn't behave well with the minimal distance: there is no known way to predict the minimal distance in the orthogonal complement so we cannot control what the minimal distance of C will be compared to that of L^2. Hopefully, our algorithm allows to decode beyond half of the minimal distance, so although a small minimal distance is problematic, some interesting things can still be done with such codes.

Understanding the Locator Space

In this section, I will detail my contributions to the Pellikan decoding scheme and particularly in the setting of binary codes.

Let me first define the notion that will be at the center of our interest:

Locator Space : Given y, let's define the associated space $S_y = \{l \in L \quad s.t. \quad yl \in \Pi\}$ as the *locator space* (often abbreviated S). This space is a vector subspace of L. The vectors in S_y will be called *locators*
Because this will be useful later, let's also split S into $S = S^* \bigoplus S^\dagger$ where S^* is a basis of the good locators ($le = (0, ..., 0)$) and $S^\dagger$ a basis of the parasite locators ($le \neq (0, ..., 0)$)

One of my main concern was to understand how parasite locators appeared, how many and how often, etc. I succeeded in understanding their appearance and I will explain it here.
To do this I introduced a useful tool which is to echelonize the matrix L on the support of the error e. Here is an example where the dimension of L is 5 and the support length of the error is 3. Because the hamming weight of e is only 3, the punctured code of L on the support of e, called $L_{|e}$ is of dimension at most 3. By doing so, the good locators appeared clearly as the 2 last rows.
We found the good locators but what about the parasite locators ?

$$\begin{pmatrix} ... & 1 & 0 & 0 & ... \\ ... & 0 & 1 & 0 & ... \\ ... & 0 & 0 & 1 & ... \\ ... & 0 & 0 & 0 & ... \\ ... & 0 & 0 & 0 & ... \end{pmatrix}$$

Here is a first result to understand their apparition:

Lemma 5. ***(Me)***
If $L_{|e}$ is full rank, then their are no parasite locators.

Proof. Recall that $\Pi \subset L^\perp$. For every locator in the locator space S,
$le \in \Pi \subset L^\perp$. It implies that $l_{|e} \in {L_{|e}}^\perp$.
Because the matrix $L_{|e}$ is full rank we have that $\dim {L_{|e}}^\perp = 0$ hence $l_{|e} = (0, ..., 0)$ and there are no parasite locators.

□

This proof visually translate in the above matrix: it is impossible to find a non-zero sum of vectors in the echelonized submatrix above that is orthogonal to all the subrows... because all the non-null rows have a single non null coefficient.
To wrap everything up, this means that if the matrix $L_{|e}$ is full rank there are no parasite and $\dim S = \dim L - p$. Each time the rank decreases by one, there are two possibilities:

1. One good locator and one parasite locator appear, which means the dimension of S increases by two

2. One good locator was created but no parasite locator, hence the dimension of S increases by one.

This led me to consider generating L from other means to guarantee the local properties. Following the advice of my supervisor I then switched my study from random Codes to Reed-Muller Codes in hope that they would solve this problem. This was a good call because I got better results from Reed-Muller codes.

The Binary Curse

Because the locators are binary, every vector has in expectation a hamming weight of $\frac{n}{2}$. The Pellikan idea to use only one locator to recover the original codeword c is bound to fail if the dimension of C is bigger than the average hamming weight of the locators because the equation the solution space $\{c \quad s.t. \quad cl = yl\}$ would have a dimension of around $\dim C - hw(l)$ [3] . The workaround around this is to combine the non-null positions of several locators. Again using a rough estimate, if we combine two good locators, we have an expected number of non-null position of $\frac{3n}{4}$ and so on. However the parasite locators complicate the task. If the matrix $L_{|e}$ has rank $p-1$, we have one parasite locator in the basis, hence half of the vectors in S are parasites. This means that by taking randomly several vectors in S to combine their non-null positions, we only get an exponantially small chance to avoid all the parasites...

Sample Decoding

Using the ideas exposed before, the first improvement on the Pellikaan algorithm is to do the following:

$S = locatorSpace(y)$
while *True* **do**
  $l_1, l_2, l_3 = sample(S)$
  $l = l_1 \cap l_2 \cap l_3$
  $c^* = solve(l, y)$
  if $hamming\_weight(c^* - y) \leqslant p$ **then**
    return c^*
  end
end

Algorithm 2: Sample Decoding

As said before, the actual number of locators to be sampled must be tuned according to the dimension of C.
Although not very elaborate, the sample decoding is actually the most efficient method when the locator space has a low dimension. When the dimension of S grows, the technique that will be shown next is the clear winner.

Lowrank decoding

One thing I noticed is that the rank of $S_{|e}$ is very small because all the vectors in S^* are null on the support of e. This is captured in the equality

$$\text{rank } (S_{|e}) = \dim S - \dim S^*$$

[3] in fact a better approximation is $\dim C - hw(l) + \dim L$, see the additional results

This is abnormal since when taking a random subset r bigger than $\dim S$, it is most likely that rank $(S_{|r}) = \dim S$.
So one decoding strategy could be to look over all the subsets of p columns of S to find the one with the lowest rank and consider it the support of e. This strategy is clearly not polynomial and there is a smarter way to achieve a similar result.

If we look at a generating matrix of S, there can be $2^{\dim S}$ different columns appearing. If $\dim S$ is sufficiently large, each column only has a small chance of appearing several times (the all zero column is an exception and behaves differently). If there are no parasite locators, $\dim S = \dim S^*$ and it follows that:

$$\dim S + p = \dim L$$

If $\dim S = \dim S^* + t$ (this means there are t parasite dimensions in S, then :

$$\dim S + p = \dim L + 2t$$

This translates in terms of $S_{|e}$ as :

$$\text{rank } S_{|e} = t$$

Hence there are only 2^t different columns appearing in $S_{|e}$. Because in small dimension, t cannot be more than 3, $\dim S \geqslant 5$ is enough to ensure we can catch them with good probability.
Here is the algorithm for $t = 2$:

$S = locatorSpace(y)$
$D = \{(column, nb\_appearance)\}$
while $hamming\_weight(c^* - y) > p$ **do**
  Take (c, c') the most appearing couple of columns not yet tried
  for $0 \leqslant i < 256$ **do**
    if $S_{|i} == (0, ..., 0), c, c'$ *or* $c + c'$ **then**
      $l_i = 0$
    else
      $l_i = 1$
    end
  end
  $c^* = solve(l, y)$
end
return c^*

Algorithm 3: LowRank Decoding

This algorithm lend itself to many heuristic adjustments, one can only try the most obvious couples, or try them all, it all depend on the likelyhood of bad events (for example a column repeated many times but having nothing to do with the support of e) appearing frequently. The version I used in all the following was to order all the couples from most to least probable and try them until I found a good candidate c^*.

Study of Reed-Muller Codes

Reed-Muller codes [8] are codes generated by two parameters r, m on a base field $\mathbf{F}_q$. They all share the following properties:

(i) $RM(r,m)^2 = RM(\min(2r, m), m)$

(ii) $RM(r,m)^\perp = RM(m - r - 1, m)$

These properties combined with the formulas given later allow to compute the dimension and the minimal distance of L, C, Π very easily.

Binary Reed-Muller Codes

$RM_{\mathbf{F}_2}(r,m)$ is a $[2^m, \sum_{0 \leqslant i \leqslant r} \binom{m}{i}, 2^{m-r}]$ binary code which means that it has:

- length $n = 2^m$
- dimension $k = \sum_{0 \leqslant i \leqslant r} \binom{m}{i}$
- minimal distance dmin $= 2^{m-r}$

Reed-Muller based codes

r	dim L	dim C	dmin *
0	1	255	2
1	9	219	10
2	37	93	48
3	93	9	128
4	163	0	
5	219	0	
6	247	0	
7	255	0	
8	256	0	

In the table above, dmin * stands for the best known minimal distance for codes of this dimension and of length 256. I use it to compare the performances of my codes, knowing thoses codes can decode up to $\lfloor \frac{\text{dmin} - 1}{2} \rfloor$ From the table above, we see that the construction works only for $r \in \{1, 2, 3\}$. The results of those codes are shown below.

Tests on 1000 random codewords and random error

r	p	dim C	nb success	nb d-error	method
1	4	219	506	494	
2	36	93	0 (kernel too big to explore)	0	pick
	35		690	20	sample
	34		922	15	sample
	34		945	16	lowrank
	33		990	8	lowrank
	32		995	3	lowrank
	31		999	1	lowrank
	30		1000	0	lowrank
3	92	9	1000	0	lowrank

Note : I am aware that my data is too small to have meaningful statistical results, but I am only interested here in giving a rough estimate of the behavior of these codes.

As always, p is the number of errors. The column d-errors captures the cases where the decrypted codeword is closer to y than the original codeword, thoses errors are bound to happen when one tries to correct errors beyond the minimal distance, what we are interested in is how often they appear.
Let's review quickly the experimental results:
The code generated from $L = RM(1, 8)$ (with $C = RM(5, 8)$)is not interesting because it's minimal distance is small (dmin $= 4$), hence with only 4 errors, half of the time, the algorithm decodes another codeword that is closer to y than the original c.
The code generated from $L = RM(2, 8)$ (with $C = RM(3, 8)$)is promising because it can correct around 30 errors with only around 1 chance in 1000 for the random error to cross the minimal distance (dmin $= 16$ this time). This is 7 more than what can be done with the best BCH codes, but we have to keep in mind that this is a tradeoff between performance and reliability .
Finally the code generated from $L = RM(3, 8)$ (with $C = RM(1, 8)$) seems interesting because it can correct 92 errors very easily since the dimension of C is only 9 : a single locator having 128 non-null positions is more than enough to decode. Also, the fact that dmin $= 128$ means that the probability of decoding another codeword closer to y than the original c is very small (but it would be interesting to compute that). Unfortunately, this turns out to be not very interesting because : (1) a naive minimum distance decoding is possible because the code only has 2^9 words so this can be done exhaustively, (2) the whole family of $RM(1, 2^n), n \in \mathbf{N}$ are well understood, and minimum distance decoding can actually be implemented in almost linear time [7] so our algorithm is far behind.

Qary Reed-Muller Codes

The Qary Reed-Muller codes are similar to the binary ones, except they live in a bigger underlying field. One way to extract binary codes from qary codes is to take $C^{\cap \mathbf{F_2}}$, the subcode of all the codewords in $\{0, 1\}$ of the bigger code C. As long as q is a power of 2, those subcodes are linear codes themselves and can be embedded in $\mathbf{F}_2$.

The following is a table of the codes that can be constructed in this manner. The main advantage of this method is that because our binary code is a subspace of the original code, we can decode it using the matrix L in the bigger field. More elements in the field means that 0 appears less often so thanks to that less locators are needed to decode.

Binary subcodes of Qary Reed-Muller codes

q	r	m	dim L	dim C	dim $C^{\cap \mathbf{F_2}}$	dmin *
4	1	4	5	241	235	4
	2		15	190	161	26
	3		35	106	85	54
16	1	2	3	250	243	4
	2		6	241	219	10
	3		10	228	191	18
	4		15	211	143	30
	5		21	190	117	40
	6		28	165	93	48
	7		36	136	81	56
	8		45	105	33	96
	9		55	78	17	112
	10		66	55	9	128
	11		78	36	1	
	12		91	21	1	
	13		105	10	1	
	14		120	3	1	
	15		136	0	0	

From the table, we can see that the promising codes (those that can hope to correct more than their BCH counterpart) are :

$$RM_{\mathbf{F}_4}(3,4), RM_{\mathbf{F}_{16}}(6,2), RM_{\mathbf{F}_{16}}(7,2)$$

Below are the experimental results for these codes.

Tests on 1000 random codewords and random error

q	r	m	p	nb success	nb d-error	method
4	3	4	32	985	0	lowrank
			31	992	0	lowrank
			30	998	0	lowrank
16	6	2	26	996	1	lowrank
			25	997	0	lowrank
16	7	2	30	987	13	lowrank

Once again, the codes allow to consistently decode a few more errors than their counterpart, with the drawback of bad events occuring with probability from 1 in a thousand to 1 in a hundred.

Additional results

Here are some interesting additional results that we did along the way but that did not fit well anywhere before.

The kernel mystery

I first started by studying randomly generated binary codes as the followup of the work of my supervisor's students. Typically, I generated a random matrix 15×200 over F_2, added a line of $(1, 1, ..., 1)$ and used it as the matrix L. Adding a line of ones is a cheap trick to increase the rank of L by one by loosing only one dimension on C. This is due to the fact that L is already included in L^2 because $\forall l \in L, ll = l \in L^2$, so adding the vector $(1, 1, ..., 1)$ to L only increases the dimension of L^2 by one (the vector $(1, 1, ..., 1)$) hence the dimension of C looses one (because $C = L^{2^\perp}$).
One nice thing about random codes is that they behave 'as expected' all the time[4]. For example the rule of thumb that product spaces always tend to fill up as much space as they can is verified with overwhelming probability. Here what interests us is that the dimension of the above code is always 16. Thus, these codes can correct up to 15 errors (16-1).
One problem I systematically stumbled upon is that I expected $\dim C_{|l} \approx min(\dim C, |l|)$. Which is what would happen if C and l did not share a common construction.
Howewer, this was never the case and I ended up having to go through a big (dimension 5 to 10) vector space to find c. Here is an explanation of what really happens.

Lemma 6. ***(Me & my supervisor)*** *For a random code family L, C and a specific $l \in L$ such that $\dim L \ll |l|$,*

$$\dim lC^5 \approx min(\dim C, |l| - \dim L)$$

Proof. With $\delta_{i,j}$ denoting the kroneker function, let $\mathcal{B}_{\bar{l}} = \{(\delta_{1,j}, \delta_{2,j}, ..., \delta_{n,j}), j \notin \text{support}(l)\}$. Notice that $\mathcal{B}_{\bar{l}} \subset lC^\perp$ since their support is disjoint.
Let also $V = \{lv, v \in L\}$. First notice that $V \subset L^2$.
I claim that $V \subset lC^\perp$. Indeed, recall that $lC = \{lc, c \in lC\}$ hence $\forall lv \in V, \forall lc \in C, < lv, lc >=< llv, c >=< lv, c >= 0$.
I have thus shown that $V + \mathcal{B}_{\bar{l}} \subset lC^\perp$.
$\dim(V + \mathcal{B}_{\bar{l}}) = \dim V + \dim \mathcal{B}_{\bar{l}}$, where $\dim \mathcal{B}_{\bar{l}} = n - |l|$, because their support is disjoint.
The vector spaces are finite, so: $\dim lC = n - \dim lC^\perp \leqslant n - (n - |l| + \dim V) = |l| - \dim V$.

Finally, if $\dim L \ll |l|$, for a randomly generated code, $\dim V = \dim L$ with overwhelming probability. □

This is what happened with random codes 15×200. It is interesting to note that this phenomenon doesn't translate well for Reed-Muller codes because the fact that the dimension of the product code LL is smaller than $\dim L^2$, forces $\dim V$ to be smaller than $\dim L$.

[4] well most of the time by definition but the bad events are extremely rare
[5] Remember $lC = \{lc, c \in C\}$

Another parasite story

The following analysis is a dual analysis of the parasite locators that my supervisor found after one of our discussions. For this I will need to introduce the notions of *punctured* and *shortened* codes.

Puncturing is the action of removing a position/column in the entire code/generator matrix. The operation $C_{|v}$ I did earlier was indeed a puncturing so I will keep the notation as such.

Shortening is the action of removing all the codewords that have a non-null value on the given position. I will denote this operation $C_{<v}$ in the following.

Notice that the set S^* of good locators can now be expressed as $S^* = L_{<e}$. Using the duality of shortening and puncturing together with the fact that $L = \Pi^\perp$ gives

$$S^* = L_{<e} = \left(L^\perp{}_{|\bar{e}}\right)^\perp = \Pi_{|\bar{e}}{}^\perp$$

Let $\Pi_{\subset e}$ the vectors of Π whose support is included in the support of e. Then $\dim \Pi_{|\bar{e}} = \dim \Pi - \dim \Pi_{\subset e}$.
Going back to the dual, this gives the formula:

$$\begin{aligned}\dim S^* &= (n - |e|) - (\dim \Pi - \dim \Pi_{\subset e}) \\ &= (n - |e|) - ((n - \dim L) - \dim \Pi_{\subset e}) \\ &= \dim L - |e| + \dim \Pi_{\subset e}\end{aligned}$$

Although maybe not apparent, this is a dual description of what I exposed earlier with the dimension of $L_{|e}$, only it is easier to compute bounds on $\Pi_{\subset e}$ than on $L_{|e}$.
I take this opportunity to introduce a conjecture I formulated based on my experiments with Reed-Muller codes: I noticed for the Reed-Muller codes that a new parasite locator always appear when a new good locator appear. From what I explained already it can be shown that $\dim S^\dagger \leqslant \dim \Pi_{\subset e}$. My claim can be reformulated as:

Conjecture 1. ***(Me)*** *For Reed-Muller codes,*

$$\dim S^\dagger = \dim \Pi_{\subset e}$$

Together with the earlier statement, this would give us a pretty good understanding of what is happening with parasite locators. I couldn't prove this so I'll let it here for me or someone else to conclude on later...

Conclusion

To the best of my knowledge, there are no real life use-cases for efficient codes with relatively high probability of decoding errors. For example, for cryptographic use, the errors must be very rare: the NIST competition allows for bad events to happen with a probability at most 2^{-128}, which is very small compared to our 0.001 bad event probability estimates.
So realistically speaking, thoses codes don't have any direct application. However, I think my work showed that this method to generate codes is interesting, and that if one could find a good balance between high decoding capabilities and large enough minimal distance, this could yield practical algorithms.

There is still a lot of work in progress as I write this report, especially experimenting the same method with other families of code, and notably projective geometric codes. Also I am currently experimenting with code puncturing, which could allow to tailor the Reed-Muller codes to the method even more.

This internship introduced me to two very interesting topics, understanding the growth of code products and predicting the minimal distance of the orthogonal complement. Apart from their use in my work, code products also have broader applications in code-based cryptography. For example in the code-based secure multiparty computing, the secret inputs are codewords so one gets the addition for free because of linearity, but multiplication is built out of a Shur product so the topic of code products surfaces again. If I am to continue doing research in code-based cryptography, I will be sure to spend some time on those.

Bibliography

[1] *Secure Multiparty Computation and Secret Sharing*
Ronald Cramer, Ivan Bjerre Damgård, Jesper Buus Nielsen-2015

[2] *Le décodage de codes linéaires par paires localisatrices d'erreur*
Elie Bouscatie, Célian Banquet - 2020

[3] *A new algorithm for decoding Reed-Solomon Codes*
Shuhong Gao - 2002

[4] *Codes correcteurs d'erreurs*
Alexis Hocquenghem - 1959

[5] *http://www.codetables.de/*
Marcus Grassl

[6] *On decoding linear codes by error correcting pairs*
Ruud Pellikaan - 1988

[7] *Hadamard Matrix Analysis and Synthesis*
R.K.YarlagaddaandJ. E.Hershey

[8] *Application of Boolean algebra to switching circuit design and to error detection*
David E. Muller-1954

[9] *Critical pairs for the Product Singleton Bound*
Diego Mirandola, Gilles Zémor-2015