Information Set Decoding to

authors
Marche Polytechnic University,
*{p.santini}@univpm.it

***Abstract*—aaaaa**

***Index Terms*—Code-based cryptography, cryptanalysis, digital signature, zero-knowledge identification scheme.**

I. Introduction

Informally, one can think of a code as a set of strings defined over some alphabet, normally a finite field $\mathbb{F}_q$, which we call codewords. The natural application of codes is that of correcting errors in communication channels. Geometrically, the problem can be formulated as follows: given a code $\mathscr{C}$ (a set of length-n strings) and some input string $\mathbf{x}$, find the codeword $\mathbf{c} \in \mathscr{C}$ which mostly resembles $\mathbf{x}$. This normally goes by the name of Maximum Likelihood Decoding (MLD) problem.

Arguably, the most studied case is that of linear codes and transmission over additive channels. In a linear code, the codewords are closed under the sum operation: the sum of any two codewords yields another codeword. Because of this property, a linear code $\mathscr{C}$ is anything but a linear subspace of the ambient vector space $\mathbb{F}_q^n$. For additive channels, one has $\mathbf{x} = \mathbf{c} + \mathbf{e}$ where $\mathbf{e}$ is a length-n string, so that also $\mathbf{x}$ is a length-n string. Codewords are seen as points of $\mathbb{F}_q^n$ and the similarity between two codewords is quantitatively measured using some distance function $\mathrm{dist} : \mathbb{F}_q^n \mapsto \mathbb{N}$. Then, MLD can be interpreted with a geometric flavour: given $\mathscr{C} \subseteq \mathbb{F}_q^n$ (a collection of points in the space) and some point $\mathbf{x} \in \mathbb{F}_q^n$, find the codeword $\mathbf{c} \in \mathscr{C}$ which minimizes $\mathrm{dist}(\mathbf{c}, \mathbf{x})$. Each codeword $\mathbf{c}$ is characterized by Voronoi region, which is the set of points $\mathbf{x} \in \mathbb{F}_q^n$ for which the closest codeword is $\mathbf{c}$. So, MLD asks, on input $\mathbf{x}$, to find the center of the Voronoi region in which $\mathbf{x}$ is contained.

It is clear that the ability to solve MLD is of great importance, since it allows to correct errors happening in communication channels. Let $\mathcal{D}$ be a decoder, i.e., an algorithm that solves MLD on input some code $\mathscr{C}$ and some string $\mathbf{x}$. Imagine that, upon transmission of some $\mathbf{c} \in \mathscr{C}$, the receiver gets $\mathbf{x} = \mathbf{c} + \mathbf{e}$: we desire that $\mathcal{D}(\mathscr{C}, \mathbf{x}) = \mathbf{c}$, since this means that we are able to correct the errors introduced by the channel and, in the end, can retrieve the transmitted sequence $\mathbf{c}$. Obviously, we also desire that $\mathcal{D}$ is an efficient algorithm (we consider this aspect later). Regardless of whether such a decoding algorithm exists, one can establish the performances of $\mathcal{D}$ using the notion of *minimum distance*, i.e., the minimum value of $d = \mathrm{dist}(\mathbf{c}, \mathbf{c}')$ existing between two (different) codewords. Indeed, each Voronoi region has a radius which is at least $d/2$. So, whenever $\mathbf{e}$ is properly bounded (say, whenever $\mathrm{wt}(\mathbf{e}) < d/2$), one guarantees that $\mathbf{x}$ remains in the same Voronoi region. Instead, if $\mathrm{wt}(\mathbf{e}) \geq d/2$, then the solution to MLD may be different from $\mathbf{c}$, because the error vector $\mathbf{e}$ moves $\mathbf{x}$ to a different Voronoi region. As a rule of thumb, linear codes having a good (to be read as high) minimum distance can correct more errors.

a) Hardness of finding the minimum distance: The problem of finding the minimum distance of a given linear code is a well known, NP-hard problem, which is even hard to approximate. This implies that, unless P≠NP, there cannot exist an *efficient* algorithm that finds the minimum distance of *any* given linear code. Yet, this does not imply that our hopes to know the minimum distance of many codes we employ is lost. First, there exist families of codes for which the minimum distance is chosen by design (think about Reed-Solomon or other algebraic codes). Moreover, it is easy to find families of codes for which finding the minimum distance is easier, perhaps solvable in polynomial time. Notice that this does non contradict the NP-hardness of the problem, since we are speaking of *specific* classes of codes: any algorithm that would work properly for these codes will not work for *any* input code.

Moreover, being NP-hard does not mean that practical instances cannot be solved in *reasonable* time. Even if known solvers take exponential time, this time may still be feasible when the input is sufficiently small.

Finally, the notion of NP-hardness is binded to the input size of the problem. Informally, NP-hardness implies that a polynomial time solver cannot exist. In other words, any algorithm $\mathcal{A}$ solving an NP-hard problem is expected to take exponential time[1] Here, polynomial/exponential time is referred to the time complexity of the algorithm, seen as a function of the input size. If we denote by n the input size, then an exponential time algorithm $\mathcal{A}$ is expected to run in time $O(2^{\alpha n})$ for some constant, positive α. This remark, which perhaps is trivial, will turn useful when we will speak about LDPC codes.

As a final remark, we would like to specify that NP-hardness is a worst case result. In other words, a problem is (informally) NP-hard if there cannot exist an algorithm solving *all instances* in polynomial time. Still, specific instances of the problem may be easily solvable. That is, the problem may be easy *on average*, or there may exist a subset of instances for which we know about an efficient algorithm.

b) LDPC codes: Low Density Parity-Check (LDPC) codes are a special family of codes, introduced by Gallager in his PhD thesis. Arguably, LDPC codes are among the most studied families of codes, because of their great error correction capabilities combined with very efficient algorithm

[1]Existence of a polynomial time algorithm for such a problem would imply collapse of the polynomial hierarchy, i.e., would imply P=NP.

for both encoding and decoding. It is also well known that, under certain ideal conditions (e.g., infinite length and no cycles), LDPC codes can achieve the channel capacity. As a matter of fact, they are employed in many applications and communication standards.

An LDPC code is, essentially, a code whose parity-check matrix admits a wide majority of zeros. For this reason, we say the matrix is *sparse*. It turns out that this sparsity is the key ingredient for all the good properties of LDPC codes. Indeed, sparsity allows for efficient encoding techniques: roughly speaking, the process gets faster since the presence of many zeros allows to skip many computations. Moreover (and more importantly) sparsity allows for fast decoding. Decoders for LDPC codes are *message-passing* algorithms: the low complexity, as well as the capacity to correct a non trivial amount of errors, is due to the wide majority of zeros in the parity-check matrix. This can be seen when the code is represented with its Tanner graph, that is, a bipartite graph representing a given parity-check matrix. The graph can be defined for any parity-check matrix (so, for any code), but its nice properties appear only when LDPC codes are considered. Namely, the graph is sparse, in the sense that it has a very few edges. Put it under graph theoretic words, the nodes in the graph have low degree. Again, the sparsity of the Tanner graph is a key ingredient to make decoding work.

c) Information Set Decoding: Information Set Decoding (ISD) is a family of generic decoders, that is, algorithms that can decode any input code. With slight modifications, these algorithms can also be used to find codewords of (upper) bounded weight. The canonical scenario is the one in which one has to find the unique solution to a given decoding instance, or has to find one codeword with the minimum weight. In the latter case, we speak of finding the minimum distance of a given code.

An ISD algorithm is an algorithm which can find the minimum distance of any given linear code. It is a randomized algorithm, whose average running time depends on the searched weight. Depending on the family of considered codes, the cost of ISD may vary. For instance, when dealing with random codes, it is well known that the minimum distance is linear in the code length n, e.g., is $d = \delta n$ for some constant $\delta \in [0; 1]$. In particular, one has $\delta = h^{-1}(1 - R)$, where $R = k/n$ is the code rate. In such a case, the cost of the algorithm is in $2^{\alpha n\left(1+o(1)\right)}$ for some positive constant α. For codes having a minimum distance which is sublinear in n (e.g., $d = o(n)$), then a well known result from Canto-Torres and Sendrier says that the cost of all ISD algorithms grows as $2^{-d\log_2(1-R)\left(1+o(1)\right)}$. Notice that, in these cases, the cost of the algorithm is sub-exponential, since d grows less than linearly with n.

When one possesses some information about the code, ISD can be sped up. This fact is easy to argue. Indeed, ISD algorithms have been designed (mostly) to attack code-based cryptosystems, in which linear codes are either random codes, or indistinguishable from random codes. In other words, nothing about the code is known, so that the algorithm cannot make any assumption on the code structure, nor on the shape of the searched codeword with minimum weight. Still, when something about the code is known, one can speed-up ISD. This is the case when the code has a non trivial automorphism group or when, for instance, the code has a very special geometric structure.

A. Our goal

At the best of our knowledge, ISD on LDPC codes has never been studied. In such a case, the main difference with respect to random codes is in that the parity-check matrix $\mathbf{H}$ defining the code is sparse. This typically does not happens when considering random codes. Yet, given the sparsity, we believe there is a strong possibility that ISD algorithms can be significantly made faster. In other words, we want to redesign ISD, considering the case in which the input parity-check matrix is sparse (we will measure sparsity by a parameter whose value will probably be crucial in setting the hardness of the problem).

Our contributions are twofold, perhaps threefold. First, we want to improve the state-of-the-art on algorithms to find the minimum distance of LDPC codes. At the best of our knowledge, many of the LDPC codes currently employed in communication standards have an unknown minimum distance. We believe our algorithms can greatly help in filling this gap.

Moreover, we want to study the hardness of the minimum distance problem, when considering codes admitting a very sparse representation. This contribution may be of fundamental importance, from a complexity theory point of view. Indeed, the sparsity of $\mathbf{H}$ plays a crucial role in the input size of the problem. Normally, a matrix with $r = (1 - R)n$ rows and n columns can be represented with $rn = O(n^2)$ bits. However, when the matrix is very sparse, the input size may change. Let γ denote the density of $\mathbf{H}$, i.e., γ is the ratio between the number of ones and the number of elements in the matrix. For each one in the matrix, it may be enough to store its row and column indices, taking respectively $\log_2(r)$ and $\log_2(n)$ bits. So, representing a sparse $\mathbf{H}$ takes binary size

$$L = \gamma rn\big(\log_2(r) + \log_2(n)\big) = O\big(\gamma n^2 \log_2(n)\big).$$

When γ is very small, the input size may no longer be polynomial in n. For instance, if $\gamma = \Omega(1/\sqrt{n})$, then $L = O\big(\log_2(n)\big)$. In such a case, the input size is logarithmic in n, instead of quadratic.

Many authors say that finding the minimum distance of LDPC codes is NP-hard, since an LDPC code is a linear code. This is false, for two reasons. First, LDPC codes are a subset of all linear codes: even if the minimum distance problem is NP-hard for linear codes, it can still happen that the problem is easy for LDPC codes. So, the complexity of the problem still needs to be understood. Moreover, the matter becomes much more interesting when one considers values of γ so that the input size is no more polynomial in n. In such a case, the problem may change complexity class (we hardly doubt this is the case). Or, it may remain NP-hard, but an exponential

time algorithm may be extremely fast. Indeed, an exponential time algorithm would take time

$$2^{\alpha L} = 2^{\alpha \log_2(n)} = n^{\alpha}.$$

So, finding the minimum distance may be polynomial in the code length: this would be a breakthrough results, as it would imply that LDPC codes are much easier to study than almost all the other codes.

Notice that such a goal is not unreasonable. Indeed, LDPC codes already allows to solve the decoding problem efficiently (when the number of errors to be corrected is sufficiently small). This behaviour is typical of LDPC codes, and is not exhibited by any other code. The decoding problem and the minimum distance problem are strictly similar; for instance, the best solvers for generic codes, for both problems, are ISD algorithms. So, the fact that the decoding problem is somewhat easier may imply that the minimum distance problem for LDPC codes is easier, as well.

Finally, as a final contribution, we would like to draw some more connections between LDPC codes and graphs. There already exist many combinatorial objects (e.g., cycles and trapping sets) that arise frequently when studying LDPC code and which are naturally described as graph theoretic objects. We would like to study these relations further. This goal is strongly motivated by some results for computational problems in coding theory. For instance, the clique problem gets significantly easier when studying sparse graphs (called graphs with low degeneracy). For a graph with degeneracy t, the problem is solvable in time $O(2^{t/4})$: when $t = \Omega\big(\log_2(n)\big)$, the problem becomes solvable in polynomial time.

II. Notation and background

A. Mathematical notation

As usual, we denote with $\mathbb{F}_q$ the finite field with q elements. Given a set $\mathscr{A}$, its cardinality (i.e., its number of elements) is indicated as $|\mathscr{A}|$. We use bold capital letters to denote matrices, and small capital letters to denote vectors. The operator $(\cdot)^\top$ denotes the transposition operation, while the null matrix is indicated as $\mathbf{0}$ (its dimensions will always be clear from the context). For a vector $\mathbf{a}$, we use $\mathrm{wt}(\mathbf{a})$ to denote its Hamming weight, that is, the number of non null coordinates. The Hamming sphere with radius w is the set of all vectors with length n and weight w and is indicated as $\mathscr{S}_{n,w}$. When clear from the context, we will simplify this notation as $\mathscr{S}_w$. We will use $\mathscr{P}_n$ to indicate the group of length-n permutations: if $\pi \in \mathscr{P}_n$ and $\mathbf{a} = (a_1, \cdots, a_n)$, we have $\pi(\mathbf{a}) = \big(a_{\pi(1)}, \cdots, a_{\pi(n)}\big)$. For an event Event, we write $\mathbb{P}\left[\mathsf{Event}\right]$ to denote the probability that Event happens. If X is a random variable, we use $\mathbb{E}[X]$ to indicate its mean value. We also recall the following version of Chernoff's bound [2], which we will use to estimate the confidence interval of our estimator.

Theorem 1 (Two-sided Chernoff's bound). *Let $X_1, \ldots, X_t$ be independent random variables, $0 \le X_i \le 1$ for each i. Let $X = \sum_{i=1}^{t} X_i$, and $\mu = \sum_{i=1}^{t} \mathbb{E}[X_i]$. Then, for any $\varepsilon \in [0,1]$, we have*

$$\mathbb{P}\left[|X - \mu| \ge \varepsilon\mu\right] \le 2e^{-\frac{\mu\varepsilon^2}{3}}.$$

We write $x \xleftarrow{\$} A$ to express that x is sampled uniformly at random from A, that is, x is distributed according to the uniform distribution over A. Finally, we define $\mathscr{I}_{n,w}$ as the ensemble of all subsets of $\{1, \ldots, n\}$ with size w.

B. Coding theory background

A linear code $\mathscr{C}$ with dimension k and length n is a k-dimensional subspace $\mathscr{C} \subseteq \mathbb{F}_q^n$ and can be uniquely identified with a generator matrix $\mathbf{G}$ for this subspace. Another way to refer to the code involves the so called parity-check matrix. We say that a matrix $\mathbf{H}$ is a parity check matrix for the code $\mathscr{C}$ if the following conditions holds:

$$c \in \mathscr{C} \iff \mathbf{H}\mathbf{c}^\top = \mathbf{0}.$$

The code rate is indicated as $R = k/n$. The weight distribution of a code is given by $\{N_{\mathscr{C}}(0), N_{\mathscr{C}}(1), \ldots N_{\mathscr{C}}(N)\}$, in such a way that $N_{\mathscr{C}}(w) = |\,\mathscr{C} \cap \mathscr{S}_w|$. We will use the following notation

$$\mathscr{C}_w = \mathscr{C} \cap \mathscr{S}_w,$$

so that $N_{\mathscr{C}}(w) = |\mathscr{C}_w|$. Due to linearity, we always have that, for any $\mathbf{c} \in \mathscr{C}_w$, also $a\mathbf{c} \in \mathscr{C}_w$, for any $a \in \mathbb{F}_q^*$. For our purposes, it will be useful to consider only codewords that are identical up to a scalar multiplication. To this end, we define $\mathscr{C}_w^* \subseteq \mathscr{C}_w$ such that, for any two distinct $\mathbf{c}, \mathbf{c}' \in \mathscr{C}_w^*$, it holds $\mathbf{c} \neq a\mathbf{c}'$ for any $a \in \mathbb{F}_q^*$. Notice that there can be several ways to define $\mathscr{C}_w^*$. For the sake of simplicity, we require that each codeword in $\mathscr{C}_w^*$ has the first non null entry which is a 1. Also, we have

$$N_{\mathscr{C}}^*(w) = |\mathscr{C}_w^*| = \frac{|\mathscr{C}(w)|}{q-1} = \frac{N_{\mathscr{C}}(w)}{q-1}.$$

We consequently modify also the definition of $\mathscr{S}_w$, and introduce $\mathscr{S}_w^*$, that is, the set of all vectors with length n, Hamming weight w and first non null entry equal to 1.

C. Random codes

We denote by $\mathcal{U}_{n,k}$ the uniform distribution of k-dimensional linear codes, with length n, over $\mathbb{F}_q$. By random code, we refer to a code sampled from $\mathcal{U}_{n,k}$. This sampling can be done by first sampling a matrix $\mathbf{V} \xleftarrow{\$} \mathbb{F}_q^{k\times(n-k)}$ and a length-n permutation $\pi \xleftarrow{\$} \mathscr{P}_n$, and then computing $\mathbf{G} = \pi\big((\mathbf{I}_k, \mathbf{V})\big)$. It is easy to see that $\mathscr{C}$, the code generated by $\mathbf{G}$, is distributed according to $\mathcal{U}_{n,k}$.

The average weight distribution of random codes is a well known quantity; for the sake of completeness, we report it in the following.

Theorem 2. ***Average weight distribution of random codes*** *The expected value of $N_{\mathscr{C}}(w)$, when $\mathscr{C}$ is distributed according to $\mathcal{U}_{n,k}$, is*

$$\binom{n}{w}(q-1)^w q^{-(n-k)} = q^{\big(h_q(w/n)-(1-R)\big)\big(1+o(1)\big)n},$$

where $h_q(x) = x\log_q(q-1) - x\log_q(x) - (1-x)\log_q(1-x)$ is the q-ary entropy function.

In this paper we are mostly interested in random codes or, at the very least, codes whose structural properties cannot be

used to somehow ease the search for codewords. To formalize this situation, we make use of the following assumption, which is standard and folklore for the study of ISD algorithms.

Assumption 1. *For any code $\mathscr{C} \subseteq \mathbb{F}_q^n$ and any integer $w \in \{0, \dots, n\}$, we assume that each codeword in $\mathscr{C}_w^*$ is uniformly distributed over $\mathscr{S}_w^*$. In particular, this implies that its support is a uniformly random subset of $\{1, \cdots, n\}$ with size w.*

III. Information Set Decoding Algorithms: a general model

In this section we formalize the operating procedure of ISD algorithms. We first consider a high level and general description, which encompasses all algorithms of this family; this modelization will turn useful, in the following sections, when we will study the best performances we can achieve, when this type of algorithms is employed to estimate the weight distribution of a linear code.

A. ISD in a nutshell

By ISD, we refer to a randomized algorithm

$$\begin{aligned} \text{ISD}\colon \mathbb{F}_q^{r\times n} \times \{0,\dots,n\} &\longrightarrow \mathscr{P}(\mathscr{C}_w), \\ (\mathbf{H}, w) &\longmapsto X. \end{aligned}$$

The algorithm receives as input a description for the code (say, a parity-check matrix $\mathbf{H}$) and the desired weight w, and returns an element X in the powerset of $\mathscr{C}_w$, i.e. a set of codewords with weight w. All ISD algorithms share a common procedure which is highlighted in Algorithm 1. In particular, the operations performed by any ISD algorithm can be divided into three main steps:

- *Partial Gaussian Elimination*: a random permutation π of length n is sampled (line 2 in the algorithm) Then, Partial Gaussian Elimination (PGE) with parameter $\ell \in \mathbb{N}$, $1 \le \ell \le n-k$ is performed. In other words, one checks whether it is possible to apply a change of basis on the permuted parity-check matrix $\pi(\mathbf{H}) \in \mathbb{F}_q^{r\times n}$, so that a fmatrix with the following structure is obtained

$$\left(\begin{array}{c|c} \mathbf{A} \in \mathbb{F}_q^{\ell\times(k+\ell)} & \mathbf{0} \in \mathbb{F}_q^{\ell\times(n-k-\ell))} \\ \hline \mathbf{B} \in \mathbb{F}_q^{(n-k-\ell)\times(k+\ell)} & \mathbf{I}_{n-k-\ell} \end{array}\right).$$

  Notice that such a matrix is not guaranteed to exist: indeed, if the leftmost $k+\ell$ columns of $\pi(\mathbf{H})$ form a matrix whose rank is $< k$, then PGE cannot be performed. In these cases, a new permutation is sampled.
- *Solving the small instance*: because of the PGE decomposition, we have

$$\mathbf{c} = (\mathbf{c}', \mathbf{c}'') \in \pi(\mathscr{C}) \iff \begin{cases} \mathbf{A}\mathbf{c}'^{\top} = 0, \\ \mathbf{B}\mathbf{c}'^{\top} + \mathbf{c}''^{\top} = \mathbf{0}. \end{cases} \tag{1}$$

  Notice that $\mathbf{c}'$ has length $k+\ell$ and is, de facto, a codeword of the code whose parity-check matrix is $\mathbf{A}$. One restricts the search for $\mathbf{c}'$ by requiring that it has some (low) weight and some specific weight partition. Notice that this is the only step that varies from one ISD algorithm to the other;
- *Producing solutions*: once $\mathbf{c}'$ has been found, one can easily compute the associated $\mathbf{c}''$ from the second row of the linear system in (1). In other words, we produce codewords of the form $(\mathbf{c}', \mathbf{c}'')$ and check whether they have the desired weight w: any such codeword corresponds to the permutation of a codeword in $\mathscr{C}_w$.

Algorithm 1: ISD operating principle

Data: subroutine Solve, parameter $\ell \in \mathbb{N}$, $1 \le \ell \le r$
Input: $\mathbf{H} \in \mathbb{F}_2^{r\times n}$, $w \in \mathbb{N}$
Output: set $Y \subseteq \mathscr{C}_w$

```
// Apply random permutation and do PGE
repeat
    Sample π ←$ S_n;
    Apply PGE on π(H);
until PGE is successful;
// Use a subroutine to find solutions for the
   small instance
X = Solve(A, ℓ);
// Test codewords associated to solutions for
   the small instance
Set Y = ∅;
for c' ∈ X do
    Compute c'' = −c'B^⊤;
    if wt(c') + wt(c'') == w then
        Update Y ← Y ∪ {π^{-1}((c0, c''))};
return Y;
```

This is a very general way to study ISD algorithms, but allows to identify the main quantities we will use for our analysis. Notice that, at this stage, we have provided all the necessary details apart from those of the subroutine Solve. Yet, its functioning is crucial to determine the computational cost of an ISD algorithm. For the moment, we keep it as a very general procedure and, to be as general as possible, consider that it will only return the codewords in $\pi(\mathscr{C}_w)$ that satisfy some constrains, e.g., some specific weight partition. In the following, we will denote by $f_{\text{ISD},\pi} : \mathbb{F}_q^{k+\ell} \longrightarrow \{0,1\}$ such a constraint, and assume that, whenever $f_{\text{ISD},\pi}(\mathbf{c}')$ is equal to 1, this means that the codeword $\mathbf{c}'$ will be among the outputs of the subroutine. Crucial quantities in our analysis will be the success probability and the average number of codewords found for each iteration. We first focus on the success probability, namely, the probability that a given permutation π is successful for an ISD algorithm. In particular, we show that, as long as we do not consider a specific code, the probability that a chosen permutation is valid will only be a function of (i) the weight w, and (ii) the constraints which are imposed by the considered ISD variant.

For what concerns the time complexity of each iteration, we can use the following estimate.

Proposition 1. ***Cost of one iteration***
On average, one iteration of ISD uses a number of elementary

operations (sums and multiplications) over $\mathbb{F}_q$ counted by

$$O\left(\frac{n(n-k+\ell)^2}{p_{\mathsf{inv}}(\ell)}+\mathbb{E}[t_{\mathsf{Solve}}]+\mathbb{E}[|X|]\right),$$

where $p_{\mathsf{inv}}(\ell)=\prod_{i=\ell+1}^{n-k}1-q^{-i}$, $t_{\mathsf{Solve}}(\ell)$ is the cost of the subroutine Solve *(as a function of ℓ) and $|X|$ is the number of solutions which are found, for the small instance.*

Proof: Performing PGE requires a number of operations which is well counted by $n(n-k+\ell)^2$ (for instance, see [4]). The number of times we need to repeated the PGE step, on average, corresponds to the reciprocal of the probability that the chosen permutation π places, on the rightmost side of $\pi(\mathbf{H})$, $n-k-\ell$ columns which form a basis for a space with dimension ℓ. Assuming that all columns of $\mathbf{H}$ behave as random vectors over $\mathbb{F}_q$, with length $n-k$, we get that this probability is

$$\begin{aligned}p_{\mathsf{inv}}(\ell)&=\prod_{i=0}^{n-k-\ell-1}\left(1-\frac{q^i}{q^{n-k}}\right)\\&=\prod_{i=0}^{n-k-\ell-1}\left(1-q^{-(n-k-i)}\right)=\prod_{i=\ell+1}^{n-k}\left(1-q^{-i}\right).\end{aligned}$$

■

Remark 1. *The term $O\left(\mathbb{E}[|X|]\right)$ is slightly optimistic, since we are omitting some polynomial factors. Indeed, executing instructions 8–9 requires to i) compute $-\mathbf{c}'\mathbf{B}^\top$, and ii) check Hamming weights. With a schoolbook approach, the calculation of $\mathbf{c}'\mathbf{B}^\top$ would require $O\left((k+\ell)^2(n-k-\ell)\right)$ operations. Yet, given that, generically, $\mathbf{c}'$ has low weight and that some precomputations can be used, this cost can be drastically reduced. Also, in practice, one can perform the check on the weight on-the-run: this technique is called early abort and, most of the times, allows to stop the computation of $\mathbf{c}''$ all its $n-k-\ell$ entries are obtained. In the end, we expect that the cost of instructions 8–10 is very limited and can be safely neglected, so that the overall cost of instructions 7–10 corresponds to $O\left(\mathbb{E}[|X|]\right)$, i.e., to the average number of performed iterations.*

B. Lee&Brickell and Stern's algorithms

Details of Lee&Brickell ISD are reported in Appendix A.

Proposition 2. ***Performances of Lee&Brickell's ISD***
The time complexity of the Lee&Brickell Solve *subroutine, with parameter $p\in\mathbb{N}$, $0\le p\le\min\{w,k\}$, is*

$$t_{\mathsf{Solve}}(p)=\binom{k}{p}(q-1)^p.$$

The probability that a codeword $\mathbf{c}\in\mathscr{C}_w$ is returned is

$$p_{\mathsf{ISD}}(w)=\frac{\binom{k}{p}\binom{n-k}{w-p}}{\binom{n}{w}}.$$

Details about Stern's ISD are reported in Appendix B.

Proposition 3. ***Performances of Stern's ISD***
The time complexity of the Stern's Solve *subroutine, with parameters $p,\ell\in\mathbb{N}$, where $0\le p\le\left\lfloor\frac{k+\ell}{2}\right\rfloor$, $0\le\ell\le n-k$ [generalizzare questi bound per qualsiasi valore di w (anche quelli grandi)] is*

$$t_{\mathsf{Solve}}(p,\ell)=L^2/q^\ell+L,$$

where $L=\binom{\frac{k+\ell}{2}}{p}(q-1)^p$. The probability that a codeword $\mathbf{c}\in\mathscr{C}_w$ is returned is

$$p_{\mathsf{ISD}}(w)=\frac{\binom{(k+\ell)/2}{p}^2\binom{n-k-\ell}{w-2p}}{\binom{n}{w}}. \tag{2}$$

Notice that, at least for the binary case, more advanced algorithms exist (e.g., MMT [3] and BJMM [1]). However, they have space complexity (i.e., amount of used memory) which is typically much larger than that of Stern. Also, they do not have a non binary counterpart: so, to avoid distinguishing between the binary and non binary cases, we will omit them from our analysis.

IV. Minimum distance of LDPC codes from Ensemble C

We formally define the ensemble of codes we consider.

Definition 1. *Let $R\in[0;1]$, $n\in\mathbb{N}$ and $v\in\mathbb{N}$, $v\le(1-R)n$. Then, we define $\mathfrak{E}_{R,v,n}$ as the set of codes whose parity-check matrices have constant column weight v. We write $\mathscr{C}\sim\mathfrak{E}_{R,v,n}$ to indicate that $\mathscr{C}$ is sampled uniformly at random from $\mathfrak{E}_{R,v,n}$; this can be done by sampling, uniformly at random, its parity-check matrix $\mathbf{H}$.*

Employing the ensemble, we can define We now study the problem of determining the minimum distance of an LDPC codes as follows. [TODO:

- scrivere formula per distanza minima
- studiare come si comporta al variare di n, assumendo a) v lineare in n, b) $v=o(n)$ (ad esempio, costante o logaritmico in n
- grafici con costo di ISD
- confronto con costo per codici random
- claim che per codici LDPC è più facile.

]

V. Improving Stern ISD

We now describe new techniques to improve Stern's ISD, when one possesses the sparse parity-check matrix $\mathbf{H}$ of a given linear code.

A. Idea I

Exploiting sparsity of $\mathbf{H}$, one can construct the small instance in a convenient way. Remember that the parameter ℓ in Stern's algorithm impacts several aspects of the algorithm:

- the success probability gets smaller as ℓ increases. It can be increased by enlarging p, but this increases the list sizes, as well;
- the lists sizes increase with ℓ, as the length of the small instance grows with ℓ;
- the probability that a pair of list elements collide decays exponentially with ℓ.

Exploiting sparsity, one can construct the small instance in Stern's ISD in a much more convenient way. The first algorithm we present in this paper is based solely on this approach. We introduce in in the next section and then analyze its performances.

B. The algorithm

We want to partition the matrix $\mathbf{U}\pi(\mathbf{H})$ as in Figure 1. Notice that our small instance is again an $\ell \times (k+\ell)$ matrix, but now has a special form as the top-right submatrix is required to be null. This can be obtained, for instance, considering the following approach:

1) select at random t rows of $\mathbf{H}$ and denote their supports by $J_1, \cdots, J_t$
2) set $J = \bigcup_{i=1}^{t} J_i$ and select a permutation σ that moves J to the leftmost positions. At this stage, we have that the first t rows of $\sigma(\mathbf{H})$ are in the form $(\mathbf{A}, \mathbf{0})$, where $\mathbf{A} \in \mathbb{F}_2^{t \times z}$ and $\mathbf{0}$ is the $t \times (n-z)$ null matrix. Notice that we are denoting $z = |J|$;
3) select another permutation $\varphi \in S_n$ that fixes (i.e., does not move) the first z coordinates and compute $\mathbf{H}' = \varphi\big(\sigma(\mathbf{H})\big)$
4) perform PGE on the last $n-(\ell-t)$ rows of $\mathbf{H}'$, aiming to produce the identity $\mathbf{I}_{n-(k+\ell)}$ in the rightmost bottom corner of the matrix.

This, in practice, corresponds to applying the permutation $\pi = \varphi \circ \sigma$ and a change of basis $\mathbf{U}$ in the form $\mathbf{U} = \begin{pmatrix} \mathbf{I}_t & \mathbf{0} \\ & \mathbf{U}' \end{pmatrix}$, where $\mathbf{U}' \in \mathbb{F}_2^{(n-k-t)\times n-k}$. Let us partition $\mathbf{x}' = \pi(\mathbf{x})$ as follows $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, with respective lengths z, $k+\ell-z$, $n-(k+\ell)$. Then, the condition $\mathbf{U}\pi(\mathbf{H})\mathbf{x}^\top = \mathbf{0}$ gives rise to the following linear system:

$$\begin{cases} \mathbf{A}\mathbf{x}_1^\top = \mathbf{0}, \\ \mathbf{B}_1\mathbf{x}_1^\top + \mathbf{B}_2\mathbf{x}_2^\top = \mathbf{0}, \\ \mathbf{C}(\mathbf{x}_1, \mathbf{x}_2)^\top = \mathbf{x}_3. \end{cases} \tag{3}$$

The first two rows of the system represent the small instance which, now, can be solved with a two-steps meet-in-the-middle approach:

- split $\mathbf{A}$ into two submatrices, each with $z/2$ columns, and find candidates for $\mathbf{x}_1$ with a meet-in-the-middle approach
- solve the second equation from (3), enumerating candidates for $\mathbf{x}_2$ and applying again a meet-in-the-middle approach
- test the pairs $(\mathbf{x}_1, \mathbf{x}_2)$ using the third equation from (3).

The full description of the algorithm is given in [PSEUDOCODE TO BE INCLUDED!!!] We now derive the analysis for the time complexity of the algorithm, under the assumption that the final permutation π leads to a solution with probability which is not affected by the choice of σ.

Proposition 4. *Let p_1 and p_2 denote the weights of $\mathbf{x}_1$ and $\mathbf{x}_2$. Then, the cost of the algorithm is given by*

$$\frac{n^3 + nL \cdot \left(2 + L2^{-t} + L2^{-\ell}\binom{k+\ell-z}{p_2}\right)}{\binom{z/2}{p_1/2}^2 \binom{k+\ell-z}{p_2}\binom{n-(k+\ell)}{w-p_1-p_2}/\binom{n}{w}},$$

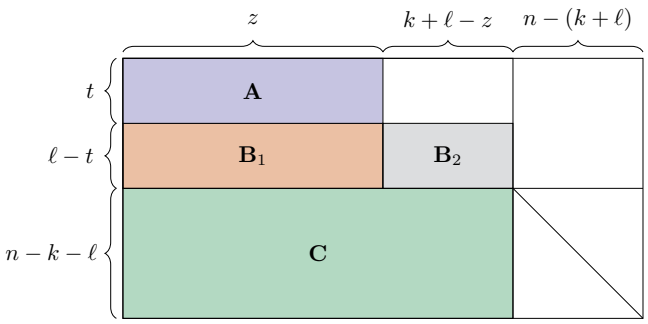

Fig. 1. Representation of $\mathbf{U}\pi(\mathbf{H})$. White areas correspond to portions that are always null.

where $L = \binom{z/2}{p_1/2}$

Proof: [to be written] ∎

VI. Estimates for ensemble C

We now consider the cost of our algorithms, on ensemble C.

A. Algorithm I

Theorem 3. *Let $\bar{z}$ denote the average value of z in Algorithm I. Then*

$$\bar{z} = n \cdot \left(1 - \left(1 - \frac{\binom{n-k-t}{v}}{\binom{n-k}{v}}\right)^t\right).$$

Proof: Since each column of $\mathbf{H}$ has constant weight v and the ones in every column are chosen uniformly at random, by the Coupon's collector problem we get that the expected value of z, which we indicate by $\bar{z}$, is $\bar{z} = n$

We first consider a column: for every set of t rows, the probability that this columns does not has a 1 in any of these rows is $\frac{\binom{n-k-t}{v}}{\binom{n-k}{v}}$. Considering t columns, we get that the probability that the column has at least a 1 in the t selected rows is

$$1 - \left(1 - \frac{\binom{n-k-t}{v}}{\binom{n-k}{v}}\right)^t.$$

If $V = \Omega(n)$ then Multiplying this by n, we get the expected value of z.

∎

VII. LDPC ensemble

We start by considering a family of LDPC codes which we use as a prototype to introduce our analysis. We start with the following definition.

Definition 2. *For $r, n \in \mathbb{N}$, $1 \le r \le n$ and $\rho \in [\![0;1]\!]$, we define $\mathscr{D}_{r,n}(\rho)$ as the distribution over $\mathbb{F}_2^{r\times n}$ such that, if $\mathbf{H} \sim \mathscr{D}_{r,n}(\rho)$, then*

$$h_{i,j} \sim \mathscr{B}(\rho), \quad \forall i \in [1;r],\ j \in [1;n].$$

In other words, to sample a matrix from $\mathscr{D}_{r,n}(\rho)$, one simply has to generate each of its elements according to $\mathscr{D}_{r,n}(\rho)$. When $\rho \ll \frac{1}{2}$, then a typical matrix sampled from

the distribution can be seen as the parity-check matrix of an LDPC code. For instance, a matrix sampled from $\mathscr{D}_{r,n}(\rho)$ is such that its expected Hamming column and row weights are, respectively, $v_r = \rho r$ and $v_c = \rho n = \frac{n}{r} v_c$. We then see that there is a clear analogy between a matrix $\mathbf{H} \sim \mathscr{D}_{r,n}(\rho)$ and the parity-check matrix of a regular LDPC codes. Note that any matrix in $\mathbb{F}_2^{r \times n}$ can be sampled from $\mathscr{D}$, but a typical matrix coming from $\mathscr{D}$ is expected to be, basically, a sparse matrix.

To turn the distribution $\mathscr{D}_{r,n}(\rho)$ into a distribution of parity-check matrices for codes with redundancy r, we need a technical refinement, which we introduce next.

Definition 3. *For $r, n \in \mathbb{N}$, $1 \leq r \leq n$ and $\rho \in [\![0; 1]\!]$, we define $\widetilde{\mathscr{D}}_{r,n}(\rho)$ as the distribution over $\mathbb{F}_2^{r \times n}$ which arises from the following experiment:*

1. *sample $\mathbf{H}$ according to $\mathscr{D}_{r,n}(\rho)$;*
2. *output $\mathbf{H}$ if $\mathrm{rank}(\mathbf{H}) = r$, otherwise restart from step 1.*

It is easily seen that any matrix sampled from $\widetilde{\mathscr{D}}_{r,n}(\rho)$ defines the parity-check matrix of a code with redundancy r and length $\leq n$. In the next section we derive the average weight distribution of the codes generated from $\widetilde{\mathscr{D}}_{r,n}(\rho)$.

A. Average weight distribution over $\widetilde{\mathscr{D}}_{r,n}(\rho)$

We start with the following theorem, where we derive the probability that a random weight-d codeword is in the kernel of a matrix sampled from $\mathscr{D}_{r,n}(\rho)$.

Theorem 4. *Let $\mathbf{c} \in S_{n,d}$ and $\mathbf{H} \sim \mathscr{D}_{r,n}(\rho)$. Then*

$$u_{r,n}(d, \rho) = \Pr[\mathbf{c} \in C(\mathbf{H})] = \left(\sum_{\substack{i \in [0;d] \\ i \text{ even}}} \binom{d}{i} \rho^i (1-\rho)^{d-i} \right)^r .$$

Proof: We are going to have $\mathbf{c} \in C(\mathbf{H})$ if and only if, for every row $\mathbf{h}$ of $\mathbf{H}$, we have $\mathbf{h}\mathbf{c}^\top = 0$. Since $\mathbf{H}$ is sampled at random from $\mathscr{D}_{r,n}(\rho)$, every row is constituted by n independent and uncorrelated random variables with distribution $\mathscr{D}(\rho)$. Then, we have

$$\Pr[\mathbf{h}\mathbf{c}^\top = 0] = \sum_{\substack{i \in [0;d] \\ i \text{ even}}} \binom{d}{i} \rho^i (1-\rho)^{d-i}.$$

Since there are r rows, we consider the r-th power of the above quantity to obtain the desired probability. ∎

Theorem 5. *Let $\mathbf{H} \sim \mathscr{D}_{r,n}(\rho)$; then*

$$\langle |C_d(\mathbf{H})| \rangle = g_{r,n,\rho}(d) = \binom{n}{d} u_{r,n}(d, \rho)$$

Proof: Let $f(\mathbf{c}, \mathbf{H})$ be the function that returns 1 if $\mathbf{c} \in C(\mathbf{H})$, and 0 otherwise. We then have

$$\langle |C_d(\mathbf{H})| \rangle = \sum_{\mathbf{H} \in \mathbb{F}_2^{r \times n}} \Pr[\mathbf{H} \mid \mathbf{H} \sim \mathscr{D}_{r,n}(\rho)] \sum_{\mathbf{c} \in S_{n,d}} f(\mathbf{c}, \mathbf{H}),$$

where $\Pr[\mathbf{H} \mid \mathbf{H} \sim \mathscr{D}_{r,n}(\rho)]$ is the probability that the distribution $\mathscr{D}_{r,n}(\rho)$ outputs $\mathbf{H}$. With a simple rewriting, we have

$$\begin{aligned} \langle |C_d(\mathbf{H})| \rangle &= \sum_{\mathbf{c} \in S_{n,d}} \sum_{\mathbf{H} \in \mathbb{F}_2^{r \times n}} \Pr[\mathbf{H} \mid \mathbf{H} \sim \mathscr{D}_{r,n}(\rho)] f(\mathbf{c}, \mathbf{H})] \\ &= \sum_{\mathbf{c} \in S_{n,d}} \sum_{\substack{\mathbf{H} \in \mathbb{F}_2^{r \times n} \\ \mathbf{c} \in C(\mathbf{H})}} \Pr[\mathbf{H} \mid \mathbf{H} \sim \mathscr{D}_{r,n}(\rho)] \\ &= \sum_{\mathbf{c} \in S_{n,d}} u_{r,n}(d, \rho). \end{aligned}$$

Since $u_{r,n}(d, \rho)$ is independent of $\mathbf{c}$, it is enough to multiply such a quantity by the number of elements in $S_{n,d}$, which is given by $\binom{n}{d}$. ∎

Notice that the quantity expressed in Theorem 5 is not exactly the average weight distribution of codes with redundancy r. Indeed, remember that $\mathscr{D}_{r,n}(\rho)$ is a distribution of matrices with r rows, but there is no guarantee about the rank. To consider the rank, we need to make use of the distribution $\widetilde{\mathscr{D}}_{r,n}(\rho)$. However, unless extreme parameters are considered, it is very likely that a matrix sampled from $\mathscr{D}_{r,n}(\rho)$ has full rank r, so that we can safely employ $g_{r,n,\rho}(d)$ for the average weight distribution of matrices sampled from $\widetilde{\mathscr{D}}_{r,n}(\rho)$. We formalize such an assumption in the following.

Assumption 2. *We assume that the distributions $\mathscr{D}_{r,n}(\rho)$ and $\widetilde{\mathscr{D}}_{r,n}(\rho)$ are indistinguishable. Then, if $\mathbf{H} \sim \widetilde{\mathscr{D}}_{r,n}(\rho)$, we can use $g_{r,n,\rho}(d)$ to obtain $\langle |C_d(\mathbf{H})| \rangle$.*

Remark 2. The validity of the above assumption will be confirmed in the remainder of the paper, where we make use of numerical simulations to confirm our treatment. Yet, we can also give some theoretical arguments to sustain it. For the sake of simplicity, let us consider the case of $\rho = 1/2$. The probability that $\mathbf{H} \sim \mathscr{D}_{r,n}(\rho)$ does not have full rank is given by $\prod_{i=1}^{r-1}(2^n - 2^i)$. If $n = r$, then this probability (for sufficiently large n) tends to 0.288. So, one can distinguish between $\mathscr{D}_{n,n}(\rho)$ and $\widetilde{\mathscr{D}}_{n,n}(\rho)$ rather easily: it is enough to sample a large enough number of matrices and check whether they have full rank or not. However, for $n > r$, then the probability to obtain a non full rank matrix from $\mathscr{D}_{n,n}(\rho)$ becomes negligible. Indeed, it is easily seen that for large n and r sufficiently lower than n, the previous probability asymptotically tends to 1.

REFERENCES

[1] A. Becker, A. Joux, A. May, and A. Meurer, "Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding," in *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*. Springer, 2012, pp. 520–536.

[2] B. Doerr, "Probabilistic tools for the analysis of randomized optimization heuristics," *Theory of evolutionary computation: Recent developments in discrete optimization*, pp. 1–87, 2020.

[3] A. May, A. Meurer, and E. Thomae, "Decoding random linear codes in $\mathcal{O}(2^{0.054n})$," in *Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17*. Springer, 2011, pp. 107–124.

[4] C. Peters, "Information-set decoding for linear codes over f q," in *International Workshop on Post-Quantum Cryptography*. Springer, 2010, pp. 81–94.

[5] J. Stern, "A method for finding codewords of small weight," *Coding theory and applications*, vol. 388, pp. 106–113, 1989.

APPENDIX A: LEE & BRICKELL ISD

The Lee&Brickell ISD uses $\ell = 0$. In other words, the matrix $\mathbf{A}$ is actually not considered (since it has zero rows) and (1) is actually solved considering that $\mathbf{c}'$ is a vector with low weight p. So, one goes through all $\binom{k}{p}(q-1)^p$ candidates for $\mathbf{c}'$ and, for each candidate, computes the corresponding $\mathbf{c}''$ as $-\mathbf{c}'\mathbf{B}^\top$.

In such a case, the constraint is simply that the permutation π permutes the solution $\mathbf{c}$ so that, on the leftmost k coordinates, only p set coordinates are placed. So, the success probability of one iteration is

$$\frac{\binom{k}{p}\binom{n-k}{w-p}}{\binom{n}{w}}.$$

APPENDIX B: STERN' ISD

B. Stern's algorithm

Among all the possible instantiations of ISD, in this paper we take into consideration Stern's algorithm [5], which is one of the most used, as well as one of the fastest on a classical computer. Given the description we have provided in the previous section, we limit ourselves to describe the subroutine Solve. This will also allow us to derive a closed form formula for $p_{\mathsf{ISD}}(\ell)$.

Algorithm 2: Stern Solve subroutine

Data: $p \in \mathbb{N}$, $0 \leq \left\lfloor \frac{k+\ell}{2} \right\rfloor$
Input: $\mathbf{A} \in \mathbb{F}_q^{\ell \times (k+\ell)}$, $\ell \in \mathbb{N}$
Output: set X with solutions of the small instance, with weight $2p$ equally partitioned

/* Partition $\mathbf{A}$ */
Write $\mathbf{A} = (\mathbf{A}', \mathbf{A}'')$, where $\mathbf{A}' \in \mathbb{F}_q^{\ell \times \left\lfloor \frac{k+\ell}{2} \right\rfloor}$, $\mathbf{A}'' \in \mathbb{F}_q^{\ell \times \left\lceil \frac{k+\ell}{2} \right\rceil}$;
/* Enumerate candidates for $\mathbf{x}'$ and $\mathbf{x}''$ */
Set $\mathscr{L}_1 = \left\{ (\mathbf{x}', \mathbf{x}'\mathbf{A}'^\top) \mid \mathbf{x}' \in \mathscr{S}_p \right\}$;
Set $\mathscr{L}_2 = \left\{ (\mathbf{x}'', -\mathbf{x}''\mathbf{A}''^\top) \mid \mathbf{x}'' \in \mathscr{S}_p \right\}$;
/* Find collisions (using efficient strategy, e.g., sorting plus binary search) */
Compute $\mathscr{X}$, the set of all pairs $(\mathbf{x}', \mathbf{x}'') \in \mathscr{S}_p \times \mathscr{S}_p$ such that $\mathbf{x}'\mathbf{A}'^\top = -\mathbf{x}''\mathbf{A}''^\top$;
return $\mathscr{X}$

At this point, the main assumption on which this algorithm relies is the following: it assumes that weight w codewords in $\mathscr{C}$ have $2p$ non null entries in the first $k+\ell$ positions (respectively divided into two block of equal length $(k+\ell)/2$ and weight p) and the remaining $w-2p$ set entries in the rightmost $n-k-\ell$ positions. In particular, the algorithm aims at finding the leftmost vector using the standard collision search technique (sometimes, also called meet-in-the-middle): It indeed creates two lists $\mathscr{L}_1$ and $\mathscr{L}_2$ through the enumeration of $\mathscr{S}_p$, together with their partial syndromes. As it is well known, the merge can be efficiently computed using a sort plus binary search approach, taking time

$$O\left(\max\left\{ |\mathscr{L}_1| \cdot \log_2\left(|\mathscr{L}_1|\right), |\mathscr{L}_2| \cdot \log_2\left(|\mathscr{L}_2|\right) \right\} \right).$$

Notice that the lists have sizes given by

$$L_1 = \binom{\left\lfloor \frac{k+\ell}{2} \right\rfloor}{p}(q-1)^p,$$

$$L_2 = \binom{\left\lceil \frac{k+\ell}{2} \right\rceil}{p}(q-1)^p.$$

Getting rid of the logarithmic factor, and taking into consideration that the resulting list $\mathscr{X}$ needs to be somehow allocated, we can consider that the overall cost of merging the two lists is given by

$$O\left(\max\left\{ |\mathscr{L}_1|, |\mathscr{L}_2|, |\mathscr{X}| \right\} \right).$$

When $\mathscr{L}_1$ and $\mathscr{L}_2$ are formed by elements without any relevant structure, we can safely consider that each pair of elements in $\mathscr{L}_1$ and $\mathscr{L}_2$ results in a collisions with probability $q^{-\ell}$. This is a frequently employed heuristic, which corresponds to assume that each entry of the associated syndromes is uniformly distributed over $\mathbb{F}_q$. In such a case, we can set

$$|\mathscr{X}| = |\mathscr{L}_1| \cdot |\mathscr{L}_2| q^{-\ell} = L_1 L_2 q^{-\ell}.$$

Remark 3. *For the sake of simplicity, we can neglect floors and ceiling. This way, we have* $L_1 = L_2 = L = \binom{\frac{k+\ell}{2}}{p}(q-1)^p$ *and* $|\mathscr{X}| = L^2 q^{-\ell}$. *This way, the cost of the overall subroutine* Solve *becomes*

$$t_{\mathsf{Solve}} = L\left(2 + Lq^{-\ell}\right).$$

so that, if we denote with

$$L = \binom{(k+\ell)/2}{p}(q-1)^p$$

the number of elements of $\mathscr{L}_0$ and $\mathscr{L}_1$, the associated complexity is given by $L^2/q^\ell + L$.