Decoding of LDPC quantum codes

Lucien Grouès

Decoding of LDPC quantum codes

Lucien Grouès

March 7, 2023

Contents

Chapter 1

Introduction

The promises of quantum computation Quantum computation is a promising way of computing by manipulating materials following the laws of quantum physics. Some of these particular laws, such as the superposition principle, can be exploited to perform fundamental tasks much faster than a classical computer. For example, Shor's famous algorithm [Sho97] provides an almost exponential speed up for the problem of integer factorization compared to the fastest classical algorithm [LLMP93]. Nevertheless, some of these laws can also be problematic during computations, for example the no-cloning theorem which forbids copying quantum information.

The challenges of physical implementations of quantum computers Several physical implementations of a quantum computer exist already, implementing qubits by particles as diverse as ions [KMW02], electrons [CDT$^+$06] or even photons [KLM01]. However, due to the fragility of quantum information none of them achieved a similar breakthrough as classical transistors, which would allow one to actually benefit from the speed-ups of quantum algorithms. Until now, no quantum computer has achieved a task (even a useless one) impossible in realistic time by a classical computer. This first milestone called quantum supremacy [Pre12] does not seem far away thanks to improvements made by physical implementations [AAB$^+$19], but at the same time it is postponed by classical computation perpetual improvements [PGN$^+$19]. Due to decoherence [Sho95]–a form of noise that arises when a quantum system interacts with the environment–it is not realistic to wait for quantum computers to be accurate enough to run useful algorithms as is. While it is possible to reduce decoherence by employing several techniques such as cooling down the quantum system, we cannot totally remove it since we must interact with the quantum system to do the computations. Thus, we need solutions to make the quantum information more robust.

Classical error correction Error correction is also well studied in the classical case with the example of wireless communication that can still be quite noisy. The solution to this problem was first found by Richard Hamming, who created the first practical class of error correcting codes [Ham50]. The idea was to add redundancy to the physical information so that even if it was damaged by noise, it was still possible to recover the logical information. Since then, many classes of error correcting codes have been discovered. One of them is the class of low density parity-check matrix (LDPC) codes [Gal62]. This class of codes introduced by Gallager is of particular interest as it contains some codes exhibiting important properties called asymptotically good codes (constant encoding rate and the size of the smallest undetectable error is proportional to the number of physical bits). Moreover, these codes can be efficiently decoded using the belief propagation decoder (BP) [RU08a].

Quantum error correction Quantum error correction is still interesting for calculus, since quantum gates are much more noisy than classical gates. To cope with this high noise, correction of the error must be very fast, despite a noisy characterization of the error. For a long time, it was not sure whether quantum error correction was possible, until Shor [Sho95] and Steane [Ste96] invented the first quantum error correcting codes. Another breakthrough came when Gottesman introduced stabilizer codes [Got97], making quantum error correction almost as convenient as classical error correction. For example, topological codes include the well-known toric code [Kit03] and its many variants [AMT12, YK17, BATB$^+$21, BK98] Aharonov and Ben-Or then proved the threshold theorem [ABO97] lay the foundation of quantum fault-tolerant computation: it is possible to turn any logical quantum circuit into a fault-tolerant one able to perform the wanted computation in presence of noise. Gottesman later improved their scheme for quantum LDPC codes coming with an efficient decoder [Got14a], by reducing the number of qubits that were required to run the fault-tolerant circuit. Indeed, while the original scheme required an overhead with a polylog dependency with the size of the circuit, the new scheme allows the overhead to stay constant: one logical qubit can always be replaced by the same number of noisy physical qubits, independently of the size of the circuit.

Good quantum codes Despite all these advances, quantum error correction still lacked good LDPC codes as existed for classical error correction. The next advance was the introduction of hypergraph product codes [TZ13] by Tillich and Zémor. These LDPC codes are almost asymptotically good (constant rate, but the size of the smallest error is only proportional to the square root of the number of physical qubits) and were given a dedicated decoder called the small-set-flip decoder (SSF) [FGL18] with which they satisfied the conditions of Gottesman's fault-tolerant scheme. Thereafter, Pantelev and Kalachev [PK21b, PK22] eventually proved the existence of asymptotically good LDPC quantum codes.

Adapting classical decoders to quantum codes While we could say that quantum error correction finally got the equivalent of classical LDPC codes, they still lack the efficient decoder that made the classical codes so useful. Unfortunately, we cannot simply apply BP to the quantum case as it suffers from the peculiarities of quantum codes despite some adaptations [PC08]. Indeed, while every non-trivial errors will damage a classical codeword, degenerate errors– not trivial errors which act trivially on the code–exist in quantum error correction. Such errors are numerous and can be small, and unfortunately classical decoders such as BP cannot handle them. They will typically add many symmetries on which BP will get stuck [PC08], unable to decide which error actually happened. Nevertheless, one of the best leads seems to combine it with other decoders [PK21a], even if a recent paper proposed a promising adaptation [DCMS22]. Finally, another powerful classical decoder based on linear programming, the LP decoder [FWK05], was adapted by Li and Vontobel to the quantum case [LV18] yet with poorer numerical results.

Local decoders These decoders need to aggregate information from the state of every part of the quantum computer and gather it in one classical processor that will compute the correction. This can be complicated because it requires a large bandwidth for the information to be corrected often enough [Del20]. Moreover, it must be compatible with keeping the quantum computer cold enough to limit the decoherence. To overcome this problem, it is possible to employ decoders only needing access to little information to compute a correction. Such decoders exist for topological codes in the form of cellular decoders [HCEK15, KP19, BDMT17a]. They are made of several local decoders which can access only nearby qubits. Surprisingly, a parallel version of SSF can run both locally and in constant time on some hypergraph product codes [Gro19a].

Outline of the thesis In this thesis, we will study the decoding of LDPC quantum codes, and in particular of hypergraph product codes with several decoders. Chapter 2 will first introduce the concepts and formalism of quantum error correction by relying on their counterparts in classical error correction. In chapter 3 we will study a first decoder solely focused on performance obtained from the combination of two already existing decoders, BP and SSF. We will show through simulations that it has improved performance on random hypergraph product codes compared to each of the decoders alone. In chapter 4 we will concentrate on the LP decoder, a decoder easier to analyze and exhibiting interesting theoretical guarantees. We will manage to prove negative results affecting the asymptotic performance on several LDPC quantum codes, while showing that it still performs well on realistic hypergraph product codes of realistic size. Finally, in chapter 5 we will concentrate on local decoders for the decoding of quantum LDPC code. In particular, we will numerically study a parallel variant of SSF in its fastest and most local implementation.

Chapter 2

Preliminaries

In this chapter, we introduce quantum computing concepts. We first present in section 2.1 the concept and formalism of classical error correction, which are easier to understand yet broadly shared with their quantum counterparts. In section 2.2 we will introduce quantum computing itself by comparison to the classical case.

2.1 Classical error correction

The goal of this section is to take advantage of the simplest case of the classical error correction to introduce as much formalism as possible, and simply adapt them later to the quantum case. In §2.1.1 we briefly introduce the representation of classical information and useful formalism. In §2.1.2 we present the relevant noise models against which we want to protect the information and in §2.1.3 the codes employed to this end. Finally, we go over interesting decoding algorithms in §2.1.4 to §2.1.8.

2.1.1 Information representation

In the classical case, the fundamental unit of information is called a bit, taking value in $\{0, 1\}$ and acting as an element of $\mathbb{F}_2$. The group operation on $\mathbb{F}_2$ is $\oplus$, the sum modulo 2, such that $1 \oplus 1 = 0 \oplus 0 = 0$ and $1 \oplus 0 = 0 \oplus 1 = 1$. It is the main bit operation we will manipulate, together with the bit flip, which turns a 1 to 0 and a 0 to 1. These two operations correspond to the logical gates XOR and NOT.

Similarly, a word made of n bits is represented as a column vector part of $\mathbb{F}_2^n$. We call n the length of the word. We can also perform the sum modulo 2 of two words w and w' of the same length n: it is the word of length n such that for every i in $[\![1, n]\!]$, $(w \oplus w')_i = w_i \oplus w'_i$.

The length of a word is not its only interesting characteristic, we can also look at the number of its bits set to 1 that we call the *Hamming weight*.

Definition 1 (Hamming weight)**.** *The Hamming weight of a word w of length n is the number of its bits set to 1:*

$$|w| = |\{i \in [\![1, n]\!] \, w_i = 1\}|.$$

Notice that the Hamming weight of a word is also equal to its ℓ^1 norm, which allows us to measure it for elements of $[0, 1]^n$.

The Hamming weight can be utilized as a metric on the set of words of length n, we can thus also define a distance between two words of same length.

Definition 2 (Hamming distance)**.** *The* Hamming distance *of two words w and w' of the same length is the Hamming weight of their sum modulo 2:*

$$d(w, w') = |w \oplus w'|.$$

The Hamming distance between two words tells how easy it is to mistake one for the other : the greater the Hamming distance the more bits have to be modified to go from one to the other and thus the stronger the noise must be. In the next section, we will formalize what we mean by noise.

2.1.2 Error models

Despite the processing of bits has been made almost noiseless thanks to the invention of transistors, some errors may still happen. In particular, wireless communication can still be subject to important noise.

The kind of noise that will corrupt the information will depend on the channel we are studying, and will be described by an error model.

First, we will look at the possible error models for a single bit. In this case, the most simple channels are the *binary erasure channel (BEC)* and the *binary symmetric channel (BSC)*. The BEC erases the value of the bit with probability p and leaves it unchanged otherwise. The BSC flips the bit with probability p and leaves it unchanged otherwise. Notice that in the case of the BEC, we know that the bit was corrupted, which isn't the case for the BSC. Moreover, any BEC with error probability p can be turned into a BSC with error probability $\frac{p}{2}$ by choosing at random the value of an erased bit.

Now that we know how the error model works for a single bit, we must extend it to a whole word by choosing the correlations between the events of having an error on two different bits. For example, an extreme case would be the one, where those events are maximally correlated, and every bit is flipped at the same time with probability p, and left unchanged otherwise. However, we will rather focus on the opposite case, where all errors are totally independent: the *i.i.d error model* (independent and identically distributed). In this case, if the error probability is p, then the error e is such that its coordinates are a collection of i.i.d variables following the Bernoulli distribution of parameter p. We can thus write the corrupted word $\tilde{w}$ obtained by passing the word w through the noisy channel using e:

$$\tilde{w} = w \oplus e.$$

To make the encoding of information resistant to noise, we need to apply error correcting codes.

2.1.3 Error correcting codes

The goal of error correcting codes is to make possible the recovery of information after it went through a noisy channel. We saw for example that it wasn't the case if we encoded naively a binary number, as each encoded number had many neighbors at distance 1, meaning that even a single corrupted bit was enough to obtain the encoding of another number. To prevent this, we need to put further apart the words of interest, and this can be done by adding *redundancy*.

To understand how it works, we can look at the simplest error correcting code: the *repetition code*. The idea behind it is to add redundancy to the word in the most naive way possible, by writing each bit multiple times. To make it able to do error correction against the BSC, we need to write each bit at least 3 times. In doing so, we can correct up to one error by recovering the logical bit following a majority vote. In this case, we see that, whereas the basic unit of information is 1 bit, the basic unit of encoded information is 3 bits. We call it a *block*, and $n = 3$ is thus the *blocklength*. To correct the encoded information, it is required to have access to the entire block of bits. When dealing with a stream of information, you want to avoid waiting for a whole block to arrive before deciding how to correct it. It is, in fact, possible

applying a different kind of error correction called continuous error correction, but that we will not study in this work.

To be more specific, we will focus on *binary linear block code*. In this case, a code of parameter $[n, k]$ is actually a *linear subspace* of $\mathbb{F}_2^n$ of dimension k, and words belonging to it are called *codewords*. To describe it, we can rely on its *generating matrix*, $G \in \mathbb{F}_2^{n\times k}$ whose set of columns is a basis of the code. For example, the generating matrix of the repetition code $[3, 1]$ is simply the all one column vector of length 3 as the code has dimension 1: one block encodes only one logical bit. The generating matrix allows us to easily encode the information, as the physical word w' corresponding to the encoded logical word w is given by $w' = Gw$. Thus, the code $\mathcal{C}$ is characterized by its generating matrix:

$$\mathcal{C} = \texttt{Im}(G).$$

The generating matrix is very convenient to encode information and adapt logical gates into physical gates. However, it does not give a good insight on whether a word is a codeword or whether it was corrupted. Moreover, it does not give any direction on how to correct the word in the case, where it was corrupted. To do so, we will rather rely on another characterization of the code, called the *parity-check matrix* $H \in \mathbb{F}_2^{m\times n}$:

$$\mathcal{C} = \texttt{Ker}(H).$$

The number of rows m of H is equal to the number of parity constraints we need to restrain $\mathbb{F}_2^n$ to the code $\mathcal{C}$. Indeed, the i-th row can be interpreted as the following parity constraint:

$$\bigoplus_{j\in[\![1,n]\!],H_{i,j}=1} w_j = 0.$$

Given H, we can compute a powerful decoding tool called the *syndrome* of w:

$$s := Hw.$$

A word is a codeword if and only if its syndrome has Hamming weight equal to 0. The syndrome can also be used as a proxy of how far we are from the code space.

With the parameters (n, k), one can define the encoding rate of the code: $R := k/n$. For example, the encoding rate of the $(3, 1)$ repetition code is $\frac{1}{3}$. This characteristic of the code informs on how efficiently it encodes information, and thus how many extra physical bits are required to protect the information. However, it doesn't provide any insight on how well the information is protected. To do so, we often add a third parameter called the *minimal Hamming distance*, or more simply the minimal distance of the code. This parameter tells us the minimal weight of an error mapping one codeword to another one:

$$d := \min\{d(w, w'), w, w' \in \mathcal{C}, w \neq w'\}.$$

We often write it to complete the two other parameters as $[n, k, d]$. For example, the (3,1) repetition code has only two different codewords with either all three bits equal to 0 or all three equal to 1. Thus, the minimal distance is equal to 3 and the complete code parameters are $[3, 1, 3]$. Here, we clearly see the trade-off when increasing the number of repetition of the code, as repeating each bit twice more will indeed double the minimal distance of the code, but also half its *encoding rate*.

The parity-check matrix can also be represented by a bipartite graph called the *Tanner graph* [Tan81] $\mathcal{T}(\mathcal{C}) = (V \cup C, E)$, where V is the set of *variable nodes* and represents the bits (*i.e.*, the columns of H) and C is the set of *check nodes* and represents the checks (the rows of H). For nodes $v_i \in V$ and $c_j \in C$, where $i \in [n]$ and $j \in [m]$, we draw an edge between v_i and

c_j if the i-th variable node is in the support of the j-th check, or equivalently if $H(i,j) = 1$. Nodes linked by an edge are neighbors, and we write $\Gamma(v_i)$ (resp. $\Gamma(c_j)$) the set of all neighbors of the node v_i (resp. c_j). Using this new representation, we can define a new property that can be held by linear block codes: a code $\mathcal{C}$ is *LDPC (low-density parity-check matrix)* [Gal62] if the associated Tanner graph has bounded degree, with left degrees (associated with nodes in V) bounded by Δ_V and right degrees bounded by Δ_C. In terms of constraints, it means intuitively that a single constraint cannot involve too many bits, and a single bit cannot be part of too many constraints. However, notice that this property is linked to the Tanner graph of the code, and though there is only one Tanner graph representing a parity-check matrix, there are many parity-check matrices to characterize the same code. Thus, poorly choosing the characterization of a code can hide its LDPC property. If all left degrees are equal to δ_V and right degrees equal to δ_C, we say that the Tanner graph and its code are (δ_V, δ_C)-regular.

Such LDPC codes are interesting as they allow for efficient decoding, and can be easily created by designing the Tanner graph. In doing so, we will often seek some extra properties such as having a biregular Tanner graph, meaning that all left degrees are equal, and so are all right degrees. In particular, it is possible to randomly generate such a Tanner graph [HEA01] such that the corresponding code has good parameters with high probability. LDPC codes built from random Tanner graphs are called *random LDPC codes*.

It is convenient to gather codes with similar designs, but different blocklength, to study how their parameters evolve as the blocklength grows. For example, we can study the family of repetition codes, for which the minimal distance is proportional to the blocklength, but the encoding rate tends towards 0 as the blocklength grows. We can then say that the repetition codes have a linear minimal distance and a vanishing rate. We say a code family is good when it has both constant rate (to be resource efficient) and linear minimal distance (to protect sufficiently the encoded information).

Once we have encoded the information by applying the generating matrix of the error correcting code, we have effectively protected it. We have indeed added redundancy by writing the logical word of length k as a larger physical codeword of length n, thus stretching apart the words of interest and increasing the Hamming distance between each other to at least d. Because of this, we can in theory draw for each codeword a ball centered on the codeword and with radius $\lfloor \frac{d-1}{2} \rfloor$ without any of these balls overlapping. Whenever the error has weight below, $\lfloor \frac{d-1}{2} \rfloor$ we can then theoretically check the only such ball it belongs to. To correct the word, we would just have to set it to the value of the codeword in the center of the ball. Since a constant error rate implies the weight of the error is proportional to the blocklength, having a linear minimal distance ensures that the correction capacity of the code family won't diminish with the blocklength. Even though this seems like a good way to correct the errors for codes with large minimal distance, it is most of the time impossible to have an explicit description of these balls. Moreover, it does not handle the case, where the balls do not cover the whole space, and an error larger than $\lfloor \frac{d-1}{2} \rfloor$ moves the word out of every ball. Actually, such a brute-force approach would have a time-complexity exponential with the blocklength, which would forbid running it for anything, but small codes.

To make the decoding algorithm, or *decoder*, scalable we want it to have a time-complexity at most polynomial. In practice, we even want them to take a linear time or be even faster, while still being able to correct as many errors as possible. This will be the topic of the next section.

2.1.4 Decoding algorithms

The goal of the decoding algorithm or decoder is not simply to find a codeword, but to find the most likely to be the one we had before applying the error. We call this *maximum likelihood*

(ML) decoding [FJ74]. The decoding problem is simplified when the noise follows the i.i.d error model, as the question we need to answer is now: what is the smallest possible correction to apply to the corrupted word to put it back to the code space? This is, in fact, the most common setting, so many decoding algorithms will try to decode the corrupted word by changing it into the closest codeword. This is precisely what the majority vote decoder does for the repetition code: it flips the least possible bits so that all the bits of the word have the same value. Another approach called *maximum a posteriori (MAP) decoding* [RU08b] approximates the ML decoding by finding for each bit its probability to have been flipped by the error, and then flip every bit for which this probability exceeds 0.5.

We will present in the next section examples of decoding algorithms functioning on random LDPC codes, but to compare them, we need to introduce a few figures of merit which will allow us to know how well they perform. We will typically have a given family of codes, and will want to know which decoder is the most suited to decode it. To do so, the first interesting value is the *word error rate (WER)* of the code at various values p_{phys} of the physical error rate. It can be estimated by sampling errors by an i.i.d distribution of Bernoulli distributions of parameter p_{phys}, applying it to the codeword, and then checking whether the decoding algorithm manages to recover the original codeword knowing only the corrupted word. If it does not succeed, either because it does not manage to go back to the code space, or because it went back to the wrong codeword, we increment the number of block errors, and finally obtain the WER by dividing the number of errors by the total number of trials. In theory, it would be necessary to measure the WER starting from each codeword, as they might not all be equally protected. Fortunately, as we work with *linear* block codes, it is enough to compute the WER for the all-zeros codeword as it is the same for every codeword. Moreover, corrupted words with the same syndrome will have the same most likely correction. Thanks to this, many decoding algorithms only require the syndrome as an input. In fact, every decoding algorithm for linear block codes can be adapted to take only the syndrome as an input, by adding an extra step, where we choose arbitrarily a corrupted word with the right syndrome (it can be done efficiently by solving a linear system), and then returning the same correction.

The lower the WER is, the better the code performs at the given p_{phys}. Moreover, for a given code family, if p_{phys} is lower than the *threshold* the larger the blocklength is the lower the WER is. In the contrary, if p_{phys} is greater than the threshold then the larger the blocklength is, the larger the WER is. By displaying together the curves corresponding to codes of the same family, we can find the threshold by looking at the p_{phys}, where the curves cross.

The value of the threshold provides information on the asymptotic performance of the code family decoded by the given algorithm. Indeed, the WER will tend to zero as the blocklength tends to infinity for any p_{phys} lower than the threshold and will tend to one for p_{phys} higher than the threshold. Because of this, we can hope to be able to protect the information for any physical error rate lower than the threshold, which makes it a very interesting figure of merit to estimate the performance of the decoding algorithm on the given code family. However, the WER may not very low around the threshold for codes of reasonable blocklength, so it can be required in practice to have a threshold much higher than the physical error rate of the error channel.

We said that two different kinds of errors could contribute to the WER, it may sometimes be interesting to count them apart. The first one occurs when the decoding algorithm fails to bring the corrupted word back to the code space, in this case we know that the decoding was unsuccessful; the decoder can return "FAILURE" instead of the wrong correction. The second type of error is called a logical error and occurs when the decoder brought back the corrupted word to a wrong codeword, in this case the failure cannot be detected. For some particular applications of the error correction, it can be interesting to look at the logical error rate (LER), where we only count logical errors. This typically tells us the maximum trust we can have in

the correction whenever the decoder does not return "FAILURE", and can be interesting when we are allowed to start the process again when we are certain that we lost the information, like asking for the message to be sent over again or redoing the computations.

We will now review three decoders adapted to LDPC codes, and that will have a particular interest when we will look at the quantum decoders.

2.1.5 Bit-flip decoder

This first decoder is called the `Flip` decoder and is the simplest allowing to decode generic LDPC codes transmitted through a BSC channel. It is thus very convenient to analyze and allows it to have proven theoretical guarantees for a particular class of LDPC codes. Yet here we will look at it in the general case, when it performs on codes for which it does not have theoretical guarantees.

The functioning of the decoder is described in algorithm 1 and is based on taking advantage of the syndrome's weight as a proxy of the distance from the corrupted word to the code space. The idea is to notice that whenever a variable node has strictly more unsatisfied check nodes (whose corresponding syndrome bit is set to 1) than satisfied ones (whose corresponding syndrome bits are set to 0) we can strictly reduce the weight of the syndrome by flipping the corresponding bit of the corrupted word. The decoder works by iteratively browsing the bits and flipping the ones that will reduce the weight of the syndrome. Eventually, it will not be able to further reduce the syndrome and will return the correction. This happens when the syndrome reached zero, meaning that we are back on the code space. But this can also happen because the decoder got stuck without reaching a codeword, in which case we still return the partial correction. It is also possible not to systematically flip a bit that decreases the syndrome weight, but to add clever constraints or even to choose to flip the bit allowing to diminish the most the syndrome weight to improve the performance of the decoder. Here, we presented the simplest version of this algorithm, since even the best versions perform poorly compared to other decoders such as the one in the following section.

Algorithm 1: `Flip`

Input: Received word $y \in \mathbb{F}_2^n$, a parity check matrix H and its Tanner graph $\mathcal{T}$
Output: $w \in \mathbb{F}_2^n$, the deduced word

$w := y$; `// Update` w `iteratively`
$\mathcal{F} = \emptyset$; `// Flippable vertices`
for $u \in V$ **do** `// Setup phase: update bits`
  If u has more UNSAT than SAT neighbors, add it to $\mathcal{F}$;
end
while $\exists u \in \mathcal{F}$ **do**
 `// While flippable vertices exist`
  flip w_u;
  Update SAT/UNSAT status of checks in $\Gamma(u)$;
  Update the set $\mathcal{F}$;
end
return w;

2.1.6 Belief Propagation decoder

We will now look at the belief propagation decoder (`BP`) [Gal62], also called sum-product decoder. Similarly to `Flip`, it has guarantees in some specific cases, but performs very well in

practice outside these settings.

Belief propagation is a message passing algorithm which allows one to efficiently compute the *likelihood* of each bit to be in error. The likelihood is defined as the probability given the syndrome of the bit to be in error over the probability given the syndrome of the bit to be correct. From the likelihood we can define the *log-likelihood* obtained by taking the logarithm of the likelihood. We will call *fidelity* the absolute value of the log-likelihood. The higher the fidelity of a bit, the more sure we are of the correction we apply to it. While this decoder was originally based on the noisy word [Pea88, KFL01], a syndrome-based can be easily derived. We present in algorithm 2 this syndrome-based version, as it will be convenient for the quantum case.

This algorithm is known to be exact on *cycle-less* graphs [KFL01], thus it would allow doing an exact MAP decoding for any code with a cycle-less Tanner graph. The problem is that having a cycle-less Tanner graph is a big constraint which cannot be met by an asymptotically good code family. In fact, a family of codes with cycle-less Tanner graphs will either have a *vanishing rate* or a *constant minimal distance* [ETV99]. The repetition code is an example of such a code, with linear minimal distance, but vanishing encoding rate, and already comes with a simpler optimal decoder.

In practice, we will thus employ this decoder on Tanner graphs with cycles, which makes its analysis much more complicated and does not guarantee anymore to compute the exact MAP decoding [RU08a]. However, it still performs very well in terms of WER as long as the Tanner graph does not have too many small cycles. Moreover, it can even be combined with other algorithms exploiting the approximated log-likelihoods to obtain a better correction [Fos01] as we will see in the next section.

Algorithm 2: `BP`

Input: Time steps T
Syndrome $s \in \mathbb{F}_2^m$
Error probability p Parity check matrix H and its Tanner graph $\mathcal{T}$

Output: Deduced error $\hat{v} \in \mathbb{F}_2^n$

Initialization: At time $t = 0$: $\forall v_i, \forall c_j \in \Gamma(v_i), m^0_{v_i \to c_j} = \lambda_i^0 = \ln\left(\frac{1-p}{p}\right)$.

for $1 \leq t \leq T$: **do**
  `sum-product-single-step`(t);
end
Terminate:
 `// Log-likelihood computation`
for $i \in [n]$ **do**
  $\lambda_i^t = \lambda_i^0 + \sum_{c_j \in \Gamma(v_i)} m^t_{c_j \to v_i}$
  if $\lambda_i > 0$ **then**
    $\hat{v}_i = 0$
  else
    $\hat{v}_i = 1$
  end
end
return $\hat{v}$

Algorithm 3: `sum-product-single-step`(t)

for $c_j \in C, v_i \in \Gamma(c_j)$ **do**
  // Checks to bits
  `Check-to-bit`(c_j, v_i)
end
for $v_i \in V, c_j \in \Gamma(v_i)$ **do**
  // Bits to checks
  `Bit-to-check`(v_i, c_j)
end

Algorithm 4: `Bit-to-Check`

Input: Variable node v_i, check node $c_j \in \Gamma(v_i)$
Output: $m^{t+1}_{v_i \to c_j}$

return $m^{t+1}_{v_i \to c_j} := \ln\left(\frac{1-p}{p}\right) + \sum_{c_{j'} \in \Gamma(v_i) \setminus c_j} m^t_{c_{j'} \to v_i}$

Algorithm 5: `Check-to-bit`

Input: Check node c_j, variable node $v_i \in \Gamma(c_j)$
Output: $m^{t+1}_{c_j \to v_i}$

return $m^{t+1}_{c_j \to v_i} := (-1)^{s_j} 2 \tanh^{-1}\left(\prod_{v_{i'} \in \Gamma(c_j) \setminus v_i} \tanh\left(\frac{m^t_{v_{i'} \to c_j}}{2}\right)\right)$

2.1.7 OSD post-processing

We will now look at a decoding algorithm called the *ordered statistics decoder (`OSD`)* [FL95] which requires knowing a priori which bits are the most likely to be erroneous. There exists different versions of this decoder, here we will refer to the fastest version called `OSD-0`. The goal is to find, among the sets of bits which bring us back to the code space when flipping them, the one having the worst combined fidelity. This is, in fact, equivalent to finding among the sets of columns of H spanning a subspace containing the syndrome s the one corresponding to the set of bits having the lowest fidelity. To approximate this, we will order the columns of H by increasing fidelity of the corresponding bit, and add them one by one to the set of columns until s belongs to the subspace spanned by this set of columns. To have a unique solution, we want this set of columns to be linearly independent, so we only add columns that do not already belong to the subspace spanned by the current set of columns. We can easily check whether a given vector is spanned by the set of columns by comparing the rank of the set of columns and the rank of the set of columns together with the vector. To describe easily the algorithm, we will introduce the notation M_J (resp. v_J) which refers to the submatrix of M (resp. subvector of v) obtained by keeping only the columns (resp. the coordinates) whose index belongs to J. We will also adopt the notation $[M|M']$ (resp. $[M, v]$) referring to the matrix obtained by juxtaposing side by side the matrix M and the matrix M' (resp. the column vector v). The pseudocode of `OSD` is given in algorithm 6.

This decoding algorithm can be employed as a post-processing to improve the WER of `BP`, as we can take advantage of the absolute value of the log-likelihood computed by `BP` to order the bits by fidelity. This combined decoder is called `BP` $-$ `OSD` and consists in first applying the

Algorithm 6: `OSD`

Input: List of bits indexes ordered by growing fidelity L
Syndromes $s \in \mathbb{F}_2^m$
Parity check matrix H

Output: Deduced error $e \in \mathbb{F}_2^n$ such that $He = s$

Initialization: $J \leftarrow \varnothing, e = 0^n$
for $i \in L$: **do**
  if $rk([H_J|s]) = rk(H_J)$ **then**
    break
  end
  if $rk([H_J|H_i]) > rk(H_J)$ **then**
    $J \leftarrow J \cup \{i\}$
  end
end
$x \leftarrow$ the solution to $H_J x = s$
Replace in e the subvector e_J by x
return e

correction computed by `BP`, updating the syndrome and generating the list L of bits ordered by increasing absolute value of their likelihood, and applying the correction computed by `OSD` using these inputs. In doing so, we can greatly improve the waterfall area compared to `BP` alone and lower its error floor at the cost of an increase in the time complexity. However, since `BP` already has a very low WER alone, and since the post-processing does not take any time when `BP` has already found a correction taking the corrupted word back to the code space, the mean computing time is almost the same and only the worst-case computing time is significantly increased. Since the correction computed by `OSD` is guaranteed to take the corrupted word back to the code space, all the decoding errors will be logical errors and the WER will be equal to the LER. Because of this, the WER is somehow minimized, in the sense that no post-processing can be applied afterward to further reduce it.

The `BP` $-$ `OSD` improves on the already excellent WER of `BP`, yet its theoretical analysis remains quite challenging. Because of this, we will focus in the next section on a decoder with good performance that can also be studied to prove interesting properties.

2.1.8 Linear programming decoder

We will present in this section a new decoder called the *linear programming decoder (LP decoder)* [Fel03, FWK05] which transforms the decoding problem into a linear program. As this decoder will be theoretically studied in this thesis, it will be reviewed with more details and several lemmas will be proven, to familiarize the reader with analyses that we will use again later.

A *linear program* is an optimization problem, where we want to minimize an *objective function* depending linearly on a set of positive real variables, which must satisfy some linear constraints. It is usually written in a systematic form, which efficiently defines it with matrices. A linear program involving n variables subject to m linear constraints described by the matrix $A \in \mathbb{R}^{m \times n}$ and the vector $b \in \mathbb{R}^m$ and minimizing the objective function described by the vector

$c \in \mathbb{R}^n$ has the following *canonical form*:

Find a vector	u
that minimizes	$c^T u$
while satisfying	$Au \leq b$
and	$u \geq 0$.

Written this way, the optimization problem can be efficiently solved by algorithms such as the *simplex algorithm*, which despite a bad worst-case complexity is extremely efficient in most cases [KM72, ST04].

The constraints that must satisfy u define a subspace of $\mathbb{R}^n$, called a *polytope* (which here implies a *convex* polytope). A polytope can be both described as the convex hull of a set of vertices, or as the restriction of the linear space by several linear constraints or *cuts*.

To transform the decoding problem into a linear program, we will define a polytope containing only words with null syndrome, the decoding problem will then consist in finding the closest point to the corrupted word in the polytope. The easiest way to create such a polytope is $\text{Conv}(\{x \in \mathbb{F}_2^n | Hx = 0\})$, where Conv performs the *convex hull* of the set. However, this ideal polytope is not easily described with cuts, and would require too many of them to define a linear program. The solution is to consider a slightly larger polytope called the *fundamental* polytope, which is defined as the intersection of m polytopes, each of them being the convex hull of words satisfying a given check.

Definition 3 (Fundamental polytope)**.** *Let H be an $m \times n$ parity-check matrix. For each row $j \in [m]$, we define the polytope*

$$\mathcal{P}_j(H) := \text{Conv}(\{x \in \{0,1\}^n | (Hx)_j = 0\}),$$

where $Conv(E)$ is the convex hull of E, that is the smallest convex containing E. The fundamental polytope *$\mathcal{P}(H)$ of the parity check matrix H is given by:*

$$\mathcal{P}(H) := \bigcap_{j \in [m]} \mathcal{P}_j(H).$$

We will see in lemma 10 that contrary to the ideal polytope we wanted to use, the fundamental polytope can be described by a realistic number of cuts. Indeed, given the code is LDPC, the number of needed linear constraints scales linearly with the blocklength.

Defining $\mathcal{V}$ as the set of vertices of $\mathcal{P}$, it is clear that we have:

$$\mathcal{V} \cap \{0,1\}^n = \mathcal{C}.$$

Since the linear program always returns an element of $\mathcal{V}$, it would be ideal for every vertex of $\mathcal{P}$ to be integral. However, this is not guaranteed and many vertices of the fundamental polytope are fractional in the general case. Such fractional vertices are called *pseudocodewords* and play a big role in the limitations of the decoding capacity of the LP decoder. Indeed, they can be optimal solutions to the linear program while not corresponding to an actual correction.

To complete the definition of the linear program, we then simply have to choose the right objective function. We said that we wanted to find the point of the fundamental polytope that is the closest to the corrupted word w, the objective function would then be $c : x \to d(w, x)$. Unfortunately, this function is not linear. However, we can replace it by $\tilde{c} : x \to \sum_{1 \leq i \leq n} (-1)^{w_i} x_i$. Indeed, this new function is clearly linear and reaches its minimum on the same vertex of the fundamental polytope.

Lemma 4. *Given $w \in \{0,1\}^n$, $S \subset [0,1]^n$ and the functions $c : x \mapsto d(w,x)$ and $\tilde{c} : x \mapsto \sum_{1\leq i\leq n} (-1)^{w_i} x_i$, we have:*

$$\arg\min_{x\in S}(c(x)) = \arg\min_{x\in S}(\tilde{c}(x)).$$

Proof. To prove this, it is enough to show that real numbers a and b exist with $a \geq 0$ such that for all x in $[0,1]^n$, $\tilde{c}(x) = ac(x) + b$. For all $x \in [0,1]^n, w \in \{0,1\}^n$:

$$\begin{aligned}
\tilde{c}(x) &= \sum_{1\leq i\leq n} (-1)^{w_i} x_i \\
&= \sum_{1\leq i\leq n, w_i=0} x_i - \sum_{1\leq i\leq n, w_i=1} x_i \\
&= \sum_{1\leq i\leq n, w_i=0} (|w_i - x_i|) - |w| + \sum_{1\leq i\leq n, w_i=1} 1 - x_i \\
&= \sum_{1\leq i\leq n, w_i=0} (|w_i - x_i|) - |w| + \sum_{1\leq i\leq n, w_i=1} |w_i - x_i| \\
&= \sum_{1\leq i\leq n, w_i=0} (|w_i - x_i|) - |w| \\
&= c(x) - |w|.
\end{aligned}$$

Taking $a = 1$ and $b = |w|$ satisfies all the conditions, which concludes the proof. □

We can now define the LP decoder effectively [FWK05].

Definition 5 (LP decoder for the BSC)**.** *Given a parity-check matrix H and a noisy word $\tilde{w}$, the decoder returns the corrected word $\bar{w}$ when it is integral, and "FAILURE" otherwise, where $\bar{w}$ is given by:*

$$\bar{w} = \arg\min_{x\in\mathcal{P}}(d(x,\tilde{w})), \tag{2.1}$$

, where $d(x,\tilde{w}) := \sum_{i=1}^{n} |x_i - \tilde{w}_i|$ is the generalization of the Hamming distance for fractional words.

We will now look at the greatest advantages of this decoder, which are its theoretical guarantees, which come directly from the clever definition of the fundamental polytope [FWK05].

The first one is called the maximum likelihood certificate and gives us guarantees on the correction whenever the decoder did not output "FAILURE" [FWK05].

Theorem 6 (Maximum likelihood decoding certificate)**.** *Suppose that the LP decoder outputs a codeword $c \in \{0,1\}^n$. Then, c is the maximum-likelihood codeword.*

Proof. We know from the definition of the fundamental polytope that it contains all the codewords. Moreover, choosing the codeword the closest to the corrupted word is equivalent to performing ML decoding on the BSC. Thus, since the returned word minimizes the distance to the corrupted word on the fundamental polytope it is more likely than any other codeword, it is thus the ML correction.

□

It can be interpreted as a guarantee that we can have high confidence in the correction returned by the LP decoder, as when it does return a correction it is the same as the optimal one. Because of this, the LP decoder has some kind of contrary approach to decoding than `BP-OSD`, as, whereas `BP-OSD` removes every error due to an incapacity to reach the code space,

but doing so increasing the LER, the LP decoder minimizes the number of logical errors by returning corrections only when they are the most likely.

It should be noticed that this theorem does not give any guarantees on the capacity of the LP decoder to successfully correct a given error: even if an error would be successfully corrected by the ML correction, the LP decoder may still output "FAILURE". Fortunately, a complementary guarantee exists, for which we first need to introduce the *fractional distance*.

Definition 7 (fractional distance)**.** *Given a code described by the parity-check matrix H with fundamental polytope $\mathcal{P}$, the minimum fractional distance* $\mathrm{d}_{\mathrm{frac}}$ *of the code is defined as*

$$\mathrm{d}_{\mathrm{frac}} = \min_{x \in \mathcal{V}, x \neq 0} (|x|),$$

where $|x| := \sum_{i=1}^{n} |x_i|$.

This is clearly a smaller variant of the minimal distance, as it does not only take the smallest distance between two codewords, but also between a codeword and a pseudocodeword. Yet, contrary to the original, it comes with an algorithm allowing to efficiently compute it by solving a series of linear problems. This is particularly interesting for random LDPC codes, for which the minimal distance is often unknown. It should be noticed that the fractional distance does not only depend on the code. It also depends on its fundamental polytope and therefore on its Tanner graph. Because of this, similarly to what we saw with codes that could lose their LDPC property to a poor choice of their Tanner graph, a code may have a smaller fractional distance if the Tanner graph chosen to describe it is inadequate. This can be a problem as the interest of the following theorem highly depends on the capacity to have codes with large fractional distances.

Theorem 8 (Decoding certificate)**.** *If the error is of weight lower than* $\lceil \frac{\mathrm{d}_{\mathrm{frac}}}{2} \rceil - 1$*, then the LP decoder will properly correct it.*

This theorem highly resembles the theoretical correcting capacity of an error correcting code with minimal distance d. This time the threshold is slightly lower, but can be achieved in practice with an actual usable decoder and not only a theoretical one.

We will now see two new characterizations of the fundamental polytope which are more tractable than the intersection of convex hulls. The first one is the definition by *cuts*, which is necessary to make explicit the linear program solved in the LP decoder. The second one relies on the concept of *valid configurations* to easily check whether a particular point is in the fundamental polytope. To give these two characterizations, we first need to define even (resp. odd) cardinality sub-neighborhoods, which are simply for a given vertex of a graph the set of all the subsets of its neighborhood with even (resp. odd) cardinality.

Definition 9 (Even/odd cardinality sub-neighborhood)**.** *Let $\mathcal{T}$ be a Tanner graph. For each check node $c_j \in C$, we define E_i^0 (resp. E_i^1), the set of even (resp. odd) cardinality sub-neighborhoods of c_j, as:*

$$E_j^0 := \{S \subseteq \Gamma(c_i) \mid |S| \equiv 0 \mod 2\},$$
$$E_j^1 := \{S \subseteq \Gamma(c_i) \mid |S| \equiv 1 \mod 2\}.$$

Intuitively, the reason we are interested in such objects is that a satisfied (resp. unsatisfied) check will have the set of its neighbors equal to 1 belonging to its set of even (resp. odd) cardinality sub-neighborhoods.

As advertised before, we can now indeed describe the fundamental polytope by a realistic number of cuts, given that the code is LDPC.

Lemma 10 (Fundamental polytope with cuts)**.** *Let x be a point in $[0,1]^n$, then x is in the fundamental polytope $\mathcal{P}$ of H iff:*

$$\forall j \in [m], \forall S \in E_j^1 \colon \sum_{v_i \in S} x_i + \sum_{v_i \in \Gamma(c_j) \setminus S} (1 - x_i) \leq |\Gamma(c_j)| - 1.$$

Proof. It suffices to show that for any $j \in [m]$ $x \in \mathcal{P}_j$ iff

$$\sum_{v_i \in S} x_i + \sum_{v_i \in \Gamma(c_j) \setminus S} (1 - x_i) \leq |\Gamma(c_j)| - 1$$

holds for any subset $S \in E_j^1$, which can easily be checked by inspection. □

The idea behind these cuts is that each cut defined by j and S will be violated by an integral point only when the set of variable nodes set to 1 in the neighborhood of check node c_j is precisely the set S. Indeed, it is easy to check that in this case the left term is equal to $|\Gamma(c_j)|$, while it is at most $|\Gamma(c_j)| - 1$ for every other integral neighborhood of c_j. Since S is an odd cardinality sub-neighborhood, we forbid one by one each of the forbidden neighborhoods of each check.

This characterization is very convenient as it can be injected directly into the linear program. However, it is not a practical way to verify whether a given point belongs to the fundamental polytope. To do so, we need the concept of valid configuration.

Lemma 11 (Valid configurations)**.** *Let x be a point of $[0,1]^n$, then x is in the fundamental polytope $\mathcal{P}$ if and only if a configuration $\{w_{j,S}\}_{j \in [\![1,m]\!], S \in E_j^0}$ exists such that:*

$$0 \leq w_{j,S} \leq 1 \qquad \forall j \in [\![1,m]\!], \forall S \in E_j^0 \tag{2.2}$$

$$\sum_{S \in E_j^0} w_{j,S} = 1 \qquad \forall j \in [\![1,m]\!] \tag{2.3}$$

$$\sum_{S \in E_j^0} w_{j,S} \mathbb{1}_S(v_i) = x_i \qquad \forall j \in [\![1,m]\!], \forall v_i \in \Gamma(c_j) \tag{2.4}$$

Proof. This definition is actually based on the original definition of the fundamental polytope as an intersection of convex hulls. Here, we will give a sketch of the proof, the complete proof can be found in [FWK05]. We will show that for a given fixed j, equations 2.2, 2.3 and 2.4 guarantee that x can be written as a convex sum of elements of $\{y \in \{0,1\}^n | (Hy)_j = 0\}$. This is equivalent to show that it can be written as a convex sum of elements of $\mathrm{Conv}(\{y \in \{0,1\}^n | (Hy)_j = 0\})$. The coefficients of the convex sum are actually given by $(w_{j,S})_{S \in E_j^0}$. Indeed, the equations 2.2 and 2.3 guarantee that those are the coefficients of a convex sum. We just have to check that this convex sum is indeed equal to x. To do so, we have to specify the exact elements of $\mathrm{Conv}(\{y \in \{0,1\}^n | (Hx)_j = 0\})$ which are summed. The element $\tilde{x}$ indexed by (j,S) will be exactly equal to x outside the neighborhood of the j^{th} check, and its support restrained to this neighborhood will be S. This is indeed an element of $\mathrm{Conv}(\{y \in \{0,1\}^n | (Hy)_j = 0\})$, since the coordinates within $\Gamma(c_j)$ satisfy the parity constraint. Moreover, all the other coordinates can be independently set to 0 or 1 to belong to $\{y \in \{0,1\}^n | (Hy)_j = 0\}$, and thus can be independently set to any number in $[0,1]$ and belong to $\mathrm{Conv}(\{y \in \{0,1\}^n | (Hy)_j = 0\})$. Finally, the fact that doing the convex sum of these elements of $\mathrm{Conv}(\{y \in \{0,1\}^n | (Hy)_j = 0\})$ with the coefficients $(w_{j,S})_{S \in E_j^0}$ will indeed give us x is guaranteed by the equation 2.4.

□

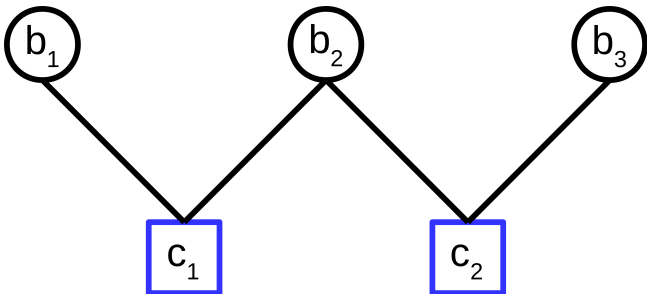


Figure 2.1: We will demonstrate how to exploit the characterization by valid configurations to easily assess that a fractional word is part of the fundamental polytope. Let's take the fractional word (0.7,0.7,0.7). To prove that it belongs to the fundamental polytope, we just have to find for each check a satisfying explanation of what it sees, meaning that its neighborhood can be written as a convex combination of integral neighborhoods satisfying its parity constraint. For the check c_1, we can explain what it sees by 0.7 of the neighborhood, where both variable nodes v_1 and v_2 are set to 1, and 0.3, where both variable nodes are set to 0. We can explain what sees c_2 the same way.

We give in figure 2.1 an example of how to exploit this characterization to easily prove the belonging of a given point to the fundamental polytope. However, it is important to notice that this definition can also be used as is to define the linear program. Indeed, each equality condition can be transformed into two inequality conditions and thus these constraints can be rewritten as a simple system of inequalities.

We saw that this decoder has very interesting theoretical guarantees and comes with powerful tools to study it. We will now see that it even performs very well in practice.

Indeed, it performs as well as the basic `BP` in terms of WER, being even guaranteed to perform at least as well on the binary erasure channel (BEC) [FWK05]. Moreover, it also comes with several variants which allow one to tighten the fundamental polytope, thus improving its performance. Its greatest weakness compared to `BP` is its comparatively low computation speed, as it still requires solving a relatively large linear program. Solving such a program would have a time complexity more than quadratic with the number of variables [CLS21] (equal to the blocklength for the definition by cuts). However, several variants have been proposed to cope with this problem. One of them is called the *adaptive LP decoder* [TSS11] and transforms the big linear program into several smaller programs by activating only a limited number of cuts at the same time. Another even more interesting variant is called the ADMM LP decoder [BLDR13] and relies on the alternating direction method of multipliers (ADMM) [GM75] together with the geometry of the fundamental polytope to implement the solving of the linear program as a message passing algorithm. Contrary to `BP`, this message passing algorithm is guaranteed to converge to the solution of the linear program even in presence of cycles in the Tanner graph. Moreover, it allows adding quadratic dependencies [LD16] to the objective function. We can take advantage of this to improve the score of integral solutions and thus reducing the odds of returning "FAILURE".

Now that we have finished presenting classical error correction, we move to quantum error correction, and will stress out the differences with its classical counterpart.

2.2 Quantum error correction

In this section, we will introduce quantum computing concepts and formalism based on what we saw in the classical case. In §2.2.1 we introduce the representation of quantum information together with its specificities. In §2.2.2 we present the codes used to protect quantum informa-

tion. From their functioning we will derive in §2.2.4 the noise against which we want to protect the information. Finally, we will go over interesting quantum decoding algorithms in §2.2.5 to §2.2.8, highlighting the differences with their classical counterparts.

2.2.1 Quantum information

In this section, we will review the differences between the representation of information in the quantum case compared to the classical case, together with the difference with the way this information can be processed. See [NC02] for more details about quantum information and quantum computation. The smallest brick of quantum information is called a *qubit*, and a *qubit state* can be any normalized vector of $\mathbb{C}^2$. It is implemented physically by a qubit state of a two-level quantum mechanical system. We see that contrary to a classical bit which can take only two values, the set of values that can take a qubit belongs to a sphere. The *Bloch sphere* is a useful representation of a qubit up to its phase, where two orthogonal states are antipodal points. Not being able to describe the global phase is actually not a problem, as two states equal up to their global phase cannot be differentiated.

To describe the state $|\psi\rangle$ of a qubit, it is convenient to adopt an orthogonal basis of $\mathbb{C}^2$. One of the most employed is $(|0\rangle, |1\rangle)$ basis with $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ and $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$. We can thus write the state of any qubit as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$ to have a normalized state. Whenever α and β are both non-zero, we say the state is in *superposition*. It is important to understand that superposition is relative to the basis adopted to describe the quantum state, and that choosing the right basis would lead to the same state not being in superposition. Because of this, a state in superposition is still a pure state and thus is a perfectly determined point of the Bloch sphere, contrary to *mixed* states that we will see later.

The operations we can apply to the qubits can be separated in two categories: *unitary evolutions* and *measurements*.

A unitary evolution is defined by a complex unitary matrix that is applied to the state vector by simple matrix multiplication. It allows us to perform rotations on the Bloch sphere, thus staying on its surface and mapping a pure state to another. The *Pauli operators* are useful examples of such unitary evolutions and are named after the operation they realize on the $(|0\rangle, |1\rangle)$ basis, namely the bit-flip X, the phase-flip Z and their combination Y:

$$X = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}, \quad Z = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}, \quad Y = iXZ = \begin{pmatrix}0 & -i\\i & 0\end{pmatrix}.$$

They form a basis of the real vector space of 2×2 Hermitian matrices.

A measurement is the only way to retrieve information about the quantum state of our qubit, as we cannot directly observe it. It is described by a hermitian operator, that is an operator equal to its transposed conjugate, like for example the X,Y, and Z operators. The orthogonal eigenspaces of this operator give a decomposition of $\mathbb{C}^2$ as their direct sum. Measuring the qubit with this hermitian operator will both project (or *collapse*) the qubit onto one of the eigenspaces, and return the eigenvalue of the corresponding eigenspace as classical information. The eigenspace onto which the state is being projected is probabilistic, the closer the eigenspace is to the original state the more likely it is to be the one it is projected on. This is formalized by *Born's rule*, which probabilistically describes what happens when measuring a qubit in state $|\psi\rangle$ by a hermitian operator A. To do so, we need two orthogonal projectors Π_0 and Π_1 such that $\Pi_0 + \Pi_1 = \mathbb{1}$ and Π_0 (resp. Π_1) projects onto the eigenspace of A labelled by 0 (resp. 1). Born's rule then tells us that:

- with probability $\langle\psi|\Pi_0|\psi\rangle$ the measurement will return 0 and the qubit will collapse onto the state $\frac{\Pi_0|\psi\rangle}{\sqrt{\langle\psi|\Pi_0|\psi\rangle}}$,

- with probability $\langle\psi|\,\Pi_1\,|\psi\rangle$ the measurement will return 1 and the qubit will collapse onto the state $\frac{\Pi_1|\psi\rangle}{\sqrt{\langle\psi|\Pi_1|\psi\rangle}}$.

For example, the hermitian operator Z has two eigenspaces, one with eigenvalue 1 and basis ($|0\rangle$), and one with eigenvalue -1 and basis ($|1\rangle$). Thus, measuring the state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ using Z will with probability $|\alpha|^2$ project the state on $|0\rangle$ and return 1, and with probability $|\alpha|^2$ project the state on $|1\rangle$ and return -1.

Working with several qubits only changes the size of the space, as an N qubits word is an element of $\mathbb{C}^{2^N}$ which is equal to the tensor product of each qubit. It is noticeable that the space dimension grows much faster than in the classical case, and one would expect an N qubits word to belong to $(\mathbb{C}^2)^N$. Such a difference comes from the possibility in quantum computation to have qubits in superposition, which are still pure states and represented by a point on the Bloch (hyper-)sphere. We can still manipulate the qubits through quantum gates and measurements, which must satisfy the same conditions as before. To efficiently implement a quantum gate, it is possible to approximate it to arbitrary precision by combining gates of a universal quantum gates set. Such a universal set can contain as little as 3 gates as proved in chapter 4.5 of [NC02], each one acting on at most 2 qubits and thus avoiding having too many qubits interact at once, which is often difficult. An interesting set of quantum gates which can also be utilized as quantum measurements is the set of Pauli operators on N qubits:

$$\mathcal{P}_N = \{\alpha P | \alpha \in \{1, -1, i, -i\}, P \in \{\mathbb{1}, X, Y, Z\}^{\otimes N}\}.$$

This set is interesting because it forms a basis of all possible matrices. Since two states are considered the same when they differ only by a global phase, different combinations of gates can actually perform the same action on the qubits.

Much like in the classical case, the system interacts with the environment, which can apply a random error to the qubits. However, contrary to what happens in the classical case, where this error is limited to what can be performed by the classical gates, here it extends further than the quantum gates we can apply to the qubits. The reason comes from the fact that, whereas we are limited to unitary evolutions on the system, it is not the case for the errors. Indeed, the qubits on which the error applies are not restrained to the system only, but can also act on qubits (i.e., 2 levels quantum systems) in the environment. Thus, even though the evolution of these qubits is necessarily unitary, it will not always be once we restrain it to the system. Such mixed states are probabilistic mixtures of pure states, which cannot be represented as a simple tensor product nor with a vector state, but require a more complex object called density matrix. Whereas we can only modify the density matrix by applying a unitary operator, the environment can apply more general operators called *completely positive trace-preserving (CPTP) maps*.

The environment can thus transform a pure state into a mixed state, which corresponds to moving a point from the Bloch sphere to its interior for single qubits. Such an operation cannot be reversed by quantum gates, as they only allow rotations on the Bloch sphere. Because of this, it is not clear that error correction is possible in the quantum case, as even if we added a lot of redundancy to the quantum word, we would at some point like to correct the error and thus move the state back to the Bloch sphere. Fortunately, we saw another way of interacting with the qubits, and we will see in the next section how quantum error correction was made possible by measurements.

2.2.2 Quantum error correcting codes

To perform error correction, we face several difficulties in the quantum case. The first one comes from the fact that the number of possible errors is infinite. Thus, it does not seem possible to

characterize it using a few bits, as it was done in the classical case with the syndrome. The second problem is that we saw that the error will most likely not be a unitary evolution, and thus cannot be undone by a quantum gate on the qubits. Fortunately, these two problems have one common solution, which is a measurement of the qubits. Indeed, in doing so, we will project the qubits onto a pure state among a finite number of possibilities. However, doing this naively for example measuring every qubit with a Z operator, the state would collapse on a state of the basis $(|0\rangle, |1\rangle)^{\otimes n}$ and thus lose any superposition. To prevent this, the measures have to be chosen cleverly, which can be done using a *stabilizer code* [Got97].

A stabilizer code on N qubits is defined by a finite set of M commuting Pauli operators $\mathcal{G} \subset \mathcal{P}_N$ called the *generators,* whose spanned space is called the set of *stabilizers.* The stabilizer code itself is then defined as the set of pure states left unchanged by the application of these generators.

Definition 12 (Stabilizer code)**.** *Let $\mathcal{G} \subset \mathcal{P}_N$ be a set of M commuting Pauli operators, it defines the stabilizer code over N qubits $\mathcal{Q}$:*

$$\mathcal{Q} = \{|\psi\rangle \in \mathbb{C}^{2^N} | \forall g \in \mathcal{G}, g|\psi\rangle = |\psi\rangle\}.$$

To avoid ending up with an empty set, we add the extra condition that $-\mathbb{1}$ cannot be obtained from a combination of generators. Requiring that the generators commute ensures the possibility to measure the qubits simultaneously on each one, which is paramount to carry out the error correction. Since each gate X,Y and Z commutes with itself and $\mathbb{1}$, but anticommutes with the others, for two generators to commute there must be an even number of qubits, where they both act non-trivially with different gates. This, however, is difficult to guarantee as it leaves $\frac{N^2-N}{2}$ different pairs of generators to check. To simplify this, a particular kind of stabilizer codes called *CSS stabilizer codes* [CS96, Ste96] exists. They require the extra condition that each generator g can either act non-trivially on qubits only with X gates or act non-trivially on qubits only with Z gates: $g \in \{I, Z\}^{\otimes N} \cup \{I, X\}^{\otimes N}$. In doing so, each generator of the same type (X or Z) is guaranteed to commute with all generators of the same type. Moreover, it allows the code to be entirely described by two classical codes.

As for classical error correcting codes, the quantum codes properties can be summarized by parameters $[\![N, K, D]\!]$. N is the blocklength of the code, and correspond to the number of physical qubits. K is the number of logical qubits, and if the M generators are independent (one cannot be obtained as a combination of the others) we have $K = N - M$. Finally, D is the *minimal distance* of the code.

Definition 13. *The minimal distance D of a stabilizer code having generators $\mathcal{G}$ and stabilizers $\mathcal{S}$ is defined as the smallest weight a Pauli error must have to commute with every stabilizer while not being one:*

$$D = \min_{e \in \mathcal{P}_N \setminus \mathcal{S} | \forall g \in \mathcal{G}, ge = eg} (|e|).$$

It is defined as the minimum weight a Pauli operator can have to commute with every generator while not being a stabilizer. Indeed, errors which are stabilizers are called degenerate errors and do not actually change the state of the qubits. On the contrary, an error that does not commute with every generator will be detected. The minimal distance is thus defined like in the classical case as the weight of the smallest logical error, where its weight is defined as the number of qubits on which it does not act trivially. In the case of CSS codes, the minimal distance is equal to the minimum between the weight of the smallest only-X logical error and the weight of the smallest only-Z logical error.

In the quantum case, errors can now be *equivalent* in the sense that they perform the same action on the code.

Definition 14 (Equivalent errors)**.** *Two errors are said to be equivalent if they are equal up to a stabilizer composition:*

$$e \sim e' \iff \exists S \in \mathcal{S} \text{ such that } e = Se'.$$

Similarly, we will write $e \nsim e'$ when e and e' are not equivalent. Because of this, some errors can have a large weight, but still be equivalent to an error with a small weight. We thus define the *minimal weight* of an error as the smallest weight over all the equivalent errors.

Definition 15 (Minimal weight)**.** *The minimal weight of an error e is defined by:*

$$|e|_{min} = \min_{e' \sim e}(|e'|).$$

We call *degenerate errors* every error whose minimal weight is equal to zero. Such error is actually a stabilizer and thus has a trivial effect on the code space.

Now that we have the code, the error correction procedure is as follows: measure the generators which will project the error onto a Pauli error and return for each one whether it commutes or anticommutes with the obtained Pauli error, then find a Pauli correction by running a decoder on the output of the measurements, and finally apply the correction to the qubits. As for the classical case, a powerful tool for the decoding is the *syndrome* of the error.

Definition 16 (Syndrome in the quantum case)**.** *The syndrome of a Pauli error e for a stabilizer code having generators $(g_j)_{1 \leq j \leq M}$ is defined as $s \in 0, 1^m$ with for all $j \in [m]$:*

$$s_j = 1 \iff eg_j = -g_j e.$$

This now looks much more like classical error correction, and we can in fact even adopt the same formalism. Indeed, a CSS stabilizer code $\mathcal{Q}$ of blocklength N can be defined by two classical codes of blocklength N, $\mathcal{C}_X$ with parity-check matrix H_X and $\mathcal{C}_Z$ with parity-check matrix H_Z. The parity check matrix H_X (resp. H_Z) describe the X-type (resp. Z-type) generators of the stabilizer code as the element at the i-th row and j-th column of H_X (resp. H_Z) means than the i-th X-type (resp. Z-type) generator acts non-trivially on the j-th qubit. The commutativity condition translates into $H_X H_Z^T = 0$. We can similarly write any Pauli error up to global phase $E = E_1 \otimes E_2 \otimes \cdots E_N$ as two elements e_X and e_Z of $\{0,1\}^N$ with $(e_X)_i = 1$ iff $E_i \in \{X, Y\}$ and $(e_Z)_i = 1$ iff $E_i \in \{Y, Z\}$. We can now compute the X and Z syndromes $s_X = H_X e_Z$ and $s_Z = H_Z e_X$, which are defined such that the error on which the error collapses commutes with the i-th X-type (resp. Z-type) generator iff $s_X = 0$ (resp. $s_Z = 0$) and anticommutes otherwise. Contrary to the classical case, if an error of non-zero weight has a zero syndrome it is not necessarily a logical error, but it can also be a degenerate error which has no effect on the qubits.

One last major difference with classical error correction is the fact that since the syndrome is computed by quantum measurement, it is subject to errors and can thus be corrupted like a classical word would be. This thus raises whether useful quantum computation can be realized in this difficult setting. The answer is actually yes and is given by the *threshold theorem* [ABO97]. This theorem states that if the probability of a *location* (the spots, where an error can occur) to be in error is lower than some constant called the *threshold*, then for any quantum circuit C and any target probability ϵ, a fault-tolerant quantum circuit that simulates C and fails with probability at most ϵ exists. This does indeed show that quantum computing can be implemented in realistic settings, but not in a very practical way, as the threshold theorem gives a polylog overhead. This means that if C' takes time t', uses m' qubits, and has v' to simulate C which takes time t, uses m' qubits and has v locations we have:

$$\frac{v'}{v} = \text{polylog}(\frac{v}{\epsilon}), \qquad \frac{t'}{t} = \text{polylog}(\frac{v}{\epsilon}), \qquad \frac{m'}{m} = \text{polylog}(\frac{v}{\epsilon}).$$

Because of this, applying this scheme to perform any useful quantum algorithm would lead to large numbers of physical qubits. Fortunately, it was later shown that under certain conditions fault-tolerant quantum computation could be achieved with constant overheads, this result is known as the *Gottesman's theorem* [Got14a].

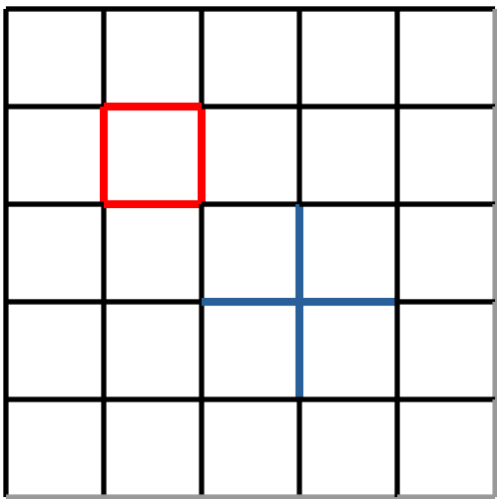
In this thesis, we will focus on the study of CSS codes, and more precisely on the code that protects against X-type errors. Because of this, we will use the same formalism as for classical error correction, and will thus refer to Z-type generators as "checks". We reserve the use of "generators" and "stabilizers" to the X-type generators and $X-type$ stabilizers. Similarly, we will always use the parity check matrix and the Tanner graph corresponding to the part of the CSS code protecting against X-type errors. We will now present the two families of CSS stabilizer codes we will employ in this thesis.

2.2.3 The toric code and the hypergraph product code

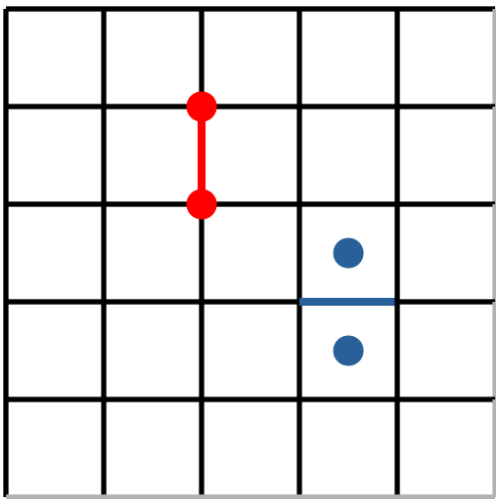
The first CSS code we will present is the toric code [Kit03] which is the quantum counterpart of the classical repetition code, as it encodes only two qubits, but has a minimal distance growing linearly with the square root of the blocklength. The toric code is defined on a two-dimensional square lattice wrapped on a torus. The qubits sit on the edges of the lattice, while the X-type (resp. Z-type) generators sit on the plaquettes (resp. on the vertices) and each generator acts non-trivially only on the 4 neighboring qubits. This construction being highly graphical, it is easier to understand in figure 2.2, where the torus is unfolded into a square, where we identify the left boundary of the square with its right boundary, and the top boundary with the bottom boundary.

The second family of CSS stabilizer codes we will study is a generalization of the toric code called *hypergraph product code* [TZ13]. The name comes from the fact that the Tanner graphs $\mathcal{T}(H_X)$ and $\mathcal{T}(H_Z)$ of the quantum code are defined from the product of the Tanner graphs of two classical codes. Let $\mathcal{T}(\mathcal{C}_1) = (V_1 \cup C_1, E_1)$ be the Tanner graph of the first classical code and $\mathcal{T}(\mathcal{C}_2) = (V_2 \cup C_2, E_2)$ be the Tanner graph of the second code. Then we define the product of $\mathcal{T}(\mathcal{C}_1)$ and $\mathcal{T}(\mathcal{C}_2)$ as the graph $\mathcal{G} = (V_1V_2 \cup C_1C_2 \cup V_1C_2 \cup C_1V_2, E)$, where $V_1V_2 \cup C_1C_2$ is the set of variable nodes corresponding to either VV-type qubits in V_1V_2 or to CC-type qubits in C_1C_2, V_1C_2 (resp. C_1V_2) is the set of check nodes corresponding to X-type generators (resp. Z-type generators). Two nodes will then be linked by an edge if they are the product of the same node and two nodes linked by an edge. The Tanner graph $\mathcal{T}(H_Z)$ (resp. $\mathcal{T}(H_X)$) can then be obtained by removing from the product graph the vertices of V_1C_2 (resp. V_1C_2) and all their incident edges. As for the toric code, the HPC construction is better understood looking at a graphical representation shown in figure 2.3. Such codes are interesting because they allow to design CSS LDPC codes with almost as good parameters as the ones of the classical codes they are based on. For example, if the hypergraph product code is obtained with twice the same classical code with a constant rate and a linear minimal distance, then the quantum code also has a constant rate and a minimal distance linear this time in the square root of the blocklength. Moreover, it has a dedicated decoder we present in §2.2.6 which can, together with a few conditions of the classical codes on which are based the hypergraph product code, satisfy the conditions of Gottesman's theorem and thus allow to perform fault-tolerant quantum computation with a constant overhead [FGL18].

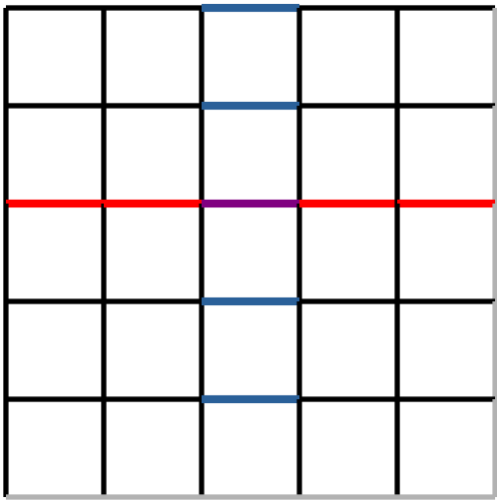
Measuring the stabilizers will project the error onto a Pauli error, it is thus enough to focus only on this type of error. In the next section, we will see common Pauli error channels that can be used to estimate the performance of a stabilizer code.



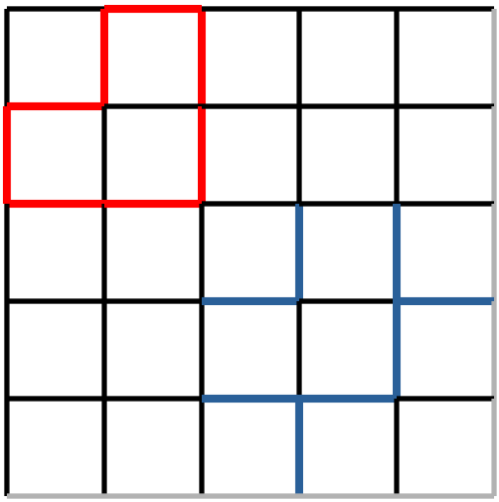
(a) Example of a Z-type generator in red and of an X-type generator in blue. The X-type generators sits on vertices, the qubits on which they act non-trivially are their neighborhood in the shape of a cross. The Z-type generators sit on plaquettes, the qubits on which they act non-trivially are their neighborhood in the shape of a square.



(b) When applying a Z-type error on a single qubit, the two neighboring plaquettes will be unsatisfied, thus informing us that an error was applied and providing us information on how to correct it. The same happens when a single qubit suffers from an X-type error, but this time the neighboring vertices are unsatisfied.



(c) When a string of X-type errors makes a loop around the torus (on the unfolded representation by going from one point of a border to the identified point on the other border, on the example in red from left to right) it commutes with all Z-type generators, but cannot be written as a combination of X-type generators, it is thus a logical error. The same happens with Z-type errors (here in blue from bottom to top), but the string of error does not appear as natural.

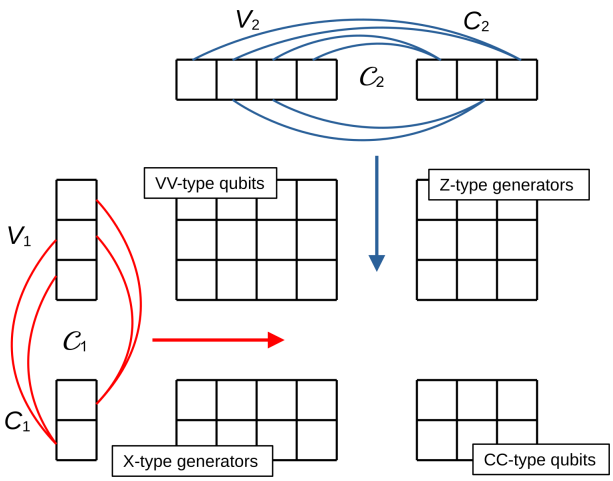


(d) Here, we show in red (resp. in blue) a sum of X-type (resp. Z-type) generators. Both are thus degenerated errors. Once again, the Z-type error's structure is not as easily understandable as the X-type error's structure. For this reason, we usually present the decoding of X-type errors on the toric code.

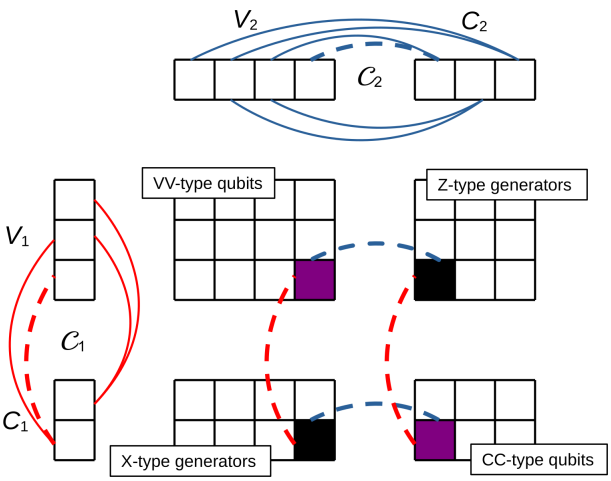
Figure 2.2: Examples of generators and different kinds of errors on a 5×5 toric code. Z-type errors are represented in blue, and X-type errors are represented in red.

2.2.4 Quantum error channels

We saw that the error will always be projected onto a Pauli error. Because of this, the quantum error channel is greatly simplified since only 3 different errors can be applied to a single qubit.



(a) Here, the nodes of the Tanner and product graphs are represented by squares. We first build the nodes of the product graph as products of one node of each graph. We then link the obtained nodes, using the edges from $\mathcal{C}_1$ and from $\mathcal{C}_2$.



(b) The X-type and Z-type generators are guaranteed to commute, as if one X-type generator and one Z-type generator act non-trivially on the same node of V_1V_2 (resp. C_1C_2) we can find a corresponding node in C_1C_2 (resp. V_1V_2), where they both act non-trivially. Here, the red edges on the product graph correspond to the same edge of $\mathcal{C}_1$, and the blue edges correspond to the same edge of $\mathcal{C}_2$

Figure 2.3: Design of an HPC from two classical codes.

We will look at the possible equivalent of the BSC, the equivalents of the BEC being the same with the extra knowledge that an error happened. As we said, the only possible errors are X, Y, and Z, but we still need to choose how likely each one is to happen. The true physical channels are very complicated in general, with various *correlations* between errors. To be able to estimate the performance of codes and decoders, we need *simplified* error channels. The most used error channel is called the depolarizing error channel which applies an error with probability p, and in case of an error picks one of the three gates at random, each one with probability $\frac{1}{3}$. This may seem like the most natural approach, as there is no reason a priori for an error to be more likely than another. However, since with CSS codes, we see the error as a combination of Z and X errors, the fact that Y errors are obtained by applying simultaneously an X and Z error will add correlations between these error types. The decoder can then take advantage of these correlations to improve its WER. To avoid this, we can adopt instead the independent $X-Z$ error channel, which first applies X errors with probability p, and then applies independently Z errors with the same probability p. In doing so, there is no correlation to take advantage of, and thus allows us to derive a lower bound on what could be achieved on a depolarizing error channel. One last possibility is to study the correction of X errors alone with the only-X error channel. Even though this last channel is highly unrealistic, it adopted in simulations to estimate the performance of a symmetric CSS code (a CSS code, where swapping X and Z generators result in the same code after reordering the qubits) which is the case of toric codes and hypergraph product codes built from one classical code. In this case, it is possible to first deal with only one type of error, the other type can then be corrected by the same method.

Like in the classical case, we will then apply this one qubit error channel to every qubit in an i.i.d way. It should be noted that studying highly correlated noise is however much more interesting in the quantum case compared to the classical case. Indeed, errors called coherent errors can occur in many physical implementations and generate a highly correlated noise by

typically applying a phase flip to many qubits [Ouy21].

Still, this does not cover the whole range of possible errors, as we must also define the error channel applied to the syndrome. Since the syndrome is made of classical bits, the model usually adopted is the i.i.d BSC channel with the same probability p. It can still be interesting to look at the performance of the decoders without syndrome noise, in which case it will be specified, the usual case being the fault-tolerant or noisy syndrome case.

We will now look at what is expected from a decoder in the quantum case, together with several examples of such decoders.

2.2.5 Quantum decoding algorithms

In the quantum case, decoders will face two important differences with the classical case. The first one is that since every generator leaves the codeword unchanged, two errors will be equivalent in the sense that applying one or the other will properly correct the code as long as we can go from one to another by applying generators. Because of this, the ML decoding does not translate anymore to the minimum weight decoding, as we must find the most likely class of errors and not the most likely error.

Definition 17 (Quantum ML correction against the depolarizing error channel)**.** *A Pauli operator c is the ML correction of an error with syndrome s on the stabilizer codes with parity-check matrix H subject to a depolarizing noise with error rate $3p$ if and only if:*

$$c \sim \underset{e \in \mathcal{P}_N, He=s}{\arg\min} \left(\sum_{\tilde{e} \in \mathcal{P}_N, \tilde{e} \sim e} p^{|\tilde{e}|} (1-p)^{N-|\tilde{e}|} \right)$$

The second problem arises from the fact that we cannot observe the corrupted word, every decoder must thus work only with the syndrome. However, we saw that any error-based decoder can be easily turned into a syndrome-based decoder as we work with linear codes. Finally, to be optimal in the depolarizing error channel model, the decoder must consider the correlations between X and Z errors. We will rather focus on decoding the uncorrelated error channels, but will mention whenever a decoder can be adapted to consider the correlation. Since the codes we study are CSS stabilizer codes which are symmetric in the sense that you obtain the same code up to a reordering of the qubits when you swap X and Z stabilizers, it is enough to check its performance against only-X error channel to assess its performance against the independent X-Z error channel (and also a lower bound on the performance on the depolarizing error channel). Because of this, we will look only at the Tanner graph and the parity check matrix corresponding to the Z-type generators and thus correcting against X-type errors. We can thus apply the classical decoders to this quantum decoding problem.

To assess their performance, we will follow the same approach as in the classical case, with the difference that the decoding will be successful even if the correction is not exactly equal to the error, but belongs to the same class of equivalence. However, the classical decoders will often need to be adapted as they will have poor performance, mostly because of the degeneration of errors in the quantum case. A typical example of how more complicated a decoder can be in the quantum case is the minimum weight perfect matching (MWPM) decoder [Edm65] for the toric codes [Kit03]. By its properties, the toric code can be considered as a quantum adaptation of the classical repetition code. While the classical decoder simply performs a majority vote, the MWPM decoder performs a two by two matching of the unsatisfied checks, which minimizes the total distance between every pair. The reason is that the smallest error that can explain two unsatisfied checks is an error string, whose support is precisely a chain of qubits that links the two checks. In the case, where more than two checks are unsatisfied, we must thus match each of them in a pair so that the sum of the error strings weights is minimal. The output correction

is then equal to one error string between each pair of checks, which is thus a minimum weight decoding.

We will now present some existing decoders for Hypergraph product codes, which are adaptations of the classical decoders we saw in the classical section. In §2.2.6 we present the SSF decoder, an adaptation of the bit-flip decoder. In §2.2.7 we present the `BP + OSD` decoder which is a combination of two classical decoders. Finally, in §2.2.8 we will see an adaptation of the LP decoder to the quantum setting.

2.2.6 SSF algorithm

This first decoder is an adaptation of the `Flip` decoder called the small-set-flip (`SSF`) decoder [LTZ15]. Because of degeneracy, the `Flip` decoder can be stuck on small constant weight errors. Indeed, given a hypergraph product code built from one classical code with even variable nodes and check nodes degrees, the `Flip` decoder will not be able to find any bit that can reduce the syndrome weight whenever the error is equal to some half generators [LTZ15]. Even if this works only in a particular case, it hints that there might be some other constant weight uncorrectable errors. In order for this decoder to work, it was then necessary to adapt it. The solution found was to not only look at one bit at the time, but at all the bits of a generator at the same time, and then finding the combination of bits to flip to reduce the most in the syndrome. In doing so, it is clear that the former problematic errors can be corrected. Moreover, it is possible to prove interesting theorems in the case, where the classical codes chosen for the product were expander codes. This allows for example to prove that the couple of HPCs with `SSF` satisfies all the conditions required for the Gottesman's theorem, namely a constant rate LDPC code combined with a decoder whose time complexity is linear with the blocklength [Got14b], and thus permits to perform fault-tolerant computations with constant overhead [FGL18]. The decoder is described in algorithm 7. Notice that it is not the equivalent of the version of `Flip` we presented earlier, but rather of a version of `Flip`, where we would browse repeatedly through all the bits and flip the one allowing to decrease the syndrome's weight the most.

Algorithm 7: `SSF`

Input: A syndrome $s_X \in \mathbb{F}_2^{nm}$ and the X-type parity-check matrix H
Output: Deduced error $\widehat{E}$

$\widehat{E} = 0^{n^2+m^2}$ `// Iteratively maintain` $\widehat{E}$
$s = s_X$ `// Iteratively maintain syndrome`
$F =$ set of errors whose support is covered by a generator
while $\exists f \in F : |s| - |s \oplus Hf| > 0$ **do**

$$\hat{f} = \arg\max_{f \in F} \frac{|s| - |s \oplus Hf|}{|f|} \quad \widehat{E} = \widehat{E} \oplus \hat{f}$$
$$s = s \oplus H\hat{f}$$

end
return $\widehat{E}$.

The proven threshold for such a decoder requires massive codes and is actually extremely low. Fortunately, it is possible to witness through simulations a larger threshold on codes with realistic sizes, as it can be seen on figure 2.4. However, this threshold ($p_t h = 4\%$) is still very low compared to what can be achieved on the toric code decoded by minimum weight perfect matching ($p_{th} = 10\%$), and has relatively poor performance compared to what can be achieved

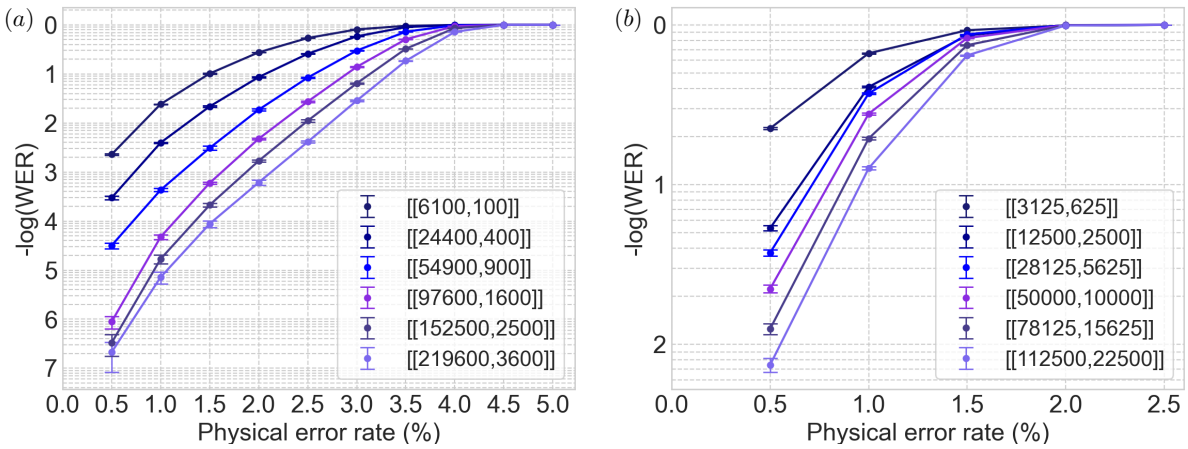


Figure 2.4: Variation of word error rate (WER) with the physical error for hypergraph product codes formed as product of regular (Δ_V, Δ_C)-regular graphs with the SSF decoder from [GK18]. The errors bars indicate 99% confidence intervals, *i.e.*, approximately 2.6 standard deviations. (a) Codes obtained as the product of $(5, 6)$-regular graphs (encoding rate of $1/61 \approx 0.016$): we observe a threshold of roughly 4.5%. (b) Codes obtained as the product of $(5, 10)$-regular graphs (encoding rate of 0.2): we observe a threshold of roughly 2%.

in the classical case. It was also proven that SSF could be run in parallel, and we will investigate this in chapter 5.

2.2.7 BP OSD decoder

It was already well understood that BP alone could not perform well on quantum LDPC codes [PC08]. The reason was also well understood and lied both in the degeneracy and in the presence of many small cycles in the Tanner graph. Such cycles naturally emerge from the necessity for generators to commute, and make the BP work in the opposite scenario it was designed for, the one without cycles. On the other hand, degeneracy generates many situations, where two valid corrections seem the most likely as they achieve the same minimal weight and are, in fact, the same up to a generator and would thus both work. However, BP does not consider degeneracy and cannot choose between the two errors. To try overcoming this problem, several approaches were tried to break the symmetry of the code, like slightly perturbing the a priori probabilities of the qubits. However, that does not address the problem of the large number of small cycles, which quickly corrupt the messages sent by each node.

It is in this context that BP-OSD was first proposed in the quantum setting, and achieved surprisingly good results [PK21a]. It was originally proposed with a version of BP considering the correlations in the depolarizing error model, allowing better decoding performance at the cost of more complicated messages, and in particular messages that are not scalar anymore. We will in this thesis run it with a simpler version of BP, which is precisely the one presented in the classical case. The reason of such good performance may partly lie in the fact that the OSD post-processing excels at bringing the word back to the code, and that it is what could not achieve BP. Indeed, when BP fails to converge in the quantum case due to small cycles and equivalent errors, it does not return a logical error, but an invalid correction. Thus, OSD can hope to successfully decoder afterward and thus improve the WER.

We proposed at the same time a similar approach by combining BP and SSF, that we will review in chapter 3.

2.2.8 LP decoder

An adaptation of the LP decoder was introduced and analyzed by Li and Vontobel in [LV18]. In that paper, they define a binary quantum LP described in 18 which is a simple adaptation of the classical LP decoder, allowing it to only require the syndrome rather than the corrupted word. To do so, they take advantage of the linearity of the stabilizer codes to compute the correction for any word with the right syndrome and then apply this correction to the actual word. This decoder is well suited to correct against the only-X error channel and the independent X-Z error channel. Indeed, they managed to prove the decoding certificate of theorem 8 while introducing a variety of other measures of the code that could be adopted in place of the fractional distance.

Definition 18 (Error-based quantum LP decoder for only-X errors on a CSS code)**.** *The error-based quantum LP decoder works as follows:*

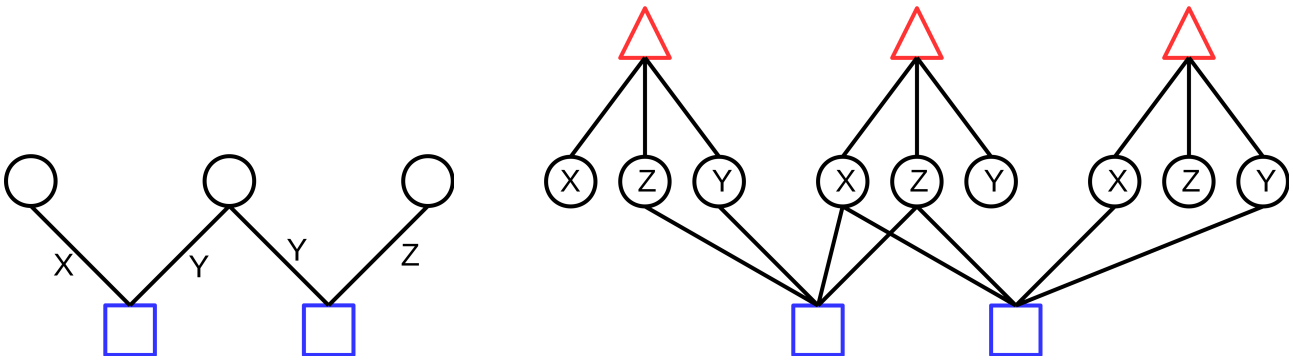
1. *Find an error $\bar{e} \in \{0,1\}^N$ such that $H\bar{e} = s$, which can be done by solving a linear system.*
2. *Compute $e = \arg\min_{x \in \mathcal{P}}(\mathrm{d}(\bar{e}, x))$, which can be done by solving a LP.*
3. *Outputs the correction $\hat{e} = \bar{e} \oplus e$ if it is integral, and "FAILURE" otherwise.*

Moreover, they introduced a non-binary version of the quantum LP decoder benefiting from the same theoretical guarantees while being adapted to the depolarizing error channel and thus able to consider the correlations between X-type and Z-type errors. Their approach [LV18] is similar to the one introduced for the classical case in [FSBG09] and first relies on rewriting the Pauli gates $\mathcal{P}_n$ usually seen as elements of $\mathbb{F}_4^n$ as elements of $\mathbb{F}_2^n \times \mathbb{F}_2^n \times \mathbb{F}_2^n = \mathbb{F}_2^{3n}$. Each Pauli error is thus represented by three elements of $\mathbb{F}_2^n$, each one describing one kind of non-trivial gate. Writing those three elements E_X, E_Y and E_Z we would have that $(E_X)_i = 1$ iff $\mathcal{P}_i = X$, $(E_Y)_i = 1$ iff $\mathcal{P}_i = Y$ and $(E_Z)_i = 1$ iff $\mathcal{P}_i = Z$. Besides, we impose $(E_X)_i + (E_Y)_i + (E_Z)_i \leq 1$ for all i, ensuring that there is only one way to write each Pauli error. Given this definition, minimizing the weight of the Pauli error is equivalent to minimizing the Hamming weight of its representation. We can also adapt the Tanner graph to suit this representation as shown in figure 2.5, from which we will be able to define the non-binary fundamental polytope. If we were to represent an equivalent of a Tanner graph for stabilizer codes, each edge between a generator node and a variable node would have to be labelled by the operation applied by the generator on the gate, as shown in figure 2.5a. And we would of course spare writing every edge labelled by the identity. Using the new representation of Pauli errors, we can replace each variable nodes by three nodes (X,Y,Z), one for each type of error. Thus, each edge labelled by a gate would be replaced by two edges going from the generator node towards the nodes corresponding to the two other errors, as shown in figure 2.5b. However, since we need to guarantee that at most one of the three nodes representing a qubit is set to 1, we have to add particular nodes we will call at-most-one nodes linked to these three nodes. The at-most-one checks are represented by the red triangles in figure 2.5b. This seems like a problem at first as we will not be able to create the fundamental polytope as before since it is not a genuine Tanner due to the at-most-one checks, fortunately it can be easily overcome.

To do so, we define $\tilde{H}$ as the parity check matrix corresponding to the Tanner graph obtained by removing every "at-most-one" nodes and all their incident edges. An element of $w \in \{0,1\}^{3n}$ represents thus a codeword iff $(Hw)_j = 0$ for every $j \in [\![1, m]\!]$ and $w_i + w_{i+n} + w_{i+2n} \leq 1$ for every $i \in [\![1, n]\!]$. Following this, we can define the fundamental polytope in the non-binary case.

Definition 19 (Non-binary fundamental polytope)**.** *Let H be an $m \times 3n$ parity-check matrix corresponding to an non-binary Tanner graph. For each row $j \in [m]$, we define the polytope*

$$\mathcal{P}_j(H) := \mathrm{Conv}(\{x \in \{0,1\}^{3n} | (Hx)_j = 0 \text{ and } \forall i \in [\![1, n]\!], x_i + x_{i+n} + x_{i+2n} \leq 1\}),$$



(a) The Tanner graph of a small stabilizer code, where each square is a check node, each circle a variable node, and each edge is labelled by the generator value for the given variable node (we omitted edges labelled with I). On this representation the variable nodes take values in $\mathbf{F}_4$, so it cannot be processed by a LP which is limited to real numbers.

(b) The factor graph for valid embeddings of the one on the left, obtained by adding triangles representing "at most one" constraints. This time, the variable nodes are binary, so they can be processed by a LP. It is clear on this figure that we could have avoided relying on "at most one" constraints by removing Y variable nodes, and represent it instead as both X and Y nodes set to 1. By doing so, it would also work with non-CSS stabilizer codes, yet we would not be able to consider the correlations in a depolarizing noise model. By removing the triangle nodes and all the edges linked to them, we recover a Tanner graph.

Figure 2.5: Adaptation of the Tanner graph to a quantum code

where $Conv(E)$ is the convex hull of E. The non-binary *fundamental polytope $\mathcal{P}(H)$ of the parity check matrix H is given by:*

$$\mathcal{P}(H) := \bigcap_{j\in[m]} \mathcal{P}_j(H).$$

If we try defining the non-binary fundamental polytope by cuts as it was done in definition 10 we will end up with a slightly different polytope, which will usually be a relaxed version of the non-binary fundamental polytope. Fortunately, we can exactly define the non-binary fundamental polytope by the valid configurations characterization. To do so, we need to define sets similar to even sub-neighborhoods, but which encapsulate the at-most-one constraints rather than the parity constraints.

Definition 20 (Valid non-binary word support)**.** *Let $V = \{v_1, ..., v_{3n}\}$ be the set of all variable nodes of a non-binary Tanner graph. Then we define the set Δ of valid non-binary words support:*

$$\Delta = \{S \subset V | \forall i \in [n], |S \cap \{v_i, v_{i+n}, v_{i+2n}\}| \leq 1\}.$$

We can now define the valid configuration characterization of the points in the non-binary fundamental polytope.

Lemma 21 (Non-binary valid configurations)**.** *Let x be a point of $[0,1]^n$, then x is in the fundamental polytope $\mathcal{P}$ if and only if a configuration $\{w_{j,S}\}_{j\in[\![1,m]\!],S\in E_j^0\cap\Delta}$ exists such that:*

$$0 \leq w_{j,S} \leq 1 \qquad \forall j \in [\![1,m]\!], \forall S \in E_j^0 \cap \Delta \tag{2.5}$$

$$\sum_{S\in E_j^0\cap\Delta} w_{j,S} = 1 \qquad \forall j \in [\![1,m]\!] \tag{2.6}$$

$$\sum_{S\in E_j^0\cap\Delta} w_{j,S} \mathbb{1}_S(v_i) = x_i \qquad \forall j \in [\![1,m]\!], \forall v_i \in \Gamma(c_j) \tag{2.7}$$

Using this characterization, we can thus turn the decoding problem into an explicit linear program. It allows to employ the similar variants as for the binary LP decoder such as the ADMM LP decoder, which can moreover be modified to penalize pseudocodewords. Other variants such as the adaptive LP decoder can also be run in the non-binary case, however since the definition of the non-binary fundamental polytope by cuts is not tight, this can lead to poorer decoding performance.

Li and Vontobel benchmarked this decoder using a MacKay et al.'s bicycle codes [MMM04], and found that the average fractional distance of those codes increased both with the code length and the code's degrees [LV18]. Moreover, they compared the performance of their decoder to the `BP` decoder both in the binary and the non-binary case. They found out that both decoders performed the same in the binary case, but that LP performed slightly better in the non-binary case.

Chapter 3

Combining the belief propagation decoder and the small-set-flip decoder

In this chapter, we present a decoder for hypergraph product codes [TZ13] obtained by combining BP [KFL01] and SSF [LTZ15]. Hypergraph product codes form a large and varied family of codes, among which we find both the toric codes [Kit03] and constant rate random LDPC quantum codes built from random classical codes. While the decoding of the toric code was already well understood, good decoders did not exist for the latter. Fortunately, a decoder inspired by the classical Flip was later designed for a small class of hypergraph product code: the SSF decoder [LTZ15]. While this decoder exhibited powerful properties for some particular codes with considerable blocklengths, it did not perform so well on more general codes with realistic sizes.

What these codes lacked was a decoder that performed well in the general case, even one benefiting from no theoretical guarantees. The perfect candidate to import from the classical case would thus be the BP decoder, which is very efficient and exhibit an excellent WER even outside its theoretical setting of cycle-free Tanner graphs [RU08a]. However, BP performs poorly in the quantum setting due to both degeneracy and the presence of many small cycles [PC08]. Several adaptations of this decoder were proposed to overcome part of these problems [PC08], but did not achieve to make it as efficient as in the classical case.

While a new adaptation finally seems to allow BP to perform on stabilizer codes [DCMS22], the best solution before was to combine it with another decoder employed as a post-process. Indeed, while BP had a poor WER, most of the decoding failures were not logical errors, so could potentially be further corrected. For example, this is what did BP+OSD [PK21a], by post-processing the output of BP by the classical decoder OSD [FL95]. In doing so, it managed to improve the WER of BP by several orders of magnitude on some constant-rate LDPC stabilizer codes. It was further improved and showed to work on a variety of LDPC stabilizer codes, which are similar to BP in the classical setting. We proposed at the same time a similar approach which improves the performance of BP, but running SSF instead of OSD as the post-process decoder. The initial goal was actually to improve the performance of SSF on realistic size hypergraph product codes. We indeed managed to improve the WER compared to SSF alone while running it on codes with a greater encoding rate. As a side effect, we found a new way to adapt BP to the quantum setting, but limited to hypergraph product codes. In addition, we designed a variant able to run with the single-shot property in the fault-tolerant case.

In section 3.1 we first analyze the behavior of BP alone on random hypergraph product codes to motivate the combination with SSF and better understand how to combine them. In section 3.2 we propose several variants of BP + SSF and assess their performance in the noiseless syndrome setting. Finally, in section 3.3 we adapt it to the fault-tolerant setting and show the results of simulations.

3.1 The BP decoder in the quantum case

As explained when presenting BP-OSD for the quantum setting, the performance of BP applied to the decoding of quantum LDPC codes gives feeble results compared to what can be expected from BP when decoding classical LDPC codes [PC08]. Moreover, these poor performances are not limited to the highly quantum setting of decoding correlated X and Z errors at once, but also to the much more classical setting of decoding X-type errors alone. Indeed, in this case, the decoder is given a classical parity check matrix H and a classical syndrome s with the goal of finding a classical error e such that $He = s$. However, two main differences are the degeneracy and the large number of small cycles.

The first one is inseparable from CSS stabilizer codes, and is the fact that two errors are equivalent if they differ only by a sum of generators, and thus if a one is a valid correction the other is one too. Since generators of LDPC codes are chosen to have small weights which do not grow with the blocklength, there will always be several valid corrections relatively close to each others. Since BP cannot consider the degeneracy, those very close corrections will not be considered equivalent, and the decoder will try to choose between the two. This will often lead to the estimated likelihoods oscillating between mixes of likelihoods corresponding to the two valid corrections, and failing to converge to any of them. It is important to understand that it does not simply oscillate between the likelihoods corresponding to each correction, but that at any given time one part of the likelihoods may tend towards the first correction, while the likelihoods in another part of the graph tend toward likelihoods corresponding to the second correction.

The second big difference is the presence of many small cycles in the Tanner graph on which BP operates. Indeed, if we go back to the whole stabilizer code defined in section 2.2, where we define a X-type Tanner graph $\mathcal{T}(H_X) = (V \cup C_X, E_X)$ and a Z-type Tanner graph $\mathcal{T}(H_Z) = (V \cup C_Z, E_Z)$, then the commutativity condition will impose the presence of a cycle of length 4 in the graph $\mathcal{G} = (V \cup C_X \cup C_Z, E_X \cup E_Z)$ whenever an X-type generator and a Z-type generator act on the same qubit. This does not necessarily induce the presence of many small cycles in $\mathcal{T}(H_Z)$, but many constructions [TZ14,Kit03] found to satisfy the commutativity share this problem. For example, the Tanner graph $\mathcal{T}(H_Z)$ of a general hypergraph product code will have many length 8 cycles, and this is particularly noticeable for toric codes. BP is proven to compute the exact likelihoods in the case, where the Tanner graph has no cycles [KFL01], and still manages to compute good approximations if there are not too many small cycles. Indeed, the problem is that a variable node is not supposed to contribute to its likelihood, which is guaranteed by the update rule for cycle less graphs. However, when there are cycles, a message sent by a variable node can come back through another path and thus be incorporated in the computation of its likelihood. Because of this, cycles of length larger than the number of BP rounds will not be a problem, whereas small cycles will corrupt multiple times the messages and thus worsen the likelihoods approximation. Since BP typically runs for tens of rounds, having each variable node part of a length 8 cycle means that every final message will have been corrupted several times.

Because of these two unavoidable particularities, BP will thus be less likely to converge, and when it does, it will converge to worse likelihood approximations than in the classical case. Nevertheless, there is one positive point, which is that correcting errors due to the poorly estimated likelihoods will most likely fail to reach the code space rather than perform a logical error. Because of this, the decoding failures of BP can be detected and fortunately corrected. In the BP-OSD decoder, this task is done by the OSD post-processing, yet here we want SSF to perform this post-processing. This actually changes a few things compared to OSD, which require us to adapt BP. Contrary to OSD, SSF is not designed to consider the exact likelihoods computed by BP, it just needs the corresponding correction (not directly, but it is applied to

update the syndrome before using it as SSF input) and thus exploits only the sign of the log-likelihoods. On the contrary, OSD needs additional information about the log-likelihoods, as it requires their relative values to order the bits. Paradoxically, this can make SSF less resilient to bad log-likelihoods approximations, in particular when it is close to 0. Indeed, when the log-likelihood is close to 0, having it change sign due to a bad approximation will not actually impact OSD, as it will still be one of the first bits added to the set of potentially flipped bits S. Because of this, its value does not actually matter, since OSD will brute force all possible combinations of bit-flips on this subset of the bits. On the contrary, SSF relies a lot on having a rather small error to correct to be successful, and thus having a terrible correction output by BP will be fatal, even if many of the erroneous bits had log-likelihoods very close to 0.

In the case of BP-SSF there are thus two properties we would like the BP output to exhibit. First, the correction should be computed with likelihoods as far from 0 as possible to avoid small approximations errors from flipping some bits of the correction. Secondly, we want the remaining error to be as small as possible, and to assess this we can take advantage of the syndrome weight as a proxy. Looking at these figures of merit, we could decide to stop BP after some condition is met, rather than running it for a predefined number of rounds. We thus studied their evolution as the number of BP rounds increased, and show in figure 3.1 a typical example of what we observed for hypergraph product codes built from a 3,4-regular random code. The figure was realized with the decoding of only one random error by BP which thus ran on the graph $\mathcal{T}(H)$. After each round, we stored the sum of the absolute value of the log-likelihoods, as well as the weight of the syndrome obtained after applying the correction described by these log-likelihoods. We then normalized these values to be able to display them together.

The behavior of the curves is consistent to what we would expect from BP running on a graph with many small cycles. Firstly, the syndrome weight reaches its minimum as the likelihood reaches its maximum, which is the normal behavior we expect when BP converges toward a good approximation of the log-likelihoods. However, in this case they do not converge, but instead reach a peak and oscillate periodically. Such behavior is typically caused by errors having a syndrome of small weight, sometimes called *trapping sets* [Ric03]. This can be interpreted by BP not being able to choose between two equivalent errors, with a part of the graph converging towards one error, and the rest of the graph converging towards the other one. As they converge, the log-likelihoods absolute values increase and the syndrome decreases because each of them cancels the syndrome, yet at some point the two different interpretations clash with each other, messages from one going against the correction of the other and diminishing the absolute values of the log-likelihoods. One good point where to stop BP would thus be one of those peaks, where the syndrome's weight is minimum. We could thus choose to keep the correction obtained at one of these 100 rounds, where the syndrome reached its minimal value. However, since the minima seem almost equal, and since the more rounds of BP we run the more corrupted we expect the messages to be by the small cycles, we decide instead to stop at the first peak. Namely, we will stop BP whenever the syndrome's weight increases compared to the previous round, and return the correction of the previous round. We present this version of BP in algorithm 8 and the combined decoder First-min BP-SSF in algorithm 9.

We will now see in the next section the results of our simulations of decoding only-X noise on hypergraph product codes using First-min BP-SSF.

3.2 The BP SSF decoder in the noiseless syndrome case

In this section, we will see how First-min BP-SSF performs on some hypergraph product codes in the case of perfect syndrome measurement. To evaluate its performance, we will estimate its

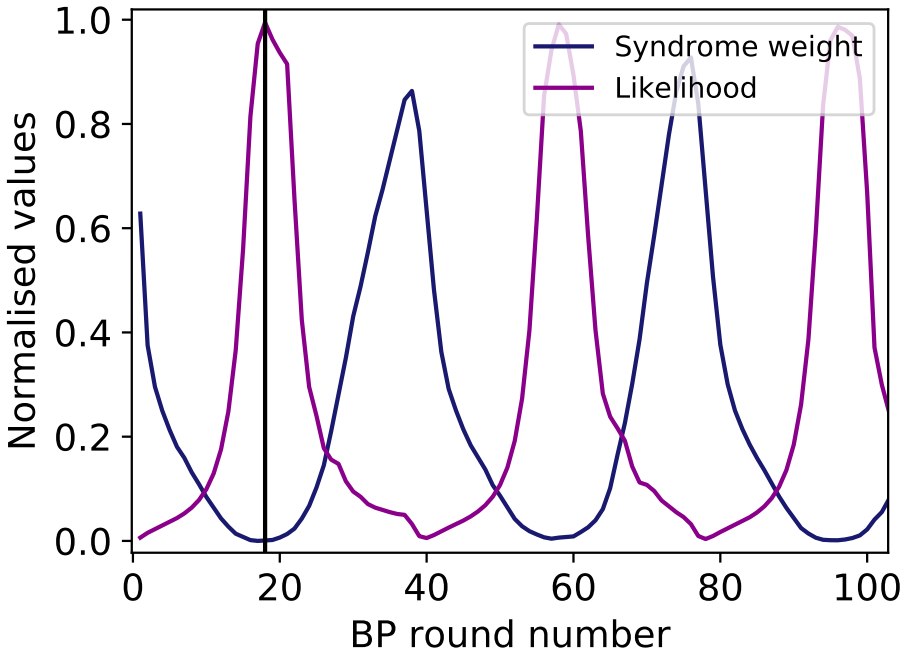


Figure 3.1: Evolution for a fixed error of the sum of the absolute value of the log-likelihoods, as well as the weight of the syndrome after correction, depending on the number of rounds of BP. It indicates that the log-likelihoods and the syndrome are highly correlated, which is expected in the case, where BP converges. However, here it does not converge, but instead start to oscillate, which could be interpreted as BP trying to converge towards two equivalent errors at once.

Algorithm 8: `First-min BP`

Input: Syndrome $\sigma_0 = \sigma_X(E)$, a parity check matrix H and its Tanner graph $\mathcal{T}$
Output: Error E' that minimizes the syndrome

$T = 0$
$\sigma_{\text{current}} = \sigma_0$
$E_{\text{BPcurrent}} = 0^n$
do
  $T = T + 1$
  $E_{\text{BPprev}} = E_{\text{BPcurrent}}$
  $\sigma_{\text{prev}} = \sigma_{\text{current}}$
  $E_{\text{BPcurrent}} = \text{BP}(T, \sigma_0, p)$
  $\sigma_{\text{current}} = \sigma_0 + \sigma_X(E_{\text{BPcurrent}})$
while $|\sigma_{current}| < |\sigma_{prev}|$
 // We repeat these steps while the syndrome decreases
$E_{\text{BP}} = E_{\text{BPprev}}$ // We choose the previous correction.
return E_{BP}

WER through simulations, we will thus start this section by explaining how those simulations are performed, which is summarized in algorithm 10. The same protocol will be followed to assess the performance of other decoders in the noiseless syndrome setting.

The simulations were performed on hypergraph product codes obtained from a single 3,4-regular code of a classical LDPC family. The design of those classical codes is detailed in the

Algorithm 9: `First-min BP + SSF`

Input: Syndrome $\sigma_0 = \sigma_X(E)$, a parity check matrix H and its Tanner graph $\mathcal{T}$
Output: Deduced error $\widehat{E}$

$E_{\texttt{BP}} = \texttt{First-min BP}(\sigma_0)$
$\sigma' = \sigma_0 + \sigma_X(E_{\texttt{BP}})$
$E_{\texttt{SSF}} = \texttt{SSF}(\sigma')$
$\widehat{E} = E_{\texttt{BP}} + E_{\texttt{SSF}}$
return $\widehat{E}$

Algorithm 10: `Noiseless-sampling`

Input: Bias probability p, a parity check matrix H and its Tanner graph $\mathcal{T}$
Output: SUCCESS if the decoding is successful, and FAIL otherwise

Sample error e from $\mu(p)^n$
`// In our case` μ `is the Bernoulli distribution.`
Compute syndrome $s = He$
Sample syndrome noise $\widetilde{\sigma}$ from $\mu(p)^m$
Compute $\hat{e} = \texttt{Dec}(s)$
$e = \hat{e} + e$
return SUCCESS if e is in the stabilizer group and FAIL otherwise

last chapter 7.1. The result of those simulations is shown in figure 3.2.

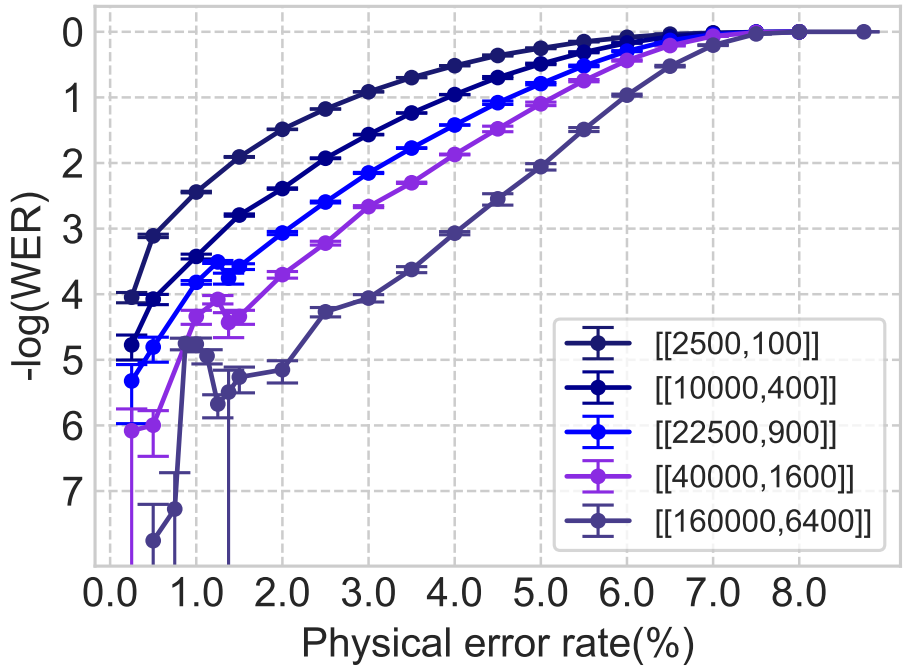


Figure 3.2: Word Error Rate (WER) as a function of the physical error for a hypergraph product code formed as a product of (3,4)-regular graphs with the `First-min BP + SSF` decoder. Both the threshold and the WER are improved compared to SSF alone, despite odd patterns at low error rates.

The WER is greatly improved compared to what could be achieved with `SSF` alone and shown in figure 2.4. The simulations are also compatible with the existence of a threshold

around 7%, which once again is better than what could be achieved with SSF alone. Moreover, the codes employed in this case have better rates and smaller generators than the one optimized for SSF, making them more practical for a physical implementation as well as more scalable. However, we notice strange patterns at low error rates, where the WER suddenly decreases as the physical error rate increases. Such patterns are clearly not wanted, as they are a sign that the decoder is suboptimal. Indeed, if one wanted to run the decoder at a physical error rate of 1% it would be better to add some extra noise before running the decoder to compute the correction. This is likely a sign that some part of the decoder does not behave as intended, which could clearly be the case, as BP is running in a worst-case scenario. However, there is possibly another problem, which is the way we decide when to stop BP. We could imagine that the typical scenario we witnessed does not apply at every physical error rates. Moreover, in figure 3.1 we showed an ideal case, where the curves are monotonous between each peak, but there were several cases, where we witnessed a first local minimum of the syndrome's weight before and higher than the periodic one, for which the stopping condition is not adapted.

To solve this problem, we introduce a new decoder based on observations we made on First-min BP-SSF. Similarly to what happens for BP alone, First-min BP-SSF does not return many logical errors and mostly fails because it was unable to reach the code space. This means that we do not actually have to chose a priori the number of rounds BP will perform, but that we can try several ones and choose the first that returns to the code space. The idea is thus to compute a correction by BP with a fixed number of iterations T, followed by SSF as a post-process, but return the correction only if it has the right syndrome and otherwise repeat the previous steps with a greater T. To ensure the decoder ends, we choose an integer T_{max} which upper bounds T. We chose to study the most time-consuming version of this decoder for a given T_{max}, where T is initialized at 0 and is incremented by only 1 between each correction attempts. We detail this decoder called Iterative BP-SSF in algorithm 11, and will now shortly review its complexity. The run time of this decoder is clearly T_{max} times the run time of BP and SSF with a naive implementation. Fortunately, it can be optimized by reusing the last messages computed for BP on T rounds as the initial messages of BP on $T+1$ rounds, which allows us to compute its correction in only one round of message passing. Because of this, the overall computation time is T_{max} times the one of SSF alone, plus only one time the one of BP. However, this is still much slower than First-min BP-SSF, particularly since SSF can be very slow when it does not manage to correct the error. To reduce the computation time, it is however possible to decrease T_{max} or increment T by more than one at the time. Both modifications could increase the WER as we may miss the right round of BP, whose correction would allow SSF to recover a codeword.

We show on figure 3.3 the result of our simulations for First-min BP-SSF with $T_{max} = 100$ against only-X errors on the same codes as for Iterative BP-SSF. We obtain a similar threshold as for First-min BP-SSF, however the WER is better with a big improvement for the largest codes at low physical error rates. Moreover, we do not witness any more strange artifacts, which was expected as the functioning of this decoder is more consistent and systematic than the previous one which was highly dependent on the behavior of the syndrome's weight.

BP-SSF is however still outperformed by BP-OSD on the very same codes, even running the same version of BP. We thus wanted to try to exploit more precisely the information output by BP by making SSF take into account the approximate likelihoods of the bits. The idea simply consisted of adjusting the cost of a small set with the likelihood of each of its bit, the greater the likelihood the higher the cost. We tried several variations of this idea, but unfortunately none improved noticeably the performance of BP-SSF.

In the next section, we will see how to adapt BP-SSF to the fault-tolerant case, and assess its performance in a more realistic setting.

Algorithm 11: `Iterative BP + SSF`

Input: Syndrome σ_0
maximal no. of `BP` iterations $T_{\max}$
a parity check matrix H and its Tanner graph $\mathcal{T}$

Output: Deduced error $\widehat{E}$ if algorithm converges and FAIL otherwise.

$T = 0$
while $T \leq T_{\max}$ **do**
  $\widehat{E} = \texttt{BP}(T, \sigma_0, p)$
  $\sigma' = \sigma_0 + \sigma_X(\widehat{E})$
  $\widehat{E} = \widehat{E} + \texttt{SSF}(\sigma')$
  $T = T + 1$
  return $\widehat{E}$ if $\sigma_X(\widehat{E}) + \sigma_0$ is zero and keep running otherwise.
end
return FAIL

3.3 The BP SSF decoder in the fault-tolerant setting

In this section, we will adapt the `BP-SSF` decoder presented in the previous section to the fault-tolerant setting and see how well it performs. To achieve this we will have to allow it to work with syndrome errors, with a technique that can be used for other decoders such as the LP decoder.

To make `BP-SSF` able to deal with syndrome errors, we first must adapt both `BP` and `SSF` to this particular setting. Such an adaptation is sometimes necessary as the decoder relies on the fact that the input is an actual syndrome, this is the case for example for the toric code decoder, the minimum weight perfect matching algorithm which needs the syndrome to have an even weight to function properly. However, in the opposite case, some decoders work as well (or at least the same way) in the case of syndrome errors. This is the case of `SSF` which is proven to exhibit a threshold for similar codes as in the noiseless syndrome case. The way it works is that it will actually ignore the potential syndrome error, and assume it is working in the noiseless measurement setting. It will thus try to decrease the noisy syndrome's weight by flipping subsets of generators, stopping when the weight cannot be further reduced. In the noiseless syndrome case, stopping when the syndrome's weight was not equal to zero necessarily meant a decoding failure. However, in this case it is more than unlikely that `SSF` manages to cancel the syndrome, even when it actually managed to correct the whole error. The reason this approach can function is that, contrary to `MWPM` that processes all of the syndrome at once, and thus has a global view of it, `SSF` only takes decisions locally based on partial information and thus cannot notice inconsistencies in the values of far apart checks. We could thus think the same will happen in the case of `BP`, since as a message passing algorithm, all the computations during `BP` are also made locally. However, this is not entirely true as we saw that `BP` could sometime fail to converge because of incompatibility of the messages sent by different areas of the graph, plus it can actually be easily adapted. This adaptation is actually not an adaptation of the decoder directly, but rather an adaptation of the Tanner graph on which it runs, and thus, can be applied to other decoders based on the Tanner graph.

The basic idea comes from the fact that measuring a check with noise is the same as perfectly measuring a check with an extra bit involved in the check. The check being corrupted would then correspond to this extra bit set to 1, and to 0 if the check was actually properly measured.

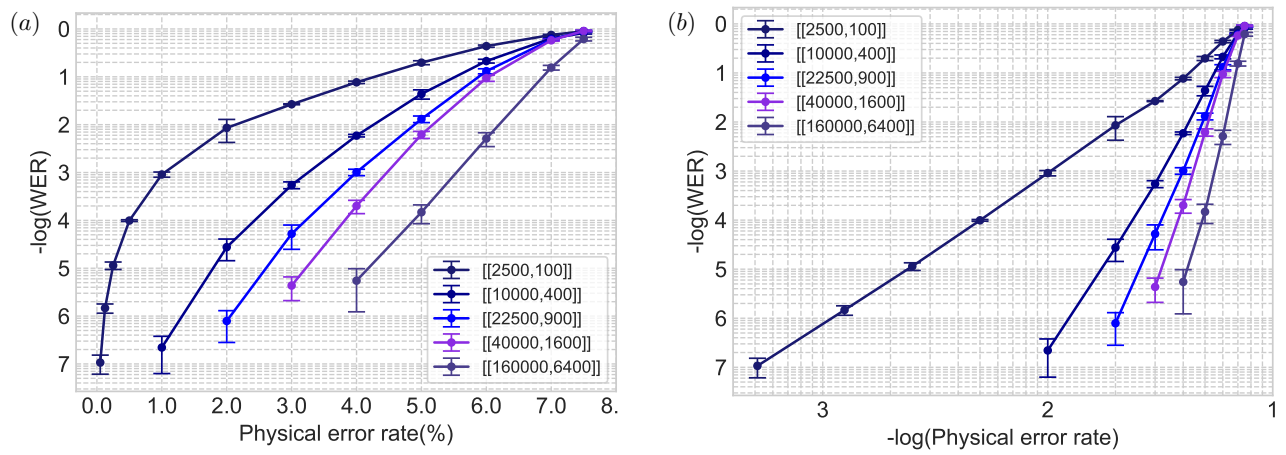


Figure 3.3: Variation of the word error rate (WER) with the physical error for a hypergraph product code formed as a product of $(3, 4)$-regular graphs with the `Iterative BP+SSF` decoder. This code family displays an encoding rate of 0.04. The error bars denote 99% confidence intervals, *i.e.*, ≈ 2.6 standard deviations. (a) Base 10 logarithm of WER versus physical error rate. We see that the threshold is above 7%. (b) Log of WER versus log of physical error rate.

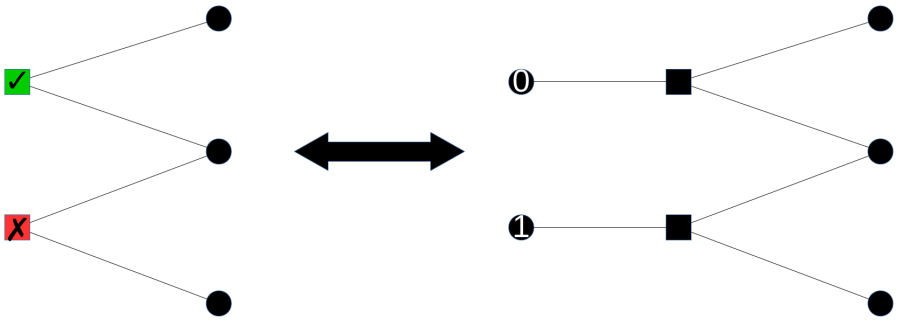


Figure 3.4: Adaptation of a Tanner graph to make it consider the potential syndrome measurement errors. When running `BP` on the graph on the right, we will treat the extra variable node of each check just like the other variable nodes, not knowing their value in the beginning, but able to make a guess in the end thanks to their estimated likelihood.

This leads to a simple adaptation of the Tanner graph, where each check node is linked to one extra variable node, as shown in figure 3.4. Notice that this modification of the graph does not introduce any cycle, so it should not negatively impact `BP`. Moreover, we will thus have an actual estimation of the syndrome error, not only defined as the difference between the initial syndrome and the syndrome of the correction as in the case of `SSF`, we can thus try to correct the syndrome before `SSF` to give it more reliable information.

Since the likelihood of the extra variable nodes is dependent on the highly corrupted messages sent by the regular variable nodes, the syndrome correction will rarely be equal to the actual error (notice that for the syndrome correction there is no degeneration). Because of this, we have to assume `SSF` will work with a faulty syndrome, even in `BP-SSF`, and thus the syndrome will most likely not be cancelled at the end of `SSF` even when the bits are properly corrected. It is thus unpractical to run `Iterative BP-SSF` in the fault-tolerant setting, and we have to go back to the more heuristic `First-min BP-SSF`.

We will now review how the performances of the decoder are assessed in the fault-tolerant setting. We could naturally follow the same protocol as for the noiseless syndrome case showed in algorithm 10, however this would, in fact, lead to a more pessimistic case than what can be expected from an actual quantum circuit. Indeed, during an actual quantum calculus we would

perform several rounds of fault-tolerant correction during the quantum calculus itself, but finally end the calculus by a measurement of the qubits and thus obtaining quantum information. We can then compute exactly the syndrome and apply a final round of error correction on the measured qubits. The protocol will thus consist of first T rounds of fault-tolerant quantum error correction alternating application of an error and computation and application of a correction, and then one last round with an exact measurement of the syndrome and a final error correction. Moreover, we can imagine running a different decoder for the last step, since it is not fault-tolerant. For example, we could choose `First-min BP-SSF` for the fault-tolerant rounds, but run `Iterative BP-SSF` in the final round. This protocol is summarized in algorithm 12 and similar to the approach adopted by Breuckman and Terhal in [BT16]. This easier protocol may still be much too challenging for some decoders, which require measuring several times the same syndrome to gather more information about it. However, this is not required for the decoders we study, as they have the *single shot* property [Bom15, Cam19a]. This allows them to compute a correction from only one noisy syndrome measurement whereas, for example, the MWPM decoder needs $O(\sqrt{n})$ noisy syndromes of the toric code to compute its correction.

Algorithm 12: `Noisy-sampling`

Input: Bias probability p
Number of faulty rounds T
Output: SUCCESS if no logical errors, and FAIL otherwise

$E = 0^n$ `// The code is initialized with no errors`
for $i \in \{1, ..., T\}$ **do**
  Sample error E_i from $\mu(p)^n$
  $E = E + E_i$
  Compute syndrome $\sigma_i = \sigma_X(E)$
  Sample syndrome noise $\widetilde{\sigma}_i$ from $\mu(p)^m$
  Let $\xi_i := \widetilde{\sigma}_i + \sigma_i$
  Compute $\widehat{E}_i = \mathtt{Dec}_1(\xi_i)$
  $E = E + \widehat{E}_i$
end
Sample error E_{T+1} from $\mu(p)^n$
$E = E + E_{T+1}$
Compute syndrome $\sigma_{T+1} = \sigma_X(E)$
Compute $\widehat{E}_{T+1} = \mathtt{Dec}_2(\sigma_{X,T+1})$
$E = \widehat{E}_{T+1} + E$
return SUCCESS if E is in the stabilizer group and FAIL otherwise

The simulations performed with this protocol also depend on T and thus have an extra degree of freedom compared to the noiseless syndrome protocol. We could then measure several thresholds by changing the value of T, and not necessarily obtaining the same value. We actually expect the threshold to diminish as T grows, as this would correspond to computing a longer quantum circuit. To assess the performance of the decoder with only one value, we will thus give the estimated limit of the threshold as T goes to infinity, which is called the *sustainable error rate* [BT16].

We decided to compare the performance of two decoders, the first one running `First-min BP-SSF` for both $\mathtt{Dec}_1$ and $\mathtt{Dec}_2$, and the second one with instead `First-min BP` alone as $\mathtt{Dec}_1$. The first decoder is the most natural one to look at given the far better performance of `BP-SSF`

compared to BP alone in the noiseless syndrome case, though it would also be interesting to look at a version with Iterative BP-SSF as $\texttt{Dec}_2$. Still, the second version is also interesting as it only employs BP in the fault-tolerant rounds, which contrary to SSF actually considers the potential syndrome errors. We give the result of these simulations in figure 3.5, where we display the evolution of the threshold as T grows and fit the curves to estimate their limit, the sustainable error rate.

We see that choosing BP alone as $\texttt{Dec}_1$ allows to have a sustainable error rate of almost 3%, whereas the sustainable error rate is slightly over 2.5% when we run the more complicated First-min BP-SSF. This is quite surprising, but as explained before, only BP considers the syndrome errors, SSF acts as if the measurement was perfect. Because of this, BP can explain part of the unsatisfied checks by syndrome errors and thus avoid performing a too large and erroneous correction on the qubits. Indeed, this is quite beneficial as an erroneous correction of the syndrome will be reset the next round as a new syndrome is measured, whereas an erroneous correction of a bit will accumulate with the errors added during the next round and will necessarily have to be corrected by a decoder. On the contrary, SSF will try to cancel the whole syndrome applying only bits correction, even when a check was not actually unsatisfied, but simply badly measured. Generally SSF will not be able to find a small set able to satisfy a corrupted check, but when it manages to do it, it will be by flipping many bits that were not erroneous. Taking too many such bad decisions can be unrecoverable by the next rounds of error correction, that will either fail to return to the code space or even perform a logical error. However, it is not in contradiction with the theoretical results proving a threshold for SSF alone in the fault-tolerant setting, since the result applies to much bigger codes with many properties that do not meet the codes used in this thesis. Moreover, the relatively large sustainable error rate difference should be put in perspective, as it did not translate into a big difference in terms of WER. Both versions thus have very similar performance on the codes tested.

To our understanding, the worse performance of the decoder running First-min BP-SSF as $\texttt{Dec}_1$ thus lies at least partially in the fact that BP has a more conservative approach in terms of bit correction than SSF as it can transfer part of the correction towards the syndrome correction. It would thus be interesting to adapt SSF to make it also more conservative, though not by allowing it syndrome correction, but rather by strengthening the conditions under which a small set can be flipped. We remind that the version of SSF we are running is flipping every small set that decreases the syndrome, but always starts by flipping the one with the best ratio between the size of the syndrome's weight reduction and the size of the small set. We thus could change this by adding the extra condition that the ratio has to be over some value to be flipped, even if it is the one with the best ratio, like what is done in the simplest version of Flip described in algorithm 1.

3.4 Conclusion of the chapter

In this chapter, we investigated the combination of the soft decoder BP with the hard decoder SSF and obtained better results than each of them in the noiseless syndrome case with two different adaptations First-min BP + SSF and Iterative BP + SSF. Though the second adaptation had a better WER, both exhibited a threshold around 7% which cannot be obtained with BP alone due to small uncorrectable error rising from both the degeneration and the presence of many small cycles in the Tanner graph imposed by the commutativity of the generators. Compared to SSF alone, the decoder achieved both a better WER and a better threshold, while working with codes having a higher rate and smaller generators weights. This is actually very beneficial, since the physical implementation of such codes at a higher rate means that less physical qubits are required to encode as many logical qubits. This reduces the

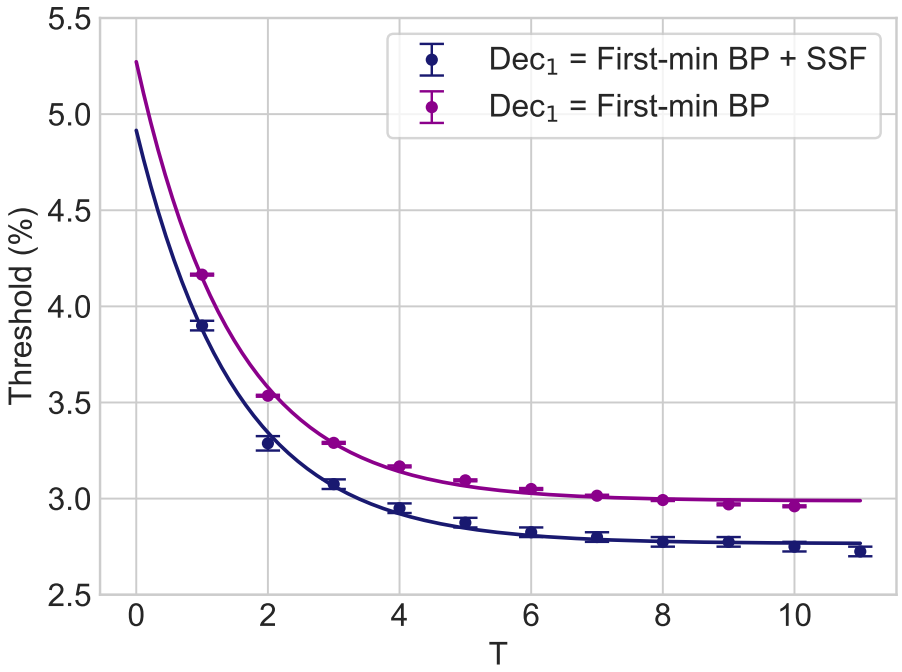


Figure 3.5: Evolution of the threshold as a function of T. Surprisingly, the simpler procedure `First-min BP` performs better than `First-min BP` + `SSF` for $\texttt{Dec}_1$, with a sustainable error rate which seems to be around 3% (compared to 2.5%).

size of the system and thus most likely its physical error rate. Moreover, smaller generators mean that less qubits have to interact together to compute a check, which reduces the syndrome error rate.

We have also investigated the performance of this decoder in the fault-tolerant setting, by trying two variants. The first one employed `First-min BP` + `SSF` for both the fault-tolerant rounds and the last noiseless syndrome round, whereas the second simply run `First-min BP` for the fault-tolerant rounds. Surprisingly, the simpler version turned out to have the better sustainable error rate, reaching almost 3% against around 2.5% for the other one. Even though we do not know the reasons why `BP` is more suited than `First-min BP`+`SSF` for fault-tolerance, whereas it was outperformed in the noiseless syndrome case, we think it is related to the fact that `BP` considers the potential syndrome errors when `SSF` acts as if there were none.

Chapter 4

The linear programming decoder

In this chapter, we will continue the study of the LP decoder in the quantum case initiated by Li and Vontobel in [LV18]. In chapter 3 we studied a combination of the BP and SSF decoders which had improved performance on hypergraph product codes compared to each individually. This heuristic decoder was focused on exhibiting good performance in terms of WER and threshold, at the cost of losing all the theoretical guarantees that made SSF so interesting. And in the end its performance fell short of the performance of BP + OSD, another similar decoder which can perform on a wider range of codes.

To recover some theoretical guarantees while still benefiting from good performance in practice, we got interested in a classical decoder based on linear programming. Indeed, the LP decoder had several interesting theoretical guarantees while exhibiting performance close to the one of BP. It was even proven to be at least as good as BP against the binary erasure channel. Its only drawback was its relatively long computation time, as it required solving a linear program whose size was proportional to the blocklength. Fortunately, several adaptations were introduced to accelerate the decoder without weakening its performance. In particular, an implementation based on the alternating direction method of multipliers (ADMM) closed the gap with BP by implementing the decoder as a message passing algorithm. Finally, the LP decoder was adapted to larger fields, such as $\mathbb{F}_4$ together with most of its different implementations.

All these reasons make the LP decoder an interesting candidate to import to the quantum case, and that is what Li and Vontobel did. In their paper [LV18] they introduced two adaptations of the LP decoder, one for the independent X-Z noise model and one for the depolarizing error channel. They proved that some theoretical guarantees were conserved in the quantum case and performed simulations on LDPC stabilizer codes. Unfortunately, the decoder performed worse than BP on some bicycle codes.

Nevertheless, we hoped that it would perform better on hypergraph product codes, so decided to further analyze it. We introduced another adaptation to the quantum case equivalent to the one of Li and Vontobel [LV18] while not requiring initially to find an error with the given syndrome. We completed the analysis by showing that all the classical guarantees were conserved in a weaker version in the quantum case. We found several error patterns that could harm the decoding performance of the LP decoder by studying the simple case of the toric code, and managed to generalize them to some random hypergraph product codes. In particular, we showed that codes lacking a good soundness property would not perform well asymptotically due to constant weight uncorrectable errors. Moreover, we manage to prove the existence of other constant weight uncorrectable errors on codes with good soundness and asymptotically good. Nevertheless, we performed simulations on several realistic-sized hypergraph product codes and showed that the LP decoder performed relatively well. Indeed, it managed to perform better than BP + SSF on some codes, while benefiting from a message passing implementation thanks to the ADMM variant.

In section 4.1 we introduce a syndrome-based adaptation of the LP decoder which we prove to be equivalent to the error-based adaptation of Li and Vontobel. In section 4.2 we both complete the study of the guarantees imported from the classical case, and introduce problematic error patterns from which we will derive several negative results. Finally, in section 4.6 we present the results of simulations of various implementations of the LP decoder on several stabilizer codes.

4.1 Syndrome-based quantum adaptation

As explained when presenting Li and Vontobel's adaptation of the LP decoder [LV18] in §2.2.8, the algorithm needs to be modified to be able to run with only the syndrome in input and not the corrupted word. To this end, Li and Vontobel took advantage of the linearity of the stabilizer codes to compute the correction for an arbitrary error with the right syndrome. Here, we want to try another approach that does not require computing the initial error to improve the overall complexity. Indeed, computing this error requires multiplying the syndrome by a pre-computed inverse of H. This matrix will generally not be sparse, and thus time taken by the matrix multiplication will be quadratic with the blocklength. While it will most likely be negligible for small codes, it will greatly improve the overall complexity of the decoder in the case, where for example we choose the ADMM implementation [BLDR13] which runs in time linear with the blocklength.

To allow the decoder to skip the computation of the initial error, we will modify the linear program so that it works directly with the syndrome. The fundamental polytope will be modified so that its integral vertices are not the codewords, but the words with the syndrome given in the input. We will thus search for the best correction among the vertices of this new polytope, which can be done by solving a linear program having for objective function the ℓ_1 norm. The choice of this objective function comes from the fact that ℓ_1 norm of a word is equal to its Hamming weight, and that this norm is linear over $[0,1]^n$.

We will call *syndrome polytope* the modified fundamental polytope to stress its dependency with the syndrome. Despite this difference, we can actually adopt very similar definitions to the ones of the fundamental polytope seen in §2.1.8. Moreover, it will be clear for each of these definitions that the fundamental polytope is actually a special case of the syndrome polytope obtained from the all-zero syndrome. The definition relying on the intersection of convex hulls is adapted by choosing convex hulls of points satisfying or not the given check, depending on the corresponding value in the syndrome.

Definition 22 (Syndrome polytope)**.** *Let H be an $m \times n$ parity-check matrix and $s \in \{0,1\}^m$ be a syndrome. For each row $j \in [m]$, we define the polytope*

$$\mathcal{P}_j^s := \mathrm{Conv}(\{x \in \{0,1\}^n | (Hx)_j = s_j\}).$$

The syndrome polytope $\mathcal{P}^s(\mathcal{C})$ *of the code $\mathcal{C} = \ker H$ and syndrome s is given by:*

$$\mathcal{P}^s(\mathcal{C}) := \bigcap_{j \in [m]} \mathcal{P}_j^s.$$

It is clear that this polytope is a generalization of the fundamental polytope, which can be obtained as the syndrome polytope of the all zero syndrome. We adapt accordingly the valid configuration characterization: a point of the syndrome polytope is not a convex combination of neighborhoods satisfying their check, but of neighborhoods satisfying or not their check depending on the corresponding value in the syndrome. When the check is not supposed to be satisfied, we will thus rely on neighborhoods whose supports are odd cardinality sub-neighborhoods instead of even cardinality ones as introduced in definition 9.

Lemma 23 (Valid configurations). *Let $\mathcal{T} = (V \cup C, E)$ be a Tanner graph and $s \in \{0,1\}^m$ be a syndrome. Then $x \in \mathcal{P}^s$ iff a configuration $\{w_{j,S}\}_{j\in[m],S\subset E_j^{s_j}}$ exists such that:*

$$0 \leq w_{j,S} \leq 1, \qquad \forall j \in [m], \forall S \subset E_j^{s_j} \tag{4.1}$$

$$\sum_{S\subset E_j^{s_j}} w_{j,S} = 1, \qquad \forall j \in [m], \tag{4.2}$$

$$\sum_{S\subset E_j^{s_j}} w_{j,S}\mathbb{1}_S(v_i) = x_i, \qquad \forall j \in [m], \forall v_i \in \Gamma(c_j) \tag{4.3}$$

Using this definition, we can describe the syndrome-based LP decoder, but this can also be done by employing its characterization by cuts. Contrary to the cuts definition of the fundamental polytope, here each satisfied check will forbid its odd cardinality sub-neighborhoods and each unsatisfied check will forbid its even cardinality sub-neighborhoods.

Lemma 24 (Syndrome polytope with cuts). *Let x be a point in $[0,1]^n$, then x is in the syndrome polytope $\mathcal{P}^s$ of H iff:*

$$\forall j \in [m], \forall S \in E_j^{s_j} : \sum_{v_i\in S} x_i + \sum_{v_i\in\Gamma(c_j)\setminus S} (1 - x_i) \leq |\Gamma(c_j)| - 1.$$

Proof. Same as the proof of lemma 10 in §2.2.8. □

We can then define the syndrome-based linear programming decoder using this characterization of the syndrome polytope.

Definition 25 (Explicit definition of the `QLPD`). *Given the syndrome s, the syndrome-based quantum LP decoder returns the output of the following LP:*

$$\begin{aligned} \textit{Minimise} & \sum_{i=1}^{n} x_i \\ s.t. \quad & 0 \leq x_i \leq 1, \quad \forall i \in [n]; \\ & \sum_{v_i\in S} x_i + \sum_{v_i\in\Gamma(c_j)\setminus S} (1 - x_i) \leq |\Gamma(c_j)| - 1, \\ & \qquad \forall j \in [m],\ \forall S \in E_j^{1-s_j}. \end{aligned}$$

This LP decoder is different from the one defined by Li and Vontobel however they always output the same correction, even in the case, where we round the fractional outputs of the linear program instead of returning "FAILURE" as it will be proven in section 7.2. For this reason, we will refer to both as `QLPD`, and to their rounded versions as `RQLPD`. Deciding which one to run will thus depend solely on which is the fastest, which will depend on the exact implementation chosen. The only advantage of Li and Vontobel's version is that the polytope never changes, so some implementation of the LP decoder could theoretically take advantage of this to do computations in advance and speed up the decoder. However, none of the implementations seen in this thesis could take advantage of this, and in particular not the ADMM implementation [BLDR13]. Nevertheless, both adaptations will be utilized in the next section, as some results are much easier to prove with one or the other.

4.2 Theoretical study

In this section, we continue the theoretical study of the LP decoder in the quantum case started by Li and Vontobel in [LV18]. In particular, §4.2.1 is focused on completing the import of the classical guarantees to the quantum case, most of them in a weaker version. In §4.2.2 we study a simple example of pseudocodeword and uncorrectable errors on the toric code, that we will generalize to other codes in §4.5.1 to §4.5. In §4.5.1 we prove the existence of constant weight uncorrectable errors for families of codes lacking a good soundness property. Finally, in §4.3 we introduce problematic error patterns that will be exploited in §4.4 to further weaken a theoretical guarantee and in §4.5 to prove the existence of constant weight uncorrectable errors even for some code families having a good soundness.

4.2.1 How the classical guarantees translate in the quantum case

In this section, we will look at how the classical guarantees of the LP decoder [FWK05] translate in the quantum case, and if necessary adapt them to make them useful.

We first adapt the maximum likelihood certificate of the classical LP decoder, which assesses that any integral correction output is the most likely integral correction possible.

Theorem 26 (Minimum weight correction certificate)**.** *If the LP decoder outputs a correction $\hat{e} \in \{0,1\}^n$, then $\hat{e}$ is the smallest correction with the right syndrome:*

$$\hat{e} = \underset{x \in \{0,1\}^n, Hx^T = s}{\arg\min} (|x|).$$

Proof. Exceptionally, this result is easier to prove with the syndrome-based LP decoder. Indeed, this result simply follows from the fact that the objective function is the weight of the word and that all the words with syndrome s are vertices of $\mathcal{P}^s$ and its only integral vertices. □

As we saw in §2.2.5 the minimum weight decoding does not translate in the quantum case into a ML decoding, thus a weaker guarantee. Despite being not as strong as in the classical case, this property is still interesting and not satisfied by decoders for general LDPC stabilizer codes. Notice that the rounded version of the LP decoders do not have this guarantee. Actually, they do not even have the guarantee of smallest weight correction in the case, where the correction has the right syndrome. This is obvious in the case of the rounded version of the syndrome-based LP decoder, since the weight of the rounded solution can be much greater than the weight of the fractional solution if many coordinates are just above 0.5. Because of this, the weight of the actual minimum correction can sit between the weight of the fractional LP solution and the weight of the rounded solution. Moreover, it is useless to try truncating the correction (set to zero every fractional entry) instead of rounding it, since it will be guaranteed to be outside the polytope and thus will not have the right syndrome.

Li and Vontobel showed in [LV18] that the classical success certificate of the LP decoder also applies in the quantum setting. However, this is only interesting if the fractional distance of the code corresponding to H_X (or H_Z) is large enough (e.g., growing with n). This is not the case for quantum LDPC codes, where the fractional distances of H_X and H_Z are upper bounded by the smallest generator weight. For this reason, we now introduce a quantum version of the fractional distance, which will also provide a sufficient condition to guarantee the success of the `QLPD`.

Definition 27 (Quantum minimum fractional distance)**.** *Given a code with fundamental polytope $\mathcal{P}$, the quantum minimum fractional distance* $\mathrm{d}_{\mathrm{qfrac}}$ *is defined as*

$$\mathrm{d}_{\mathrm{qfrac}} = \min_{x \in \mathcal{V}, x \nsim 0} (|x|).$$

Remark 1. *This is an adaptation of the classical minimum fractional distance. Since this new definition takes degeneracy into account, it is unclear whether it can be computed efficiently by solving linear programs similar to the ones given in [FWK05].*

We now adapt to the quantum setting, the classical success certificate proven in [FWK05].

Theorem 28 (Success certificate)**.** *If the error is equivalent to an error of weight lower than* $\lceil \frac{\mathrm{d}_{\mathrm{qfrac}}}{2} \rceil - 1$*, then* `QLPD` *will properly correct it.*

Proof. We want to show that if the error which occurred e with syndrome s is equivalent to an error $\tilde{e}$ of weight lower than $\lceil \frac{\mathrm{d}_{\mathrm{qfrac}}}{2} \rceil - 1$, then the correction $\hat{e}$ output by the LP decoder is equivalent to e for any error $\bar{e}$ chosen at the beginning of Li and Vontobel's decoder. Since the performance of the LP decoder is independent of the error $\bar{e}$ we can choose it to be $\tilde{e}$, thus we have to prove that $\bar{x} = \arg\min_{x \in \mathcal{P}}(\mathrm{d}(\tilde{e}, x))$ is equivalent to 0. We will show that there is a contradiction, if not the case. Let $\bar{x}$ be either fractional or not equivalent to 0. We know it is an element of $\mathcal{V}$, and that it is the closest to $\tilde{e}$. However, since 0 is also an element of $\mathcal{V}$ and $\mathrm{d}(\tilde{e}, 0)$ is lower or equal to $\lceil \frac{\mathrm{d}_{\mathrm{qfrac}}}{2} \rceil - 1$, then $\mathrm{d}(\tilde{e}, \bar{x})$ has to be lower than $\lceil \frac{\mathrm{d}_{\mathrm{qfrac}}}{2} \rceil - 1$. Thus, by triangular inequality we know that $\mathrm{d}(\bar{x}, 0)$ is lower than $\mathrm{d}_{\mathrm{qfrac}}$, which is in contradiction with the definition of $\mathrm{d}_{\mathrm{qfrac}}$, since $\bar{x}$ is in $\mathcal{V}$ and either fractional or not equivalent to 0. □

This theorem is the same as in the classical case, but with the quantum fractional distance. It thus can be useful again since the quantum fractional distance is not upper bounded by the weight of the generators, at the cost of not being computable by the same efficient method. However, despite not being trivially upper bounded, we will see in section 4.4 that it is still independent of the blocklength in some particular cases, even allowing for some constant weight errors to be uncorrectable. Such problematic errors are easy to exhibit in the case of the toric code, and they can even be generalized to other stabilizer codes.

4.2.2 Limitations on the toric code

It is clear that the toric code is not the kind of code on which we want to run the LP decoder, since it already has excellent dedicated decoders such as `MWPM`. However, its very graphical representation allows us to easily design fractional words and check whether they are in the fundamental polytope. We can then better understand the problematic errors the decoder has to face, and try to generalize them to other LDPC codes.

It is indeed straightforward to prove that a fractional word on the toric code is in the fundamental polytope by employing its valid configurations characterization as demonstrated before in figure 2.1. We adopt the same method in figure 4.1 to create a fractional word that belongs to the fundamental polytope. This particular example is first interesting as it clearly is independent of the blocklength. Because of that, if it were to be a pseudocodeword it would prevent the quantum fractional distance of the toric code to grow with the blocklength, and thus make theorem 28 nearly useless for this code. We have not seen yet characterizations of pseudocodewords, that is fractional vertices of the fundamental polytope, but it is treated in §4.4.

The example of figure 4.1 is actually interesting even in the case, where the fractional word is within the fundamental polytope without being a pseudocodeword. Indeed, it is clear that the X-type error whose support corresponds to the solid red edges is minimal and of weight 5, meaning that the smallest valid correction is also of weight 5. However, the distance between the fractional word and the error is only 4 (8 variable nodes differ by 0.5), thus smaller than the distance to any codeword equivalent to the all-zero codeword. This is, in fact, enough to guarantee that this error is uncorrectable by the LP decoder.

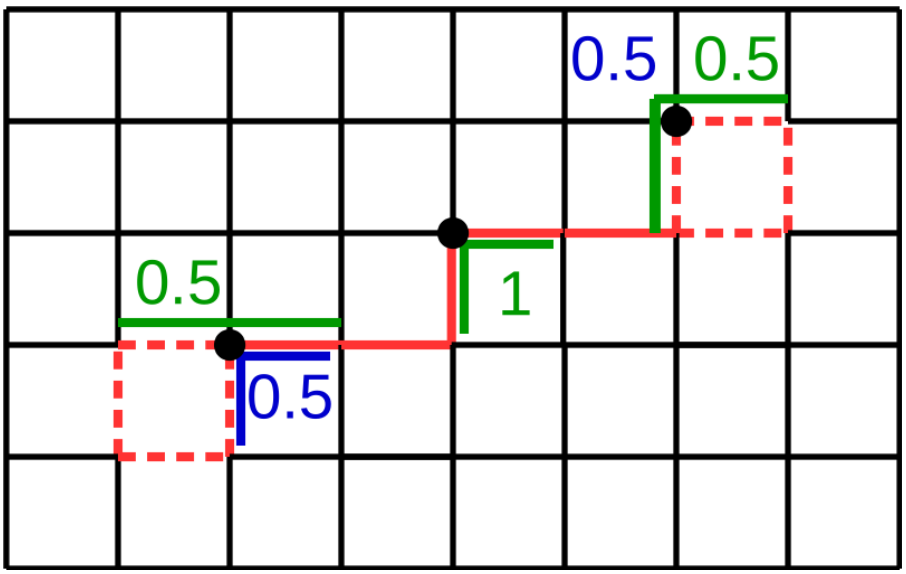


Figure 4.1: We look at a portion of the toric code, where every variable node in solid red is set to 1, every variable node in hashed red is set to 0.5, and all the others are set to 0. We satisfy the checks at each end of the error string by setting a neighboring generator to 0.5. Since it is a fractional word, we cannot compute its syndrome to check whether it is part of the code. We thus use the valid configuration characterization of the fundamental polytope to ensure it is a fractional codeword. We check that this fractional word is indeed part of the fundamental polytope by giving for each check an explanation of the neighboring bits as a convex combination of even-cardinality sub-neighborhoods. We draw in blue and green these sub-neighborhoods for checks with the black dots, with the weight given to them in the same color.

To characterize uncorrectable errors, we will study the LP decoder of Li and Vontobel, and to make it as simple as possible we will assume that the original codeword was the all-zero codeword and that the noisy word initially chosen by the decoder is the actual noisy word (this can be done as none of these choices impact the decoder's output). It is then clear that the decoder is successful if and only if the vertex output by the linear program is the all-zero codeword or any equivalent codeword (that is, a degenerate error). However, because of the objective function chosen, the vertex output will always be the one the closest to the noisy word. We already exploited this observation to prove the success certificate in theorem 28, but it can also be utilized to prove a negative result.

Lemma 29 (Failure certificate)**.** *Let $\mathcal{P}$ be the fundamental polytope of $\mathcal{C}$ and e an error such that a point x of $\mathcal{P}$ exists with ℓ_1 distance to e is smaller than the ℓ_1 distance from e to any codeword equivalent to the all-zero codeword, then the LP decoder cannot correct the error e on the code $\mathcal{C}$.*

Proof. Follows from the fact that knowing that x is in $\mathcal{P}$ guarantees that the linear program cannot output any vertex whose distance to e is greater than $d(e, x)$. □

Notice that the fractional point close to the error does not need to be a pseudocodeword, but simply to belong to the fundamental polytope. Applying lemma 29 to the error and fractional word of figure 4.1 we see that we actually have a weight-5 uncorrectable error. To generalize this kind of errors to other stabilizer codes, we can first study codes with poor soundness, which is the subject of §4.5.1.

However, there is another possible approach based on the fractional word obtained by taking the smallest error string as possible. Such fractional words can actually even be problematic

for codes with good soundness. We will study such fractional words together with associated errors in the appendix. In section 4.3 we first explain how the fractional words are created for CSS codes, and in particular for hypergraph product codes. In §4.4 we will then deduce from these fractional points a constant upper bound on the quantum fractional distance of some a priori good stabilizer codes. In §4.5 we will rather focus on the associated errors and show that they can be constant weight uncorrectable errors for some (again a priori good) stabilizer codes.

4.3 Introduction of a problematic pattern: bolases

We will now propose another generalization of the fractional word seen in figure 4.1 so that it can be a problematic pattern on codes having a good soundness. To formally guarantee that these fractional words have the properties required to be problematic, we will have to restrain ourselves to hypergraph product codes built from a Tanner graph having a large girth. However, this constraint is usually linked to better codes in the classical case, in particular since BP performs better on graphs with a large girth. For this reason, we doubt that the large girth conditions weaken the code.

We saw in §4.5.1 a possible generalization of the example of uncorrectable error seen in §4.2.2. We will now try another generalization of the fractional word of figure 4.1, where instead of starting from a rather long error string, the fractional word is initially a weight one error. As we did before, we will then return it to the fundamental polytope by setting a generator to 0.5 in the neighborhood of each unsatisfied check. To make it more general, we can in fact even rely on one larger stabilizer linked to every unsatisfied check at once. We will call this particular kind of fractional word a *bolas*.

Definition 30 (Bolas)**.** *A fractional word x is a bolas if only one variable node is equal to* 1*, a few variable nodes are equal to* 0.5*, and all the others are equal to 0. The variable nodes equal to* 0.5 *must form the support of a stabilizer linked to every neighbor of the variable node equal to* 1*. We call* central bit *the variable node set to 1, and* central checks *its neighbors. We call* peripheral bits *the variable nodes set to* 0.5*, and* peripheral checks *their neighbors that are not central checks:*

$$\begin{aligned} x \in \mathcal{B} \Longleftrightarrow & x \in \{0, 1/2, 1\}^n \text{ with} \\ & \{v_i | x_i = 1\} = \{v_{i_0}\} \text{ and} \\ & \{v_i | x_i = 1/2\} = S \text{ with } S \text{ the support of a stabilizer and } \Gamma(v_{i_0}) \subset \Gamma(S). \end{aligned}$$

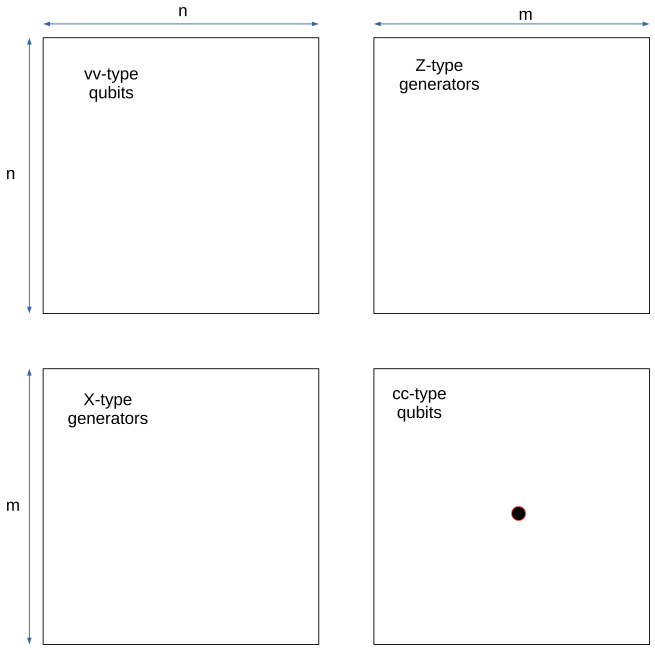
We give in figure 4.2 an example of bolas on a hypergraph product code, and how to create it step by step. Bolases must be understood as analysis tools having properties that will be exploited extensively in the rest of the chapter. We will often focus on bolases of hypergraph product codes, for this reason, it is good to have in mind their construction explained in figure 2.3.

These fractional words are of particular interest as they are guaranteed to belong to the fundamental polytope while having a relatively small weight as only one bit is set to 1.

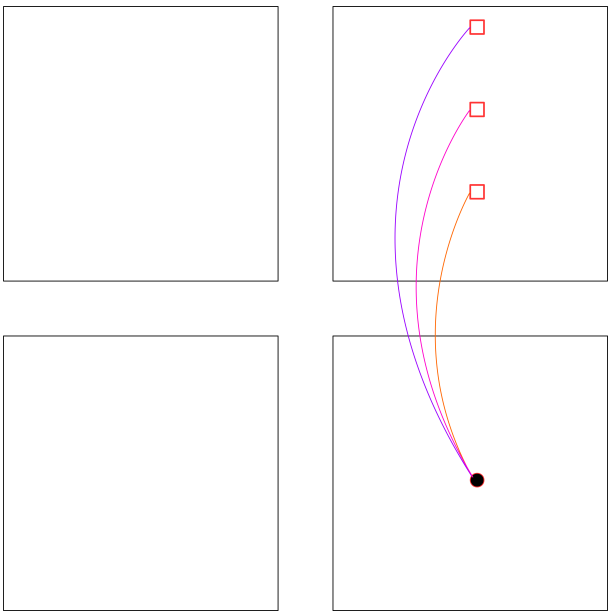
Lemma 31. *A bolas always belongs to the fundamental polytope.*

Proof. Similarly to what we did in figure 4.1 we will give valid configurations for every check node. By construction each check node has either in its neighborhood only nodes set to 0, or an even number of nodes set to 0.5 and the rest set to 0 (in this case it is a peripheral check), or 1 node set 1, an even number of nodes set to 0.5 and the rest set to 0 (in this case it is a

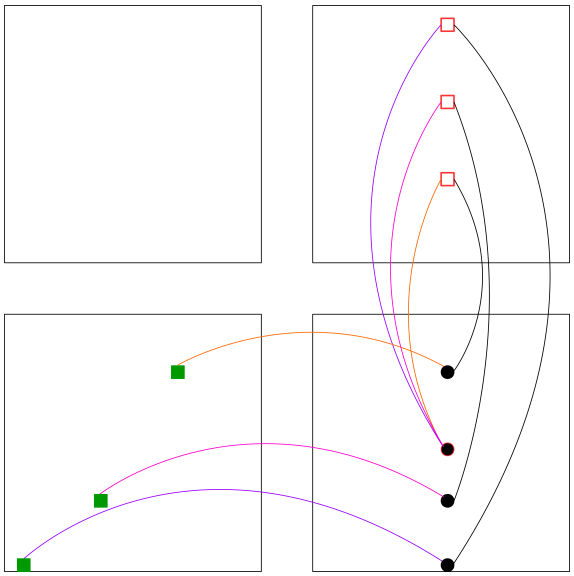
central). A possible valid configuration for a central check is half the weight on the central bit and one neighboring peripheral bit set to 1, and half the weight on the central bit and the rest of the neighboring peripheral bits set to 1. For a peripheral check, half the weight can be on all the neighboring peripheral bits set to 1, and half on no variable nodes set to 1. Finally, for the other checks, all the weight can be on all variable nodes set to 0. □



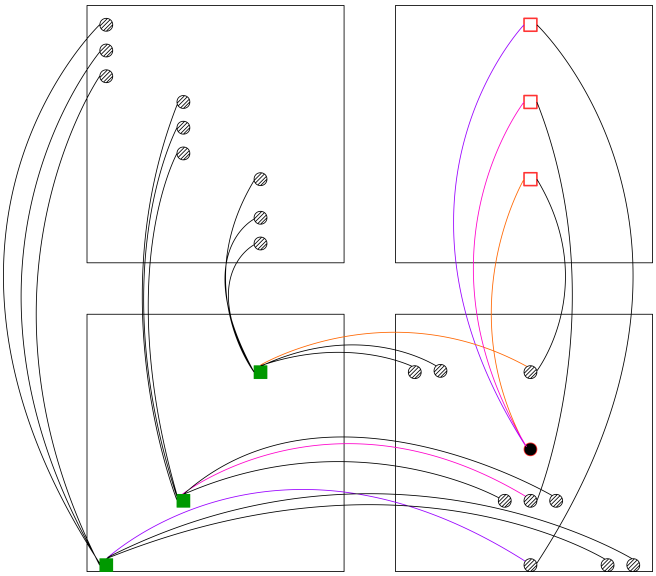
(a) We first chose the central bit (here of type cc, but we could have chosen it of type vv).



(b) Because we flipped the central bit, d_c checks are now unsatisfied.



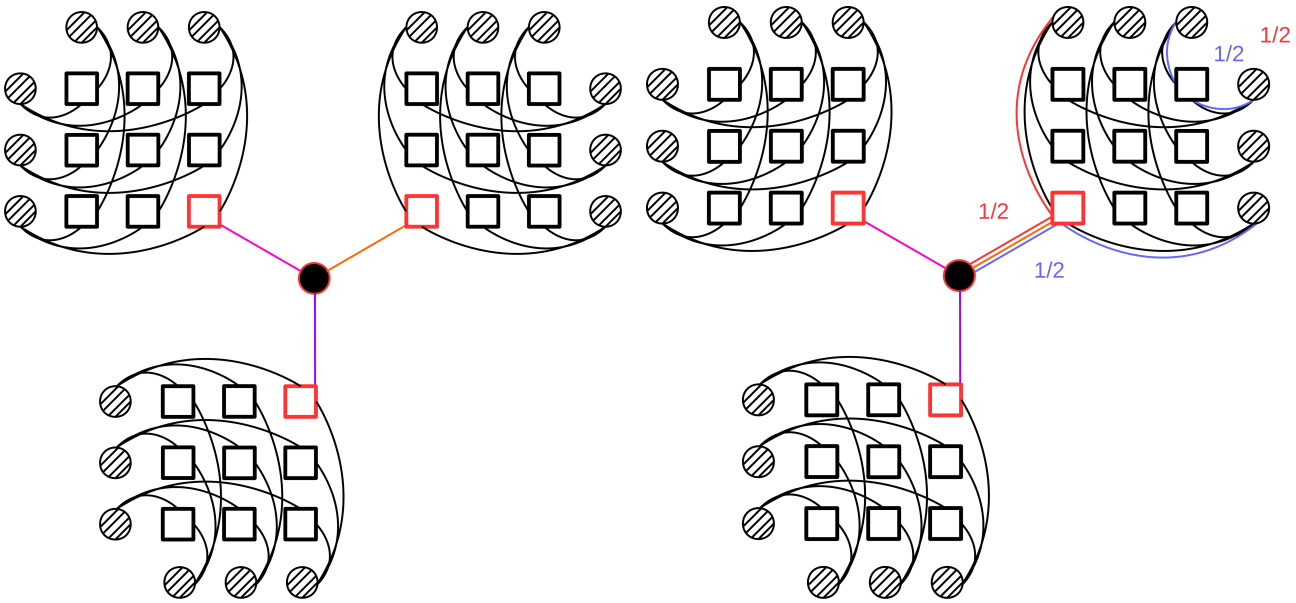
(c) To satisfy these checks, we have to choose for each one a neighboring generator. The colored edges with the same color correspond to the same edge on the classical Tanner graph.



(d) By giving to each bit of these generators the value 0.5 we go back to the fundamental polytope, as we will show in practice by giving a valid configuration.

Figure 4.2: Design of bolases

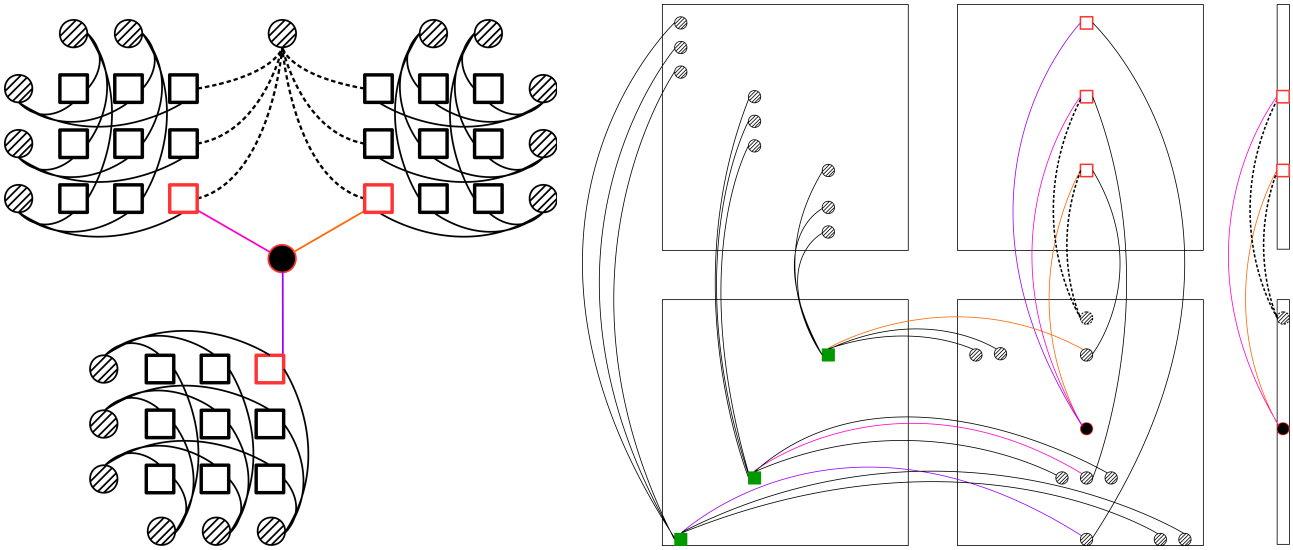
We give an example of such a bolas in figure 4.2 for a hypergraph product code obtained from a classical code having check degree equal to bit degree equal to 3. We also present a simpler representation of this fractional word displaying only the variable nodes not set to 0,



(a) In this representation, bits and checks part of the same generator are vertically or horizontally aligned are aligned the same way in the precedent representation. Moreover, each check linked to the central bit are vertically aligned with it (they would be aligned horizontally if we had chosen the central bit of type vv.

(b) There are, in fact, only two kinds of checks: the central checks and the peripheral ones. Giving the weights of the valid configuration for one example of each will thus be enough. We only wrote down the positive weights, and for each weight, the edges of the same color shows the even cardinal set of bits set to 1.

Figure 4.3: Simpler representation



(a) Case, where a peripheral bit belongs to several generators. Because of the way we chose the colored edges, it has to be vertically aligned with the central bit.

(b) In this case, we see on this representation that we can, in fact, extract a cycle of size 4 from the classical code.

Figure 4.4: Case, where two peripheral bits overlap

together with the valid configurations on figure 4.3. Notice that the central variable node and the generators have been carefully chosen. Here, we will describe how we designed it for a more general hypergraph code obtained as the product of a general classical code having $\mathcal{T}_0$ as a Tanner graph.

Definition 32 (Standard bolas for hypergraph product codes)**.** *Let $\mathcal{Q}$ be a hypergraph product code obtained by the product of a Tanner graph $\mathcal{T}_0$ with itself, a standard bolas is thus defined by:*

- *a central bit $(c_0 \times c_0)$ obtained as the product of twice the same check node of $\mathcal{T}_0$,*
- *the set of central checks is then equal to $\{(v_{i_1} \times c_{i_0}), ..., (v_{i_\delta} \times c_{i_0})\}$ with δ the degree of c_0,*
- *for each central check $(v_{i_j} \times c_{i_0})$ we chose a first peripheral bit $(c_{i'_j} \times c_{i_0})$ with $c_{i'_j} \in \Gamma(v_{i_j}) \setminus \{c_0\}$,*
- *the set of peripheral bits of central check $(v_{i_j} \times c_{i_0})$ is then equal to $\Gamma((c_{i'_j} \times v_{i_j})) = (\Gamma(c_{i'_j}) \times v_{i_j}) \cup (c_{i'_j} \times \Gamma(v_{i_j}))$*

In this definition, we do not ensure that the peripheral bits of two different central checks do not overlap. Because of this, the set of peripheral checks may not be the support of a stabilizer. However, we will see in the next lemma that whenever the classical Tanner graph has a girth large enough, any standard bolas is a bolas according to definition 30.

Lemma 33 (Standard bolases are actual bolases)**.** *Any standard bolas built on a hypergraph product code obtained from a classical Tanner graph with girth greater than 4 will be a bolas according to definition 30.*

Proof. For a standard bolas not to be an actual bolas, the peripheral bits obtained from two different central checks must overlap. We take the same notations as in definition 32 and assume the peripheral bits of $(v_{i_j} \times c_{i_0})$ and $(v_{i_k} \times c_{i_0})$ overlap. The two sets of peripheral bits are $\Gamma((c_{i'_j} \times v_{i_j})) = (\Gamma(c_{i'_j}) \times v_{i_j}) \cup (c_{i'_j} \times \Gamma(v_{i_j}))$ and $\Gamma((c_{i'_k} \times v_{i_k})) = (\Gamma(c_{i'_k}) \times v_{i_k}) \cup (c_{i'_k} \times \Gamma(v_{i_k}))$, so for them to overlap we need either $v_{i_j} = v_{i_k}$ or $c_{i'_j} = c_{i'_k}$. The first equality is impossible by definition of a standard error, we thus have $c_{i'_j} = c_{i'_k}$. We can then exhibit a length 4 cycle on $\mathcal{T}_0$: $(c_{i'_k}$— v_{i_k}— c_{i_0}— v_{i_j}— $c_{i'_j} = c_{i'_k})$. Such a cycle cannot exist if the classical Tanner graph has a girth larger than 4, thus the result. □

It is important to understand that not only such a small cycle would be easy to avoid when designing $\mathcal{T}_0$, but that it also has no reason to improve the performance or the decoding of the classical code or the hypergraph product of this code. Moreover, we saw that such small cycle can be problematic for `BP` and that the performance of both decoders are linked, thus requiring a large girth should even improve the decoding performance of `QLPD`. We can then make the hypothesis that $\mathcal{T}_0$ has a large girth without restricting ourselves to a class of worse hypergraph product codes. We will see that having a large girth can guarantee that the standard bolas has interesting properties.

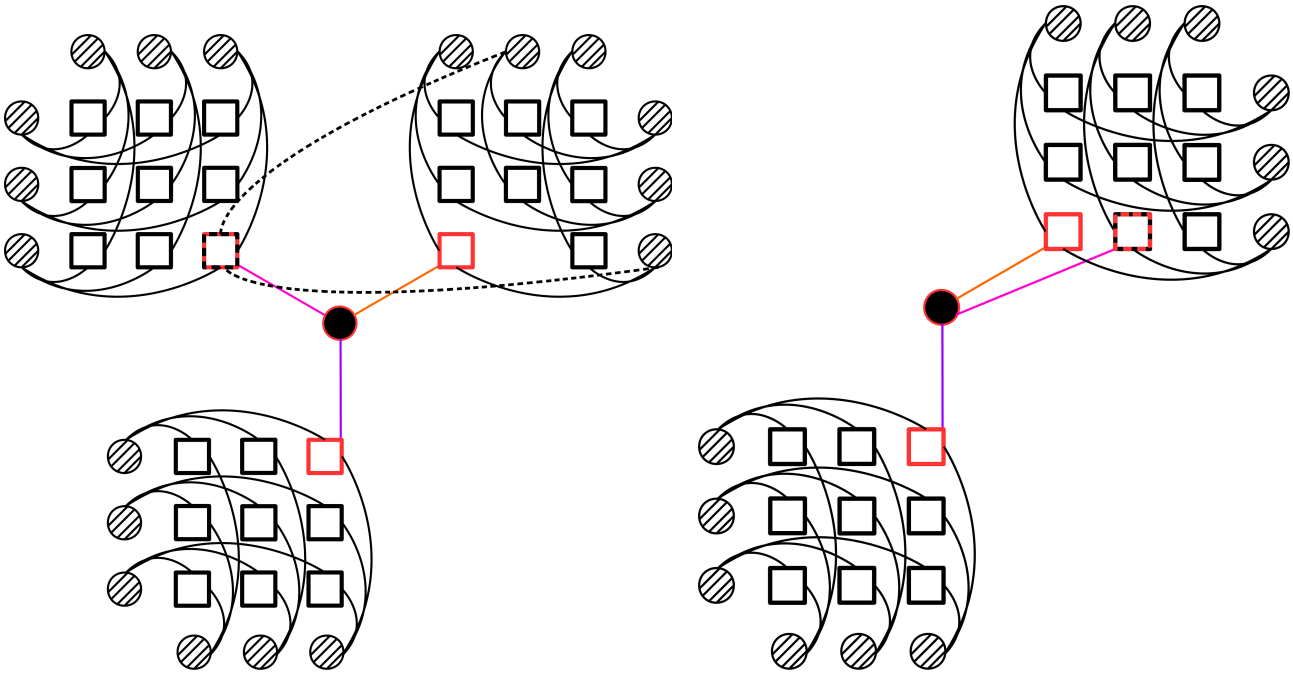
We saw that bolases always belong to the fundamental polytope, but they are generally not pseudocodewords. To guarantee they are a vertex of the fundamental polytope, we have to focus on a particular kind of bolases, we call *ideal bolases*. To formally define them, we first have to define particular sets of variable nodes called *2-regular supports*.

Definition 34 (2-regular support)**.** *We call 2-regular support a set of variable nodes S such that every check node has exactly 2 or 0 elements of S in its neighborhood.*

It is clear that 2-regular supports are the supports of some particular codewords, having rather a small weight or at least locally small weight. In most CSS stabilizer codes, generators are 2-regular supports, and it is the case for every hypergraph product codes. Moreover, if two 2-regular supports are far enough from each other, in the sense that their neighborhoods do not overlap, then their union is also a 2-regular support. Using this definition, we can define a particular type of bolas that can be proven to be pseudocodewords.

Definition 35 (Ideal bolas)**.** *A bolas with peripheral checks* $(c_1, \cdots, c_d)$ *and peripheral variable nodes* P *is an ideal bolas iff* $P = \bigcup_{i \in [d]} P_i$*, where for every* i c_i *is in the neighborhood of* P_i*, and the* P_i *are 2-regular supports with non-overlapping neighborhoods.*

Having these extra constraints will be useful to prove that ideal bolases are not only inside the fundamental polytope, but are actually vertices of the fundamental polytope, namely pseudocodewords. Proving this result is the subject of §4.4. These constraints are in fact often naturally met when setting one generator to 0.5 for each central check since generators are often 2-regular supports, and since they can usually be chosen far enough from each other so that their neighborhoods are distinct.



(a) Here, we look at what would happen if a check was linked to two different generators, and being the central check of one, but a peripheral check for the other.

(b) We see that it would, in fact, allow us to simplify the design of the bolas, as one generator can satisfy two of the central checks at once. This would then not be a problem, as long as it is detected.

Figure 4.5: Case, where a central check and a peripheral check overlap

Lemma 36 (Standard bolases are ideal)**.** *Any standard bolas built on a hypergraph product code obtained from a classical Tanner graph with girth greater than 6 will be an ideal bolas according to definition 35.*

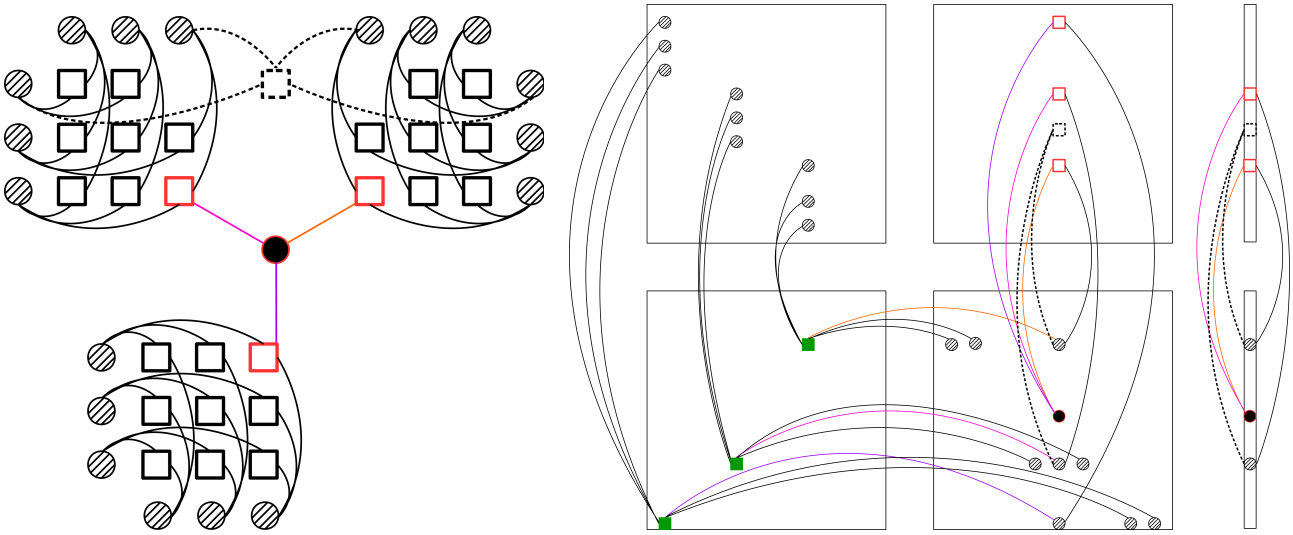
Proof. We take the same notations as in definition 32 and proof of lemma 33 and assume that $\mathcal{T}_0$ has girth greater than 6. By lemma 33 we already know that any standard bolas is indeed a bolas according to definition 30. To prove that they are also ideal, it is enough to check that each peripheral check is linked to peripheral bits coming from the only same central check. Let's assume it is not the case, and that the peripheral check $(v_f \times c_g)$ is linked to both $\Gamma((c_{i'_j} \times v_{i_j}))$ and $\Gamma((c_{i'_k} \times v_{i_k}))$. Then in $\mathcal{T}_0$ we have v_f linked to both $c_{i'_j}$ and $(c_{i'_k}$, and c_g linked to both v_{i_j} and v_{i_k}. We can thus exhibit the following length 6 cycle of $\mathcal{T}_0$: $(v_f$— $c_{i'_j}$— v_{i_j}— c_{i_0}— v_{i_k}— $c_{i'_k}$— v_f. This cycle can be actually smaller if $v_f = v_{i_j}$ or $v_f = v_{i_k}$, however it will never be trivial as we cannot have both equalities at the same time. It is important to check that the cycle can

never be trivial (for example by only going back and forth on a few edges), since a cycle on the product graph does not always translate into a cycle on the original graph. Since the girth exceeds 6, such a cycle cannot exist and any standard bolas is thus ideal. □

For now, we focused on studying the bolas itself which is a fractional word, but it can be useful to also design errors close to it. We will focus in particular on one kind of error guaranteed to be very close to the bolas, which we will call *bolas spawned errors*.

Definition 37 (Bolas spawned error)**.** *Given a bolas x having a central bit v_{i_0} and l peripheral bits $\{v_{i_k}\}_{1\leq k\leq l}$, we say that an error e is spawned by x iff its support is equal to the central bit plus at least half of the peripheral bits.*

The basic idea behind this definition is that a bolas spawned error will always be closer to the bolas which spawned it than to the all zero codeword, and that is what we exploit in section 4.5 to prove some errors uncorrectable.



(a) We look once again at the case, where a check is linked to two generators, but this time as a peripheral check for both.

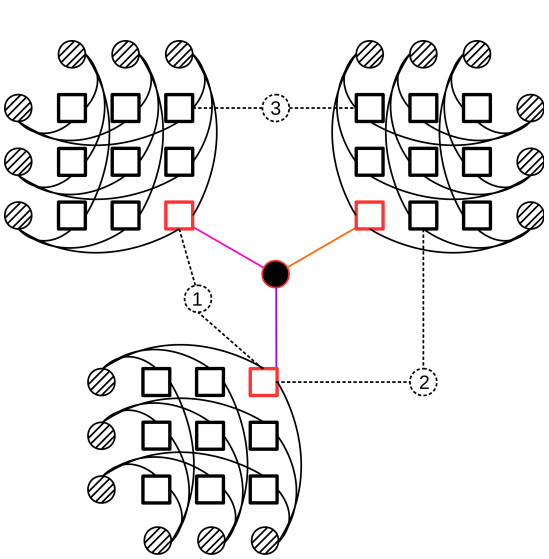
(b) We see that it is enough to have a cycle of length 6 in the classical code to allow such a connection.

Figure 4.6: Case, where two peripheral checks overlap

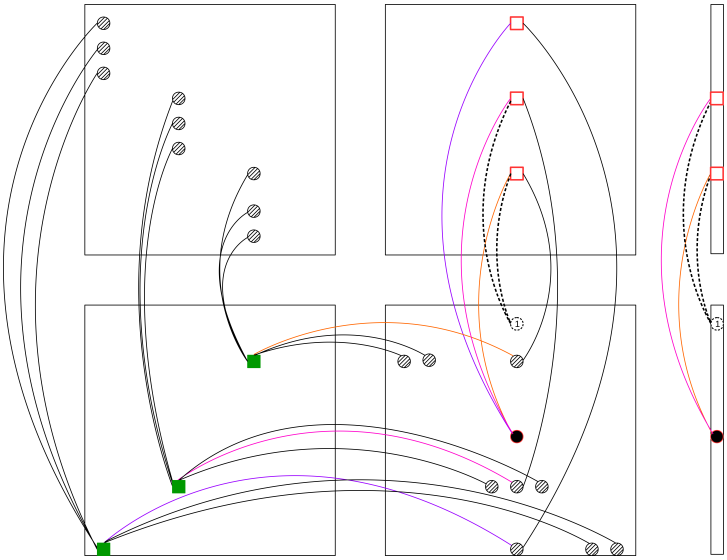
When working on bolases of hypergraph product codes, we will sometimes want the checks of two different generators (both the central and peripheral check nodes) to be linked to the check nodes of another generator only by the central variable node. Namely, the neighborhood of a peripheral check should not overlap with the neighborhood of any check of another generator, and the neighborhood of a central check has only the central bit in common with the neighborhood of another central check. We will see that standard bolases always have this property if the classical Tanner graph used for the hypergraph product has girth greater than 8.

Lemma 38 (Standard bolases have distant peripheral checks)**.** *Any standard bolas built on a hypergraph product code obtained from a classical Tanner graph with girth greater than 8 has the following property: if we gather the peripheral checks and central checks in clusters so that two check nodes are in the same cluster if they are linked by a peripheral bit, then only the central bit links two different clusters.*

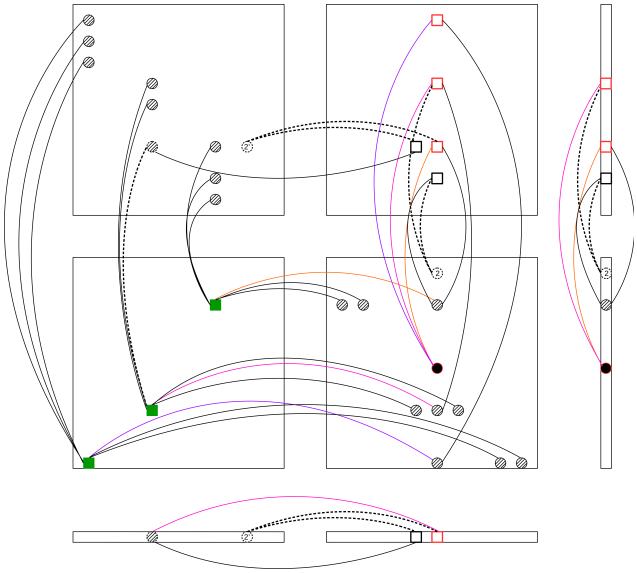
Proof. We take the same notations as in definition 32 and proof of lemma 33 and assume that $\mathcal{T}_0$ has girth greater than 8. Each cluster will actually be equal to the neighborhood of the



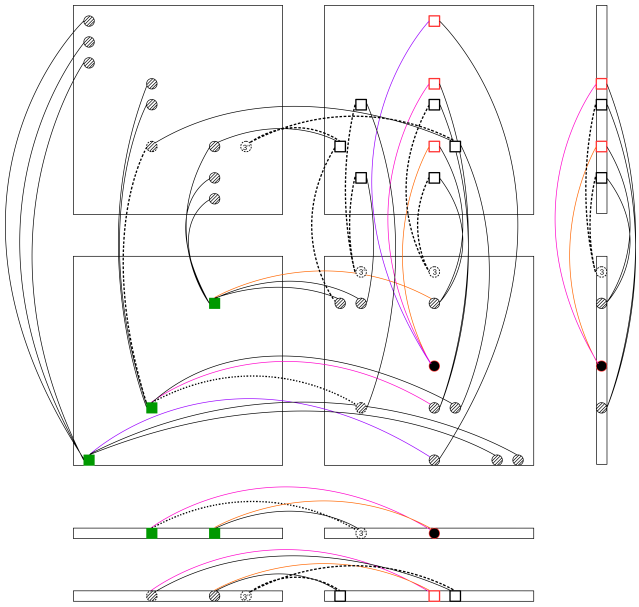
(a) We now look at the case, where a bit links two checks of different generators. The check can both be peripheral, or both central, or one central and one peripheral.



(b) In the case, where both are central, we can exhibit a length 4 cycle in the classical graph.



(c) In this case, there are two different possibilities, one which requires a length 4 cycle, and one which requires a length 6 cycle in the classical graph.



(d) In this last case there are three different possibilities, which require the classical graph to have a cycle of length 4, 6, or 8.

Figure 4.7: Case, where a bit links two different generators

peripheral bits obtained from a single central check, and will thus contain this central check and several peripheral checks. we have to differentiate the case, where the bit linking the two clusters is a vv-type variable node or a cc-type variable node. If the vv-type variable node $(v_f \times v_g)$ links the neighborhoods of the sets of peripheral bits $\Gamma((c_{i'_j} \times v_{i_j}))$ and $\Gamma((c_{i'_k} \times v_{i_k}))$, then v_f is linked to both $c_{i'_j}$ and $c_{i'_k}$ we can thus exhibit the following length 6 cycle on $\mathcal{T}_0$: $(v_f$— $c_{i'_j}$— v_{i_j}— c_{i_0}— v_{i_k}— $c_{i'_k}$— v_f. This cycle can be actually smaller if $v_f = v_{i_j}$ or $v_f = v_{i_k}$, however it will never be trivial as we cannot have both equalities at the same time. Similarly, if the cc-type variable node $(c_f \times c_g)$ links the neighborhoods of the sets of peripheral bits $\Gamma((c_{i'_j} \times v_{i_j}))$ and $\Gamma((c_{i'_k} \times v_{i_k}))$, then c_g is linked to both a variable node $v_{i'_j}$ linked to $c_{i'_j}$ and to a variable node $v_{i'_k}$ linked to $c_{i'_k}$. We can thus exhibit the following length 8 cycle on $\mathcal{T}_0$:

(c_g— $v_{i'_j}$— $c_{i'_j}$— v_{i_j}— c_{i_0}— v_{i_k}— $c_{i'_k}$— $v_{i'_k}$— c_g. None of these cycles could exist if the girth of $\mathcal{T}_0$ exceeds 8, thus the result. □

In the two following sections, we will employ bolases to prove negative results on the decoding of stabilizer codes by the LP decoder. In section 4.4 we first prove a upper bound on the quantum fractional distance of some stabilizer codes. Then in section 4.5 we exhibit uncorrectable errors guaranteed to exist on some hypergraph product codes.

4.4 Using bolases to upper bound the quantum fractional distance

We showed in section 4.3 that bolases are always part of the fundamental polytope. Moreover, their weight can be independent of the blocklength, as a code with bit degree d and generators weight w has some bolases of weight $1 + \frac{wd}{2}$. Because of this, bolases are great candidates to prove an upper bound of $\mathrm{d}_{\mathrm{qfrac}}$the quantum fractional distance independent of the blocklength. However, for the weight of a fractional word to restrict $\mathrm{d}_{\mathrm{qfrac}}$ the fractional word must not only be in the fundamental polytope, but more precisely be a pseudocodeword and thus a vertex of the fundamental polytope. This section will thus be dedicated to proving that ideal bolases are pseudocodewords.

To prove that an element of the fundamental polytope is one of its vertices we will manipulate a characterization of the vertices based on the number of its *valid directions* which are directions (or vectors) along which it can move in both senses a small, but non-null distance while staying in the fundamental polytope.

Definition 39 (Valid directions)**.** *Let x be a point of $\mathcal{P}$, the set $\mathcal{V}_x$ of valid directions of x is defined as:*

$$\mathcal{V}_x = \{\alpha \in [0,1]^n, |\alpha| = 1 \ s.t. \ \exists \epsilon > 0, \forall \delta \in [-\epsilon, \epsilon], x + \delta\alpha \in \mathcal{P}\}.$$

We illustrate this definition in 4.8 on a two-dimensional polytope. It is clear that the number of valid directions depends on whether the point sits on a vertex, an edge, or strictly inside the polytope, and that's what we prove in lemma 40.

Lemma 40. *Let x be a point of the fundamental polytope $\mathcal{P}$ having $\mathcal{N}$ as the set of its vertices. Then x belongs to $\mathcal{N}$ if and only if it is an extreme point, that is a point with no valid directions:*

$$x \in \mathcal{N} \iff \mathcal{V}_x = \varnothing.$$

Proof. We will prove the contrapositive one implication at the time.

$\overline{x \in \mathcal{N}} \implies \overline{\mathcal{V}_x = \varnothing}$:

We adopt the same notations and hypothesis as in the lemma, but we also assume that x does not belong to $\mathcal{N}$, meaning that x is either in the interior of $\mathcal{P}$ ($x \in \overset{\circ}{\mathcal{P}}$) or x is in the interior of an edge of $\mathcal{P}$ ($\exists \beta, \gamma \in \mathcal{N} \mid x \in]\beta, \gamma[$).
In the first case, because of the definition of the interior of $\mathcal{P}$, any α with norm equal to 1 is in $\mathcal{V}_x$.
In the second case, the vectors $\alpha = \frac{x-\beta}{\mathrm{d}(x,\beta)}$ and $-\alpha = \frac{x-\gamma}{\mathrm{d}(x,\gamma)}$ are in $\mathcal{V}_x$.
In both cases, $\mathcal{V}_x$ is not empty, which concludes the proof of this implication.

$\overline{x \in \Gamma} \impliedby \overline{\mathcal{V}_x = \varnothing}$:

We adopt the same notations and hypothesis as in the lemma, but we also assume that $\mathcal{V}_x$ is

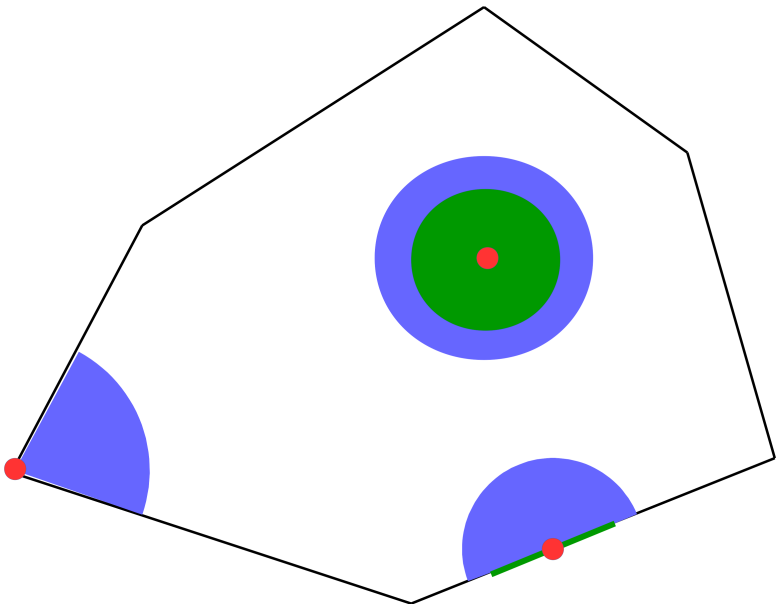


Figure 4.8: Example of valid directions on a two-dimensional polytope. For each of the three red points, we represented in blue the directions towards which we can move from the point while staying in the polytope, and in green the valid directions. We see that the point strictly inside the polytope has an infinite number of valid directions, while the one on the edge has only 2, and the one on the vertex has none.

not empty. Then by definition, there exists a point α with norm 1 and a positive number ϵ such that $\beta = \alpha + \epsilon\alpha$ and $\gamma = \alpha - \epsilon\alpha$ both belong to $\mathcal{P}$. Moreover, $0.5\beta + 0.5\gamma = x$, thus x can be written as a convex combination of points of $\mathcal{P}$, so x is not a vertex of this polytope. It concludes the proofs of this implication and of the lemma. □

Thanks to this lemma, we simply have to show that a bolas has no valid direction to prove that it is a pseudocodeword. To do so we will prove several properties a valid direction of a bolas must satisfy and show that under some condition all these conditions cannot be met, or more precisely that every coordinate of the valid direction should be equal to zero.

The first result is just a consequence of the fact that the points of the polytope have coordinates between 0 and 1, but will be very useful since bolases typically have most of their coordinates set to 0.

Lemma 41. *Let x be a point of $\mathcal{P}$ with x_i integral for some i, then:*

$$\alpha \in \mathcal{V}_x \implies \alpha_i = 0.$$

Proof. Trivial by exploiting the fact that the fundamental polytope $\mathcal{P}$ is included in the hypercube $[0, 1]^n$. □

This result tells us that all the coordinates of the valid direction corresponding to integral coordinates of x will have to be null, which highly reduces the degrees of liberty that will have the valid direction. However, it does not constraints the coordinates of the valid direction corresponding to coordinates of x set to 0.5, to do so we will need to exploit the characterization of the fundamental polytope by cuts.

The second result shows that under some circumstances the value of a coordinate of a valid direction propagates to other coordinates, meaning that once you chose the value of the direction for some coordinates, the value for some other coordinates will be determined.

Lemma 42. *Let c be a check node of degree d and of neighborhood $(v_{i_j})_{j\leq d}$, if x is a point of $\mathcal{P}$ such that $(x_{i_j})_{j\leq d} = (0.5, 0.5, 0, \cdots, 0)$ then:*

$$\alpha \in \mathcal{V}_x \Longrightarrow (\alpha_{i_j})_{j\leq d} = (\alpha_{i_1}, \alpha_{i_2}, 0, \cdots, 0) \text{ and } \alpha_{i_1} = \alpha_{i_2}.$$

Proof. We already know from the previous lemma that $(\alpha_{i_j})_{3\leq j\leq d} = (0, \cdots, 0)$. To prove $\alpha_{i_1} = \alpha_{i_2}$, we will exploit the cut-based definition of the fundamental polytope, and in particular the cut obtained from the check c and the set $S = \{v_{i_1}\}$:

$$y \in \mathcal{P} \Longrightarrow y_{i_1} + \sum_{2\leq j\leq d} (1 - y_{i_j}) \leq d - 1 \ (1).$$

If α is in $\mathcal{V}_x$ then there exists $\epsilon > 0$ such that $x + \epsilon\alpha$ and $x - \epsilon\alpha$ are both in $\mathcal{P}$. Thus, they must satisfy the inequality (1), which applied to $y = x + \epsilon\alpha$ gives:

$$y_{i_1} + \epsilon\alpha_{i_1} + \sum_{2\leq j\leq d} (1 - (y_{i_j} + \epsilon\alpha_{i_j})) \leq d - 1 \Longleftrightarrow 0,5 + \epsilon\alpha_{i_1} + 1 - 0,5 - \epsilon\alpha_{i_2} + d - 2 \leq d - 1$$
$$\Longleftrightarrow \epsilon\alpha_{i_1} \leq \epsilon\alpha_{i_2}.$$

Similarly, by applying (1) to $y = x - \epsilon\alpha$ we get $\epsilon\alpha_{i_1} \leq \epsilon\alpha_{i_2}$, thus by combining both inequalities we get $\epsilon\alpha_{i_1} = \epsilon\alpha_{i_2}$ which since $\epsilon \neq 0$ implies $\alpha_{i_1} = \alpha_{i_2}$ which concludes the proof. □

This lemma shows us that given a point x of $\mathcal{P}$, choosing to increase or decrease some coordinates set to 0.5 will require to apply the same modification to another coordinate to stay in $\mathcal{P}$. This propagation of the modification is due to some check nodes having in their neighborhood only 2 variable nodes set to 0.5 and all the rest set to 0. Because of the constraint they induce on valid directions, we call them growth propagating nodes.

Definition 43 (Growth propagating node)**.** *Let x be a point of $\mathcal{P}$, then the check node c of degree d and of neighborhood $(v_{i_j})_{j\leq d}$ is called growth propagating node of x between v_{i_1} and v_{i_2} if:*

$$(x_{i_j})_{j\leq d} = (0.5, 0.5, 0, \cdots, 0).$$

We illustrate this definition for the toric code in figure 4.9 with x a non-ideal bolas. We see that growth propagating nodes can force a whole set of coordinates of a valid direction to be equal, and we call the set of variable nodes corresponding to these coordinates a growth propagating cluster.

Definition 44 (Growth propagating cluster)**.** *Let x be a point in the fundamental polytope of the Tanner graph $\mathcal{T}$, let $\mathcal{G}$ be the graph whose nodes are the variable nodes of $\mathcal{T}$, and, where there is an edge between two nodes v_i and v_j iff there is a check node $\mathcal{T}$ which is a growth propagating node of x between v_i and v_j. The connected components of $\mathcal{G}$ are called the growth propagating clusters of x.*

We illustrate this definition in figure 4.10 on the toric code and the same bolas.

We will now formally state the result that makes growth propagating clusters interesting, namely that the coordinates of a valid direction corresponding to variable nodes in the same cluster need to be the same.

Lemma 45. *Let x be a point of the fundamental polytope, and let v_{i_1} and v_{i_2} be two variable nodes in the same growth propagating cluster of x, then:*

$$\alpha \in \mathcal{V}_x \Longrightarrow \alpha_{i_1} = \alpha_{i_2}.$$

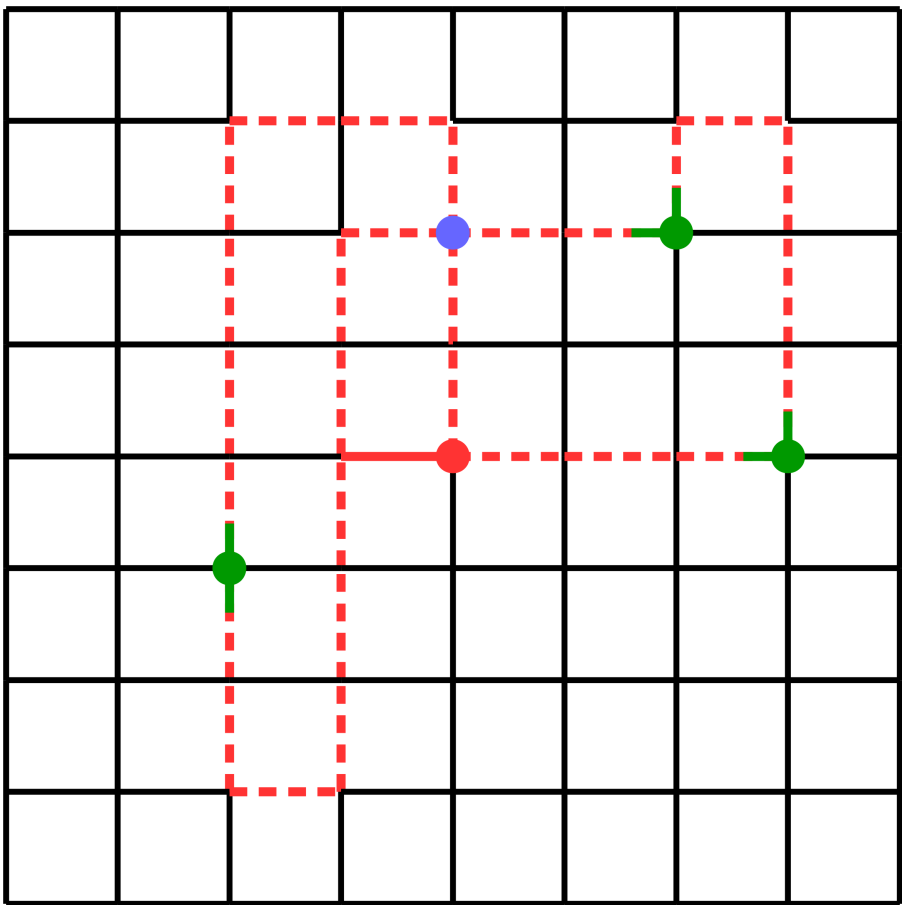


Figure 4.9: Example of growth propagating nodes. In this example, we represented in green some growth propagating nodes and colored partly in green the two variable nodes between which they propagate the growth. The red check node isn't a growth propagating node, as it is linked to a variable node set to 1. The blue check node isn't a growth propagating node either, as it isn't linked to exactly 2 variable nodes set to 0.5.

Proof. Using lemma 42 together with the definition of growth propagating node, we know that if there is a path made of growth propagating nodes between two variable nodes v_{i_1} and v_{i_2}, then their corresponding coordinates in a valid direction will be the same. Such a path is guaranteed to exist by the definition of growth propagating clusters. □

Growth propagating clusters are clearly linked to 2-regular supports, but they are not the same nor does one imply the other. Indeed, growth propagating clusters can contain only one variable node contrary to 2-regular neighborhoods, while being always connected which is not necessary the case for 2-regular supports. However, it is clear that if x has all its coordinate equal to 0 except the variable nodes of a connected 2-regular support, then the 2-regular support will also be a growth propagating cluster of x.

After showing that some nodes forced some coordinates of valid directions to be equal, we will show that some other nodes can force a given coordinate to be null, which allows us in the end to show in some cases that all coordinates are in equal to zero.

Lemma 46. *Let c be a check of degree d and with neighborhood $(v_{i_j})_{j\leq c}$, if x is a point of $\mathcal{P}$ such that $(x_{i_j})_{j\leq d} = (0.5, 0.5, 0, \cdots, 0, 1)$ and if α is in $\mathcal{V}_x$ and $\alpha_{i_1} = \alpha_{i_2}$ then:*

$$(\alpha_{i_j})_{j\leq d} = (0, \cdots, 0).$$

Proof. We already know from lemma 41 that $(\alpha_{i_j})_{3\leq j\leq d} = (0, \cdots, 0)$. To prove $\alpha_{i_1} = \alpha_{i_2} = 0$, we will exploit the definition of the fundamental polytope based on cuts, and in particular the

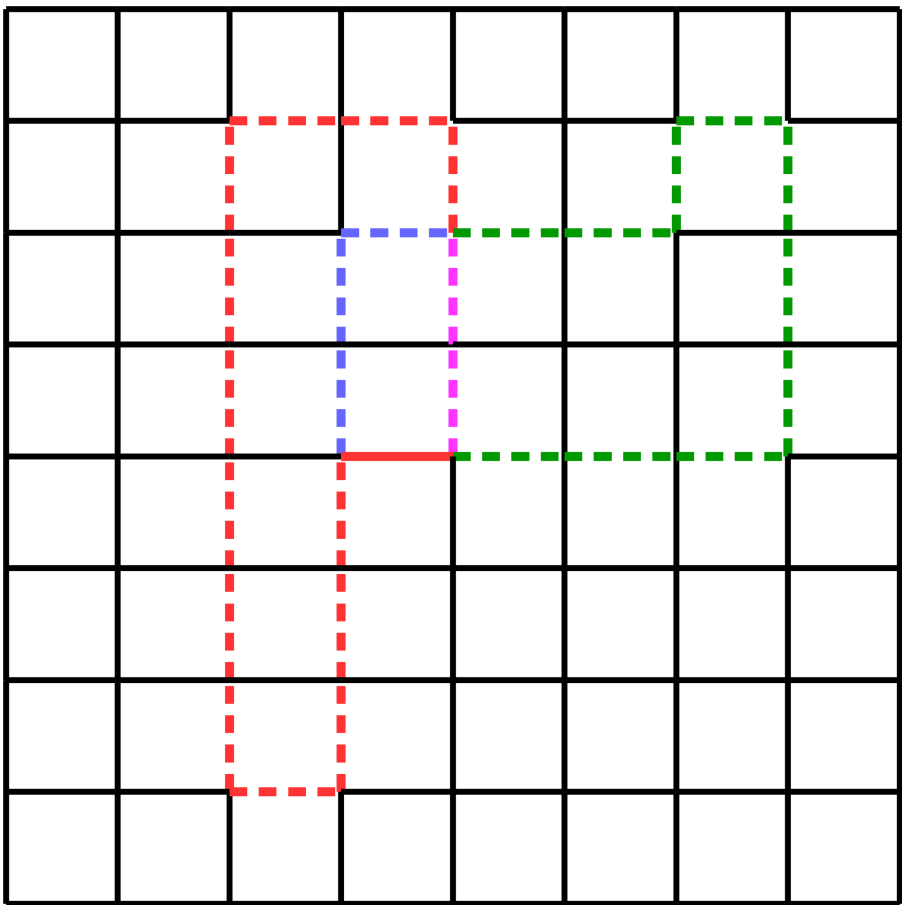


Figure 4.10: Example of growth propagating clusters. In this example, we represented the different growth propagating clusters. The dashed bits of the same cluster are in the same color, while all the black bits and the continuous red bit are alone in their clusters. We see that there are many clusters of size 1 and only 4 clusters of bigger size, and that they form a partition of the set of variable nodes.

cut obtained from the check c with the set $S = \{v_{i_1}, v_{i_2}\}$:

$$y \in \mathcal{P} \Longrightarrow y_{i_1} + y_{i_2} + \sum_{3 \leq j \leq d} (1 - y_{i_j}) \leq d - 1 \qquad (1).$$

If α is in $\mathcal{V}_x$ there exists $\epsilon > 0$ such that $x + \epsilon\alpha$ and $x - \epsilon\alpha$ both are in $\mathcal{P}$. Then they both must satisfy inequality (1), which applied to $y = x + \epsilon\alpha$ gives the following inequality:

$$\begin{aligned} x_{i_1} + \epsilon\alpha_{i_1} + x_{i_2} + \epsilon\alpha_{i_2} + \sum_{3 \leq j \leq d} (1 - (x_{i_j} + \epsilon\alpha_{i_j})) \leq d - 1 &\iff 0.5 + \epsilon\alpha_{i_1} + 0.5 + \epsilon\alpha_{i_2} + \sum_{3 \leq j \leq d} (1) \leq d - 1 \\ &\iff \epsilon\alpha_{i_1} + \epsilon\alpha_{i_2} + d - 1 \leq d - 1 \\ &\iff \epsilon\alpha_{i_1} + \epsilon\alpha_{i_2} \leq 0. \end{aligned}$$

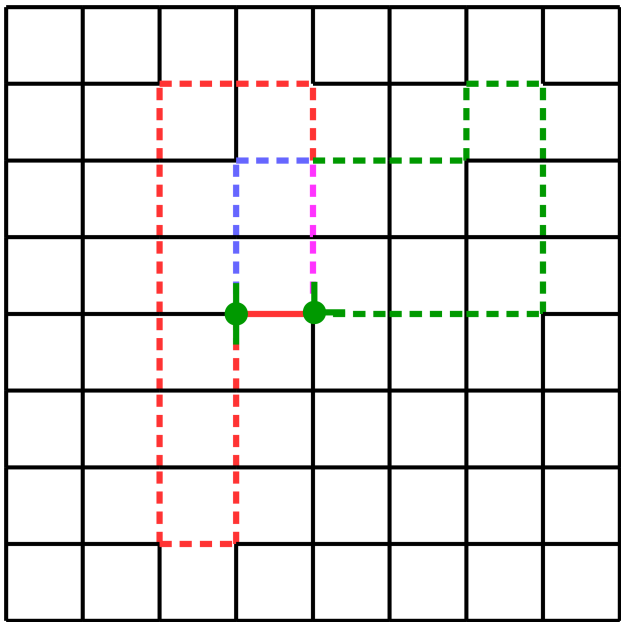
When applying it instead to $y = x - \epsilon\alpha$, we find $\epsilon\alpha_{i_1} + \epsilon\alpha_{i_2} \geq 0$ which combined with the previous inequality and the fact that ϵ is not zero gives us $\alpha_{i_1} = -\alpha_{i_2}$. Yet, we assumed $\alpha_{i_1} = \alpha_{i_2}$, so finally $\alpha_{i_1} = \alpha_{i_2} = 0$. □

This result is powerful when combined with growth propagating clusters, since two bit-nodes of this cluster we always have the same corresponding value in a valid direction. Because of this, whenever a check node is linked to only variable nodes set to 0 except one variable node equal to 1, and two variable nodes from the same growth propagating cluster set to 0.5, then

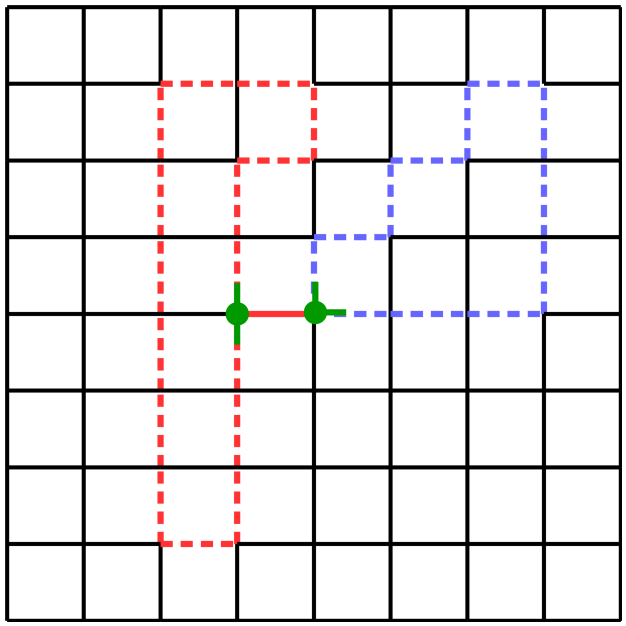
all the coordinates of a valid direction corresponding to bit-nodes of the growth propagating cluster will be null. Because of this, we call such clusters *growth cancelling clusters*.

Definition 47 (Growth cancelling cluster)**.** *Let x be a point of $\mathcal{P}$, then a growth propagating cluster is also a growth cancelling cluster iff two of its variable nodes v_i and v_j are neighbors of the same check node which has only one other non-zero variable node in its neighborhood and that this variable node is set to 1.*

We illustrate this definition on the toric code in figure 4.11. It is particularly easy to decide whether a growth propagating cluster is a growth cancelling cluster for a bolas since only the central bit is set to 1 and can thus cancel the growth.



(a) Here, we show the growth propagating clusters of figure 4.10 and the two only check nodes linked to a variable node equal to 1. We see that none of these checks is linked to two bits of the same clusters, meaning that there are no growth cancelling clusters. It is clear that growth cancelling clusters do not form a partition of the variable nodes.



(b) Here, we slightly modified the example to have an ideal bolas: there are no more checks neighbor of the two 2-regular supports. In this case, we see that we have only two growth propagating clusters, and that each of them has two variable nodes linked to the same central check node, meaning that these growth propagating clusters are also growth cancelling clusters.

Figure 4.11: Example of growth cancelling clusters.

We now formalize the result that makes growth cancelling clusters so interesting to constrain valid directions.

Lemma 48. *Let x be a point of the fundamental polytope, and let v_i be a variable node in a growth cancelling cluster of x, then we have :*

$$\alpha \in \mathcal{V}_x) \Longrightarrow \alpha_i = 0$$

Proof. We first take advantage of the definition of a growth cancelling cluster and the lemma 45 to show that all the conditions of lemma 46 are met, we then apply it together with 45 to conclude that all coordinates of the growth propagating cluster must be null. □

We then have a simple condition to assess that a point of $\mathcal{P}$ is a vertex of $\mathcal{P}$.

Theorem 49. *Let x be a point in the fundamental polytope $\mathcal{P}$ of H, if all the fractional variable nodes (with non-integral value) of x belong to growth cancelling clusters of x, then x is a vertex of $\mathcal{P}$.*

Proof. We know that if a direction is valid, then only the coordinates corresponding to bits having fractional values can be non-zero. Yet by hypothesis, all the fractional bits are part of growth cancelling clusters, which means that the associated coordinates have to be zero in the valid direction. In conclusion, every coordinate of the valid direction has to be equal to zero meaning that there is in fact no valid direction, so the point has to sit on a vertex of $\mathcal{P}$. □

This theorem does apply to the modified example introduced in 4.11 which is an ideal bolas, yet it does not apply to the original example as there are many variable nodes set to 0.5, but no growth cancelling clusters. Indeed, we show in figure 4.12 that we can even prove that this example doesn't sit on a vertex of the fundamental polytope, as it can be obtained as the barycenter of two points of the polytope.

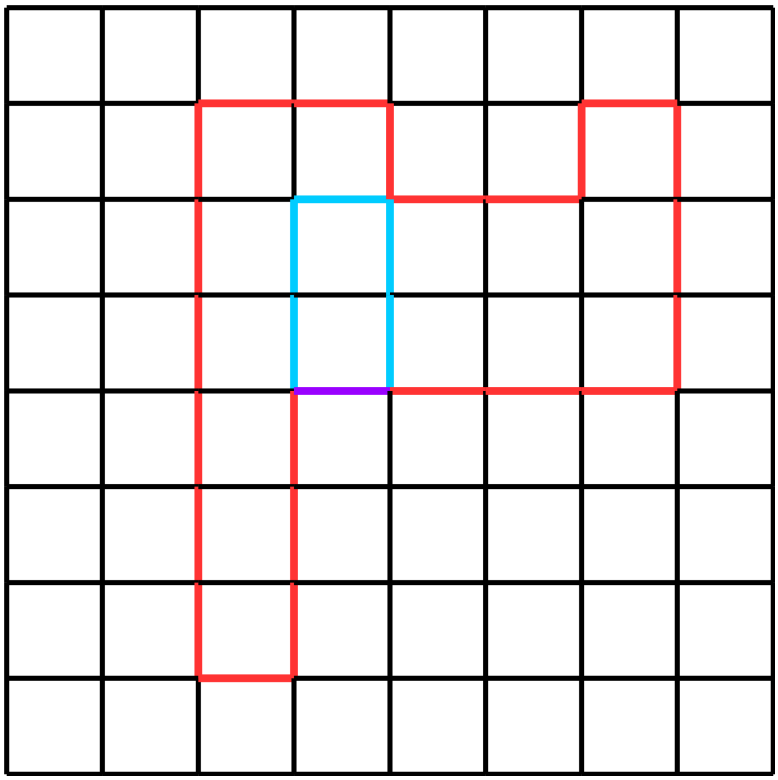


Figure 4.12: Example of a barycentric combination of codewords. Going back to the example of 4.11 we see that it can be described as the mean of two codewords, the one in red and purple and the one in blue and purple. Only the purple variable node is set to 1 in both codewords, meaning that it will be the only variable node set to 1 in the barycenter. The blue and the red variable nodes will be set to 0.5, and the black variable nodes will remain to 0. It is indeed the very same bolas as in the example of figure 4.2, meaning that it does not sit on a vertex of the fundamental polytope, as it can be obtained by a convex combination of codewords which are vertices of the polytope.

We cannot apply directly this theorem on ideal bolases, since a 2-regular support P_i chosen for the central check node c_i is not necessarily connected. Thus, instead of having all its variable nodes part of the growth cancelling cluster linked to c_i, there might also be 2-regular supports disconnected from it and forming their own growth propagating cluster. However, we can, in fact, remove these variable nodes from P_i and obtain a new 2-regular support satisfying all

the conditions. This leads us to the following theorem, which links ideal bolases and quantum fractional distance.

Theorem 50. *A fundamental polytope having an ideal bolas of weight w has a quantum fractional distance upper bounded by w.*

Proof. Let $\{c_{i_j}\}_{j\leq d}$ be the central check nodes of the bolas x, and $\{P_{i_j}\}_{j\leq d}$ their neighboring 2-regular supports. We call P'_{i_j} the connected component of P'_{i_j} linked to c_{i_j}. We can then design a new bolas having the same central variable nodes as x, but having peripheral check nodes equal to $\bigcup_{j\leq d} P'_{i_j}$. This new bolas x' has now all its peripheral variable nodes part of growth cancelling clusters, so according to theorem 49 it is a vertex of the fundamental polytope. Since it is a fractional vertex, its weight thus upper bound the quantum fractional distance. Moreover, since its weight is by construction lower than the weight w of x, w also upper bounds the quantum fractional distance. □

We demonstrate this on the toric code in figure 4.11, and show the importance of having ideal bolases and not simply bolases to guarantee that it is a vertex of the fundamental polytope in figure 4.12. The toric code allows to easily check whether a bolas is ideal or not thanks to its graphical representation, as for the example of figure 4.13. We can thus employ theorem 50 to prove that the quantum fractional distance of a toric code is at most 5, regardless of its blocklength.

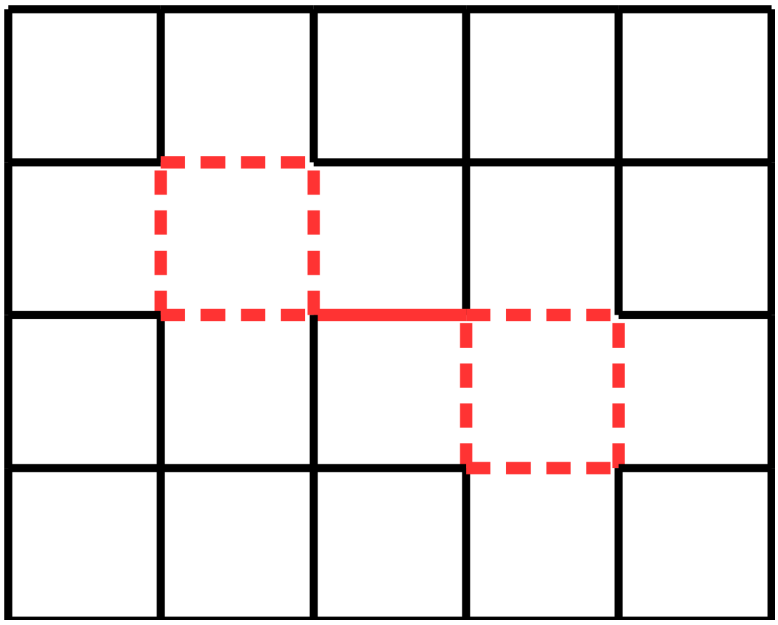


Figure 4.13: Example of a bolas on the toric code with two maximally distant generators as peripheral variable nodes. It is easy to check that this bolas is, in fact, ideal, as the two generators have no common check in their neighborhood. Theorem 50 then tells us that the quantum fractional distance of toric codes is at most 5.

We don't know how to guarantee the existence of small ideal bolases for general stabilizer codes, even though it seems realistic that most codes having generators which are also 2-regular supports (which is the case with many codes and all hypergraph product codes) do have ideal bolases. Fortunately, we can guarantee the existence of ideal bolases in the case of hypergraph product codes given conditions on the girth of the Tanner graph of the classical code, which leads to the following theorem.

Theorem 51. *A hypergraph product code obtained from a Tanner graph having girth greater than 6 and degrees lower or equal to δ has a quantum fractional distance lower or equal to δ^2+1.*

Proof. Let x be standard bolas presented in definition 32. Lemma 36 guarantees that x is an ideal bolas, since the girth of the classical Tanner graph exceeds 6. We then simply have to measure the weight of the standard bolas and apply theorem 50. The weight of a standard bolas is equal to half the number of peripheral bits plus 1. The number of check nodes linked to the central variable node is upper bounded by δ, so there are at most δ central checks. Each central check implies a set of peripheral bits of size at most 2δ variable nodes. Thus, there are at most $2\delta^2$ peripheral bits. The standard bolas has thus weight at most δ^2+1, which since the standard bolas is ideal, upper bounds the quantum fractional distance according to theorem 50.

□

The conditions of this theorem makes it usable on only a fraction of the hypergraph product codes family. However, it should be taken as a sign that the decoding certificate will most likely not be useful when the blocklength is much larger than the generators weight. Indeed, even though several hypotheses were required to formally demonstrate the result, we doubt that not meeting these requirements will allow the quantum fractional distance to grow larger. If we look in particular at the girth constraints for hypergraph product codes, it should be understood that to prevent every standard bolas to be ideal, one small cycle would not be enough. Instead, the classical Tanner graph should be filled with small cycles, which does not seem reasonable. Moreover, even if there are indeed enough small cycles we think that small pseudocodewords would still exist, but in a different and maybe more complicated form like for example what we showed in figure 4.5.

Showing that the quantum fractional distance most likely cannot grow with the blocklength for LDPC stabilizer codes definitely remove any interest from the quantum version of the decoding certificate. However, the quantum version was already less interesting than the original version in the classical case since we could not compute the quantum fractional distance the same way as the fractional distance, and we did not find another efficient way to compute it.

Finally, it is important to stress out that even though the quantum decoding certificate cannot guarantee that every constant weight error can be corrected, the result does not say that there are indeed uncorrectable constant weight errors. Here, we proved that the fundamental polytope does have indeed fractional vertices very close to the all zero codeword, but for them to be a problem we would also need an objective function to reach its optimal value on one of these pseudocodewords. The objective function is given by the noisy word we want to correct, and so can take only a given set of values. Because of this, some points of the fundamental polytope might never be returned by the linear program even when carefully choosing the input codeword. Because of this, having pseudocodewords close to codewords may not be a problem and does not mean that there are small uncorrectable errors. However, we will investigate the existence of such errors in the next section, with bolas spawned errors as candidates.

4.5 Constant weight uncorrectable errors from bolases

The goal of this section is to design constant weight uncorrectable errors with bolases, and find conditions guaranteeing their existence. To do so, we will first give a simple characterization of uncorrectable errors, and then exploit the properties of bolases to exhibit one having a weight independent of the blocklength.

Lemma 52. *Any error closer to a bolas that to any codeword equivalent to the all-zero codeword cannot be corrected by the LP decoder.*

Proof. Follows from lemma 29 and the fact that every bolas belongs to the fundamental polytope. □

To employ this result, we not only need a bolas, but also an error close to this bolas. It is to this end that we introduced bolas spawned errors in definition 37. The interest of these errors is that by construction they are closer to the bolas which spawned them than to the all-zero codeword. Indeed, the distance from a spawned error to its bolas is always equal to half the number of peripheral variable nodes, while its weight is at least 1 plus half the number of peripheral variable nodes. This is still not enough to ensure it is uncorrectable, since we also need to check that it is closer to the bolas than to any word equivalent to the all-zero codeword. However, this is guaranteed if the error is minimal, since the all-zero codeword will be the element in its equivalence class the closer to the error.

Lemma 53. *Any bolas spawned minimal error is uncorrectable by the LP decoder.*

Proof. Follows from lemma 52 and the fact that the smallest distance between a minimal error and a codeword equivalent to the all-zero codeword is precisely the weight of the error. □

Similarly to what we had we tried to upper bound the quantum fractional distance, we end up with a negative result requiring conditions we cannot guarantee in the general case. Indeed, we did not prove the existence of bolases in the general case, but only for hypergraph product codes given light conditions on the classical code, and we did not investigate at all the minimality of bolas spawned errors. To exhibit a case, where we are guaranteed to have constant weight uncorrectable errors, we will investigate the conditions for a bolas spawned error to be minimal on hypergraph product codes.

Let x be a bolas on a hypergraph product code designed, and e an error spawned by x. Since the peripheral variable nodes form the support of a degenerate error, flipping all the peripheral variable nodes in e will result in an error $\tilde{e}$ equivalent. For e to be minimal it is thus necessary that $|\tilde{e}|$ to be greater than $|e|$, and thus that at most half of the peripheral variable nodes are equal to 1 in e. Since e is a bolas spawned error, at least half of the variable nodes of x are equal to 1. We thus need to have exactly half of the variable nodes equal to 1, thus the number of variable nodes must be even. Moreover, since in this construction we choose non-overlapping generators (one for each central check node) to form the peripheral variable nodes, each of these generators must also have even weights (otherwise at least one would have more than half its bit set to 1 and thus flipping it would reduce the error weight).

For this reason, we choose the check degrees and the bit degrees of the classical code to be all equal to d, making generators weight even and equal to $2d$. We will moreover require the girth of the classical code to be at least 10, thus ensuring that no variable node links peripheral check nodes from different central check nodes. With this condition, we have a better control over the connectivity of the bolas spawned error, which we can take advantage of by maximizing the weight of the syndrome. Indeed, it is clear that since the degree of the nodes is d, any error of weight w and syndrome weight wd has to be minimal since smaller errors cannot have such a large syndrome. With the bolas constraints, we will never be in this ideal scenario, but we can still maximize the weight of the syndrome by choosing to set only vv-type (or cc-type) peripheral bits to 1. This maximizes the weight of the syndrome while guaranteeing that half the peripheral variable nodes are set to 1. Employing this design of bolas spawned errors, we can actually guarantee that it will be minimal.

Theorem 54 (Failure certificate of the LPD with HP codes)**.** *Let $\mathcal{C}$ $[n, k]$ be a classical code defined by a Tanner graph $\mathcal{T}$ of girth at least 10 and check nodes degrees equal to variable nodes degrees equal to δ. Then the quantum code defined by the hypergraph product of $\mathcal{T}$ with itself has at least $\mathcal{O}(n)$ uncorrectable X errors of weight $\delta^2 + 1$.*

Proof. To prove this theorem, we will show that given a central variable node product of twice the same node of $\mathcal{T}$, we can build an uncorrectable error of weight $\delta^2 + 1$. For this purpose, we will adopt the design described above and the guarantees we have from the large girth of $\mathcal{T}$. Let x be a standard bolas, it is guaranteed to be an actual bolas by lemma 33. Let e be an error spawned by x, with support equal to the central variable node and the vv-type peripheral variable nodes. To ensure that e is uncorrectable, we only need to prove that it is minimal and then apply lemma 53. Let s be the syndrome of e. By construction, $|e|$ is equal to $\delta^2 + 1$ (generators have weight 2δ and we set δ of them half to 1 plus the central variable node) and $|s|$ is equal to $\delta(\delta^2 - 1)$ (all the peripheral check nodes are unsatisfied). It follows that every error equivalent to e must have weight at least $\delta^2 - 1$, since it must have syndrome $|s|$. Thus, if e is not minimal, there must exist an equivalent error $\tilde{e}$ of weight equal to $\delta^2 - 1$ or δ^2. We will show that both cases are impossible, thus proving that e is minimal and thus uncorrectable by application of lemma 53.

We will first exploit the fact that $|\tilde{e}|$ cannot be equal to δ^2. This actually comes from the fact that all the generators have even weight, and thus two equivalent errors must have weights with the same parity. This is not true in this case, $\tilde{e}$ thus cannot have this weight and be equivalent to e.

In the second case, every variable nodes of the support of $\tilde{e}$ needs all its δ neighbors to be unsatisfied (thus peripheral checks for a bolas). Since there are $\delta^2 - 1$ peripheral check nodes from one central check, for each variable node of the support of $\tilde{e}$ to be linked to δ peripheral check nodes, some would need to link peripheral check nodes from different central check nodes. However, since the girth is larger than 8, lemma 38 guarantees that such variable nodes cannot exist.

We can thus conclude that e is indeed minimal and thus by application of lemma 53 it cannot be corrected by the LP decoder. □

As in the case of the existence of ideal bolases, we can prove the existence of constant weight uncorrectable errors only under severe conditions. However, we believe once again that even though those constraints are necessary to analyze the error, not having these conditions will most often still be compatible with the existence of constant weight uncorrectable errors, only not as simple to exhibit. Moreover, we tried such a design on some hypergraph product codes built from 3,4-regular classical codes (replacing each generator by two overlapping generators to have even weight degenerated errors) and they were sometimes uncorrectable by the LP decoder even though the degrees and the girth of the classical Tanner graph did not satisfy the conditions. Interestingly enough, every such errors we tried could not be corrected by `BP` showing once again the link between these two decoders.

In §4.5.1 and §4.5 we saw two examples of constant weight uncorrectable errors and codes for which they are guaranteed to happen. To simplify the analysis, we assumed they were decoded by the binary LP decoder. However, the negative result would still hold in the case of the non-binary LP decoder, since any point of the fundamental polytope in the binary case corresponds to a point of same weight within the non-binary fundamental polytope. To derive from it a lower bound on the WER, independent of the blocklength, we must not only show that these errors are uncorrectable, but that any error having only a fixed number of bits identical to it will also be uncorrectable. For example, the idea in the case of the minimal bolas spawned errors would be to show that any error identical to it for every bit closer to the central bit than some value independent of the blocklength would also be uncorrectable. Although we think this is the case, we did not prove it, but will consider these error patterns to be incompatible with a threshold. While the second type occurs on codes lacking a useful property, the one based on bolases seem to arise when we try on the contrary to design good codes, thus hinting that it can actually occur on every code. This does not seem compatible with good performances, but such error pattern will most likely be minority for small to medium codes. Instead, we expect

the dominating uncorrectable error patterns to have their odd diminish with the blocklength. Because of that reason, the performance of the LP decoder on a code family for which there exists constant weight uncorrectable error could still exhibit good performance on codes with a reasonable size, and even be compatible with a threshold. For this reason, despite proving several negative results for the LP decoder applied to some code families, it is still interesting to look at the actual performance of the decoder through simulations, which is the goal of the next section.

4.5.1 Limitation on codes with poor soundness

The goal of this section is to generalize the uncorrectable error seen on the toric code in figure 4.1. The idea behind the example of figure 4.1 is that any word (in this case the length 5 error-string) can be brought back to the fundamental polytope by setting for each unsatisfied check a generator in its neighborhood to 0.5. This is always possible as long as every qubit belongs to at least one generator, which would otherwise mean that some errors on a single qubit are logical errors. We can thus create for any minimal error of syndrome s a fractional point within the fundamental polytope whose distance to the error is at most $\frac{w|s|}{2}$, where w is an upper bound on the generators weight. Notice that the distance from the minimal error to this fractional point is independent of the error's weight, so it can be theoretically smaller than the error's weight and thus guaranteeing that the error is uncorrectable.

Theorem 55 (Failure for errors with small syndrome)**.** *Given a CSS code with generators of weight at most w such that each qubit belongs to at least one generator. Consider an error with syndrome s, and let e be an equivalent error with minimal weight. If $|e| > \frac{1}{2}w|s|$, then this error cannot be corrected by the QLPD.*

Proof. We focus on the part of the code correcting X-type errors, thus only considering X-type errors and Z-type generators. Given a syndrome s, we design the candidate fractional word e_{fract} which has a weight smaller than $|e|$, and then prove that it belongs indeed to the syndrome polytope.

Let $G = \{g_1, ..., g_{|s|}\}$ be a multiset of generators such that:

$$\forall j \in [m], s_j = 1 \Rightarrow c_j \in \Gamma(g_j).$$

This can be done whenever each bit belongs to at least one generator, as we allow one generator to appear several times.
Then e_{fract} is defined by:

$$e_{fract}(i) = \begin{cases} 0.5 \text{ if } \exists g \in G \text{ s.t. } v_i \in g \\ 0 \text{ otherwise.} \end{cases}$$

e_{fract} may not have the support of a degenerated error if some generators of G overlap, but it always has the property that any check has either no neighbor set to 0.5 or at least 2 (it cannot have just one as elements of G are generators and must commute with each check).

It is clear that if w is the weight of the largest generator then the weight of e_{fract} cannot exceed $0.5w|s|$, and reaches it only if the chosen generators do not overlap and have the maximum weight.

To show that e_{fract} is indeed in the syndrome polytope we have to give for each type of check its part of the valid configuration, which have to fulfil the 3 conditions of the lemma 23 (which can the easily be checked by inspection as in figure 4.1). We use the same notations as in lemma 23, and give for every satisfied (resp. unsatisfied) check of e an explanation of its neighborhood in e_{fract} as a convex combination of integral errors with support of even (resp.

odd) cardinal. To simplify this, we differentiate between different types of checks, characterized by whether they are satisfied in e and what kind of neighborhood they have in e_{fract}. Moreover, we define for each check c the set $S_c := \{i \in \Gamma(c) : e_{fract}(i) = 0.5\}$.

Type 1: a satisfied check with $S = \varnothing$.
In this case, we chose $w_{c,\varnothing} = 1$.

Type 2: a satisfied check with S having an even cardinal greater or equal to 2.
In this case, we choose $w_{c,\varnothing} = w_{c,S} = 0.5$.

Type 3: a satisfied check with S having an odd cardinal greater or equal to 3.
In this case, we chose for all $i \in S$ $w_{c,S\setminus\{i\}} = \frac{0.5}{[|S|-1]}$ and $w_{c,\varnothing} = 1 - |S|\frac{0.5}{|S|-1}$.

Type 4: an unsatisfied check with S having an even cardinal greater or equal to 2.
In this case, we pick an element $i \in S$ and choose $w_{c,\{i\}} = w_{c,S\setminus\{i\}} = 0.5$.

Type 5: an unsatisfied check with S having an odd cardinal greater or equal to 3.
In this case, we chose $w_{c,\varnothing} = w_{c,S} = 0.5$.

This proves that there is in the syndrome polytope a fractional word e_{fract} having weight smaller than e, so with weight smaller than any error equivalent to the actual error. As the QLPD returns the point of the syndrome polytope with the smallest weight, we are certain that it will not output an error equivalent to the actual error, so it will fail to decode it.

□

This result implies in particular that the `QLPD` can only perform well for CSS quantum codes having the property that a correctable error cannot have a syndrome that has a weight significantly smaller than the weight of the minimal equivalent error. This property of a code is sometimes called soundness [LTZ15, Cam19b].

A famous example of a stabilizer codes family lacking this property are the toric codes, but we already had a glimpse in §4.2.2 of how easy it was to create uncorrectable errors for the LP decoder on this code. Nevertheless, this type of construction only works on codes lacking an interesting property. On the contrary, we will see in §4.3 to §4.5 examples of problematic patterns that can exist on a priori good stabilizer codes.

4.6 Simulations results

While the previous section was dedicated to theoretical properties and asymptotic performance of the LP decoder, this section will be focused on assessing its actual performance on realistic size codes through simulations. In §4.6.1 we will show the result of simulations on `QLPD` and its rounded version `RQLPD`. In §4.6.2 we will compare the performance of `QLPD` with the performance of its ADMM implementation.

4.6.1 Simulations on the LP decoder

In this section, we will estimate the actual decoding performance of the LP decoder on quantum hypergraph product codes by simulations. To do so, we followed the protocol described by algorithm 10 running `RQLPD` in the place of $\texttt{Dec}_1$, and employing either the parity-check matrix H_X of toric codes or more general hypergraph product codes in the place of H. We will thus focus here on the correcting capacity against only-X noise in the noiseless syndrome case.

We decided to opt for the rounded version of the decoder to be able to compare it fairly to previously mentioned decoders, as they are focused on minimizing the WER regardless of theoretical guarantees.

To solve the linear program appearing in both `QLPD` and `RQLPD` we ran a LP solver called Gurobi [Gur22]. This allowed us to perform this step of the decoder with greater efficiency than if had implemented ourselves, at the cost of losing control over the resource consumption of the decoder. This actually led to a very large usage of RAM, which forbid us to employ the same codes as for `BP` + `SSF`, but restricts us to similar codes with a smaller blocklength. Fortunately, reducing the blocklength is not the only way to reduce the size of the linear program. Indeed, we can also run the adaptive cuts implementation of the LP decoder, which transforms the resolution of a linear program, where each check provides many inequality constraints to the resolution of several linear programs, where each check provides at most one inequality constraint. This allowed us to employ larger codes and overall speed up the decoder.

We first assess in figure 4.14 the performance of `RQLPD` on the toric code to verify that they are consistent with the theoretical results of sections 4.5.1 and 4.5. Indeed, we showed that the toric codes exhibit many small uncorrectable errors as they have error strings with small syndromes, but also uncorrectable bolas spawned errors. By looking at the WER for toric codes of different blocklengths we see that contrary to what we would expect in the presence of a threshold, the curves do not cross, and larger codes do not perform better even at very low error rates. This is what we expected because of the existence of so many constant weight errors proved in sections 4.5.1 and 4.5.

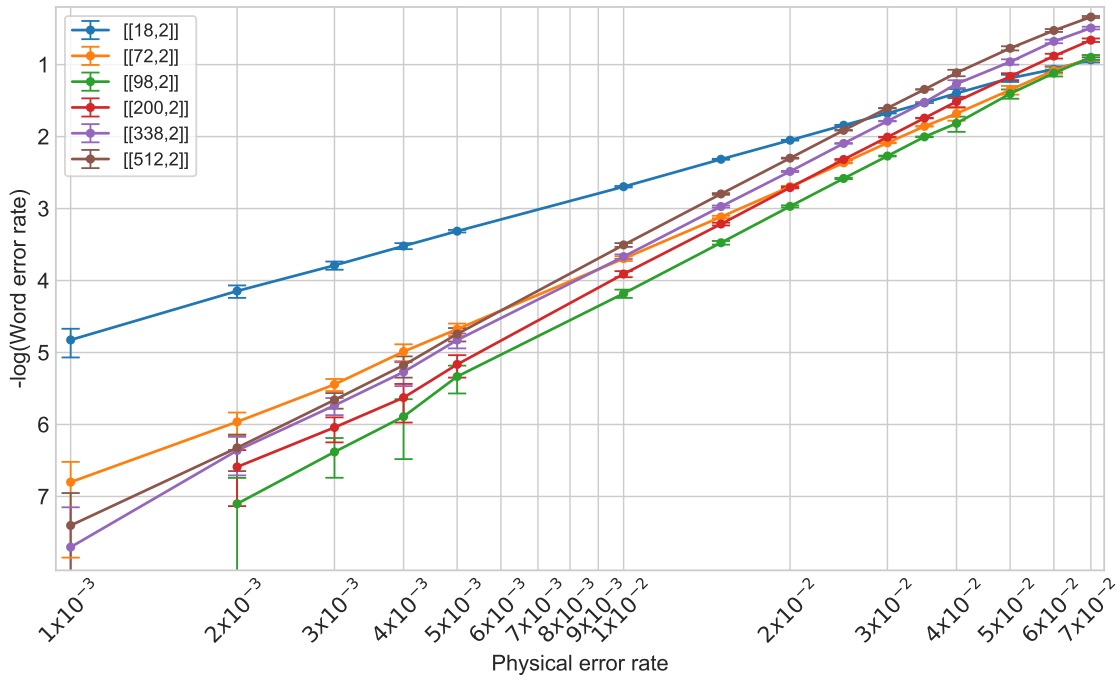


Figure 4.14: We computed the WER for the RQLPD on toric codes against independent X-type errors as noise model. The curves are never in the right order and cross chaotically, which is a sign of a lack of threshold. This is consistent with the theoretical results of sections 4.5.1 and 4.5 which prove QLPD has many uncorrectable constant weight errors; and as expected rounding the fractional solutions of the linear program is not a solution to overcome these errors.

In figures 4.15a and 4.15b we show the results of similar simulations, but on families of hypergraph product codes built from random 3,4-regular LPDC codes. The design of the

codes is detailed in section 7.1, but the main differences are that the codes employed in 4.15b have larger blocklengths, less small cycles than the one of 4.15a and are expected to be quite homogeneous. The curves of both figures cross around 7% of physical error rate, which would be compatible with a threshold in this region. However, on figure 4.15a the code of parameters $[[225, 9]]$ performs better than codes with larger blocklength, while still crossing them around 7%. This does not necessarily mean there is no threshold, since contrary to the toric codes family which is systematic and is made of codes which have the same structure, the codes we study are made from randomly generated classical codes so their quality may not be homogeneous in particular for the one of figure 4.15a.

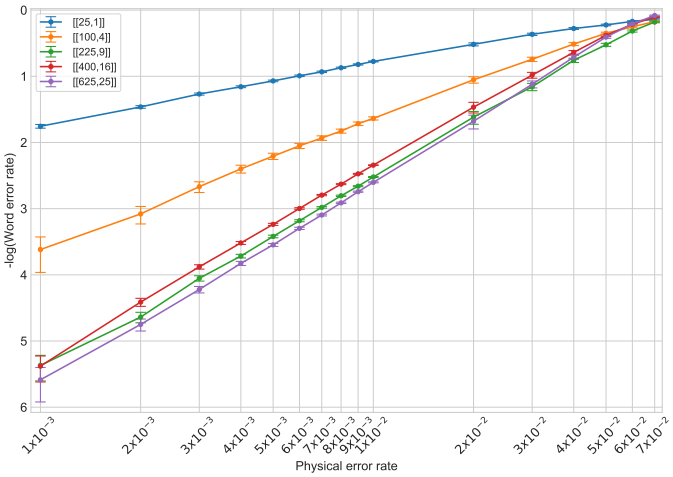
However, while this may not be incompatible with a threshold, it should not either be taken as the guarantee of the existence of a threshold. The first reason is that it is difficult to assess the performance of a decoder for random families of codes, which we further discuss in section 7.1. The second reason is that contrary to BP + SSF for which we had good reasons to believe in the existence of a threshold, since SSF is proven to have a threshold for particular families of hypergraph product code; it is the opposite for RQLPD as we proved results leaning toward the lack of threshold for QLPD on particular families of hypergraph product codes. We think that what these curves show is positive, but only means that in this range of blocklength and physical error rates the performance of RQLPD can be improved by increasing the size of the code;, but that this will not be the case anymore for too small physical error rates and too large blocklengths.

Fortunately, the threshold is not the only figure of interest, and we can also compare the performance of RQLPD with other decoders on the same codes. That is what we did in figure 4.16a for the two extreme codes of figure 4.15a and in figure 4.16b for the smallest code of figure 4.15b. We compared the performance of BP, Iterative BP + SSF, BP + OSD and RQLPD on these codes subject to only-X errors. While the WER is very close for the smallest code of 4.16a, it is clear for the largest code that RQLPD performs better than BP + SSF having even a WER closer to the one of BP + OSD0 while still not performing as well. We will see in section 7.1 that codes with less small cycles greatly improve the performance of BP and RQLPD, while not affecting so clearly the performance of BP + SSF and BP + OSD0. However, RQLPD performs worse in figure 4.16b. Indeed, it only matches the performance of BP + SSF, and is very far from performing as well as BP + OSD. This can be explained partly by the fact that we ran a version of OSD introduced in [RWBC20] called OSD-CS, which is expected to perform better than OSD0. However, this does not explain the fact that RQLPD does not perform better than BP + SSF. This would be consistent with the theoretical results of section 4.5 as we expect such codes to be difficult to correct by RQLPD as they grow larger, however this was not visible in figure 4.15b.

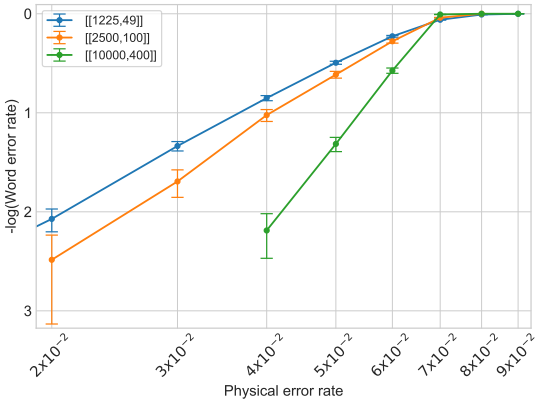
Overall, the LP decoder seems to compete with BP + SSF for medium size codes, while even performing better for small codes;, but falls short to the performance of the BP + OSD decoder. Fortunately, there is an implementation of the LP decoder called ADMM LP decoder which could both improve the performance of the decoder in terms of WER while making it more practical. We will assess the performance of this implementation in the next section for decoding various hypergraph product codes.

4.6.2 Simulations on the ADMM LP decoder

In this section, we study the performance of the ADMM implementation of the LP decoder on several hypergraph product codes. The ADMM implementation transform the resolution of the linear program into running a message passing algorithm over the Tanner graph, just like for BP. However, contrary to BP, the messages involved in this algorithm are guaranteed to converge, allowing to approximate with arbitrary precision the solution of the linear program. Moreover, the number of rounds required to obtain a given average precision does not depend



(a) WER for the `RQLPD` on a family of small random hypergraph product codes subject to only-X noise. All the curves cross around 7% of physical error rate which would be consistent with a threshold in this area, however the code of parameter [[225,9]] performs better than larger codes. This could be because this code family can be poorly homogeneous, since the initial classical codes were both small and randomly generated. Overall, we think these curves are rather hinting towards the existence of a threshold around 7% of physical error rates.

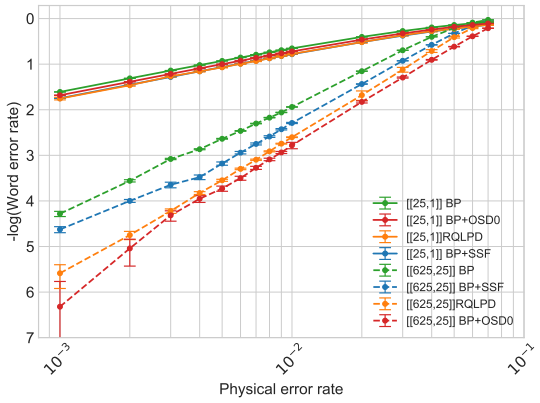


(b) WER for the `RQLPD` on a family of random hypergraph product codes subject to only-X noise. These codes are larger, have less small cycles, and should be more homogeneous than the previous ones. The curves are in the right order and all crossing around 7%, while the one corresponding to the largest code is much steeper. This is consistent with a threshold around 7% of physical error rates.

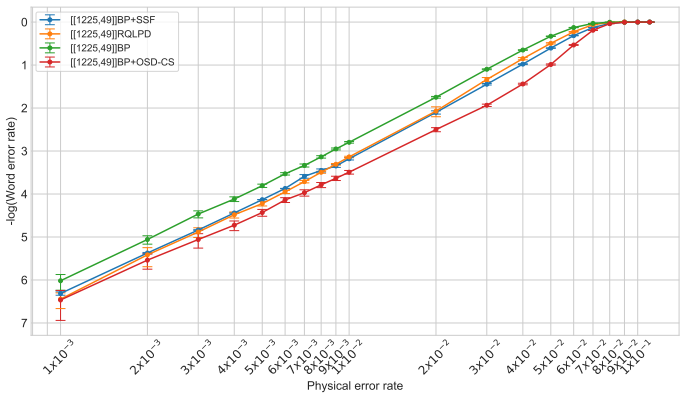
Figure 4.15: WER for codes of two slightly different families of hypergraph product codes. While these figures are rather leaning towards the existence of a threshold at 7%, we believe this may be the case only in this range of physical error rates and blocklength and that the curves would probably be swapped if we increase too much the latter. These results are however positive, as they show we can improve the decoding performance of `RQLPD` by increasing the size of the code for codes of small to medium blocklengths.

on the blocklength, making the overall complexity linear with the blocklength. Unfortunately, this is only true for the precision averaged over every message, and not for the precision of every message. Because of this, the average precision can sometimes be satisfactory while some messages are far from having converged, leading to the approximation of the solution to the LP to have some erroneous coordinates. If we want every message to be precise enough, we need to increase the number of rounds with the blocklength, and thus worsen the complexity. Because of this, we introduce several variants of the ADMM implementation of the LP decoder, or more simply `ADMM`. The first one called `Fast ADMM` stops whenever the average precision is satisfying or when the correction obtained by the current messages returns the noisy word to the code space. The second one called `Slow ADMM` will instead stop only once the precision of every message is satisfying. Similarly to what we did for `QLPD`, we also introduce the rounded variants `RADMM`, `Fast RADMM`, and `Slow RADMM`.

We compare in figure 4.17 the performance of `Slow RADMM` and `Fast RADMM` to the performance of `RQLPD` on a random hypergraph product code of parameters [[1225,49]]. As expected, `Slow RADMM` achieves the same WER as `RQLPD`, since every message had enough time to converge, thus making an excellent approximation of the solution to the LP. What is more interesting is that while `Fast RADMM` does not perform as well, its WER is still very close to that of the slower decoder. Because of this, the fact that the constant time convergence is only for the average precision does not seem to be a problem. Moreover, it is even possible to apply another



(a) Comparison of the WER of various decoders on small random hypergraph product codes built from 3,4-regular classical codes already studied in figure 4.15a. While the smallest code exhibit similar performance for every decoder, we see for the largest one that `RQLPD` performs better than `Iterative` `BP` + `SSF` and almost as well as `BP` + `OSD0`.



(b) Comparison of the WER of various decoders on a small random hypergraph product codes built from a 3,4-regular classical code already studied in figure 4.15b. Contrary to what we would expect from the results of 4.16a and the fact that this code is expected to favor `BP` and `RQLPD` over `BP` + `SSF` and `BP` + `OSD`, `RQLPD` performs only as well as `BP` + `SSF`. While the relatively large gap with `BPOSD` can be explained by the fact that we ran a variant slightly better than `BP` + `OSD0`, it cannot explain the fact that `RQLPD` does not perform better than `BP` + `SSF`.

Figure 4.16: Comparison of the decoding performance of several decoders on hypergraph product codes.

stopping condition to even further reduce the number of rounds. This decoder can thus run in time linear with the blocklength, but it can be even run in parallel in constant time at the cost of having the number of processors proportional to the blocklength.

Being able to change the resolution of the linear program into a message passing algorithm is already of import, as it greatly reduces the complexity of the decoder and allows solving it in parallel. Nevertheless, this is not the only interest of this implementation, since it also allows adding non-linear terms to the objective function. Indeed, the objective function of a linear program has to be linear, and most LP solving algorithms will not function otherwise. However, this is not the case of the ADMM as it allows adding non-linear terms to the objective function, which can be employed to penalize the fractional solutions and thus lower the chance of reaching a pseudocodeword. Yet, this is not a magical solution as we lose the convergence guarantees we had before, in particular when trying to penalize pseudocodewords too much.

We tried the different penalty functions proposed in [LD16], but did not manage to sensibly improve over the performance without penalties. However, when trying it on an hypergraph product code of parameters [[1922,50]] based on a cyclic code we managed to have a great improvement of the WER at high physical error rates as it can be seen on figure 4.18. Because of a lack of time, we employed parameters identical for every physical error rates, while the original paper insisted that the optimal parameters usually vary with the sound to noise ratio. By carefully choosing the parameters, `Fast Penalized ADMM` should never perform worse than the `Fast ADMM`, and sometimes greatly improving the WER. The reason the penalized version works much better with this code is not clear, but could either stem from the fact that the penalty tackles particularly well errors mostly present in the [[1922,50]] code or because the optimal parameters for the [[1225,49]] code were out of the range of parameters we tried.

ADMM thus seems like an interesting option to improve the performance of the LP decoder

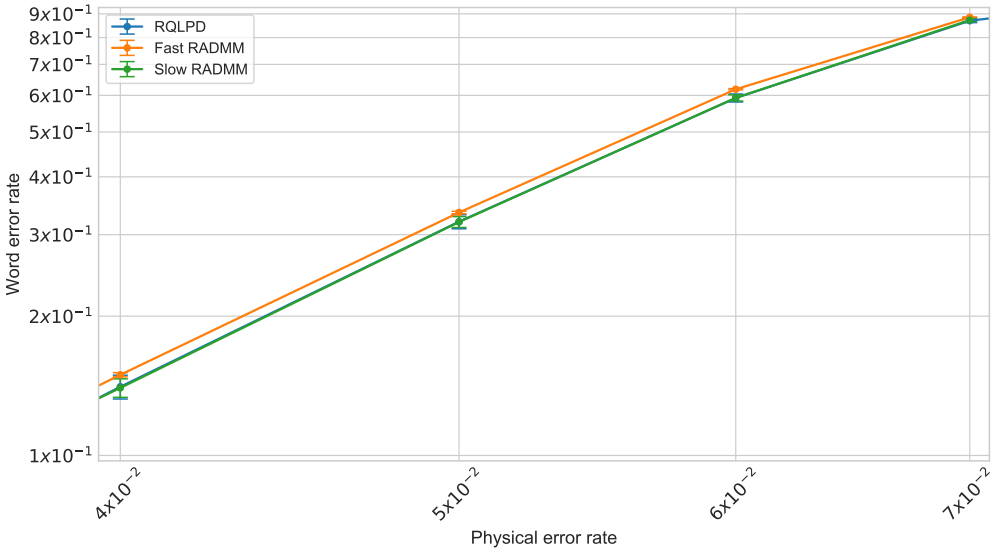


Figure 4.17: Comparing the WER of random [[1225,49]] hypergraph product code built from a 3,4-regular classical code already studied in figures 4.15b and 4.16b corrected with `RQLPD`, Fast RADMM or Slow RADMM. While only Slow RADMM performs as well as `RQLPD`, Fast RADMM has very close performance, making it a suitable choice as it has a better computation time.

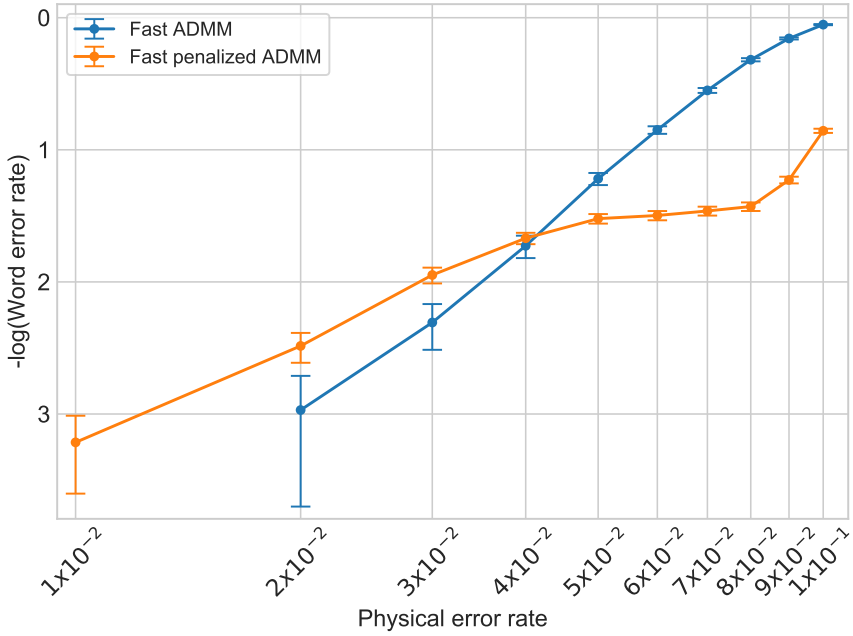


Figure 4.18: WER of the binary LP decoder with the ADMM and the penalized ADMM implementation on hypergraph product code of parameters [[1922,50]] obtained from a cyclic code 3,3-regular used in [PK21a]. The penalized version shows great improvement at the higher physical error rates, which is encouraging. The fact that it performs worse than the ADMM at lower physical error rates comes from the fact that we chose the same parameters for the whole range of physical error rates, whereas the optimal parameters depend on the sound to noise ratio.

both in terms of WER and of computation time. Yet, the results seem highly dependent upon the code, and it can in some occasions perform worse than the usual LP decoder, but with the

possibility to trade the computation time off for a better WER.

4.7 Conclusion of the chapter

Despite a theoretical analysis proving the existence of problematic errors threatening the correcting capacity of the LP decoder on some hypergraph product codes, the decoder actually performed quite well on small to medium size random hypergraph product codes. While we still believe that the performance of this decoder will in most cases be asymptotically bad, it does not prevent it from being competitive for codes with realistic sizes. Moreover, it comes with the ADMM implementation allowing to compute the correction through message passing. Thus, it is as practical as `BP` while having the same performance as `BP` + `SSF`. The possibility to penalize pseudocodewords has shown good results for some codes, but the improvement seems very variable and code dependent.

Chapter 5

The parallel small-set-flip decoder

All the decoders we have seen until then have a computation time that increases with the blocklength, which can be problematic for actual fault-tolerant computation [ABO97]. Indeed, schemes such as code concatenation typically require a decoding time constant with the blocklength, which is difficult to achieve. Moreover, quantum decoders usually need the whole syndrome to compute a correction for each qubit. Information from each generator must thus be gathered in one processor every time we want to perform an error correction. At the same time, such error corrections must be performed as often as possible to prevent the error to grow out of control. It would then require a considerable flow of information that can be difficult to achieve [Del20].

To overcome this problem, we can instead employ local decoding algorithms, which allow the correction applied to each qubit to be computed by a local processor with access to a small part of the syndrome. This has already been done several times for topological codes [HCEK15,KP19,BDMT17a]. In this setting, the local decoder is made of cells acting in parallel. Each cell is located somewhere on the code and can act on the neighboring qubits, access the neighboring generators, but also communicate with the neighboring cells.

This definition of a local decoder, however, is only convenient for topological codes for which the physical implementation of the code itself has a geometrical structure. In a more general case, the notion of distance between the different qubits or generators does not make sense. Instead, we propose another definition based on the number of syndrome bits on which the correction of a qubit depends. A decoder is said local if this number does not depend on the blocklength. Such decoders will have lower correction capacities than the non-local decoders. In particular, they will not be able to decode most errors in a single round, but would instead decrease it little by little and need several rounds of decoding to correct it entirely. We will thus focus on their capacity to reduce the error, which is characterized by the ratio between the weight of the error after and before the correction.

Creating a local decoder able to reduce many errors with a good factor is not trivial. However, Grospellier showed in its PhD thesis [Gro19a] that a constant time variant of `SSF` met these two conditions. Indeed, when applied to expander codes, it was both local and able to reduce every error of weight smaller than a variable scaling with the square root of the blocklength. This property relied a lot on the good expansion of the code, which allows small errors to have relatively large syndrome weights. On the contrary, the toric code is well known for having small errors able to grow while preserving a small syndrome weight. For this reason, it seems unlikely that a decoder having similar properties could be created for the toric code. We manage indeed to prove that no decoder could have strictly the same properties, but it is still possible to perform local error correction with a different approach.

In section 5.1 we make explicit what we mean by local decoder, and give other existing definitions together with examples of such decoders. In particular, we will present the theoretical

guarantees of the parallel SSF decoder. In section 5.2 we prove that no decoder on the toric code could have the same guarantees. In section 5.3, we make explicit the exact version of the parallel decoder that we will study. Finally, in section 5.4 we show the result of simulations done on the parallel SSF decoder.

5.1 Local decoders

The decoders exhibiting the best performance in terms of WER and threshold often require a global view of the syndrome for computing each correction. For example, it is the case for the MWPM [Edm65] on the toric code and the BP+OSD [PK21a] on general hypergraph product codes. Some local variants have, however, been already studied, in particular for topological codes. On such codes, the local decoders take the form of cellular decoders made of many cells located on the lattice. These cells can only have access to the value of the neighboring generators, but can also communicate classically with the neighboring cells. Their behavior is thus purely local, which, however, allows information to travel across the lattice at a very slow pace, whereas a non-local decoder would allow it to travel much quicker.

One such example we will look at is the Harrington's decoder [Har04, BVC+17] which relies on cells of different dimensions. The lattice is tiled with small cells which have access to the neighboring generators and qubits, but also able to communicate with the neighboring cells. We can then add several layers of virtual cells, each one obtained by tiling the previous layer with larger and fewer cells. These cells will act like the physical ones, but their communication will be much slower because implemented by the physical cells. This algorithm allows decoding the toric code with almost constant resources, only a clock scaling logarithmically with the blocklength is required for the larger virtual cells.

In the case of a general LDPC quantum code, the actual position of the qubits is not known and would depend on the physical implementation. We thus cannot exploit the position of the qubits to define a local decoder. Instead, we will require a local decoder to be able to compute the correction of each qubit with the values of only a constant number of generators.

Definition 56. *A decoding algorithm $d : \{0,1\}^m \to \{0,1\}^n$ is called* α-local *if sets $T_i \subseteq [m]$ exist for each $i \in [n]$ with $|T_i| \leq \alpha$ and functions $d_i : \{0,1\}^{T_i} \to \{0,1\}$ such that the following holds. For any $s \in \{0,1\}^m$ and $i \in [n]$, the i-th bit of $d(s)$ is given by $d_i(s_{|T_i})$. Here, $s_{|T_i} \in \{0,1\}^{T_i}$ denotes the substring of s indexed by the elements in T_i.*

The idea behind this definition is that each qubit could be attributed a dedicated processor wired to a finite number of generators and able to compute extremely efficiently the correction of the qubit. It is clear that since the size of the input is independent of the blocklength, each decoder computes the correction in a constant time. Notice however that less than a processor may be required for each qubit, with one processor computing the correction of several qubits.

Furthermore, we see that the Harrington's decoder is not local according to definition 56, since the largest virtual cells have access to a number of qubits proportional to the blocklength. This, however, does not mean it is unpractical for large-scale computers, as this definition of local does not cover every realistic decoder. Such local decoders will often have lower performance in terms of WER than the ones which have access to the whole syndrome. Still, they are actually not required to correct the whole error, but rather contain it for long enough. Indeed, at the end of the computation we can apply a slower and non-local decoder which has a better WER, similarly to what we do for fault-tolerant simulations. Because of this, we are interested in the capacity of a decoder to decrease the minimal size of an error, that is, the size of the smallest equivalent error. We call this property error-reducing.

Definition 57. *For $\gamma \in [0,1)$, a decoder d is said to be* γ-error-reducing *for a family of errors $\mathcal{N} \subseteq \{0,1\}^n$ if, for any $e \in \mathcal{N}$, we have $|e + d(He)|_{\min} \leq \gamma |e|$. When the syndrome is noisy, we say that d is* (γ, c)-error-reducing *for some $c \geq 0$ and a family of errors $\mathcal{M} \subseteq \{0,1\}^n \times \{0,1\}^m$ if for any $(e,f) \in \mathcal{M}$, we have $|e + d(He + f)|_{\min} \leq \gamma |e| + c|f|$.*

While these definitions of local decoders 56 and error reduction 57 do not perfectly translate the requirements for a decoder to function in realistic conditions, they are a useful proxy for codes lacking topological properties. Indeed, having a decoder able to reduce every small error by a large enough factor would allow us to restrain the size of the error for the whole computation given a small enough physical error rate. However, creating a decoder being both local and error-reducing for a large class of error with a good coefficient is not easy, and their mere existence is not trivial. Somewhat unexpectedly, a parallel version of `SSF` has been proven to meet those two conditions. Indeed, in his PhD thesis [Gro19b], Grospellier studied a constant time decoder made of a large (but constant) number of rounds, where several generators were corrected in parallel. He was then able to show that such a decoder applied to an expander code could reduce any error of weight smaller than a variable scaling with the square root of the blocklength.

This parallel version of `SSF` has not only a constant time of computing, but is also local according to definition 56. Indeed, the number of checks on which the correction of any qubit depends is upper bounded by a value depending solely on the weight of the generators, the degrees of the qubits and the number of rounds of parallel `SSF`. Since all these values do not depend on the blocklength, each qubit can thus be corrected by looking at only a constant number of syndrome bits. However, while this number is constant, it may still be huge. To cope with this problem we will study in section 5.4 the correcting capacity of the parallel `SSF` with only a few number of parallel rounds on hypergraph product code of medium size which we present in section 5.3.

As we said, it is difficult to design a decoder being both local and able to reduce errors of any weight, given that we take a blocklength large enough. This may actually not even be possible for every code. Indeed, the parallel `SSF` is proven to reduce small errors only for codes having a large enough expansion. This expansion guarantees the small errors to have a large enough syndrome and thus that the code itself has an excellent soundness. On the contrary, the toric code is known to have a deplorable soundness, with long error strings having a constant syndrome weight. We thus expect local decoders not to be able to reduce every constant weight error, even by a poor factor. This in indeed what we will prove in the next section, by exhibiting on the toric code a class of constant weight errors having for any decoder a constant proportion of unreduced errors. However, we will see that this result is far insufficient to deduce any no go result on scalable decoders for the toric code.

5.2 A no-go theorem for the toric code

It seems a priori complicated for a decoder on the toric code to be both local according to definition 56 and be able to reduce (even slightly) every constant weight errors. The reason is that local decoders must take decisions having only access to a small portion of the syndrome, yet the parts of the syndrome detecting the error can already be minuscule. Typically, error strings are detected by only two syndrome bits, and thus seem like good candidates to prove for any local decoder the existence of constant weight unreduced errors. This section will thus be focused on proving this result, which would then guarantee that no decoder applied to the toric code can have the properties proved by Grospellier [Gro19a] for the local SSF. However, the goal of this theorem is rather to highlight the rarity of decoders having such a property rather than deduce a negative result on the local decoding of toric codes. Our main result is

summarized in the following theorem.

Theorem 58. *Let α be a positive number, then three positive constants c_1, c_2 and c_3 exist such that any α-local deterministic decoder acting on the toric code of blocklength n will not be able to reduce at least $c_1 n - c_2\sqrt{n}$ errors of weight at most c_3.*

This theorem tells us that a local decoder for the toric code will not reduce some errors of constant weight, meaning that increasing the blocklength of the code will not prevent some small errors not to be reduced. On the contrary, the parallel version of SSF applied to expander codes is local and reduces every error of weight lower than some value proportional to the square root of the blocklength, thus all the errors of a given weight will eventually be reduced if the blocklength is great enough.

While we have demonstrated that no decoder on the toric code could have strictly the same properties as the parallel `SSF`, it could still have an almost identical guarantee. Indeed, the problem here is that while we proved that a constant fraction of the errors in E_γ, the probability for an i.i.d error to belong to E_γ exponentially diminishes with the blocklength. Indeed, such an error requires every qubits outside a given row to be exact, which happens independently for each with probability $1 - p_{phys}$, where p_{phys} is the physical error rate. Usually, when we try to exhibit a large family of uncorrectable error, it is enough to find a small family of constant weight uncorrectable error. Indeed, an error will often stay uncorrectable after we modify her action on qubits far from the support of the error. Because of this, we can greatly increase the size of the uncorrectable errors family, by adding every error that is equal to the original uncorrectable errors only inside a small bubble around its support. Having an initial family of constant weight errors guarantees then that the probability for an error to belong to the increased family stays constant with the blocklength. Unfortunately, this method does not work anymore in the case, where we want to prove that the error cannot be reduced. Indeed, while an incorrect decoding of some qubits will not generally be rectified by a perfect correction of other qubits, a poor reducing of some part of an error can be compensated by a perfect reduction in another area. For this reason, we can create a reduced error by combining a small error that is not reduced with a larger error acting on different qubits and very well reduced.

While we managed indeed to exhibit errors of constant weight that cannot be simultaneously reduced by a local decoder on the toric code, finding a large family of such errors is not so simple. Nevertheless, we think that larger families than the one we exhibit should exist, but probably more complex and requiring a more detailed analysis. This partially shows how unique are the properties of the parallel `SSF` on hypergraph product codes.

5.3 The parallel SSF decoder

In this section, we propose an algorithm for deciding which generators will be corrected by the decoder each decoding round. We transform this problem into the coloring of a graph obtained from the Tanner graph.

To make `SSF` run in parallel, it is important to check that the generators corrected at the same time are compatible. It is clear that compatible generators must not overlap, as it could require a qubit to receive different corrections at the same time. Moreover, we want the corrections proposed for two different generators to be consistent, in the sense that their corrections should not try to satisfy the same check by flipping different qubits. Since the processors computing the correction cannot communicate, one way to ensure this is to require compatible generators not to see the same checks. Looking at the hypergraph product, two generator nodes are compatible if and only if they are at distance more than 4 from each others. We can thus build a graph whose nodes are generators and, where two generators are linked by an edge if and only if they are incompatible.

Using this graph, we will be able to choose which generators shall be corrected during a given round of error correction by coloring as many nodes as possible so that no edge links two colored nodes (most generators will still be left uncolored). In doing so, all the colored generators can be safely corrected simultaneously, and we just have to do one coloring per round of error correction. Since we want to avoid correcting always the same generators, we make the coloring of one round depend on the previous ones. We thus decide to keep a log of which generators have been the most recently corrected in the form of a queue: the more recently a decoder has been corrected the further it is in the queue. Coloring a generator for a given round would thus consist into trying to color each generator in the order given by the queue, and updating the queue by putting every generator we color at the end. The coloring algorithm would then output both the coloring for this round and the queue for the coloring of the next round. We describe this method in algorithm 13.

Algorithm 13: Generators coloring

Input: T the number of rounds,G the list of generators, I a list of sets with I_g the set of generators incompatible with generator g
Output: C a list of sets with C_i the set of generators corrected at round i

```
C ← list of T empty lists;
Q ← G;
i ← 1;
while c ≤ T do
    for g in Q do
        if g in C_i then
            break;
        end
        if I_g ∩ C_i = ∅ then
            move g at the end of Q;
            append g to C_i;
        end
    end
    i ← i + 1;
end
return C
```

It is important to note that the coloring does not depend at all on the syndrome, so it can be done in advance. Each processor will thus know at each round whether it must correct its associated generator. If the number of error correcting rounds exceeds the number of coloring computed in advance, we can simply loop from the first coloring. Since the transition between the last coloring and the first coloring may not be as smooth as between successive coloring, the more rounds are colored, the better the decoder should perform.

Applying this algorithm to the hypergraph product code of parameters [[2500,100]] we notice that only around 350 qubits can receive a correction during one round, meaning that we would need about 7 rounds to look at every qubit. This could be problematic, in particular as we want to look at the extreme case, where only one round is done in a row. To increase the correction capacity of the decoder, but keeping it parallel and local, we propose to complete the `SSF` correction by a `Flip` correction for some compatible qubits. Once again, we mean by compatible that it must not share a check with another qubit being potentially corrected during this round. As for the generators, we can build the graph having qubits for nodes and, where two nodes are linked if the corresponding qubits are incompatible. This can once again be done from the hypergraph product, where two qubits are not compatible iff they are at distance at

most two from each others. Similarly to what we did for the coloring of the generators, we relied on a queue to decide in which order we try to color the qubits, but also consider which qubits will be corrected this round by `SSF`. The algorithm will browse the queue trying to color each qubit in order. In the case, where the qubit is already to be examined by `SSF`, we put it at the end of the queue without coloring it. If it is not in the list, we color it if possible and move it to the end of the queue.

Completing the `SSF` decoder with `Flip` allows us to double the number of qubits which can be corrected each round. Through simulations, we will see in the next section that it also allows us to better limit the increase in the error's weight.

Algorithm 14: Qubits coloring

Input: T the number of rounds,B the list of qubits, S a list of sets with S_i the set of qubits potentially corrected by `SSF` during round i, I a list of sets with I_b the set of qubits incompatible with qubit b

Output: C a list of sets with C_i the set of qubits corrected at round i

```
C ← list of T empty lists;
Q ← B;
i ← 1;
while c ≤ T do
    for b in Q do
        if b in C_i then
            break;
        end
        if b in S_i then
            move b at the end of Q;
        end
        else if I_b ∩ C_i = ∅ then
            move b at the end of Q;
            append b to C_i;
        end
    end
    i ← i + 1;
end
return C
```

5.4 Our simulation results

In this section, we will assess the performance of the parallel `SSF`$-$`Flip` decoder by simulations. We will in particular compare its performance to a toy decoder based on MWPM. Contrary to `BP` $+$ `SSF` or `QLPD`, the parallel version of `SSF` is not expected to correct most errors, even the small ones, but is only guaranteed to reduce them given that the code has a good enough expansion. Moreover, here we look at an extreme version of the parallel `SSF`, where only one round of correction is applied for each syndrome measured, making the number of qubits potentially corrected only a fraction of all the qubits, even when we correct some of them by `Flip`. However, this comes with the benefit of being able to process these corrections much faster than with the other decoders. To assess the performance of this decoder, we will thus apply a noise channel with a much lower error rate, trying to protect the information for as many round as possible. Similarly to what we do in the fault-tolerant case, after several of

error correction (with or without syndrome noise) with the local decoder we do a last correction round with a slower non-local decoder having good WER performance to assess whether the information has been protected. We will typically run $\texttt{OSD}_0$, which offers good performance while benefiting from a practical computation time.

In figure 5.1 we first focus on the capacity of the decoder to limit the growth of the error depending on whether only a parallel `SSF` is run or a parallel $\texttt{SSF} - \texttt{Flip}$. Their performances are compared for the [[2500,100]] hypergraph product code, which does not perform as well with `SSF` as the [[6100,100]] code. However, it allows faster computations, so will be chosen for these preliminary simulations alone. While the two decoders have similar performance, combining `SSF` with `Flip` does seem to allow a better control over the error growth. It should be noted that the curves have a rather erratic behavior, which, we think, is because we look directly at the final error rate. For this reason, we will now rather look at whether the information is recoverable at the end of the rounds. To estimate this, we will run a non-local decoder at the end of the simulations to try recovering the initial information. Another possible approach is to look at the number of rounds we can perform before losing the information, as done for the Harrington decoder. While arguably more pertinent, it is much more time-consuming, as it requires simulating after each round a correction by the non-local decoder. One of the interest of looking at the mean time for which the information is protected is that there is no need to decide an arbitrary threshold for the WER, above which we say the number of rounds is impractical.

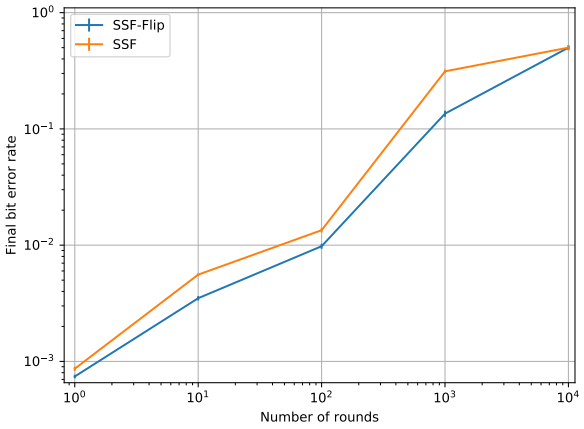


Figure 5.1: Final bit error rate of the [[2500,100]] hypergraph product code protected for several rounds by either the parallel `SSF` alone or by the parallel $\texttt{SSF} - \texttt{Flip}$. The physical error rate of the noise added each round is 10^{-4} The fact that the second decoder can act on more qubits each round seems beneficial despite that part of the correction is proposed by `Flip` which is not adapted to the quantum setting.

In figure 5.2 we look at the performance of the parallel $\texttt{SSF} - \texttt{Flip}$ on the [[6100,100]] hypergraph product code subject to various levels of syndrome noise. While looking at the noiseless syndrome case is interesting to have an idea of the potential of the decoder, it is not a realistic setting, thus the need to look also at the noisy syndrome case. We are working in conditions where the qubit noise is extremely low because correction rounds are done very frequently. However, this does not impact that much the syndrome error rate, which is thus most likely several times larger. For this reason, we not only look at the usual noisy syndrome case, where each syndrome bit has the same error rate as the qubits, but also the case, where its error rate is ten times greater. Surprisingly, we see that the syndrome noise does not impact too much the WER, the information being protected in each case for at least 10^5 rounds. Even more surprisingly, a high syndrome noise seems to decrease the WER when the number of

rounds gets considerable, however we do not know how to explain nor interpret it.

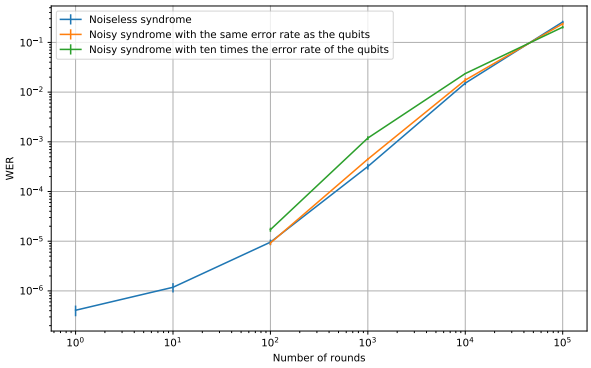


Figure 5.2: Word error rate of the [[6100,100]] hypergraph product code protected for several rounds by the parallel $\texttt{SSF} - \texttt{Flip}$ against only-X noise of rate 10^{-4} and various levels of syndrome noise. We end each simulation by one last round without syndrome noise corrected by $\texttt{BP} + \texttt{OSD} - 0$ to assess whether the information was properly protected. Interestingly enough, the syndrome noise does not seem to penalize too much the WER, and even lowers it when the number of rounds gets larger than the inverse of the noise rate. At this physical error rate, the parallel $\texttt{SSF} - \texttt{Flip}$ can protect the information for at least 10^5 rounds.

Until now, we focused on the most extreme version of the parallel $\texttt{SSF}-$ decoder, where only one color is corrected each round. In figure 5.3 we look at the impact on the WER of an increase in the number of colors corrected in a row. To do so we employed the code of parameters [[6100,100]] without syndrome noise and the rate of the only-X error channel equal to 10^{-4}. As expected, increasing the number of colors corrected per round reduces the WER. This offers a trade-off between computation time and decoding performance, which allows adapting the decoder to the error channel. The ideal number of colors corrected in a row will thus most likely depend on the properties of the physical implementation.

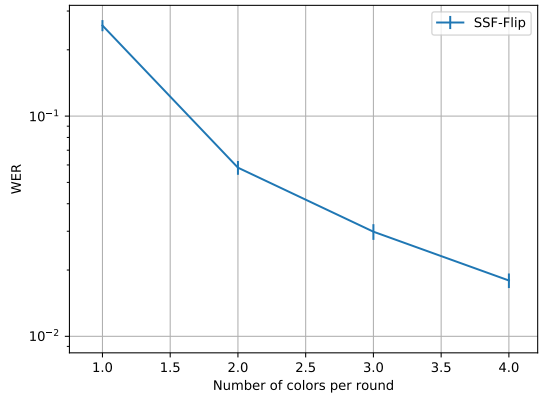


Figure 5.3: Word error rate of the [[6100,100]] hypergraph product code protected for 10^5 rounds by the parallel $\texttt{SSF} - \texttt{Flip}$ against only-X noise of rate 10^{-4}. Here, we vary the number of colors corrected each round, meaning the number of serial rounds of decoding we apply for one syndrome measurement. As expected, the WER decreases as we increase the number of color, but at the cost of increasing the decoding time. This trade off between computation time and accuracy allows adapting the decoder to the properties of the noise channel by finding the optimal number of colors to correct each round.

To assess the performance of the decoder, we decided to compare them to a toy decoder

based on MWPM. The idea was to create a local decoder acting on a code of blocklength similar to ours, and also able to see only the same number of syndrome bits at the same time. To turn MWPM local, we simply tiled the lattice with constant size rectangles, which are each corrected by MWPM. The tiling is shifted vertically by half a tile each round and horizontally every two rounds in order for consecutive corrections to be as complementary as possible. The parallel SSF can see $d_c d_v$ syndrome bits at once, where d_c (resp. d_v) is the check degree (resp. the bit degree) of the classical code used as the basis of the hypergraph product. Thus, we chose tiles of dimension $d_c \times d_v$ for the local MWPM to see as many syndrome bits at the same time as the parallel $\texttt{SSF} - \texttt{Flip}$. To compare to the performance of the parallel $\texttt{SSF} - \texttt{Flip}$ decoder on the [[6100,100]] hypergraph product code made from a 5,6-regular classical code, we thus ran the local MWPM decoder on the 50×60 toric code with parameters [[6000,1]] and with tiles of dimension 5×6.

We show the results of those simulations in figure 5.4, where the codes are subject to only-X noise of rate 10^{-4}, but no syndrome noise. While the toy decoder performs better at low error rates, it comes only from the fact that the decoder employed for the last step is the usual MWPM, which performs better than the $\texttt{BP} - \texttt{OSD0}$ ran after $\texttt{SSF} - \texttt{Flip}$. Indeed, the local MWPM manages to protect the information during at most 10 times fewer rounds than the parallel $\texttt{SSF} - \texttt{Flip}$. Moreover, it performs on a code encoding much less logical qubits than the hypergraph product code: the overhead is 100 times smaller for the hypergraph product code. When comparing its performance to the Harrington decoder which also performs on toric codes the difference is even larger, as the Harrington decoder can protect the qubits for much longer, but also against more noise [BDMT17b].

Similarly, the Harrington decoder seems to perform also much better than $\texttt{SSF} - \texttt{Flip}$ in particular because it can protect against a much higher error rate. However, the comparison is difficult as the codes on which they perform have highly unequal encoding rates. However, the simulation results on the $\texttt{SSF} - \texttt{Flip}$ are promising as they show that such a decoder can work even on codes of realistic size, and by decreasing to the minimum the number of colors corrected each round. The obtained decoder thus requires processes to access a much more practical number of generators, while acting on a more general LDPC code still benefiting from a constant encoding rate.

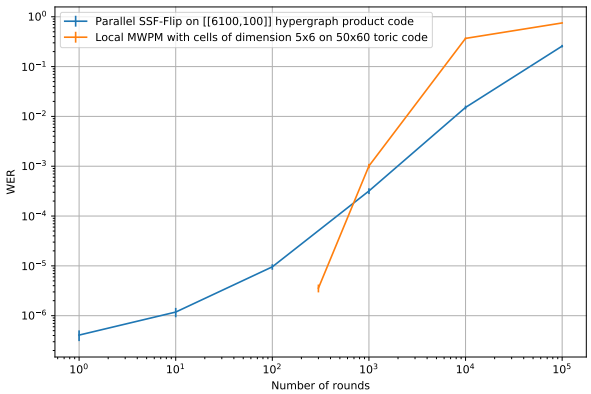


Figure 5.4: Word error rate of a toric code and a hypergraph product code, protected for several rounds by the local MWPM and the $\texttt{SSF}-\texttt{Flip}$ against only-X noise of rate 10^{-4}. Each decoder acts on codes with similar blocklengths (6100 and 6000) with local decoders having access to the same number of qubits at the same time. The performance of the local MWPM may seem impressive given that it is a toy decoder which performs almost as well as $\texttt{SSF}-\texttt{Flip}$. However, MWPM protects a code encoding only one logical qubit instead of 100, and it performs very poorly compared to the Harrington decoder able to protect the information much longer and against a higher physical error rate.

5.5 Conclusion of the chapter

We presented an alternative definition of a local decoder relevant for non-topological codes and translating requirements for a scalable decoder. Such decoders will often exhibit worse decoding performance than their non-local counterparts. However, they do not actually need to entirely correct the error, but rather be error-reducing. While creating a local decoder error-reducing enough is complicated, it was proven for a variant of `SSF` on expander codes. We showed that no decoders could have the same properties as the toric code, but this actually did not imply any practical negative result. Finally, we made the `SSF` variant as simple and fast as possible at the cost of losing the theoretical guarantees. However, we managed to obtain good results on hypergraph product codes of realistic size, thus opening the way to scalable decoders on constant rate quantum codes.

It could be interesting to try allowing each processor to communicate with a constant number of other processors, similarly to what is allowed for cellular decoders on topological codes [HCEK15,KP19,BDMT17a]. In doing so, we could implement message passing algorithms in a local way, without ever aggregating the whole syndrome on the same processor. The most known message passing decoder–`BP`–does not perform well in the quantum setting. However, we saw in chapter 4 that the ADMM adaptation [BLDR13] of the LP decoder exhibit good performance while requiring a constant number of rounds to converge to arbitrary precision. This makes it the ideal candidate to expand the definition of local decoder 56 with limited classical communication between processors.

Chapter 6

Conclusion

In this thesis, we studied the decoding of LDPC quantum codes and in particular of hypergraph product codes using different decoders. The first decoder was obtained by combining the `BP` and `SSF` decoders, and improved performance on hypergraph product codes compared to each individually. While this first decoder was solely focused on the performance, the next decoder– the LP decoder–also allowed theoretical analysis. We thus managed to exhibit several small uncorrectable errors that would most surely worsen its asymptotic performance. Nevertheless, it performed surprisingly well on hypergraph product codes of medium blocklength, even improving over the performance of `BP` + `SSF`. Finally, we focused on more practical problems raised by decoders by studying a parallel version of `SSF` having the property of being local.

While we mostly focused on hypergraph product codes as their simple construction allow easier analysis, it would be interesting to perform simulations with the same decoders, but on the good families of LDPC quantum codes introduced in [PK22]. We could hope in particular that it will improve the decoding performance of the two constant time decoders. Another interesting lead would be to make better local decoders by applying schemes employed in the Harrington decoder [BDMT17a] to message passing decoders. In doing so, we could hopefully make a local variant of decoders such as `BP`, but also as the ADMM implementation of the LP decoder which would then be as practical as the parallel `SSF` while benefiting from better performance.

Chapter 7

Appendix

7.1 Design of the random LDPC codes

In this section, we will present algorithms that can be run to design random regular Tanner graphs and some post-processing methods that can make them better and more homogeneous. Finally, we will see that these methods select codes that are actually more beneficial to some decoders over others, preventing a perfectly fair comparison.

To assess the performance of a decoder for some code family, it is important to simulate it on standard codes to be able to compare it fairly to other decoders. For some code families, it is easy as there is only one code for a given blocklength, and it can be built systematically without resorting to randomness. This is unfortunately not the case for the general hypergraph product codes we study, which are constructed from a random biregular graph, which guarantees it will have good properties with high probability. To build such graphs, we resort to different algorithms employing a random number generator (RNG). However, even if the codes are generated the same way and are expected to have good properties, relying on RNG will inevitably lead to heterogeneous samples of codes. Not only some codes will be better than others, but the best performing codes will depend on the decoder chosen.

The simplest algorithm to generate a random graph with a given degree distribution is called the *configuration model* and works as follows: starting from a graph with the right number of vertices, but no edges and a list, where each vertex appears as many times as its desired degree, we remove two elements picked randomly from the list and add an edge between them, and repeat until the list is empty. We will run this algorithm to design a biregular graph–that is, having two kinds of vertices which we will call left vertices and right vertices–requires two different lists and remove one element of each list to create a new edge. Another problem is the possibility of having several edges between the same two vertices, which we will correct afterward. Once the two lists are emptied, we remove copies of edges by randomly swapping copy edges [Tho03]. For example, if there exist two edges (c_1, v_1), we pick an edge (c_2, v_2) at random and replace one of the edges (c_1, v_1) and the edge (c_2, v_2) by the edges (c_2, v_1) and (c_1, v_2). In algorithm 15 we describe the version of the configuration model we adapt to generate random regular Tanner graphs.

Apart from the length two cycles which are prevented as we removed every double edges, this algorithm does not try to avoid small cycles. Yet trying to maximize the girth of the graph can be useful since small cycles can alter the performance of some decoders such as `BP`. While it would be infeasible to create a graph with maximal girth due to too large a complexity, it is possible to run a greedy algorithm called progressive edge growth (PEG) [HEA01]. Similarly to configuration model, PEG builds the graph by starting from a graph with all the vertices, but no edges, but instead of adding edges randomly it picks the edge reducing the least the current girth. To meet this condition, we choose a left vertex and pick the right vertex, which is the

Algorithm 15: Configuration model for generating a biregular graph

Input: d_L the left degree, d_R the right degree, n the number of left vertices, m the number of right vertices (with $d_L n = d_R m$)

Output: R the list of right vertices, L the list of left vertices and E the list of edges of a random d_l,d_r-biregular graph with no double edges.

$E \leftarrow$ empty list;
$R \leftarrow$ list of integers between 1 and m;
$L \leftarrow$ list of integers between 1 and n;
$Re \leftarrow d_R$concatenated copies of R;
$Le \leftarrow d_L$ concatenated copies of L;
for v *i* Re **do**
    $c \leftarrow$ a random element of Le;
    Remove one c from Le;
    Append the edge (c, v) to E;
end
while E *contains an edge which is not unique* **do**
    Pick and edge (c, v) which is not unique in E; Pick a random edge (c', v') in E;
    Remove one (c, v) and one (c', v') from E;
    Append the edges (c, v') and (c', v) to E;
end
return C

furthest from the left vertex in the current graph. In case of equality, we chose the one with the smallest degree, and we pick it at random if there is once again equality. We describe the version of this algorithm employed for generating Tanner graphs with better girth in algorithm 16. Notice that it is not needed to remove double edges as the girth is guaranteed to be at least 4 as long as the degrees are greater than 2 [HEA01]. Running this algorithm, we obtain graphs which are more homogeneous and have less small cycles than with the configuration model method.

Since PEG is an efficient algorithm, it is possible to generate many graphs and select the best one according to some score. Here, we adopt the same score as the one employed in Grospellier's thesis [Gro19b], which aggregates over each node the size of the smallest cycle to which it belongs and the number of such cycles. Initially, this score was exploited in a post-processing to the configuration model method. It consisted of randomly swap edges of the graph, but keeping only the modifications that increased the score. In doing so, he managed to improve the performance of SSF on the hypergraph product codes compared to the performance obtained on codes generated by the configuration model method.

In this thesis, most of the hypergraph product codes employed are the ones of Grospellier's thesis [Gro19b], which thus were generated applying the score-based post-processing and the configuration model method. However, since generating 1000 graphs with PEG and keeping the one with the higher score allowed to have codes of similar quality in less time, we also generated some codes with this method. This is the case of the codes with parameters [[1225,49]], [[2500,100]] and [[10000,400]] employed in figures 4.15b,4.16b and 4.17. We also used small codes generated by only PEG, this is the case of codes with parameters [[25,1]], [[100,4]], [[225,9]], [[400,16]] and [[625,25]] of figures 4.15a and 4.16a. Finally, the hypergraph product code built from a cyclic code in figure 4.18 is the code C2 used by Pantelev and Kalachev in [PK21a].

We compared the performance of codes generated with the configuration model method, PEG, and the one with the best score over 1000 generated with PEG. We show in figure 7.1 their performance when decoded with either `BP`,`BP+SSF`, `BP+OSD-0` or `QLPD`. Codes generated

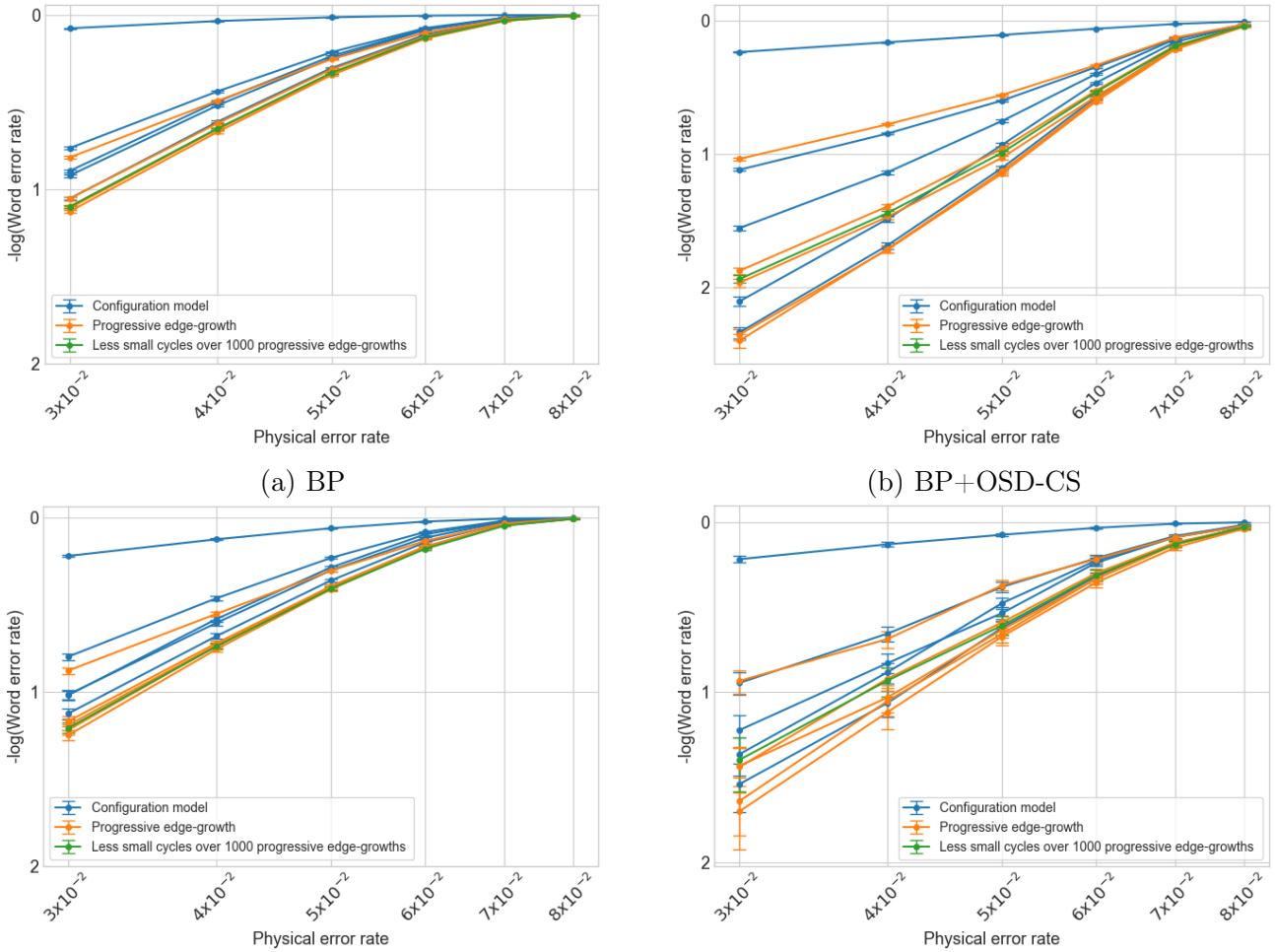
Algorithm 16: Progressive edge growth algorithm

Input: d_L the left degree, d_R the right degree, n the number of left vertices, m the number of right vertices (with $d_L n = d_R m$)

Output: R the list of right vertices, L the list of left vertices and E the list of edges of a random d_l,d_r-biregular graph with no double edges.

$E \leftarrow$ empty list;
$R \leftarrow$ list of integers between 1 and m;
$L \leftarrow$ list of integers between 1 and n;
$Re \leftarrow d_R$ concatenated copies of R;
$Le \leftarrow d_L$ concatenated copies of L;
for v *in* Re **do**
    $c \leftarrow$ a random element of Le among the ones the furthest from v and with lowest degree in the current graph;
    Remove one c from Le;
    Append the edge (c, v) to E;
end
return C

by PEG tend to perform better than the ones generated by the configuration model method for `BP`, `BP + SSF`, `BP + OSD-0` and `QLPD` which seems to indicate that all these decoders are bothered by small cycles. However, the decoders `BP + OSD-0` and `BP + SSF` which have a post-processing seem more resilient to cycles as long as the girth is large enough. Indeed, the code with the best score among 1000 generated with PEG does not perform better than most of the others generated with PEG. On the contrary, the performance of `BP` and `QLPD` seem more correlated with the score of the decoder, since the green curve is among the lowest ones.



(a) BP (b) BP+OSD-CS

(c) QLPD (d) BP+SSF

Figure 7.1: Study of the incidence of the code design on the WER for several decoders subject to the only-X-error channel. All codes are HPC built from one 3,4-regular random LDPC code of blocklength 28. However, we employed different methods to create the classical codes. The first method (5 curves in blue) is the configuration model algorithm, the second one (5 curves in orange) is the progressive edge growth algorithm (PEG), and the last one (1 curve in green) consists in choosing the code with the less small cycles among 1000 codes created with PEG. First we notice that it seems better for all decoders to use PEG rather than the configuration model algorithm as the latter sometimes gives codes with a very high WER. On the contrary, even though choosing the code with the fewer cycles seems to improve the performance for BP and QLPD, it is not so clear anymore when we combine BP with another decoder.

7.2 Equivalence of the two quantum LP decoders

In this section, we will show that the syndrome-based LP decoder is equivalent to the error-based LP decoder introduced by Li and Vontobel in [LV18].

More precisely, we want to show that whenever one outputs an integral correction, then the other one outputs the same correction. Because of this, we need the linear program to have only one solution, meaning that the objective function reaches its minimum on only one vertex of the polytope. This is also useful to guarantee the convergence of some adaptations of the LP decoder, such as the adaptive LP decoder. Fortunately, it is possible to guarantee this property by adding very little noise to the objective function, which will be enough to break the symmetries [TSS11].

To accomplish this, we will use the fact that the polytope $\mathcal{P}^s$ can be obtained by applying several reflections to the fundamental polytope. First, we need to define precisely what we call the reflection function.

Definition 59 (Reflection)**.** *Let β be in $[\![0, n-1]\!]$, we then define the function* $\underset{\beta}{\text{sym}}$*:*

$$\begin{aligned}[0,1]^n &\longrightarrow [0,1]^n,\\ (x_0,\cdots,x_\beta,\cdots,x_{n-1}) &\longrightarrow (x_0,\cdots,1-x_\beta,\cdots,x_{n-1}).\end{aligned}$$

$\underset{\beta}{\text{sym}}$ computes the symmetry of a point through the hyperplane of equation $x_\beta = 0.5$.

We will sometime apply $\underset{\beta}{\text{sym}}$ to a set of points, which will mean that we apply it to each point of the set.

For convenience, we also define the multiple reflections function.

Definition 60 (Multiple reflections)**.** *Let e be in $\{0,1\}^n$, we then define the function* $\underset{e}{\text{Sym}}$ *as:*

$$\begin{aligned}[0,1]^n &\longrightarrow [0,1]^n,\\ x &\longrightarrow (\prod_{i<n} \underset{i}{\overset{e_i}{\text{sym}}})(x).\end{aligned}$$

Similarly, we will sometime apply $\underset{e}{\text{Sym}}$ to a set of points, which means that we apply it to each point of this set. Notice that in the $\underset{e}{\text{Sym}}$ case, where x has integral coordinates we have that $\underset{e}{\text{Sym}}(x) = x + e$, it will be exploited in several proofs. Notice also that the order in which we apply the reflections is not explicit, which is not a problem since all the $\underset{\beta}{\text{sym}}$ (and by extension all the $\underset{e}{\text{Sym}}$) commute.

Using these reflections we can show a simple link between the syndrome polytopes obtained from 2 different syndromes which correspond to errors that differ by only one bit.

Lemma 61. *If two syndromes s and s' correspond to two errors which differ by only the bit v_β, then we have:*

$$\mathcal{P}^{s'} = \underset{\beta}{\text{sym}}(\mathcal{P}^s).$$

Proof. The goal is to show that:

$$\bigcap \text{Conv}(\{x \in \{0,1\}^n | (Hx)_i = s_i\}) = \underset{\beta}{\text{sym}}(\bigcap \text{Conv}(\{x \in \{0,1\}^n | (Hx)_i = s'_i\})).$$

Yet since the reflection commutes with both the intersection function and the convex hull function, it is actually enough to show that for every i in $[\![0, m-1]\!]$:

$$\{x \in \{0,1\}^n | (Hx)_i = s_i\} = \underset{\beta}{\text{sym}}(\{x \in \{0,1\}^n | (Hx)_i = s'_i\}).$$

To achieve this we will handle separately the following two possible cases: either c_i is a neighbor of v_β (case 1) or it is not (case 2).

Case 1:

In this case, $s'_i = s_i \oplus 1$:

$$\begin{aligned}
\{x \in \{0,1\}^n | (Hx)_i = s'_i\} &= \{x \in \{0,1\}^n | \bigoplus_{v_j \in \Gamma(c_i)} x_j = s'_i\} \\
&= \{x \in \{0,1\}^n | \bigoplus_{v_j \in \Gamma(c_i)} x_j = s_i \oplus 1\} \\
&= \{x \in \{0,1\}^n | x_\beta = 0, \bigoplus_{v_j \in \Gamma(c_i) \setminus \{v_\beta\}} x_j = s_i \oplus 1\} \\
&\quad \cup \{x \in \{0,1\}^n | x_\beta = 1, \bigoplus_{v_j \in \Gamma(c_i) \setminus \{v_\beta\}} x_j = s_i\} \\
&= \operatorname*{sym}_\beta(\{x \in \{0,1\}^n | x_\beta = 1, \bigoplus_{v_j \in \Gamma(c_i) \setminus \{v_\beta\}} x_j = s_i \oplus 1\}) \\
&\quad \cup \operatorname*{sym}_\beta(\{x \in \{0,1\}^n | x_\beta = 0, \bigoplus_{v_j \in \Gamma(c_i) \setminus \{v_\beta\}} x_j = s_i\}) \\
&= \operatorname*{sym}_\beta(\{x \in \{0,1\}^n | \bigoplus_{v_j \in \Gamma(c_i)} x_j = s_i\}) \\
&= \operatorname*{sym}_\beta(\{x \in \{0,1\}^n | (Hx)_i = s_i\}).
\end{aligned}$$

Case 2:
If c_i is not a neighbor of v_β, then $s'_i = s_i$ so $\{x \in \{0,1\}^n | (Hx)_i = s'_i\} = \{x \in \{0,1\}^n | (Hx)_i = s_i\}$. We then just have to show that one of these sets is invariant by application of $\sim_\beta$:

$$\begin{aligned}
\{x \in \{0,1\}^n | (Hx)_i = s_i\} &= \{x \in \{0,1\}^n | \bigoplus_{v_j \in \Gamma(c_i)} x_j = s_i\} \\
&= \{x \in \{0,1\}^n | x_\beta = 0, \bigoplus_{v_j \in \Gamma(c_i)} x_j = s_i\} \\
&\quad \cup \{x \in \{0,1\}^n | x_\beta = 1, \bigoplus_{v_j \in \Gamma(c_i)} x_j = s_i\} \\
&= \operatorname*{sym}_\beta(\{x \in \{0,1\}^n | x_\beta = 1, \bigoplus_{v_j \in \Gamma(c_i)} x_j = s_i\}) \\
&\quad \cup \operatorname*{sym}_\beta(\{x \in \{0,1\}^n | x_\beta = 0, \bigoplus_{v_j \in \Gamma(c_i)} x_j = s_i\}) \\
&= \operatorname*{sym}_\beta(\{x \in \{0,1\}^n | \bigoplus_{v_j \in \Gamma(c_i)} x_j = s_i\}) \\
&= \operatorname*{sym}_\beta(\{x \in \{0,1\}^n | (Hx)_i = s_i\}).
\end{aligned}$$

This concludes the proof. □

This result is a first step toward understanding the structure of the syndrome polytope, and it can easily be generalized to any extrinsic polytopes.

Corollary 62. *If two syndromes s and s' correspond respectively to the errors e and e' then:*

$$\mathcal{P}^{s'} = \operatorname*{Sym}_{e \oplus e'}(\mathcal{P}^s).$$

Proof. This can be proven easily by induction on $|e \oplus e'|$ using lemma 61. □

Using this result with one of the errors being trivial, we see that the syndrome polytope is equal to the fundamental polytope gone through $|e|$ reflections.

Theorem 63. *Any extrinsic polytope $\mathcal{P}^s$ obtained from a syndrome s corresponding to an error e is linked to the fundamental polytope $\mathcal{P}$ in the following way:*

$$\mathcal{P} = \operatorname*{Sym}_e(\mathcal{P}^s).$$

We made the link between the fundamental and the extrinsic polytope, yet we still have to study how this link translates when applying a LP over one or the other. It is the goal of the following lemma.

Lemma 64. *If $\mathcal{P}^s = \operatorname*{Sym}_e(\mathcal{P})$ then:*

$$\arg\min_{x \in \mathcal{P}^s}(\mathrm{d}(e, x)) = \operatorname*{Sym}_e(\arg\min_{x \in \mathcal{P}}(|x|)).$$

Proof.

$$\begin{aligned}
\arg\min_{x \in \mathcal{P}^s}(\mathrm{d}(e, x)) &= \arg\min_{x \in \mathcal{P}^s}(\mathrm{d}(\operatorname*{Sym}_e(0), x)) && \text{because } \operatorname*{Sym}_e(0) = e \\
&= \arg\min_{x \in \mathcal{P}^s}(\mathrm{d}(\operatorname*{Sym}_e(\operatorname*{Sym}_e(0)), \operatorname*{Sym}_e(x))) && \text{because the symmetry conserves the distances} \\
&= \arg\min_{x \in \mathcal{P}^s}(\mathrm{d}(0, \operatorname*{Sym}_e(x))) \\
&= \arg\min_{x \in \mathcal{P}^s}(|\operatorname*{Sym}_e(x)|) \\
&= \arg\min_{x \in \mathrm{Sym}_e(\mathcal{P})}(|\operatorname*{Sym}_e(x)|) \\
&= \operatorname*{Sym}_e(\arg\min_{x \in \mathcal{P}}(|x|)).
\end{aligned}$$

□

We can easily derive from this lemma an equality between the output of the syndrome-based `QLPD` and the error-based `QLPD`, in the case one of the outputs is integral.

Theorem 65. *The decoding based on the syndrome and the decoding based on the error are equivalent in the sens that one succeeds iff the other one succeeds.*

Proof. We just have to show that they always output the same correction. This is clear if both output "FAILURE", so we will now assume that at least one of them did not. We just have to get back to the definition of the two decoders and use lemma 64. By definition of the error-based LP decoder (resp. the syndrome-based LP decoder) the decoder outputs a correction iff $\arg\min_{x \in \mathcal{P}}(\mathrm{d}(\bar{e}, x))$ (resp. $\arg\min_{x \in \mathcal{P}^s}(|x|)$) is integral. According to 64 one is integral iff the other one is, we thus have by the hypothesis that at least one outputs a correction that both LP outputs are integral. Thus, by calling $\hat{e}_{error}$ the correction of the error-based decoder and $\hat{e}_{syndrome}$ the correction of the syndrome-based decoder, we have $\hat{e}_{error} = \bar{e} + \arg\min_{x \in \mathcal{P}}(\mathrm{d}(\bar{e}, x))$

and $\hat{e}_{syndrome} = \arg\min_{x \in \mathcal{P}^s}(|x|)$:

$$\begin{aligned}
\hat{e}_{error} &= \bar{e} \oplus \arg\min_{x \in \mathcal{P}}(\mathrm{d}(\bar{e}, x)), \text{ where } H\bar{e} = s \text{ by definition of the error-based LP decoder} \\
&= \bar{e} \oplus \operatorname*{Sym}_{\bar{e}}(\arg\min_{x \in \mathcal{P}^s}(|x|)) \text{ application of lemma 64} \\
&= \bar{e} \oplus \bar{e} \arg\min_{x \in \mathcal{P}^s}(|x|) \text{ because the solution of the LP is integral} \\
&= \arg\min_{x \in \mathcal{P}^s}(|x|) \\
&= \hat{e}_{syndrome} \text{ by definition of the syndrome-based LP decoder.}
\end{aligned}$$

□

It is interesting to notice that these two results also work in the case, where we modify the decoders to make "rounded" variants which always output a correction and no "FAILURE". To do so, we round every fractional output to the closest integral output (by rounding each coordinate inferior 0.5 to 0 and every coordinate superior or equal to 0.5 to 1).

7.3 Proof of theorem 58

To prove this result, we want some checks to take part in the corrections of only a little number of bits. This can be derived from the locality of the decoder.

Lemma 66 (Check locality)**.** *Let D be a α-local decoder for the toric code, we call β_j the number of qubits whose correction depends on the j^{th} X-type syndrome bit :*

$$\beta_j = |\{i | j \in T_i\}|.$$

Moreover, the average value of β_j can be upper bounded:

$$\frac{2}{n} \sum_{j=1}^{\frac{n}{2}} \beta_j \leq 2\alpha$$

.

Proof. The assertion comes from two different ways of counting the number of couples (i, j), such that $j \in T_i$. □

We can then show that if a check has enough locality we can find other checks which are independent of it, in the sense that no bit has its correction depending on both checks.

Lemma 67 (Independent checks)**.** *Two checks j and j' are said to be independent if there is no bit i such that both j and j' are in T_i. If two checks are independent, then the correction returned when they are both unsatisfied is exactly equal to the sum ($\oplus$) of the corrections when only one of them is unsatisfied (1). Moreover, for every check j, there is at least one independent check j' among any set of checks (not containing j) of size $\alpha\beta_j + 1$ (2).*

Proof. Assertion (1) comes from the fact that if two checks are independent, the correction applied to a bit depends at most on only one of those checks, and from the fact that we can safely assume that the correction applied to the bit i is of weight 0 whenever $w(s_{|T_i}) = 0$ (otherwise there would be corrections applied even when there are no errors).
Assertion (2) comes from the fact that the number of checks not independent of check j is at most $\alpha\beta_j$ as j appears in the correction of β_j bits, and each bit correction depends on at most α checks. □

To simplify the problem of finding unreduced errors, we will focus on horizontal error strings. However, we cannot control the form of their corrections, which may not be contained in a single row. To cope with this problem, we introduce a projection that turns any error or correction into one contained in a single row.

Lemma 68 (Horizontal projection)**.** *The horizontal projection of an error e is the sum ($\oplus$) of every row of e and is written $P(e)$ with $P(e)$ in $\{0,1\}^L$. The projection of a sum of two errors is the sum of the projections of each error (1) and two equivalent errors have the same horizontal projections (2).*

Proof. Assertion (1) comes from the sums $\oplus$ commute with each others. Assertion (2) comes from assertion (1) and the fact that a generator has a projection equal to 0^L. □

Based on this projection, we can introduce a new kind of weight of an error, which is linked to the minimal weight.

Lemma 69 (Horizontal weight)**.** *The horizontal weight of an error e is the weight of $P(e)$ and is written $|e|_h$. The minimal weight of an error is greater or equal to its horizontal weight.*

Proof. Use assertion (2) of lemma 68 and the fact that the horizontal weight of an error is smaller or equal to the weight of the error. □

Using this new weight, we can find conditions for the correction of a decoder to reduce a horizontal error string.

Lemma 70 (Horizontal error string reduced by a decoder)**.** *An error e is said to be reduced by the decoder D iff $|e + D(He)|_{min} < |e|_{min}$. A horizontal error string e is reduced by a decoder iff the horizontal projection of the correction it generates has at least one more bit set to one on the support of $P(e)$ than outside its support.*

Proof. We call w_{in} the number of bits in the support of both $P(D(He))$ and $P(e)$, and we call w_{out} the number of bits in the support of $P(D(He))$, but not in the support of $P(e)$. Using lemma 69 it is enough to show that $|e \oplus D(He)|_h \geq |e|_{min} \iff w_{in} \geq w_{out}$:

$$\begin{aligned}
|e \oplus D(He)|_h &= |P(e \oplus D(He))| && \text{by definition of } |.|_h \\
&= |P(e) \oplus P(D(He))| && \text{by assertion (1) of lemma 68} \\
&= |P(e)| + w_{out} - w_{in} && \text{by definition of } w_{out} \text{ and } w_{in} \\
&= |e|_{min} + w_{out} - w_{in} && \text{because } e \text{ is an horizontal error string.}
\end{aligned}$$

The result directly comes from this equality. □

Using this condition, we can create an artificial example, where a horizontal error string is not reduced by the decoder.

Lemma 71 (Unreduced horizontal error)**.** *Let D be a local decoder and γ be a number in $[n]$. If $\{c_i, i \in [\![0, 2\gamma+3]\!]\}$ is a set of $2\gamma+4$ ordered checks belonging to the same row, such that for all $i \in [\![0, 2\gamma+2]\!]$, c_i and c_{i+1} are independent and such that the error strings between c_0 and c_1, and $c_{2\gamma+2}$ and $c_{2\gamma+3}$ have a weight lower or equal to $\gamma+1$. Then there exist $i \in [\![0, 2\gamma+2]\!]$ such that the horizontal error string between c_i and c_{i+1} is not reduced by D.*

Proof. We will assume that for every $i \in [\![0, 2\gamma+3]\!]$ the error between c_i and c_{i+1} is reduced by the decoder D, and find a contradiction. To do so, we introduce for every $i \in [\![1, 3\gamma+1]\!]$ the horizontal projection p_i of the correction computed by D when only check c_i is unsatisfied. We then introduce m_i the number of bits set to 1 in p_i between checks c_i and c_{i-1}, and l_i (resp.

r_i) the number of bits set to 1 in p_i between checks c_{i-1} and c_0 (resp. c_i and $c_{2\gamma+3}$). We then introduce the sequence $(u_i)_{i\in[\![1,2\gamma+3]\!]}$ with $u_i = m_i - g_i - d_i$. First, it is clear that u_i is upper bounded by the weight of the horizontal error string between c_{i-1} and c_i. Moreover, since we assumed this error string to be reduced by D, lemma 70 tells us that the correction induced by c_{i-1} and c_i must have a horizontal projection with at least one more bit set to 1 between c_{i-1} and c_i than outside. Combining this with assertion (1) of lemma 67 and the fact that c_{i-1} and c_i are independent, we know that $m_i + r_{i-1} - r_i - l_i - m_{i-1}$ have to be greater or equal to 1. Since $l_{i-1} \geq 0$ we also have $m_i - r_i - l_i - m_{i-1} + r_{i-1} + l_{i-1} \geq 1$ and we recognize $u_i - u_{i-1} \geq 1$. Moreover, since the weight of the reduced horizontal error string between the two independent checks c_0 and c_1 is at most $\gamma + 1$, applying lemma 70 and assertion (1) of lemma 67 we obtain that $u_1 \geq -\gamma$. Since $u_i - u_{i-1} \geq 1$ and $u_1 \geq -\gamma$, we have $u_{2\gamma+3} \geq \gamma + 2$, which is impossible since the weight of the horizontal error string between $c_{2\gamma+2}$ and $c_{2\gamma+3}$ is at most $\gamma + 1$. We found a contradiction, and thus there exist $i \in [\![0, 2\gamma + 2]\!]$ such that the horizontal error string between c_i and c_{i+1} is not reduced by D. □

To prove the main theorem 58 we have to demonstrate that we can actually find many sets of checks having all the properties required by lemma 71 and that the error found has a weight that does not depend on the blocklength.

Proof of theorem 58. Let D be a α-local decoder on the toric code, then according to lemma 66 the average value of β_j is upper bounded by 2α. Thus, if we tile the lattice by horizontal strings of l consecutive checks, at least half will have their average β_j upper bounded by 4α. On such a set of checks, at least half of them will thus have a β_j upper bounded by 8α. Assertion (2) of lemma 67 tells us that if β_j is upper bounded by 8α, then we can find a check independent of check j among any set of $8\alpha^2 + 1$ checks on the right of j and on the same row. Moreover, if every check j' in this set has $\beta_j \leq 8\alpha$, then we can repeat the operation to find a chain of independent checks. If we have $(8\alpha^2 + 1)k + 1$ checks with $\beta_j \leq 8\alpha$, we can thus create a chain of $k + 1$ checks such that two consecutive checks are independent. $2(8\alpha^2 + 1)k + 2$ checks with at least half of them with $\beta_j \leq 8\alpha$, we can thus create a chain of $k + 1$ checks such that two consecutive checks are independent. Moreover, the last check does not need to have $\beta_j \leq 8\alpha$, thus we can choose them so that the distance between the last check and the one before is at most $8\alpha^2 + 1$. Similarly to the first check, which can be found at distance at most $8\alpha^2 + 1$ by going right from the second check. Choosing $l = 2(8\alpha^2 + 1)k + 2$ and $k = 16\alpha^2 + 4$, we can use lemma 71 with $\gamma = 8\alpha^2$, which guarantees that one of the horizontal error string between two consecutive checks is not reduced by the decoder. The number of such errors is proportional to the number t of length l strings with which we can tile the lattice without overlap. If L is the size of the lattice, then we have

$$t = \sum_{i=1}^{L} \lfloor \frac{L}{l} \rfloor.$$

By property of the floor function t is lower bounded by $\frac{n}{2l} - \sqrt{n}$ and the weight of the unreduced error is at most l which does not depend on n, thus the result.

□

Bibliography

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[ABO97] Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 176–188, 1997.

[AMT12] Iryna Andriyanova, Denise Maurice, and Jean-Pierre Tillich. New constructions of CSS codes obtained by moving to higher alphabets. Technical report, February 2012. 9 pages, 9 figures, full version of a paper submitted to the IEEE Symposium on Information Theory.

[BATB+21] J. Pablo Bonilla Ataides, David K. Tuckett, Stephen D. Bartlett, Steven T. Flammia, and Benjamin J. Brown. The xzzx surface code. *Nature Communications*, 12(1), Apr 2021.

[BDMT17a] Nikolas P Breuckmann, Kasper Duivenvoorden, Dominik Michels, and Barbara M Terhal. Local decoders for the 2d and 4d toric code. *Quantum Information & Computation*, 17(3-4):181–208, 2017.

[BDMT17b] Nikolas P. Breuckmann, Kasper Duivenvoorden, Dominik Michels, and Barbara M. Terhal. Local decoders for the 2d and 4d toric code. *Quantum Info. Comput.*, 17(3–4):181–208, mar 2017.

[BK98] Sergey B Bravyi and A Yu Kitaev. Quantum codes on a lattice with boundary. *arXiv preprint quant-ph/9811052*, 1998.

[BLDR13] Siddharth Barman, Xishuo Liu, Stark C Draper, and Benjamin Recht. Decomposition methods for large scale lp decoding. *IEEE Transactions on Information Theory*, 59(12):7870–7886, 2013.

[Bom15] Héctor Bombín. Single-shot fault-tolerant quantum error correction. *Physical Review X*, 5(3):031043, 2015.

[BT16] Nikolas P Breuckmann and Barbara M Terhal. Constructions and noise threshold of hyperbolic surface codes. *IEEE transactions on Information Theory*, 62(6):3731–3744, 2016.

[BVC+17] Nikolas P Breuckmann, Christophe Vuillot, Earl Campbell, Anirudh Krishna, and Barbara M Terhal. Hyperbolic and semi-hyperbolic surface codes for quantum storage. *Quantum Science and Technology*, 2(3):035007, 2017.

[Cam19a] Earl T Campbell. A theory of single-shot error correction for adversarial noise. *Quantum Science and Technology*, 4(2):025006, 2019.

[Cam19b] Earl T Campbell. A theory of single-shot error correction for adversarial noise. *Quantum Science and Technology*, 4(2):025006, Feb 2019.

[CDT+06] L Childress, Gurudev Dutt, Jacob Taylor, A. Zibrov, F Jelezko, P Hemmer, and M Lukin. Coherent dynamics of coupled electron and nuclear spin qubits in diamond. *Science (New York, N.Y.)*, 314:281–5, 11 2006.

[CLS21] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021.

[CS96] A Robert Calderbank and Peter W Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098, 1996.

[DCMS22] Julien Du Crest, Mehdi Mhalla, and Valentin Savin. Stabilizer inactivation for message-passing decoding of quantum ldpc codes. In *2022 IEEE Information Theory Workshop (ITW)*, pages 488–493. IEEE, 2022.

[Del20] Nicolas Delfosse. Hierarchical decoding to reduce hardware requirements for quantum computing. *arXiv preprint arXiv:2001.11427*, 2020.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[ETV99] Tuvi Etzion, Ari Trachtenberg, and Alexander Vardy. Which codes have cycle-free tanner graphs? *IEEE Transactions on Information Theory*, 45(6):2173–2181, 1999.

[Fel03] Jon Feldman. *Decoding error-correcting codes via linear programming*. PhD thesis, Massachusetts Institute of Technology, 2003.

[FGL18] Omar Fawzi, Antoine Grospellier, and Anthony Leverrier. Constant overhead quantum fault-tolerance with quantum expander codes. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 743–754. IEEE, 2018.

[FJ74] G David Forney Jr. Convolutional codes ii. maximum-likelihood decoding. *Information and control*, 25(3):222–266, 1974.

[FL95] Marc PC Fossorier and Shu Lin. Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory*, 41(5):1379–1396, 1995.

[Fos01] Marc PC Fossorier. Iterative reliability-based decoding of low-density parity check codes. *IEEE Journal on selected Areas in Communications*, 19(5):908–917, 2001.

[FSBG09] Mark F Flanagan, Vitaly Skachek, Eimear Byrne, and Marcus Greferath. Linear-programming decoding of nonbinary linear codes. *IEEE Transactions on Information Theory*, 55(9):4134–4154, 2009.

[FWK05] Jon Feldman, Martin J Wainwright, and David R Karger. Using linear programming to decode binary linear codes. *IEEE Transactions on Information Theory*, 51(3):954–972, 2005.

[Gal62] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[GK18] Antoine Grospellier and Anirudh Krishna. Numerical study of hypergraph product codes. *arXiv preprint arXiv:1810.03681*, 2018.

[GM75] Roland Glowinski and Americo Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, 9(R2):41–76, 1975.

[Got97] Daniel Gottesman. *Stabilizer codes and quantum error correction.* California Institute of Technology, 1997.

[Got14a] Daniel Gottesman. Fault-tolerant quantum computation with constant overhead. *Quantum Info. Comput.*, 14(15–16):1338–1372, nov 2014.

[Got14b] Daniel Gottesman. Fault-tolerant quantum computation with constant overhead. *Quantum Information & Computation*, 14(15-16):1338–1372, 2014.

[Gro19a] Antoine Grospellier. *Constant Time Decoding of Quantum Expander Codes and Application to Fault-Tolerant Quantum Computation.* Theses, Sorbonne universités, November 2019.

[Gro19b] Antoine Grospellier. *Constant time decoding of quantum expander codes and application to fault-tolerant quantum computation.* Theses, Sorbonne Université, November 2019.

[Gur22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[Ham50] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.

[Har04] James William Harrington. *Analysis of quantum error-correcting codes: symplectic lattice codes and toric codes.* PhD thesis, Caltech, 2004.

[HCEK15] Michael Herold, Earl T Campbell, Jens Eisert, and Michael J Kastoryano. Cellular-automaton decoders for topological quantum memories. *npj Quantum information*, 1(1):1–8, 2015.

[HEA01] Xiao-Yu Hu, Evangelos Eleftheriou, and D-M Arnold. Progressive edge-growth tanner graphs. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No. 01CH37270)*, volume 2, pages 995–1001. IEEE, 2001.

[KFL01] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[Kit03] A Yu Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[KLM01] E Knill, R Laflamme, and G Milburn. A scheme for efficient quantum computation with linear optics. *Nature*, 409:46–52, 02 2001.

[KM72] Victor Klee and George J Minty. How good is the simplex algorithm. *Inequalities*, 3(3):159–175, 1972.

[KMW02] David Kielpinski, C.R. Monroe, and D.J. Wineland. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417:709–11, 07 2002.

[KP19] Aleksander Kubica and John Preskill. Cellular-automaton decoders with provable thresholds for topological codes. *Physical review letters*, 123(2):020501, 2019.

[LD16] Xishuo Liu and Stark C Draper. The admm penalized decoder for ldpc codes. *IEEE Transactions on Information Theory*, 62(6):2966–2984, 2016.

[LLMP93] Arjen K Lenstra, Hendrik W Lenstra, Mark S Manasse, and John M Pollard. The number field sieve. In *The development of the number field sieve*, pages 11–42. Springer, 1993.

[LTZ15] Anthony Leverrier, Jean-Pierre Tillich, and Gilles Zémor. Quantum Expander Codes. In *FOCS 2015 - IEEE Annual Symposium on the Foundations of Computer Science*, pages 810–824, Berkeley, United States, October 2015. IEEE.

[LV18] July X Li and Pascal O Vontobel. Lp decoding of quantum stabilizer codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1306–1310. IEEE, 2018.

[MMM04] David JC MacKay, Graeme Mitchison, and Paul L McFadden. Sparse-graph codes for quantum error correction. *IEEE Transactions on Information Theory*, 50(10):2315–2330, 2004.

[NC02] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[Ouy21] Yingkai Ouyang. Avoiding coherent errors with rotated concatenated stabilizer codes. *npj Quantum Information*, 7(1):1–7, 2021.

[PC08] David Poulin and Yeojin Chung. On the iterative decoding of sparse quantum codes. *Quantum Info. Comput.*, 8(10):987–1000, nov 2008.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

[PGN+19] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. Leveraging secondary storage to simulate deep 54-qubit sycamore circuits, 2019.

[PK21a] Pavel Panteleev and Gleb Kalachev. Degenerate quantum ldpc codes with good finite length performance. *Quantum*, 5:585, 2021.

[PK21b] Pavel Panteleev and Gleb Kalachev. Quantum ldpc codes with almost linear minimum distance. *IEEE Transactions on Information Theory*, 68(1):213–229, 2021.

[PK22] Pavel Panteleev and Gleb Kalachev. Asymptotically good quantum and locally testable classical ldpc codes. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 375–388, 2022.

[Pre12] John Preskill. Quantum computing and the entanglement frontier. *arXiv preprint arXiv:1203.5813*, 2012.

[Ric03] Tom Richardson. Error floors of ldpc codes. In *Proceedings of the annual Allerton conference on communication control and computing*, volume 41, pages 1426–1435. The University; 1998, 2003.

[RU08a] Tom Richardson and Ruediger Urbanke. *Modern coding theory*. Cambridge university press, 2008.

[RU08b] Tom Richardson and Ruediger Urbanke. *Modern coding theory*. Cambridge university press, 2008.

[RWBC20] Joschka Roffe, David R White, Simon Burton, and Earl Campbell. Decoding across the quantum low-density parity-check code landscape. *Physical Review Research*, 2(4):043423, 2020.

[Sho95] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52(4):R2493–R2496, 1995.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, oct 1997.

[ST04] Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004.

[Ste96] Andrew M Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793, 1996.

[Tan81] R Tanner. A recursive approach to low complexity codes. *IEEE Transactions on information theory*, 27(5):533–547, 1981.

[Tho03] Jeremy Thorpe. Low-density parity-check (ldpc) codes constructed from protographs. *IPN progress report*, 42(154):42–154, 2003.

[TSS11] Mohammad H Taghavi, Amin Shokrollahi, and Paul H Siegel. Efficient implementation of linear programming decoding. *IEEE transactions on information theory*, 57(9):5960–5982, 2011.

[TZ13] Jean-Pierre Tillich and Gilles Zémor. Quantum ldpc codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2013.

[TZ14] Jean-Pierre Tillich and Gilles Zémor. Quantum LDPC Codes With Positive Rate and Minimum Distance Proportional to the Square Root of the Blocklength. *IEEE TIT*, 60(2):1193–1202, Feb 2014.

[YK17] Theodore J Yoder and Isaac H Kim. The surface code with a twist. *Quantum*, 1:2, 2017.