Words of Minimal Weight and Weight Distribution in Binary Goppa Codes

Matthieu Finiasz

Abstract

The weight distribution of a Goppa code is proven to be "close" to the binomial weight distribution found in a random linear code. Some bounds have been found on the distance between the two distributions and this distance is incredibly small for some particular weights. However, for smaller weights the bound on the distance is of no use.

Using an algorithm to find words of minimal weight we were able to perform some statistics on these weight distributions and show that even for weights close to the minimal weight bound the distribution is still binomial-like.

Key Words: Goppa codes, weight distribution, minimal weight, syndrome decoding.

1 Introduction

By construction, a Goppa code is able to decode up to t errors uniquely [1]. Consequently, the minimal distance between two words of such a code (which is also the minimal weight of a non-zero code word) is at least $2t + 1$. What we are interested in is telling whether or not it is possible to find such minimal weight words and then evaluate their number.

Little is known about the weight distribution of the Goppa codes, however it is sure that it is close to a binomial distribution [4]. That is, the number of words of weight w in a code of length n is approximately $\binom{n}{w} \times 2^{k-n}$. This is true when w is not too small, but when w is close to 0 the distribution is not the same: for instance, for any weight from 1 to $2t$ the number of words is 0. To compensate for this lack of low weight words, the number of words of weight just above the minimal weight should be a little larger than the expected number. We are going to see if this is the case and how this small amount is distributed over the other weights.

2 An Algorithm for Finding Words of Minimal Weight

Suppose we have a binary Goppa code Γ constructed over the field $\mathbb{F}_{2^m}$ with the polynomial g of degree t. This code will correct up to t errors, so we will try to find code words of weight $2t+1$.

Using a locator polynomial. We call L_c the locator polynomial of a code word c. This is the polynomial of smallest degree which has for roots the field elements associated to the non-zero positions of c. All words in Γ have the following property: the derivative of L_c can be divided by g^2.

For words of minimal weight this is interesting because L_c is of degree $2t+1$ and L'_c is of degree $2t$, like g^2. Moreover, both L'_c and g^2 are unitary. So $L'_c = g^2$. As we are in a field of characteristic 2 we only know half the coefficients of L_c. Anyhow, we can then try to guess the other coefficients so that L_c is split over $\mathbb{F}_{2^m}$. Each split L_c will correspond to a code word of minimal weight. However if we call N_{2t+1} the number of words of minimal weight this algorithm will take, in average, $2^{m(t+1)}/N_{2t+1}$ tries to find one minimal weight word.

This is not the best that can be done. For instance another algorithm can find minimal weight words in a much shorter time.

Using the decoding algorithm. This technique will use the following idea: we have an algorithm able to decode up to t errors [2]. If we try to decode a word of weight $t+1$ and the decoding works, it will give us a word of weight between 1 and $2t+1$. As there is no word of weight 1 through $2t$ in Γ we will necessarily have a word of weight $2t+1$. However this will only work if the decoding succeeds. This time the average number of tries before a decoding succeeds will be: $\binom{n}{t+1}/(N_{2t+1}\binom{2t+1}{t+1})$.

Whatever the number of minimal weight words, this algorithm will require less attempts than the previous one: n is smaller than 2^m so $\binom{n}{t+1}/\binom{2t+1}{t} < \binom{2^m}{t+1}/\binom{2t+1}{t} < 2^{m(t+1)}t!/(2t+1)!$. There is a factor $t!/(2t+1)!$ between the two algorithms, so even if testing if a polynomial is split is a little faster than trying to decode a word, the first algorithm will be much slower than the second one.

3 Experimentation

In this section we will discuss the results obtained through experimentation, but first of all, we will try to see what results we should expect.

In [5] the case of decoding a random syndrome in a Goppa code is studied. It is shown that for a binary Goppa code correcting t errors, the ratio of decodable syndromes is approximately $1/t!$. This means that a random word has an average probability of $1/t!$ of being at a distance less or equal to t of a code word. This is true for a random word, but in our algorithm we only consider words of weight $t+1$ for which the average probability could be quite different.

For instance, this is the case for smaller weights: all words of weight t or less can be decoded, not only $1/t!$ of them.

Known exact values. In [3], Goppa codes correcting 3 errors are studied. They are classified and for each class the exact number of minimal weight word can be computed. The following table compares the obtained results to the one which should be expected if there were exactly one decodable word of weight $t+1$ among $t!$: $N_{2t+1} = n^{t+1}/(2t+1)!$

n	exact number	expected number
16	~ 4	13
32	128	208
64	$\sim 2\,640$	3 328
128	47 616	53 261
256	$\sim 806\,000$	852 176
512	13 264 896	13 634 817

This table shows that when the length of the code increases our approximation gets closer to the real value. In fact the error decreases exponentially: 225%, 62%, 26%, 12%, 5.7%, 2.7%...

Experimental values. To be able to evaluate more precisely the quality of the estimation we can compare it to experimental results. Applying the decoding algorithm several times to a given Goppa code, we can check the ratio of decodable words among those of weight $t+1$. Our approximation will be good if this ratio is, in average, close to $t!$.

We computed the following experiment:

- ◇ generate a random Goppa code of length n correcting t errors
- ◇ compute 50 words of minimal weight
- ◇ compute the average number of tries necessary for each of these 50 words
- ◇ repeat this 20 times for each set of parameters (n, t)
- ◇ compute the average value (Σ) and standard deviation (σ) for each set of parameters

The obtained results are shown in the following table:

t	5		6		7		8		9	
n	Σ	σ	Σ	σ	Σ	σ	Σ	σ	Σ	σ
512	146	21	866	129	5 903	882	45 491	5 128	–	–
1 024	138	30	755	100	5 308	755	44 172	5 387	425 400	52 409
2 048	125	16	721	73	4 892	673	44 827	5 094	367 767	48 077
4 096	119	15	769	144	4 773	962	38 685	6 250	368 646	48 756
8 192	120	17	750	112	5 235	790	41 036	5 041	383 443	56 764
16 384	123	14	732	91	5 470	846	39 351	6 242	374 139	59 313
32 768	120	18	662	99	5 193	933	42 309	8 629	357 590	39 353
65 536	116	16	693	81	5 372	914	39 643	5 719	360 973	41 858
Theory	120	17	720	102	5 040	713	40 320	5 702	362 880	51 319

The "Theory" line corresponds to the results which should be expected if there were exactly 1 among $t!$ decodable words for every binary Goppa code correcting t errors. The average value Σ is of course $t!$ and the standard deviation is $\frac{t!}{\sqrt{50}}$. This is because, for each Goppa code, the average number of necessary tries is computed as the average of 50 attempts, each one with a standard deviation of $t!$. Results could be more precise if these averages were computed using more attempts but this could take very long as for example, with $t = 9$ each attempt already takes a few minutes.

What we can see from this table is that the obtained results are always really close to the theoretical ones. Looking closer we can also note that for small values of n, Σ is always a little larger than $t!$. This is probably due, like for the 3-errors correcting codes, to the approximation errors: the number of words of minimal weight is a little smaller for small values of n, hence the average number of decoding attempts is greater. To minimize this error we should probably use instead of $\frac{1}{t!}$ the value $\frac{1}{n^t} \times \binom{n}{t}$ which for example for parameters $(512, 8)$ would give $\Sigma = 42598$. However, when n increases the two approximations tend to be the same.

Conclusion From these experimentations and known results we can say that there is a good probability that, for codes of great length, the number of words of minimal weight tends to be what we expected at the beginning of this section: $N_{2t+1} = \frac{n^{t+1}}{(2t+1)!}$. However, for codes of smaller length, the number of minimal weight words is a little smaller.

The computational time required to make some statistics on codes correcting more errors is huge and we therefore cannot really check that our assumption remains true for a larger t, however it seems reasonable to believe that it will.

4 Weight Distribution

In this section we consider as true the assumption that, among a set of word of any given weight greater than t, there are always a ratio of $1/t!$ decodable words. We've seen in the previous part that this was certainly the case for words of weight $t + 1$, therefore this must also be true for words of greater weight.

Trying to decode words of weight $t + 1$ we find that $N_{2t+1} = \frac{n^{t+1}}{(2t+1)!}$. If we take a word of weight $t+2$ and try to decode it we get the following possibilities:

- ⋄ it is not decodable: this happens with probability $1 - \frac{1}{t!}$
- ⋄ it decodes in a word of weight $2t+2$: this happens with probability $N_{2t+2} \times \frac{\binom{2t+2}{t+2}}{\binom{n}{t+2}}$
- ⋄ it decodes in a word of weight $2t+1$: this happens with probability $N_{2t+1} \times \frac{\binom{2t+1}{t+2}}{\binom{n}{t+2}} = \frac{1}{n(t-1)!}$

From this we can deduce that: $N_{2t+2} = (1 - \frac{t}{n})\frac{n^{t+2}}{(2t+2)!}$. In a more general manner, starting from words of weight $t + \alpha$, we have the following formula:

$$\frac{1}{t!} = \sum_{w=\alpha}^{2t+\alpha} N_w \frac{\binom{w}{\beta}}{\binom{n}{t+\alpha}\binom{t+\alpha}{\beta}} \quad \text{where} \quad \beta = \left\lceil \frac{w+\alpha}{2} \right\rceil$$

From this we get:

$$N_{2t+\alpha} = \frac{\binom{n}{t+\alpha}}{\binom{2t+\alpha}{t+\alpha}} \times \left(\frac{1}{t!} - \sum_{w=\alpha}^{2t+\alpha-1} N_w \frac{\binom{w}{\beta}}{\binom{n}{t+\alpha}\binom{t+\alpha}{\beta}} \right) = \frac{n^{t+\alpha}}{(2t+\alpha)!} \times \left(1 - t! \sum_{w=\alpha}^{2t+\alpha-1} N_w \frac{\binom{w}{\beta}}{\binom{n}{t+\alpha}\binom{t+\alpha}{\beta}} \right)$$

Which in first order approximation in $\frac{1}{n}$ gives: $N_{2t+\alpha} = \frac{n^{t+\alpha}}{(2t+\alpha)!} \times (1 - \frac{t}{n} + O(1/n^2))$. Hence, for small values of α (before the $\frac{t}{n}$ term can cause a significant error), we will have $N_{2t+\alpha} = \frac{n^{t+\alpha}}{(2t+\alpha)!}$.

Note that this is exactly the binomial distribution: $\frac{\binom{n}{2t+\alpha}}{n^t} \simeq \frac{n^{2t+\alpha}}{(2t+\alpha)!n^t} = \frac{n^{t+\alpha}}{(2t+\alpha)!}$. Therefore we can say that, if our assumption is true, even for the smaller weights the weight distribution of a binary Goppa code is the standard binomial distribution we get with a random code.

So a Goppa code will have, as expected, a binomial weight distribution, even for small weights. But what about the "gap" created by the absence of words of weight 1 to $2t$? We came to our conclusion under the assumption that for any weight w the probability a word of weight w was decodable is $\frac{1}{t!}$. As we've seen with the statistics in section 3 this assumption will be more exact when n is large, and in this case the "gap" becomes really small compared to the number of words of weight $2t+1$. For this reason it seems normal that under our assumption we do not notice any compensation for this "gap".

5 Conclusion

Starting from the assumption that for a given $w \geq t+1$, a word of weight w has the same probability of being decodable as any word, we could show that the weight distribution of a binary Goppa code is close to the binomial distribution, even for words of small weight and especially when n is large. Thanks to the decoding algorithm we could make some statistics on the probability a word of weight $t+1$ is decodable and show that the assumption was reasonable. With this algorithm we also have a way to find minimal weight code words in any Goppa code. However its complexity grows in $t!$ so it will only work for t small (that is $t \leq 10$ in practice). As this algorithm requires to be able to decode to work it can be seen as a trap which might find a use for cryptography, but this is another problem...

References

[1] V. D. Goppa. "A new class of linear error-correcting codes" In *Probl. Inform. Transm.*, vol. 6, pp. 207-212, 1970.

[2] E. R. Berlekamp. "Goppa codes" In *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 590-592, 1973.

[3] Anne Canteaut. *PHD Thesis* : "Attaques de cryptosystèmes à mots de poids faible et construction de fonctions t-Résilientes" chapitre 2, exemple 2.14.

[4] F. Levy-dit-Vehel and S. Litsyn. "Parameters of Goopa Codes Revisited" In *IEEE Transactions on Information Theory*, vol. 43, no. 6, November 1997.

[5] N. Courtois, M. Finiasz, and N. Sendrier. "How to achieve a McEliece-based digital signature scheme" In *ASIACRYPT 2001*, Springer-Verlag, 2001. `http://eprint.iacr.org/2001/010` and RR-INRIA 4118.