Efficient Message-Passing Decoding of Quantum LDPC Codes

Julien Du Crest

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : Laboratoire d'Informatique de Grenoble

Décodage efficace de codes quantiques LDPC par passage de messages

Efficient Message-Passing Decoding of Quantum LDPC Codes

Présentée par :

Julien DU CREST

Direction de thèse :

Valentin SAVIN DIRECTEUR DE RECHERCHE, CEA CENTRE DE GRENOBLE	Directeur de thèse
Mehdi MHALLA CHARGE DE RECHERCHE, CNRS Délégation Alpes	Co-encadrant de thèse

Rapporteurs :

BANE VASIC
FULL PROFESSOR, UNIVERSITY OF ARIZONA, TUCSON
ANTHONY LEVERRIER
DIRECTEUR DE RECHERCHE, CENTRE INRIA DE PARIS

Thèse soutenue publiquement le **4 novembre 2024**, devant le jury composé de :

GILLES ZEMOR, PROFESSEUR DES UNIVERSITES, UNIVERSITE DE BORDEAUX	Président
VALENTIN SAVIN, DIRECTEUR DE RECHERCHE, CEA CENTRE DE GRENOBLE	Directeur de thèse
BANE VASIC, FULL PROFESSOR, UNIVERSITY OF ARIZONA, TUCSON	Rapporteur
ANTHONY LEVERRIER, DIRECTEUR DE RECHERCHE, CENTRE INRIA DE PARIS	Rapporteur
IRYNA ANDRIYANOVA, PROFESSEURE DES UNIVERSITES, CY CERGY PARIS UNIVERSITE	Examinatrice
MNACHO ECHENIM, PROFESSEUR DES UNIVERSITES, GRENOBLE INP - UGA	Examinateur

Invités :

MEHDI MHALLA
CHARGE DE RECHERCHE, CNRS DELEGATION ALPES

Contents

List of Publications

- **A Blindness Property of the Min-Sum Decoding for the Toric Code**
  Julien du Crest, Mehdi Mhalla and Valentin Savin
  arxiv preprint, 2024, Invited presentation at ISIT24.

- **Check-Agnosia based Post-Processor for Message-Passing Decoding of Quantum LDPC Codes**
  Julien du Crest, Francisco Garcia-Herrero, Mehdi Mhalla, Valentin Savin and Javier Valls
  Quantum 8, 1334, 2024

- **Layered Decoding of Quantum LDPC Codes**
  Julien du Crest, Francisco Garcia-Herrero, Mehdi Mhalla, Valentin Savin and Javier Valls,
  12th International Symposium on Topics in Coding (ISTC23), 2023

- **Stabilizer Inactivation for Message-Passing Decoding of Quantum LDPC Codes**
  Julien du Crest, Mehdi Mhalla and Valentin Savin
  IEEE Information Theory Workshop (ITW22), 2022

Introduction

In order to build a quantum computer able to perform meaningful tasks, one needs to deal with errors that accumulate during the computation, that would otherwise render the output unusable. The first cause of errors is the still immature technologies that will be used to build such a device, while the second cause is inherent to quantum physics, due to *quantum decoherence*. One could draw a comparison between today's situation and the early days of classical computation, in the late 40's and 50's, where the problem of immature technologies was a big concern[1]. But, while one can easily achieve fault-tolerance classically by performing the same computation several times, combining that with regular backups, this turns out to be impossible quantumly, as the computations results are usually probabilistic, and because even such a simple idea as doing a backup on a quantum state is precluded by the so called *No-Cloning Theorem* that states that a quantum state cannot be copied perfectly. Hopefully this is where the field of error correction – widely used today in classical communication and storage – comes into play and finds a new application, by enabling to compute over encoded (*i.e.* redundant) information protected from noise by error-correction schemes.

Several challenges arise:

- **Characterizing the noise model that the qubits will be subjected to for each quantum device.** It might be different for the several architectures in competition today, making it yet unclear which error-correction scheme would be best.
- **Protecting the quantum information from noise.** This means designing good error-correcting code, and efficient decoders to recover and correct the errors faster than they accumulate.
- **Computing over encoded information.** This is by no mean an easy task, and for many quantum error-correcting schemes, it is still unclear how to perform such a task efficiently.

[1] One funny (if true?) explanation of the IT term *bug* is that it would come from the living bugs that would get fried near the triodes used to store information before the use of transistors.

In this thesis, we will focus on the second problem, and more specifically on the design and analysis of decoding algorithms used to recover and correct errors. We focus on a type of decoding algorithms called message-passing (MP) decoders, used to decode the class of low-density-parity-check (LDPC) codes. Although classically, this class of codes and the associated decoders were discovered in the early 60' by R. Gallager and are now well understood, much work remains to be done on their quantum analogs, qLDPC codes. This is a very active area today, that ranges from very recent results on finding the best achievable parameters for qLDPC theoretically [8, 38, 53, 67] as well as finding (theoretical) decoders for those codes [52] to implementing and comparing practical decoders [60, 68, 76]. It should also be noted that there are back and forth exchanges between this dynamic field and its classical counterpart, as the tools used to construct good quantum codes have also been reused to construct locally-testable-codes [67], a long-standing open question in classical error correction that was also solved independently in [20]. qLDPC codes have taken a huge importance following the breakthrough work of Gottesman [33], where the existence of a fault-tolerant quantum computation scheme with constant overhead was shown using qLDPC codes. Since then, qLDPC codes represent the main approach to performing fault-tolerant quantum computing, and are a very active area of research with a a lot of standing open questions.

The main contributions of this thesis can be summarized as follows :

- A rigorous analysis of the min-sum decoder on the toric code showing the inherent limitations of a message-passing based approach on a highly degenerate code (Chapter 3).
- The first rigorous analysis of layered scheduling decoding for quantum codes, as well as a generic optimal construction of layers for the wide class of quantum hypergraph-product codes (Chapter 5).
- The proposal of several new pre and post-processings that allow to drastically improve the decoding performance of the message-passing decoders, relying on the structure of the codes and the understanding of the specific issues related to degeneracy (Chapters 4, 7, and 8).

We will first introduce formally classical error correction in Chapter 1, and then build on that to explain the specificities of quantum error correction in Chapter 2. After those first introductory Chapters, we will start presenting our research results. In Chapter 3, we will discuss the decoding of the toric code, widely used today, and considered by many to be the most likely candidate for near term 2D-connectivity constrained devices [31]. We will show the inherent limitations of a specific message-passing decoder, the min-sum algorithm, and this will serve as both an insight on the hardness of decoding qLDPC codes with message passing decoders, as well as a

motivation for introducing the new decoding techniques described in subsequent Chapters. In a direct followup, in Chapter 4, wewill introduce a pre-processing that allows us to match the bounds predicted by the analysis of Chapter 3. This pre-processing may be of practical interest as its complexity is linear, hence the non negligible gains at practically no cost are very interesting, notably when using it with a costly post-processing like ordered-statistics-decoding (OSD), the main used post-processing technique today. This Chapter and the preceding one are based on the preprint [13]. In Chapter 5 we provide the first analysis of layered decoding for quantum codes, provide constructions for the wide family of hypergraph product codes, and give numerical results showing the interest of such a method that is a good compromise for decoding speed and performance. In Chapter 6, we investigate the importance of the ordering for serial and layered decoding, and show that the ordering can be an important tool to alleviate the effects of degeneracy on the decoding performance. This Chapter and the preceding one are based on the article [21], as well as additional unpublished material for Chapter 6. Finally, in the last two chapters we introduce hardware friendly post-processings. In Chapters 7 (based on article [23]), we introduce a new post-processing based on inactivating stabilizers involved in degenerate errors. We show that its error-correction performance is state-of-the-art, and that its complexity is much lower than the previous state-of-the-art OSD post-processing. Additional content that was not included in the published article discusses how to properly take into account the correlations when using the post-processing for depolarizing noise. In Chapter 8 (based on published article [22]), we discuss another post-processing based on the previously presented post-processing that introduces several new hardware driven optimizations, as well as a complete analysis of the power consumption and latency of the decoder when implemented on an FPGA board.

Notations

In the rest of this thesis, we use the following conventions.

Linear Algebra

$\mathbf{v}$	A vector.
$\mathbf{v}(i)$	The i-th coordinate of $\mathbf{v}$.
$\|\mathbf{v}\|$	Hamming weight of $\mathbf{v}$.
$\mathbf{M}$	A matrix.
$\mathbf{M}^\mathsf{T}$	Transpose of $\mathbf{M}$.
$\mathbf{M}_{[i,j]}$	The coordinate on row i and column j.
rows($\mathbf{M}$)	Rows of matrix $\mathbf{M}$.
cols($\mathbf{M}$)	Columns of matrix $\mathbf{M}$.
$\mathbf{M}_{[i,*]}$	i-th row of $\mathbf{M}$
$\mathbf{M}_{[*,j]}$	j-th row of $\mathbf{M}$
$\mathcal{M}_{m,n}(\mathbb{F})$	The set of matrices of size (m, n) over field $\mathbb{F}$.

Fields

$\mathbb{N}$	Integer numbers
$\mathbb{R}$	Real numbers.
$\mathbb{C}$	Complex numbers.
$\mathbb{F}_2$	Galois field of 2 elements.

Quantum Notations

c^*	conjugate of complex number c.
$\mathbf{v}^\dagger$	Conjugate transpose of $\mathbf{v}$

Chapter 1

Classical LDPC Codes and Message-Passing Decoding

Today, Low-Density Parity Check (LDPC) codes are used everywhere in classical error correction, and their quantum counterparts have also quickly become a staple of quantum error correction. It is only right to start by presenting the classical notions of channel-coding and LDPC codes, before building on and introducing their quantum analogs.

The description of classical LDPC codes and their decoding dates from the early 60' with the seminal work of R. Gallager [28]. In this Chapter, we review the basics of error correction in the setting of communication, as defined by Claude Shannon in the late 40' under the name *channel coding* [81], and describe the main concepts used to define LDPC codes the mainly used message-passing decoding approaches, as presented in [28]. Toward the end, we present the notion of decoding tree, introduced by Niclas Wiberg [92]. We will use it in Chapter 3 to show the limits of min-sum decoding on the family of toric codes.

1.1 Channel Coding

The first mathematical formal description of the problem of communication over a noisy channel is due to Claude Shannon [81] . It can be summarized in the box diagram of Figure 1.1, describing how two parties can communicate over a noisy communication channel. Here we only describe

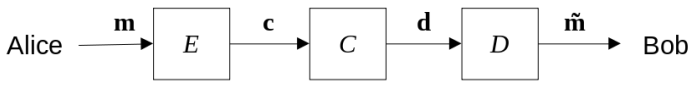


Figure 1.1: The Graphical Representation of Channel Coding

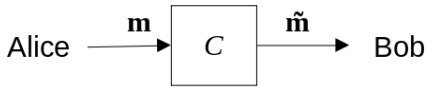


Figure 1.2: Communication on a noisy channel without error-correction

the barebone setting, but the more general setting can be found in Shannon's paper.

1.1.1 The Bit-Flip Channel

Suppose Alice wants to send a message $\mathbf{m} \in \mathbb{F}_2^k$ to Bob over a noisy communication channel, where each bit is likely to be flipped. If we further assume that the noise is applied identically and independently on each bit sent through the channel, refered to as **identically independently distributed (i.i.d.) noise**, then formally the behavior of the channel can be described by specifying the probability distribution associated to the transmission of a single bit where Alice sends bit b and Bob receives bit b':

$$\Pr(b' \mid b), \forall b, b' \in \mathbb{F}_2$$

Definition 1. *The* **bit-flip channel** *is defined by :*

$$\Pr(b' \mid b) = \begin{cases} p, & \text{if } b' = 1 - b \\ 1 - p, & \text{if } b' = b \end{cases}$$

In the rest of this chapter, we always consider the noise defined by the i.i.d. bit-flip channel, unless stated otherwise.

If Alice were to send a message $\mathbf{m} \in \mathbb{F}_2^k$ to Bob directly over a bitflip channel C of parameter p as in Figure 1.2, then the probability of success of the communication would be $(1-p)^k$ which would decrease exponentially fast as the length of the message sent by Alice increases. In order to protect the message from the noise, some redundancy is added to it. Formally, we associate to each message $\mathbf{m} \in \mathbb{F}_2^k$ a **codeword** $\mathbf{c} \in \mathbb{F}_2^n$ (the redundant message) where $n > k$. This mapping from message to codeword is given by the encoder function $E : \mathbb{F}_2^k \to \mathbb{F}_2^n$.

Then the codeword is sent over the noisy channel, meaning the noise will be applied on each bit independently, and a **noisy codeword** $\mathbf{d} = \mathbf{c} + \mathbf{e} \in \mathbb{F}_2^n$ is received by Bob. Now Bob will try to recover Alice's message using a decoder D, outputting message $\tilde{\mathbf{m}}$.

The performance of an error-correcting scheme (E, D) over a channel C will be defined by probability that the received message is the same as the input message, knowing the noise source

characterisation[1] :

$$\Pr(\tilde{\mathbf{m}} = \mathbf{m})$$

This simple diagram is already enough to present all the main questions of error correction. First, one has to decide how information is encoded, that is, to specify the encoding function E. The two main characteristics of an error correcting code are :

- The **rate** $R = \frac{k}{n}$ is the ratio between the number of bits of the initial message and the number of bits of the codeword. This characterizes how much redundancy was added.
- The **minimum distance** d is the minimum Hamming distance between any two distinct codewords. It characterizes how many errors the code can correct. It is a parameter that must be chosen relative to the noise channel – the stronger the noise, the bigger the minimum distance must be to have a chance to recover the original message.

Secondly, one has to process the noisy received message to try to recover the most-likely input message. If one is interested in maximizing the probability that $\tilde{\mathbf{m}} = \mathbf{m}$, the best theoretical decoder, called maximum-likelihood[2] (ML), estimates the probability of each input message depending on the received message and outputs the most-likely depending on the channel noise. However this decoder is usually impractical as it requires exponential computing power for most codes. In practice, one has to come up with a decoder that is both fast (at least polynomial, sometimes even log-linear) without loosing too much decoding performance compared to the ML decoder.

1.2 Linear Codes

1.2.1 Definition

The first mention of linear codes, although not stated in this exact framework, was by R. Hamming in 1950 [37]. Linear codes are a broad class of codes such that the code space is linear (*i.e.* the bitwise sum of two codewords is a codeword). Linear codes are simpler to analyse theoretically, but also proved to be very successful in practice, so much so that most of the encoding schemes used today are using linear codes.

[1] This also sometimes referred to in the literature as *block-error probability*, as opposed to *bit-error probability*, where one tries to optimise the probability of error of each transmitted bit.

[2] The maximum-likelihood decoder is properly defined in Section 1.3.3 for the subclass linear error-correcting codes.

Definition 2. *A code* $\mathtt{C}$ *is* **linear** *if there is an injective linear map* $\mathbf{G}$ *from the message space to the code space.*

$$\forall \mathbf{c} \in \mathtt{C},\ \exists! \mathbf{m},\ \mathbf{c} = \mathbf{G}\mathbf{m}$$

Alternatively, a linear code can be defined by its **parity check matrix** $\mathbf{H}$ *such that the codespace is the kernel of* $\mathbf{H}$*.*

$$\mathbf{c} \in \mathtt{C} \iff \mathbf{H}\mathbf{c} = 0$$

To describe a linear code, one uses parameters $[n, k, d]$*, where* n *is the* **codelength** *(number of bits of the code space),* k *is the code* **dimension** *(number of bits of the messages), and* d *is the* **minimum distance***, which turns out to also be the weight of the smallest codeword.*

1.2.2 Syndrome-Based Decoding

On the bit-flip channel, for a linear code $\mathtt{C}$ defined by parity check matrix $\mathbf{H}$, supposing an error $\mathbf{e}$ happens when transmitting codeword $\mathbf{c}$, the maximum-likelihood decoding problem can be stated as:

Find the codeword $\tilde{\mathbf{c}}$ in $\mathtt{C}$ that is the closest to $\mathbf{d} = \mathbf{c} + \mathbf{e}$ *i.e.*:

$$\tilde{\mathbf{c}} = \arg\min_{\mathbf{c}'} \{|\mathbf{c}' + \mathbf{d}| \quad | \quad \mathbf{H}\mathbf{c}' = 0\}$$[3]

By defining $\mathbf{s} = \mathbf{H}(\mathbf{c}+\mathbf{e}) = \mathbf{H}\mathbf{e}$, one can see that the problem of decoding can be equivalently stated as:

Find the smallest weight error $\tilde{\mathbf{e}}$ such that $\mathbf{s} = \mathbf{H}\tilde{\mathbf{e}}$ *i.e.*:

$$\tilde{\mathbf{e}} = \arg\min_{\mathbf{e}'} \{|\mathbf{e}'| \quad | \quad \mathbf{H}\mathbf{e}' = \mathbf{s}\}$$

We call the later formulation **syndrome-based decoding**.

Clearly the two approaches are equivalent, but since in the context of quantum computing, we will always be doing syndrome-based decoding, in the following, we present all the (classical) decoding notions in the context of syndrome-based decoding.

1.2.3 Tanner Graph Representation

Any linear code has a graph representation, called the **Tanner graph** [84] – named after its inventor. It is a graph with two types of vertices (called nodes), variable nodes (one for each column of the matrix) representing the coded bits and check nodes (one for each row) representing linear constraints on them. For every non-zero entry in the matrix there is an edge between the corresponding variable and check nodes.

[3] Recall $|\mathbf{v}|$ denotes the Hamming weight of vector $\mathbf{v}$.

Definition 3. Tanner Graph

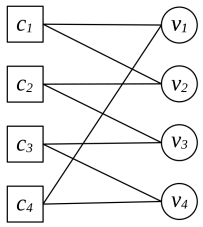
Given a code $\mathtt{C}$ *defined by the parity check matrix* $\mathbf{H} \in \mathcal{M}_{m,n}(\mathbb{F}_2)$*, the Tanner graph is a bipartite graph* $(\mathcal{C}, \mathcal{V}, E)$ *where :*

$$(c, v) \in E \iff \mathbf{H}_{[c,v]} = 1, \ \forall (c, v) \in \mathcal{C} \times \mathcal{V}$$

As an example, Figure 1.3 shows the 4-repetition code in parity-check matrix form (1.3a) and Tanner graph representation (1.3b).

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & & \\ & 1 & 1 & \\ & & 1 & 1 \\ 1 & & & 1 \end{bmatrix}$$

(a) Parity-check matrix representation



(b) Tanner graph representation

Figure 1.3: The 4-repetition code as represented by its parity-check matrix and its Tanner graph .

For the Tanner (bipartite) graph representation of a code $(\mathcal{C}, \mathcal{V}, E)$, we use the notation $\mathcal{N}(\_)$ to refer to the vertex neighbourhood (of a vertex):

$$\mathcal{N}(v) = \{c \in \mathcal{C} \mid (c, v) \in E\}, \qquad \mathcal{N}(c) = \{v \in \mathcal{V} \mid (c, v) \in E)\}$$

Note that we also sometimes use this notation abusively directly on a variable/check, implicitly referring to the neighborhood in the associated Tanner graph.

1.3 LDPC Codes and Message Passing Decoders

1.3.1 Low-Density Parity Check codes

Low-Density-Parity-Checks (LDPC) codes were invented in the 60' by Gallager [28]. Up until then, people would come up with code families with good rate and minimum distance, and only then look for efficient decoders for those codes. The breakthrough of Gallager's approach was to define a family of codes made to be decoded by a specific type of efficient decoders. However, this approach was not used in practice until the 90', where LDPC codes were rediscovered and found use with the new computing power resources available. Now LDPC codes are used everywhere in communications, among which wifi [46], 5G [74], satellite communications [54], and many others.

Definition 4. *A family of* **Low-Density-Parity-Check (LDPC) codes** *is a family of linear codes* $\{\mathtt{C}_l\ [n_l, k_l, d_l]\}_{l\in\mathbb{N}}$ *each defined by parity check matrices* $\mathbf{H}_l \in \mathcal{M}_{n_l,k_l}(\mathbb{F}_2)$ *such that there exists a constant* Δ *such that:*

$$n_{l+1} > n_l,\ \forall l \in \mathbb{N}$$

$$|\mathbf{H}_{l[i,*]}| \leq \Delta,\ \forall i \in \{1 \dots k_l\}$$

$$|\mathbf{H}_{l[*,j]}| \leq \Delta,\ \forall j \in \{1 \dots n_l\}$$

Note that since they are linear codes, LDPC codes naturally have a representation as Tanner graph, and the added property of low-density then translates to a bounded degree on every node in the graph.

Although the notion is defined asymptotically, with a slight abuse of terminology, whenever a specific code has the property that $\Delta \ll n$, implying that one will be able to decode it quite efficiently using message passing decoders, we refer to it as an "LDPC code".

1.3.2 Message-Passing Decoding

As mentioned earlier, the LDPC codes were invented together with an efficient decoding algorithm family, usually referred to as *message-passing* (MP) decoders. We will present those decoding algorithms in the context of syndrome-based decoding, as this is what will be used later for quantum codes.

In the following, we suppose we are given a Bipartite Tanner Graph $(\mathcal{C}, \mathcal{V}, E)$ associated to a parity check matrix $\mathbf{H}$, and are trying to decode a syndrome $\mathbf{s}$ *i.e.*, finding the smallest error $\tilde{\mathbf{e}}$ such that $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$.

Those decoders work by assigning a belief value to each nodes of the Tanner graph, and exchanging information between neighbouring nodes, updating those beliefs at each iteration. Here we give the formulas for the message exchanges of the three main used decoders, that will be referred to throughout this thesis: min-sum, belief-propagation, and normalized min-sum. But before introducing those decoders, we define the *skeleton* of an MP algorithm and then start with the simplest example of MP decoder : the majority-voting algorithm (also known as Gallager-B decoder).

To define an MP algorithm, one has to define the initial values of the variable nodes, known as *a priori* values. For syndrome-based decoding, the *a priori* value of a variable node corresponds to the *a priori* belief that the variable node is in error. Hence, in the case where the noise is *i.i.d.* on all bits, all the *a priori* values are equal. Then one has to define the two functions that will be used at each iteration to exchange information between neighbouring nodes, and an additional

function that outputs the belief of the decoder for each variable node after a given number of iterations, known as *a posteriori* values, as well as the hard decision, the error syndrome. One key thing about *all* message passing algorithms is that they must exchange only extrinsic information, meaning the information that one node will send to another should not depend on information received at the last round by this node. Note that $\mathbf{s}$ and $\mathbf{e}$ take value in $\mathbb{F}_2$, but all other values are in $\mathbb{R}$, and that a negative value represents the belief of being in error ($\mathbf{e}(v) = 1$) while a positive or zero[4] value represents the belief of not being in error ($\mathbf{e}(v) = 0$).

$\gamma(v), \forall v \in \mathcal{V}$	:	*a priori* (soft) information
$\mu_{v\to c}^{(i)}, \forall (v,c) \in E$	:	variable to check message at iteration i
$\mu_{c\to v}^{(i)}, \forall (v,c) \in E$	:	check to variable message at iteration i
$\tilde{\gamma}^{(i)}(v), \forall v \in \mathcal{V}$	:	*a posteriori* (soft) information at iteration i
$\tilde{\mathbf{e}}^{(i)}(v), \forall v \in \mathcal{V}$	:	the estimated error

Note that $\tilde{\mathbf{e}}^{(i)}(v) = (1 - \text{sign}\ \gamma^{(i)}(v))/2$ for all the decoders presented here. Also, when we use the notation $\tilde{\gamma}$ or $\tilde{\mathbf{e}}$ without specifying the iteration, this means we are considering the value outputted by the algorithm at its termination (whether it succeeded to converge to a solution or reached the maximum number of iterations), and that when describing the recursive functions computed by the message-passing algorithms, we use as a convention $\tilde{\gamma}^{(0)} = \gamma$ and $\mu_{c\to v}^{(0)} = 0$.

Majority-Voting is the simplest example of a message passing algorithm. At each round, the *a posteriori* of every variable node is ± 1, and each variable node sends its current belief to its neighbouring check nodes. Each check node c then transmits to each neighbouring variable node v the belief in ± 1 that would be required for c to be satisfied (assuming all other neighbouring variable nodes keep the same value). Then each variable node updates its own belief by doing a majority-voting (hence the name) on the values received from its neighbouring checks and its initial *a priori* value. Note that in order for the messages to remain *extrinsic*, the belief that a variable node sends to a check node is not the the majority it computed for its own belief (the *a posteriori*), but the majority-voting of what it received only from the other checks at the last iteration together with its *a priori* value. Note that here, to break the ties when no strict majority can be found, here we assume that the function sign is defined as : $\text{sign}(x) = \begin{cases} +1 \text{ if } x \geq 0 \\ -1 \text{ else} \end{cases}$

[4] A belief of 0 is indeterminate, in the following we take the (usual) convention of assigning it to no-error, but ties can also be broken by assigning a random value.

$$
\begin{aligned}
\gamma(v) &= 1 \\
\mu^{(i)}_{v\to c} &= \text{sign}\left(\gamma(v) + \sum_{c'\in\mathcal{N}(v)\setminus v} \mu^{(i-1)}_{c'\to v}\right) \\
\mu^{(i)}_{c\to v} &= (-1)^{\mathbf{s}(c)} \times \prod_{v'\in\mathcal{N}(c)\setminus v} \mu^{(i)}_{v'\to c} \\
\tilde{\gamma}^{(i)}(v) &= \text{sign}\left(\gamma(v) + \sum_{c'\in\mathcal{N}(v)} \mu^{(i)}_{c'\to v}\right)
\end{aligned}
$$

Min-Sum (MS) can be seen as an improvement on the majority-voting, where the belief of each variable-node is not binary (± 1) anymore, but takes its value in $\mathbb{Z}$. The sign of the belief sent by a check c to a variable node v is the same as before, meaning that the message sent by c to v still represents the value v should take in order to satisfy c, provided all other variable neighbours of c keep the same values. But now the "amplitude" of the message is equal to the amplitude of the belief of "least reliable" variable node used to compute the message.

$$
\begin{aligned}
\gamma(v) &= 1 \\
\mu^{(i)}_{v\to c} &= \gamma(v) + \sum_{c'\in\mathcal{N}(v)\setminus c} \mu^{(i-1)}_{c'\to v} \\
\mu^{(i)}_{c\to v} &= (-1) \times \prod_{v'\in\mathcal{N}(c)\setminus v} \text{sign}\left(\mu^{(i)}_{v'\to c}\right) \times \min_{v'\in\mathcal{N}(c)\setminus v} |\mu^{(i)}_{v'\to c}| \\
\gamma^{(i)}(v) &= \gamma(v) + \sum_{c'\in\mathcal{N}(v)} \mu^{(i)}_{c'\to v}
\end{aligned}
$$

Belief-Propagation (BP) is used to estimate the bit-wise probability of error of each bit. Here we only give the binary version, that we present using log-likelihood ratios instead of probabilities for numerical stability[5]. This approach results in a faster and more stable implementation of the decoder. Note that in order to run the belief-propagation, it is required to know (an estimate of) the error probability of each bit of the channel. In the following, we assume an *i.i.d.* noise on all bits defined by a bitflip channel of probability p.

[5] See for example [28] for more details

$$
\begin{aligned}
\gamma(v) &= \log \frac{1-p}{p} \\
\mu_{v\to c}^{(i)} &= \gamma(v) + \sum_{c' \in \mathcal{N}(v)\setminus c} \mu_{c'\to v}^{(i-1)} \\
\mu_{c\to v}^{(i)} &= (-1)^{\mathbf{s}(c)} \times \prod_{v' \in \mathcal{N}(c)\setminus v} \mathsf{sign}\left(\mu_{v'\to c}^{(i)}\right) \times \Phi\left(\sum_{v' \in \mathcal{N}(c)\setminus v} \Phi\left(|\mu_{v'\to c}^{(i)}|\right)\right) \\
\gamma^{(i)}(v) &= \gamma(v) + \sum_{c' \in \mathcal{N}(v)} \mu_{c'\to v}^{(i)}
\end{aligned}
$$

Where $\Phi(x) = -\log\left(\tanh\frac{x}{2}\right)$.

Normalized Min-Sum (NMS) is a modified version of the MS. The min-sum can also be seen an approximation of the belief-propagation, where as a first order approximation, $\Phi\left(\sum_{v' \in \mathcal{N}(c)\setminus v} \Phi(\mu_{v'\to c}^{(i)})\right)$ is replaced by $\min_{v' \in \mathcal{N}(c)\setminus v} |\mu_{v'\to c}^{(i)}|$. It can be shown that this approximation is always an upper bound, so in order to get a better approximation of the belief-propagation, one can improve this linear approximation by using a constant scaling factor $c_{sf} \in]0,1[$.

$$
\begin{aligned}
\gamma(v) &= 1 \\
\mu_{v\to c}^{(i)} &= \gamma(v) + \sum_{c' \in \mathcal{N}(v)\setminus c} \mu_{c'\to v}^{(i-1)} \\
\mu_{c\to v}^{(i)} &= (-1)^{\mathbf{s}(c)} \times \prod_{v' \in \mathcal{N}(c)\setminus v} \mathsf{sign}\left(\mu_{v'\to c}^{(i)}\right) \times c_{\mathsf{sf}} \times \min_{v' \in \mathcal{N}(c)\setminus v} |\mu_{v'\to c}^{(i)}| \\
\gamma^{(i)}(v) &= \gamma(v) + \sum_{c' \in \mathcal{N}(v)} \mu_{c'\to v}^{(i)}
\end{aligned}
$$

1.3.3 MS, BP, localized Maximum likelihood and Maximum A Posteriori

One can show that on acyclic graphs (*i.e.* trees) one can state exactly what the MS and BP converge to. Here we always assume an *i.i.d.* noise on all bits defined by a bitflip channel of probability p. For that we first define three theoretical decoders. We first restate for completeness the maximum-likelihood decoder defined earlier.

Definition 5. *For a matrix* $\mathbf{H}$*, a syndrome* $\mathbf{s}$*, the* **Maximum-likelihood** *(ML) decoder outputs an error* $\mathbf{e}$ *of minimum weight such that* $\mathbf{He} = \mathbf{s}$.

Definition 6. *For a matrix* $\mathbf{H}$*, a syndrome* $\mathbf{s}$*, and a variable node* v*, we define the* **localized Maximum-Likelihood (lML) belief** *as :*

$$
\gamma^{lML}(v) = \min_{\mathbf{He}_1=\mathbf{s}, \mathbf{e}_1(v)=1} |\mathbf{e}_1| - \min_{\mathbf{He}_0=\mathbf{s}, \mathbf{e}_0(v)=0} |\mathbf{e}_0|
$$

This immediately translates to a **localized Maximum-likelihood** *decoder* $lML(\mathbf{s})$ *such that :*

$$IML(v) = \begin{cases} 1 \text{ if } \gamma^{IML}(v) > 0 \\ 0 \text{ if } \gamma^{IML}(v) < 0 \\ \_ \text{ if } \gamma^{IML}(v) = 0 \end{cases}$$ [6]

One can see that the IML will always fail to decode any syndrome where there are several errors satisfying the syndrome of minimum weight.

Definition 7. *For a matrix* $\mathbf{H}$*, a syndrome* $\mathbf{s}$*, and a qubit* q*, we define the* **maximum a posteriori** *belief as:*

$$\begin{aligned}
\gamma^{MAP}(v) &= \log \frac{\Pr(\mathbf{e}(v)=0 \mid \mathbf{s})}{\Pr(\mathbf{e}(v)=1 \mid \mathbf{s})} \\
&= \log\left(\sum_{\mathbf{He}_0=\mathbf{s},\, \mathbf{e}_0(v)=0} \Pr(\mathbf{e}_0)\right) - \log\left(\sum_{\mathbf{He}_1=\mathbf{s},\, \mathbf{e}_1(v)=1} \Pr(\mathbf{e}_1)\right) \\
&= \log\left(\sum_{\mathbf{He}_0=\mathbf{s},\, \mathbf{e}_0(v)=0} p^{|\mathbf{e}_0|}(1-p)^{n-|\mathbf{e}_0|}\right) - \log\left(\sum_{\mathbf{He}_1=\mathbf{s},\, \mathbf{e}_1(v)=1} p^{|\mathbf{e}_1|}(1-p)^{n-|\mathbf{e}_1|}\right) \\
&= \log\left(\sum_{\mathbf{He}_0=\mathbf{s},\, \mathbf{e}_0(v)=0} \left(\frac{p}{1-p}\right)^{|\mathbf{e}_0|}\right) - \log\left(\sum_{\mathbf{He}_1=\mathbf{s},\, \mathbf{e}_1(v)=1} \left(\frac{p}{1-p}\right)^{|\mathbf{e}_1|}\right)
\end{aligned}$$

This allows to define the decoder MAP such that :

$$MAP(v) = \begin{cases} 1 \text{ if } \gamma^{MAP}(v) > 0 \\ 0 \text{ if } \gamma^{MAP}(v) < 0 \\ \_ \text{ if } \gamma^{MAP}(v) = 0 \end{cases}$$

This decoder is optimal in terms of bitwise error rate, but remark that this one also might output an error that does not satisfies the syndrome.

As explained in [92], one can show that on acyclic graphs of diameter δ, for the MS, the *a posteriori* of each variable node v at iteration δ is exactly $\gamma^{\mathsf{IML}}(v)$, while for the BP, the *a posteriori* of variable node v at iteration δ is exactly $\gamma^{\mathsf{MAP}}(v)$.

[6] Note that for this decoder and the MAP decoder defined below, we allow the decoder to output a third symbol (_) when the belief is indeterminate. Note that those decoders are sometimes presented as breaking ties by taking a value at random in $\mathbb{F}_2$. In this work, we wanted to emphasize the failure of those decoders in case of indeterminacy, whereas the random version has an (exponantially) small chance of answering a valid answer.

This does not translates unfortunately to graphs with cycles, but the powerful **decoding trees** introduced by Wiberg in [92] allow us to reason on graphs with cycles. Informally, this is done by unfolding the Tanner graph around a variable node into a tree (duplicating the nodes if necessary to replace cycles). In fact one can show that what the MS and the BP are computing for Tanner graph with cycles are respectively IML and MAP on the corresponding decoding tree. This will be properly defined in Chapter 3 although in a slighly less generic manner than Wiberg's, and readers interested should refer to [92] for the generic construction.

However, for practical purpose, it is important to note that for classical good LDPC codes (of size n), their diameter (*i.e.* the minimun distance between any two variable nodes) is of order $\log n$, as is their girth (*i.e.* the size of the smallest cycle). This implies that in $\mathcal{O}(\log n)$ iterations, the decoder has access to the whole information of the graph, while avoid information loops. Which means that in practice, message-passing decoders work very well. This is unfortunately not the case for quantum codes as we will see later, where the girth is usually of constant size, because of degeneracy (see Section 2.4).

1.3.4 Schedulings

Once one has defined the message exchange rules, there only remains to decide in which order the message exchanges will take place, or alternatively, the update order on the variables/checks, and this is called a scheduling. There are 3 main types of scheduling.

- The first one is **flooded scheduling** where there are only two rounds in an iteration, first processing all variable-to-check messages in parallel then all check-to-variable functions in parallel. This decoder is particularly suited for parallel architecture [6].

- The second type is **serial scheduling** where each check (and its neighbouring variable nodes) are processed one after the other. This scheduling is clearly not parallel, but its interest lies in the fact that it converges faster, since information propagates further than just neighbouring nodes in a single iteration. It can be shown that if the Tanner graph of the code is a tree, the convergence is twice faster than flooded and that this also translates in practice for good classical codes [82].

- Finally, there is a third scheduling that tries to combine the best of both words. This is the **layered scheduling**. On thing to realize is that processing check nodes in parallel can lead to a slowdown in computing time as there can be multiple read-write calls at the same time on variables, which will necessarily create a slowdown in the clock speed. One can see that check nodes that do not have a variable node in common can be processed at the

same time without the read-write collisions. Hence the idea is to split the check nodes in layers of check nodes such that in each layer, there are no check nodes having neighbouring variable nodes in common, such that a single layer can be processed in parallel, without read/write collision. If done accordingly, this can be implemented very efficiently on a parallel architecture, benefiting from the advantages of serial (converges twice as fast) while also being highly parallel and working at high clock speed [41].

1.3.5 The OSD post-processing

Because of so called stopping sets and trapping sets (see for example [70] for a survey on the subject), while decoding LDPC codes with message-passing decoders, at small error probability, it can happen that the syndrome associated to specific small error patterns is undecodable because the decoder cannot converge, even though the weight of those errors is far smaller than the minimum distance. When that happens, the slope of the error rate worsen for small error probabilities, this phenomenon being known as an **error floor**. The **ordered-statistics-decoding** (OSD) post-processing is traditionally used to alleviate error-floors [26]. It can be used to always output a syndrome-satisfying error (by solving a linear system) when the message passing decoder failed to converge to an error pattern satisfying the syndrome.

Formally, suppose we are given a parity check matrix $\mathbf{H}$ for an $[n, k, d]$ code, and reliabilities $\tilde{\gamma}$ on the variable nodes, such that the corresponding hard decision does not satisfy the syndrome. The variable nodes are sorted according to their reliability, and a set of the k most unreliable variable nodes is chosen, such that the corresponding columns in the matrix form a maximal set of linearly independent columns. This is done is the following way : Starting with a set $S = \emptyset$, we iterate over the bits sorted by increasing reliability[7], and for variable node v if $\mathbf{H}_{[*,S\cup\{v\}]}$ is full rank, we set $S = S \cup \{v\}$. Then, we fix the value of the error vector on $[n] \setminus S$ according to their reliabilities, and solve a linear system on S to output an error satisfying the syndrome.

Complexity : Algorithmically, the complexity of the OSD comes from performing Gaussian elimination. The naive algorithm for Gaussian elimination has complexity $\mathcal{O}(n^3)$. Although there have been a lot of research on the complexity of the related problems of matrix multiplication/matrix inversion, leading to a complexity of $\mathcal{O}(n^{2.373})$ (see for example [30] for a survey or [93] for the most recent advances), it should be noted that those bounds are achieved with so called *galactic algorithms* that cannot be used in practice as the hidden constant factors are too big for practical use. To date, in most real-life scenarii, the simple $\mathcal{O}(n^3)$ algorithm is still used. Since in the rest of this work, we are mostly concerned about practical aspects of decoding, we will keep $\mathcal{O}(n^3)$

[7] Several metrics can be used, but here we consider $|\tilde{\gamma}(v)|$.

for the complexity of OSD to convey the "real" cost of performing Gaussian elimination when comparing it to other post-processings for (quantum) error correction.

Chapter 2

Quantum Error Correction

We now describe the basics of quantum computing and quantum error correction (for a comprehensive and extensive review on the topic of quantum computation, see for example [64]), before introducing the families of quantum codes that will be used throughout this dissertation.

2.1 Quantum States and Measurements

The analog of a bit of information in quantum computing is a *qubit*, and instead of taking a value of 0 or 1, the **state** of a single qubit can be represented by a unit vector of $\mathbb{C}^2$.[1] Also, this description of quantum state is known in the literature as **pure state**, for a more complete description of quantum states, using density matrices, refer to [64]. To describe the state of a qubit $|\psi\rangle$, we will always use the standard orthogonal basis of $\mathbb{C}^2$, denoted $|0\rangle$ and $|1\rangle$.

$$|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1$$

Alternatively, we may write $|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$, not mentioning that the basis is $|0\rangle\,, |1\rangle$, and we also write $\langle\psi| = (\alpha^*, \beta^*)$ for the dual vector, which is the $\mathbb{C}$-valued linear operator defined by the inner product of $\mathbb{C}^2$. The subtlety of quantum information is that although a single qubit could theoretically contain infinite information (for example you could encode all the decimals of π in α), this information is not readily accessed, as *the fundamental limitation of quantum computing* is that one can only measure a qubit in a basis, thus extracting only *one bit* of information from a qubit. As there are an infinity of measurement basis, one could hope to still recover the decimals of π by doing copies of the state and doing repeated measurements, but this is impossible due to the *no-cloning theorem* that states that it is impossible to copy a quantum state perfectly.

[1] In fact, states are indistinguishable up to a global phase, so although this representation is entirely correct, if one wants an isomorphism, $\mathbb{C}^2/\mathbb{C}^*$ should be taken.

We also define the Pauli operators in the $|0\rangle, |1\rangle$ basis by the following matrices: $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, the bit-flip operator that transforms $|0\rangle \to |1\rangle$ and $|1\rangle \to |0\rangle$, and $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, the phase-flip operation that invert the phase of a quantum state. The group of the Pauli matrices also contains a third operator, $Y = iXZ$.

When describing a system of several qubits, called a **qubit register**, things differ once again from classical computation, because of **entanglement**. That is, while a classical system of n bits is represented by a vector of $\{0,1\}^n$, the system composed of n qubits is represented by a unit vector in $\left(\mathbb{C}^2\right)^{\otimes n}$ which takes into account the fact that qubits can be entangled (that is their state cannot be described by a probability distribution over pairs of states).

We generalize the Pauli matrices as the so called **Pauli operators** on qubit registers.

Definition 8. *Given a quantum register of n qubits, we denote by X_i the Pauli operator applying the Pauli X matrix on the i-th qubit of the register i.e.,*

$$X_i = \bigotimes^{i-1} I \otimes X \otimes \bigotimes^{n-i-1} I$$

In the following, for a vector $\mathbf{v} \in \{0,1\}^n$, we define the Pauli X operator of $\mathbf{v}$ as

$$X_{\mathbf{v}} = \prod_{i \in 0 \ldots n} X_i^{v(i)}$$

We define the same notations for Z.

We will sometimes use those operators to define Pauli measurements in the context of error correction.

Quantum Projective Measurements

To access quantum information, one has to perform measurements on the quantum states. Here we describe **projective measurements**, but a more complete description of measurements can again be found in [64].

Definition 9. *A* **projective measurement** *is defined by a Hermitian operator:*

$$M = \sum_m m P_m$$

where P_m are projectors with respective eigenvalue m. When measuring $|\psi\rangle$ with measurement M, the probability of outcome m is $\Pr(m) = \langle\psi| P_m |\psi\rangle$.

For example, the Pauli matrix Z defines a projective measurement since:

$$Z = (+1) \times \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + (-1) \times \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Hence it defines a measure in the $\{|0\rangle, |1\rangle\}$ basis. Similarly, the operator X can be decomposed as $(+1) \times |+\rangle \langle +| + (-1) |-\rangle \langle -|$, hence defining a measure in the $\{|+\rangle, |-\rangle\}$ basis.

It can be seen that all the Pauli operators defined above all yield a projective measurement (see for example [64] for more details), and we will use them later in the context of error correction.

The Pitfalls of quantum information

The No-Cloning Theorem The simplest classical error-correction scheme is the repetition code, where the input message (say a bit 0 or 1) is repeated a number of times, and the decoder is just a majority choice. Unfortunately such a simple idea cannot be applied to protect quantum information as the no-cloning theorem states that it is *impossible to copy perfectly a quantum state.*

Continuous Errors and Measurements On one side, there can be infinitely many errors that can happen, and on the other side, whenever we look at a quantum state (measure it), we destroy it and project to one of the finite number of basis states in which it was measured. It seems hard to believe that it is possible to correct an infinite number of errors by just having access to a finite information given by the measurements. Moreover, if the quantum information is destroyed each time we do a measurement, it seems impossible to preserve a quantum state. Thankfully, this turns out to be wrong as we will see in the following.

2.2 Quantum Codes

It should be clear that classical error correction, designed to protect one bit of information is not enough to protect a quantum state. However, surprisingly enough, it turns out that since in the end we cannot access the whole quantum state and are only left anyway with classical information, all of the quantum power is still achievable by correcting only against X and Z errors. This can be done in several ways, but here we only describe the Calderbank-Shor-Steane (CSS) construction, that encompasses a wide class of error correction schemes, and the only one that will be used throughout this work. It is also quite simple to grasp coming from classical error correction.

CSS code construction

The quantum analog of linear codes are stabilizer codes [32], of which CSS codes are a subset. But since our work focus exclusively on decoding CSS codes, we only present this framework here, in words that will make it the most comprehensible for an audience coming from classical error correction.

A CSS code is defined by a tuple of two classical parity check matrices $(\mathbf{H}_X, \mathbf{H}_Z)$ both in $\mathcal{M}_{(-,n)}(\mathbb{F}_2)$, respectively used to correct Z and X errors. From the physicist point of view, each row of the matrix is associated to a measurement of an either X-type or Z-type Pauli operator. The syndrome associated to that row is just the extracted measurement outcome. It is required that all those Pauli operators commute with each other, which is trivially satisfied for operators of the same type (X or Z), but since Pauli X and Z anti-commute, in order for the X and Z operators to commute, it is required that rows of H_X and H_Z be orthogonal, which can be stated concisely as:

$$\mathbf{H}_X \mathbf{H}_Z^\mathsf{T} = 0$$

The codespace of the CSS code defined by $\mathbf{H}_X$ and $\mathbf{H}_Z$ is defined as

$$\mathtt{C} = \text{span}\left\{ \sum_{s \in \text{im}\, \mathbf{H}_Z} |c + s\rangle \, , c \in \ker \mathbf{H}_X \right\}$$

It is described as an $[[n, k, d]]$ code where n is the number of **physical qubits**, k the number of **logical qubits** that it encodes, and d the **quantum minimum distance**.

Quantum CSS codes are said to be **degenerate**[2], as the quantum minimum distance of the code is greater than the classical minimum distance of the classical codes defined by $\mathbf{H}_X$ and $\mathbf{H}_Z$.

For degenerate codes, the Pauli X operators $X_\mathbf{e}$ and $X_{\mathbf{e}'}$, are said to be **equivalent** if $\mathbf{e} + \mathbf{e}' \in \text{im}\, H_X$, (and similarly for Z errors). Equivalent operators act the same on the codespace, so if an error that occured is $X_\mathbf{e}$ happens and one applies $X_{\mathbf{e}'}$ as a correction, the state will be mapped back to the original codeword. The operators that map a codeword to another are called **logical operators**. Since there can be equivalent logical operators, it make sense to define the equivalence class of logical errors as:

$$\mathcal{L}_X = \ker \mathbf{H}_Z / \text{im}\, \mathbf{H}_X$$

$$\mathcal{L}_Z = \ker \mathbf{H}_X / \text{im}\, \mathbf{H}_Z$$

Given that the classical code associated to $\mathbf{H}_X$ is a $[n, k_1]$ code and the classical code associated to $\mathbf{H}_Z$ is a $[n, k_2]$ code, then the parameters of the CSS code are computed as:

$$k = n - k_1 - k_2$$

[2] This notion can in fact be defined properly for all stabilizer codes [32]

$$d = \min(d_X, d_Z)$$

where

$$d_X = \min\{|\mathbf{e}|, \forall \mathbf{e} \in \ker \mathbf{H}_Z \setminus \operatorname{im} \mathbf{H}_X\}$$

$$d_Z = \min\{|\mathbf{e}|, \forall \mathbf{e} \in \ker \mathbf{H}_X \setminus \operatorname{im} \mathbf{H}_Z\}$$

Since in the following we consider the simpler model of an unbiased noise (meaning that X and Z errors are equally likely), we are interested with codes that can (roughly) correct as many X and Z errors. This justifies the fact that we will usually only specify the quantum minimum distance instead of d_X and d_Z, since for the codes that interest us, they are either equal or very close.

Here we give the fundamental theorem of CSS codes (see [64] for the proof):

Theorem 1. *A $[[n, k, d]]$ CSS code encodes k logical qubits into n physical qubits and can correct any error (even continuous one) supported on at most $\lfloor d/2 \rfloor$ physical qubits.*

Of course, as will be the subject of this work it is not sufficient to theoretically be able to correct a certain number of errors, one also has to come up with decoding algorithms that can infer such corrections from the syndrome.

The physical way quantum error correction is performed using a CSS code is the following:

(i) Quantum measurements are made of the set of operators $\{X_{\mathbf{v}}, \mathbf{v} \in \text{rows}(\mathbf{H}_X)\}$ and $\{Z_{\mathbf{v}}, \mathbf{v} \in \text{rows}(\mathbf{H}_Z)\}$, which project the quantum error (even a continuous one) into two errors $X_{\mathbf{e}_X}, Z_{\mathbf{e}_Z}$ while yielding classical syndromes $\mathbf{s}_Z$ and $\mathbf{s}_X$ (such that $\mathbf{H}_X \mathbf{e}_Z = \mathbf{s}_Z$ and $\mathbf{H}_Z \mathbf{e}_X = \mathbf{s}_X$).

(ii) Now the problem of decoding is to find corrections $\tilde{\mathbf{e}}_X, \tilde{\mathbf{e}}_Z$ satisfying $\mathbf{H}_X \tilde{\mathbf{e}}_Z = \mathbf{s}_Z$ and $\mathbf{H}_Z \tilde{\mathbf{e}}_X = \mathbf{s}_X$.

(iii) Applying operator $X_{\tilde{\mathbf{e}}_X}$ and $Z_{\tilde{\mathbf{e}}_Z}$ maps the quantum state back in the codespace. If the corrections applied belonged to the same logical class as the errors $\mathbf{e}_X$ and $\mathbf{e}_Z$, the correction was successful.

When doing message passing decoding of a quantum code, we denote by $\mathcal{C}_X$ the set of X-checks (relative to $\mathbf{H}_X$), $\mathcal{C}_Z$ the set of Z-checks, and $\mathcal{Q}$ the set of qubits.

CSS and qLDPC codes

We have not yet talked about quantum LDPC codes, and their relation to the code construction we explained. We said earlier that in order to do quantum error correction with a CSS code, one would need to measure the stabiliser operators defined by the rows of $\mathbf{H}_X$ and $\mathbf{H}_Z$. Although theoretically, those operators can be of any weight, in practice, it is very hard to perform measurement of stabilisers of high weight fault-tolerantly, and the few quantum code families that are not LDPC (see for example quantum polar codes [24]) must rely on more convoluted strategies to perform syndrome measurement, to avoid measurement errors spreading uncontrollably through the code . However, it should be noted that having low-weight stabilizers is no *panacea*, and depending on the computing architecture used (for example a 2D grid), implementing a qLDPC code that cannot be easily embedded on the plane can be challenging [3, 15, 87]. Also, the notion of qLDPC is not restricted to CSS codes, and can be defined for the more generic class of stabiliser codes (see for example [32] for an introduction to the stabilizer formalism). Since in the rest of this thesis we are interested with the message-passing decoding, a method that requires the codes to be LDPC, in the following, we only present families of CSS codes that happen to also be qLDPC. Note also that in the rest of this thesis, we will only talk about the MP decoding of CSS qLDPC codes, and never of the more generic class of stabiliser codes. Although it is possible to perform message-passing decoding of non-CSS stabilizer codes using a non-binary MP decoder, here we focus on the simpler task of correcting CSS codes, where all the challenges of decoding a quantum code already arise.

Some Families of CSS qLDPC Codes

Here we present a few families of CSS qLDPC codes that will be discussed later in this thesis.

Hypergraph Product Codes

The hypergraph product codes (HP) are a subclass of CSS codes discovered by Tillich and Zemor [86]. At the time of their discovery, they offered the best asymptotic constructions in term of minimum distance and rate, as well as a practical scheme to build quantum codes.

Hypergraph product codes are constructed as follows: Given two classical parity check matrices $\mathbf{A}$ and $\mathbf{B}$, the code is constructed as:

$$\mathbf{H}_X = [\mathbf{A} \otimes \mathbf{I}, \mathbf{I} \otimes \mathbf{B}^\mathsf{T}]$$

$$\mathbf{H}_Z = [\mathbf{I} \otimes \mathbf{B}, \mathbf{A}^\mathsf{T} \otimes \mathbf{I}]$$

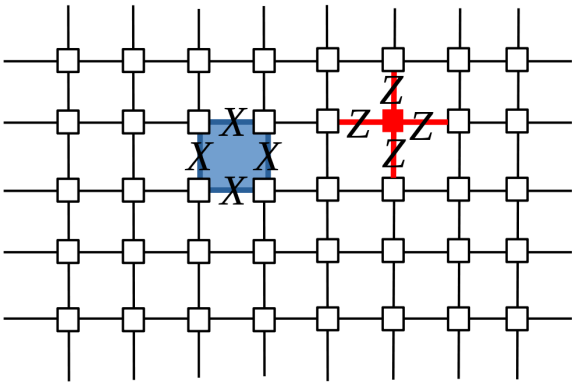


Figure 2.1: The tiling of the Torus

where $\mathbf{I}$ is the identity matrix (the size is implicit). Suppose the classical parameters of the codes defined by $\mathbf{A}, \mathbf{B}, \mathbf{A}^\mathsf{T}, \mathbf{B}^\mathsf{T}$ are as follows:

$$\mathtt{C}(\mathbf{A}) : [n_\mathbf{A}, k_\mathbf{A}, d_\mathbf{A}]$$

$$\mathtt{C}(\mathbf{A}^\mathsf{T}) : [r_\mathbf{A}, k_\mathbf{A}^\mathsf{T}, d_\mathbf{A}^\mathsf{T}]$$

$$\mathtt{C}(\mathbf{B}) : [n_\mathbf{B}, k_\mathbf{B}, d_\mathbf{B}]$$

$$\mathtt{C}(\mathbf{A}^\mathsf{T}) : [r_\mathbf{B}, k_\mathbf{B}^\mathsf{T}, d_\mathbf{B}^\mathsf{T}]$$

Then the HP code $\mathtt{CSS}(\mathbf{H}_X, \mathbf{H}_Z)$ has quantum parameters:

$$\mathtt{CSS}(\mathbf{H}_X, \mathbf{H}_Z) : [[n_\mathbf{A} n_\mathbf{B} + r_\mathbf{A} r_\mathbf{B}, k_\mathbf{A} k_\mathbf{B} + k_\mathbf{A}^\mathsf{T} k_\mathbf{B}^\mathsf{T}, \min(d_\mathbf{A}, d_\mathbf{A}^\mathsf{T}, d_\mathbf{B}, d_\mathbf{B}^\mathsf{T})]]$$

The Toric Code

The toric code is one of the most widely used and studied family of quantum codes, and it is a simple illustration of a hypergraph product code. We first describe it in the hypergraph product formalism, before giving a topological and more graphical representation.

The toric code is generated from two repetition codes (with added final redundant line).

For example, here we provide the matrices $\mathbf{A} = \mathbf{B}$ for a 4-repetition code used to create the toric code of parameters $[[32, 2, 4]]$.

$$\mathbf{A} = \mathbf{B} = \begin{bmatrix} 1 & 1 & & \\ & 1 & 1 & \\ & & 1 & 1 \\ 1 & & & 1 \end{bmatrix}$$

The toric code is usually represented as a tiling of the torus as depicted in figure 2.1. In that case, the edges represent the qubits (referred to as vertical or horizontal qubits), the Z stabilizers

are located on the vertices, and the X stabilizers on the plaquettes. Note that we will always depict the toric code using the tiling representation and not the Tanner graph. But whenever we refer to the neighbourhood of a check/variable using the notation $\mathcal{N}(\_)$, we refer to the vertex-vertex adjacency in the Tanner graph, which would translate in the tiling graph as an edge-vertex adjacency.

Generalized Hypergraph Product Codes

Definition 10. *A* **circulant** *matrix* $\mathbf{M}$ *is a square matrix of size* l *such that the*

$$\forall i, j, k, \quad \mathbf{M}_{i+k,\ j+k} = \mathbf{M}_{i,j}$$ [3]

In the following, we associate to each binary circulant matrix $\mathbf{M}$ its unique polynomial $p(x) = \sum_{j\in 0\dots l-1} \mathbf{M}_{0,j} x^j \in \mathbb{F}_2[x]/(x^l-1)$. This map is one to one. We denote by $\mathbb{B}(p(x))$ the circulant matrix of size l associated to polynomial $p(x) \in \mathbb{F}_2[x]/(x^l-1)$, and we denote by $p^\mathsf{T}(x)$ the polynomial such that $\mathbb{B}(p^\mathsf{T}(x)) = \mathbb{B}(p(x))^\mathsf{T}$.

Given a matrix $\mathbf{A} \in \mathcal{M}_{m,n}\Big(\mathbb{F}_2[x]/(x^l-1)\Big)$, we extend the function $\mathbb{B}$ to mean the binary matrix $\mathbb{B}(\mathbf{A}) \in \mathcal{M}_{m\times l, n\times l}(\mathbb{F}_2)$ as the matrix where all the polynomial coefficients of $\mathbf{A}$, $p_{i,j}(x) = \mathbf{A}_{[i,j]}$ are replaced by their associated binary circulant matrix $\mathbb{B}(p_{i,j}(x))$.

Definition 11. *The* **Generalized hypergraph product (GHP)** *code associated to matrix* $\mathbf{A} \in \mathcal{M}_{m,n}\Big(\mathbb{F}_2[x]/(x^l-1)\Big)$ *whose coefficients are* $(a_{i,j})$*, and polynomial* $b(x) \in \mathbb{F}_2[x]/(x^l-1)$*, is the CSS code defined by the binary matrices*

$$\mathbf{H}_X = [\mathbb{B}(\mathbf{A}), \mathbb{B}(b(x)) \otimes \mathbf{I}], \qquad \mathbf{H}_X = [\mathbb{B}(b(x))^\mathsf{T} \otimes \mathbf{I}, \mathbb{B}(\mathbf{A}^*)]$$

where $\mathbf{A}^* \in \mathcal{M}_{n,m}\Big(\mathbb{F}_2[x]/(x^l-1)\Big)$ *is the matrix such that if* $\mathbf{A}_{[i,j]} = p(x)$*,* $\mathbf{A}^*_{[j,i]} = p^\mathsf{T}(x)$*.*

The B1 and C2 Codes

The B1 code $[[882, 24]]$ is a (3,6)-regular[4] GHP code defined by:

$$\mathbf{A} = \begin{bmatrix} x^{27} & 0 & \dots \\ x^{54} & x^{27} & \\ 1 & x^{54} & \ddots \\ 0 & 1 & \ddots \\ \vdots & & \ddots \end{bmatrix} \qquad b(x) = 1 + x + x^6$$

[3] Where all indices are (mod l)

[4] A matrix is said to be (α, β)-regular if all rows (resp. columns) have weight exactly α (resp. β). By extension, a quantum CSS code is said (α, β)-regular if both $\mathbf{H}_X$ and $\mathbf{H}_Z$ are (α, β)-regular.

The C2 code $[[1922, 50, 16]]$ is a (3,6) regular HP code constructed out of a single circulant matrix of size 31, $\mathbf{A} = \mathbf{B} = \mathbb{B}(p(x))$ where $p(x) = 1 + x^2 + x^5 \in \mathbb{F}_2[x]/(x^{31} - 1)$

Asymptotic constructions

In the recent years, a very active research area has been on proving the existance of so-called "good" quantum codes with linear rate and minimum distance. Not much improvement had come since the hypergraph product codes of [86] (around 2013), until 2020 when the fiber bundle codes [38] convincingly broke[5] the $\sqrt{n}$ distance barrier. Using the homological formalism of chain complexes, the longstanding question was finally answered independently in [20, 53, 67]. Quickly after, it was also shown that those codes accept log-linear decoders [35, 52]. Interestingly, those decoders are based on the small-set-flip decoder of [51], which is the quantum analog of the fondational Sipser Spielman bit-flip decoder [83]. Some work has been done to make those theoretical decoders practical [34], but as is usually the case, the theoretical decoders on which one can hope to get mathematical guarantees of performance are rarely the best for practical use.

Similarly, those constructions have not yet yielded interesting practical constructions, and the state of the art constructions for less than a few thousand qubits are constructed using the methods presented above, which will be the codes we use later on for benchmarking our algorithms.

2.3 Error Models

The noise model is a very important parameter to take into account when designing an error-correction scheme. Today, many technologies are competing to build the first intermediate scale quantum computer (NISQ device), each technology having its own noise model, sometimes not even clearly defined yet (see for example [12] for a review on the different technologies). This noise can depend on the architecture and the computation for example SPAM errors focusing on state preparation and measurement errors, or circuit level noise where the noise is dependant on the physical circuit implemented (see for example [29]). Without specific assumptions on the noise model, we consider the simpler model that encoded qubits are affected by Pauli errors, where each qubit is independently acted on by a Pauli I, X, Y, Z error, with probability $\Pr(I), \Pr(X), \Pr(Y), \Pr(Z)$. The total error probability is denoted by $p = 1 - \Pr(I) = \Pr(X) + \Pr(Y) + \Pr(Z)$. Since $Y = iXZ$, we only need to correct for X and Z errors. In most of the literature, the depolarizing noise is used, which is an unbiased noise (meaning as many X, Y, Z errors), and has the merit to capture the difficulty of decoding quantum noise.

[5] Up until then, the only improvements knows over the $\sqrt{n}$ minimum distance barrier where poly-log [25].

More formally, the **depolarizing channel** of parameter p is an one qubit channel with probabilities:

$$\Pr(X) = \Pr(Y) = \Pr(Z) = p/3 \qquad \Pr(I) = 1 - p$$

For us, since in this work we focus on binary decoding using message passing, for balanced codes (meaning they have the same error correction capability for X and Z error), we almost exclusively only decode X errors, and to keep the analysis simple, we use the simpler model of X noise defined as follows:

$$\Pr(X) = p \qquad P(I) = 1 - p$$

Of course, when having access to a decoder for X and Z errors, one can design a decoder for depolarizing noise by taking into account the correlations induced by the relation $Y = iXZ$. This will be discussed more in details in Section 7.4.

2.4 The Challenges of Decoding qLDPC codes with Message Passing: Degeneracy And Short Cycles

Contrary to their classical counterparts, were we know how to design LDPC codes with good properties for message-passing based decoding, in quantum, the additional orthogonality requirement of CSS codes makes it hard – if not impossible – to construct qLDPC codes with good decoding properties, because of the reason listed below.

2.4.1 Degenerate and Non-Degenerate Errors

Here we introduce the (non-standard) notion of **degenerate error** that we defined in [13]. This formalism will allow us to capture many ideas developed through several papers: It will be used throughout this thesis, in particular in Chapter 3, and in Chapter 7 where it generalizes the notion of stabilizer-spitting errors of [23]. It also encompasses the notion of *symmetric stabilizer trapping set* from [73].

For CSS codes, the fact that the quantum minimum distance is greater than the classical minimum distance of the constituent codes is known as **degeneracy**. Once again, this comes from the fact that errors are equivalent up to stabiliser addition as defined in Section 2.2. The notions of degenerate syndrome and degenerate error, introduced below, capture the fact that there may be several errors of minimal weight explaining a given syndrome.

For a given parity-check matrix $\mathbf{H}$, for any syndrome $\mathbf{s}$, we denote by $\mathcal{E}(\mathbf{s})$ the set of errors satisfying this syndrome.

$$\mathcal{E}(\mathbf{s}) = \{\mathbf{e} \ | \mathbf{He} = \mathbf{s}\}$$

Moreover, we denote by $|\mathbf{e}|$ the (Hamming) weight of an error $\mathbf{e}$, and define $\mathcal{E}_{\min}(\mathbf{s})$ as the subset of errors of minimal weight.

$$\mathcal{E}_{\min}(\mathbf{s}) = \{\mathbf{e} \in \mathcal{E}(\mathbf{s}) \mid |\mathbf{e}| = \min_{\mathbf{e}' \in \mathcal{E}(\mathbf{s})} |\mathbf{e}'|\}$$

Definition 12. *A syndrome* $\mathbf{s}$ *is said* **degenerate** *if* $|\mathcal{E}_{\min}(\mathbf{s})| > 1$.

Definition 13. *An error* $\mathbf{e}$ *is said* **degenerate** *if there exists* $\mathbf{e}' \neq \mathbf{e}$ *such that* $|\mathbf{e}'| \leq |\mathbf{e}|$ *and* $\mathbf{H}\mathbf{e}' = \mathbf{H}\mathbf{e}$.

Lemma 1. *If* $\mathbf{e} \in \mathcal{E}_{\min}(\mathbf{s})$, *then* $\mathbf{e}$ *is degenerate iff* $\mathbf{s}$ *is degenerate.*

We also refine the classical notion of decoding radius to discuss more precisely the decoding of non-degenerate errors.

Definition 14. *The* **non-degenerate** *decoding radius* ω, *for a given decoder and code, is the largest integer* ω *such that all non-degenerate errors of weight* $\leq \omega$ *are correctly decoded.*

Degenerate syndromes are hard to decode since there are several most likely errors. Since message passing decoders try to estimate the error locally, without having a notion of degeneracy and of the logical equivalence of errors, they usually get stuck on degenerate errors, not being able to decode them. This is a trouble since small errors consisting of half of the support of a stabilizer (see [13, 23, 73]) have been shown to produce undecodable errors.

2.4.2 Short Cycles in qLDPC code

It is well known from classical LDPC theory that short cycles impair the decoding performance [36, 72, 84]. By construction, qLDPC (CSS) codes suffer from many short cycles since the matrices are low-weight and orthogonal, each line of $\mathbf{H}_X$ is a small codeword of $\mathbf{H}_Z$ and respectively. Since their discovery, much work has been done on how to cope with short cycles in CSS codes [44, 69, 73]. Hence much of the research on message-passing decoding of quantum codes has been on trying to cope with those short cycles, using post-processings [22, 23, 68] or changing slightly the decoder itself [21, 50] or the parity-check matrix [60].

In Figure 2.2, we give a visual example of the challenges of decoding quantum codes.One can see that instead of converging to an error (even possibly one that would not satisfy the syndrome), the reliability values cycle, sometimes reaching very high beliefs in one direction or the other, and then going back to indecision. This is due to the loopy nature of the decoding graph, where the convergence is hindered by the biases of self-confirmation arising from small loops.

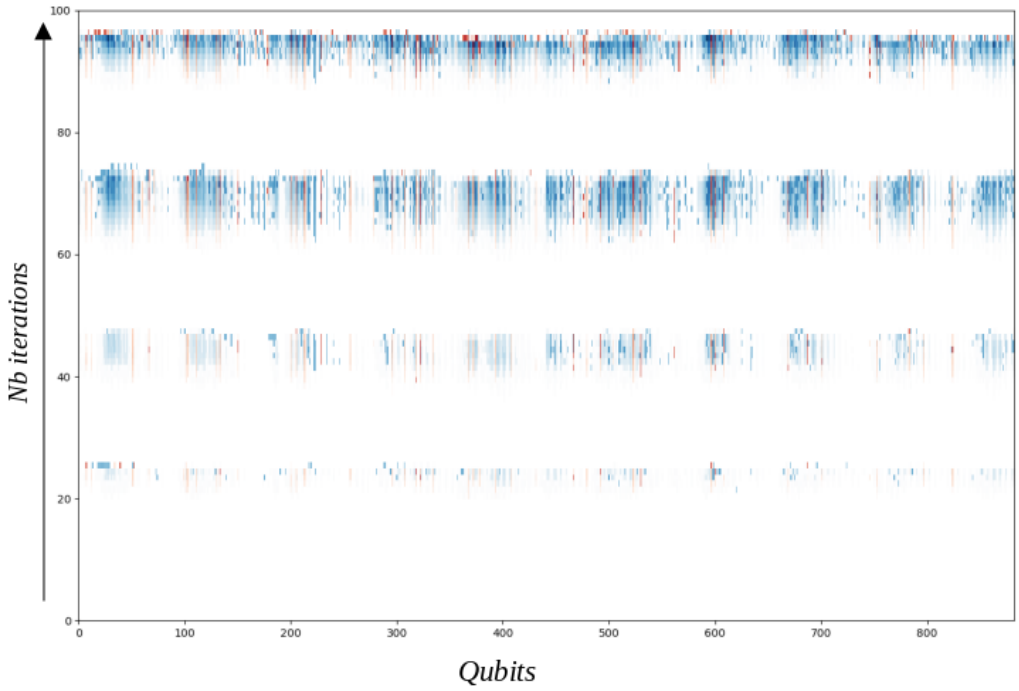


Figure 2.2: BP convergence failure after 100 iterations. The x-axis represents the qubits of the code, while the z-axis represents the number of iterations. At each iterations, the reliability of each qubit are represented by a color ranging from dark red (strong belief of error) to dark blue (strong belief of no-error).

2.5 Decoding Techniques

Here we quickly discuss two additional decoding techniques related to message-passing decoding of qLDPC codes that will be mentioned throughout this work.

2.5.1 Decoding Depolarizing Noise

When decoding depolarizing noise with a message-passing decoder, to take into account the correlations between the X and Z syndrome, it seems natural to use a quaternary alphabet (I, X, Y, Z) to decode both X and Z errors at the same time, taking into account the correlations at each iteration. This has been described and used many times [1,66,68] and for readers interested with the implementation details, one can find a good explanation in [66], Appendix B.

In this work, we are only concerned about decoding of X and Z errors separately. Taking into account the correlations using two separate binary decoders has been studied in [14, 75] and very recently in [71]. In Section 7.4, we discuss how correlations can be taken into account when using a binary message-passing decoder together with a post-processing step, and compare it to

the quaternary decoder.

2.5.2 The Ordered Statistic Decoding (OSD) Post-Processing

The application of the OSD post-processing to quantum CSS codes was proposed in [68]. MP+OSD has been shown to produce effective results on several code families, when the MP decoder is a normalized version of the MS (NMS) using a serial scheduling. Even though its prohibitive cost makes it unpractical, it is now considered as the state-of-the-art decoder for message passing based decoding of qLDPC codes, and in the following we always take it as a baseline when discussing our post-processings. Furthermore, as was shown in [76], MP+OSD is able to approximate ML decoding on the toric code, a topic that will be mentioned again in Chapter 4.

Also note that very recently, independent works [40, 94] have been focusing on designing post-processing inspired by OSD and much less costly. As those results appeared on arxiv at around the time this thesis was written, it is yet too early to know if those promising attempts will make OSD-like post-processing hardware friendly, a perspective that would no doubt be a big step towards efficient decoding.

Chapter 3

A Blindness Property of the Min-Sum Decoding for the Toric Code

This Chapter and the next are based on a preprint [13]. Although it came last, this work is a good introduction to the different works presented in this thesis. To the best of our knowledge, this is the first work showing formally the limitations of a message-passing decoder on a quantum code.

3.1 Introduction

Quantum LDPC codes are known to be classically degenerate [69], thus the sparsity of their Tanner graph does no longer fulfill its role of enabling efficient MP decoding, but merely acts as an enabler for fault-tolerance (*e.g.*, fault-tolerant syndrome extraction and fault-tolerant operations on logical qubits). Consequently, a significant effort has been devoted over the last few years to efficient decoding of quantum LDPC codes, by either combining the MP decoding with a post-processing step [22, 23, 68, 76], and/or improving the MP decoding performance itself [50, 65, 75, 91, 95].

Here, we consider the MP decoding of the Kitaev toric code [47], the planar version of which is currently the dominant error-correction solution to achieve fault-tolerance in large-scale quantum computers, especially for connectivity constrained technologies [31]. It should be noted that message passing is currently not the go-to solution to decode the toric code. The minimum weight perfect matching decoder of [19], (see [27] for a very good description of the algorithm) achieves ML decoding for X-noise and is usually used as a benchmark for decoding the toric code. The union-find decoder of [16] is achieving almost linear time decoding with very good error correction performances, and the tensor network decoder of [7] is able to approximate quantum ML at a reasonable complexity. Among the large class of quantum LDPC codes, the toric code is known to be the less responsive to MP, due the the presence of weight-2 degenerate errors

undecodable by vanilla MP decoders, such as BP or MS. It is very interesting to study message-passing decoding on the toric code because its high degeneracy allows to study the behavior of the decoder in "stress conditions", and its simple structure gives hope for some explicability. Because of the weight-2 degenerate errors, the logical error rate of these vanilla decoders scales as p^2, where p is the physical error rate, well behind the $p^{\lceil \frac{d}{2} \rceil}$ scaling of a minimum distance decoding (*i.e.*, correcting any error of weight $\leq \lfloor \frac{d-1}{2} \rfloor$), where d is the minimum distance of the code. A few approaches proposed in the literature, such as BP with memory [50], generalized BP [65], or neural-BP [55, 90], succeeded to improve the logical error rate performance for small distance toric codes ($d \leq 9$), but they all failed to scale to larger distance codes. Precisely, for $d \leq 9$, the above approaches may yield a logical error rate that scales as $p^{\lceil \frac{d}{2} \rceil}$, but this scaling does not longer improve with increasing minimum distance[1], indicating the presence of undecodable errors of weight 5. Here, we try to understand if such difficulties come from an intrinsic limitation of the toric code itself, hindering further improvement of MP-based decoding for $d > 9$.

Our goal is to understand how the information is spreading on the toric code, and what is the maximum length it can travel. To provide a formal statement, we first introduce the notion of local blindness of an MP decoder. Informally, given an error syndrome $\mathbf{s}$ and an unsatisfied check c, *i.e.*, such that $\mathbf{s}(c) = 1$, we consider a fake syndrome $\mathbf{s}^c$, having c as the only unsatisfied check (note that $\mathbf{s}^c$ is not a valid error syndrome, since no error can generate it). We say that the MP decoding of $\mathbf{s}$ is *locally blind* in the neighborhood of c, if running the MP decoding on the error syndrome $\mathbf{s}$, or on the fake syndrome $\mathbf{s}^c$, yields the same *a posteriori* value for any qubit q neighboring c, and for any number of decoding iterations. Put differently, at no iteration are the qubits neighboring c aware that it is not the only unsatisfied check of the syndrome: the MP decoder fails to convey the information from the other unsatisfied checks. Intuitively, this may happen when c is too far away from the other unsatisfied checks of the syndrome.

Main Results

Here, we focus on the MS decoding, with the conventional flooded scheduling. The reason is twofold. First, MS is an MP decoding algorithm that is aimed at solving the maximum likelihood (ML) decoding problem in a *localized* manner (as explained in 1.3.3). It actually succeeds in doing so for codes defined by acyclic graphs. However, quantum LDPC codes (as well as good classical LDPC codes) are defined by graphs with cycles, preventing local information from being

[1] The Adaptive BP with Memory (AMBP) in [50] may improve the slope of the logical error rate curve beyond $d = 9$, which comes from the use of many MBP decoders (with different meta-parameter values). Put differently, this improvement is attributable to a form of decoding diversity, rather than the intrinsic error correction capacity of the MBP decoder, which saturates at $d = 9$ [50, Section 4.4].

spread effectively, and causing a degradation of the error correction performance with respect to ML decoding. On that subject, we start in Section 3.2 by a brief discussion on the decodability of small weight errors, relative to MS and IML.

Second, MS presents a number of practical advantages, *e.g.*, low computational complexity (only requires additions and comparisons) and robustness to low precision arithmetic (*e.g.*, below 6-bit messages), being the *de facto* solution used in practical applications and hardware implementations [6]. Moreover, assuming a Pauli channel model for physical qubit errors, with X and Z errors decoded independently, MS does not need an *a priori* knowledge of the channel X, Y, Z error probabilities [23].

Our first result states that the **local** information exchange of the MS is not spreading **globally**.

Consider an error syndrome $\mathbf{s}$, and let c be an unsatisfied check, *i.e.*, such that $\mathbf{s}(c) = 1$. Let also $\mathbf{s}^c$ denote the binary vector, such that $\mathbf{s}^c(c') = \begin{cases} 1, & \text{if } c = c' \\ 0, & \text{otherwise} \end{cases}$.

Note that $\mathbf{s}^c$ is not a valid error syndrome, as no error can generate it. If one runs MS or any other MP decoder on $\mathbf{s}^c$, the decoder will run for infinitely many iterations, without finding a valid explanation of the input "fake" syndrome $\mathbf{s}^c$. We are interested in the *a posteriori* values computed by the decoder for the qubits neighboring c. We denote by $\text{APP}(q, i)$ the *a posteriori* value of a qubit q at iteration i, when the decoder is supplied with the real error syndrome $\mathbf{s}$. Likewise, we denote by $\text{APP}^c(q, i)$ the *a posteriori* value of a qubit q at iteration i, when it is supplied with the fake error syndrome $\mathbf{s}^c$.

Definition 15. *Consider the MP decoding of a syndrome $\mathbf{s}$, and let c be un unsatisfied check. We say that* **the decoding of $\mathbf{s}$ is locally blind in the neighborhood of** *c, if $\text{APP}(q, i) = \text{APP}^c(q, i)$, for any qubit q neighboring c, and any iteration $i \geq 0$.*

Theorem 1 (MS Local Blindness)**.** *Consider an error syndrome $\mathbf{s}$ on a toric code, such that all the unsatisfied checks are at distance at least 5 from each other. Then, under the i.i.d. noise assumption, the MS decoding of $\mathbf{s}$ is locally blind in the neighborhood of any unsatisfied check.*

In particular, it follows that such a syndrome is undecodable by MS. Further, we are interested in determining the smallest weight of an undecodable non-degenerate error. Recall that an error $\mathbf{e}$ is **non-degenerate** if it generates a syndrome that cannot be generated by any other error $\mathbf{e}' \neq \mathbf{e}$, such that $|\mathbf{e}'| \leq |\mathbf{e}|$ (see Section 2.4.1). For instance, 1-dimensional errors $\mathbf{e}$ of weight $|\mathbf{e}| \leq \lfloor \frac{d-1}{2} \rfloor$ and generating a syndrome of weight 2 are non-degenerate. But some 2-dimensional errors may also be non-degenerate.

Finally, before introducing the next Theorem, we refine the classical notion of decoding radius

(*i.e.*, the largest integer ω, such that any error of weight $\leq \omega$ is decoded correctly), by introducing the **non-degenerate decoding radius**, which only takes into account non-degenerate errors.

Theorem 2 (MS Non-Degenerate Decoding Radius)**.** *For any toric code of distance ≥ 9, the non-degenerate decoding radius of the MS is 3.*

We give formal proofs of Theorems 1 and 2, based on the formalism of decoding trees. We also conjecture that Theorem 1 holds for the normalized MS decoder, irrespective of the normalization factor (note also that for an appropriate choice of the normalization factor, the normalized MS provides a good approximation of BP). Although the proof techniques we use do not translate directly to normalized MS, we provide numerical evidence to corroborate our conjecture.

3.2 A Preliminary Discussion on Decoding Small Weight Degenerate and Non-Degenerate Errors

Before going on to show the main theorems of this chapter, we start by giving some insight on what can and cannot be decoded by the MS (and when this differs from IML), for small degenerate and non-degenerate errors on the toric code.

3.2.1 Weight 2 Degenerate Errors are Undecodable by MS

There are only two degenerate errors of weight 2 up to translation and rotations (depicted in Figure 3.1). To show that they are both undecodable by the MS, we use the fact that both syndromes are invariant under some symmetries. The two cases are depicted in Figure 3.1 (a) and (b). For the weight 2 syndrome of (a), it is clear that by the symmetry about the ℓ line, $\text{APP}(q_k, i) = \text{APP}(q'_k, i)$ for $k \in \{0, 1\}$, $\forall i \geq 0$. Hence the estimate error around c will never satisfy the syndrome $\mathbf{s}$. A similar argument applies for (b).

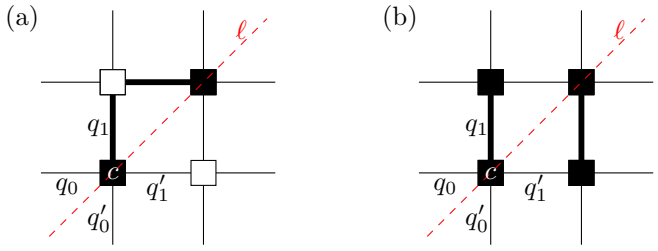


Figure 3.1: Undecodable degenerate errors of weight 2

3.2.2 A Weight 3 Degenerate Error Decodable by MS

Although we said earlier that degenerate syndromes are key to understand the failures of the MS, and we will emphasize it later on using numerical simulations, although all degenerate errors are undecodable by the IML, it is not true that degenerate implies undecodable by MS. In Figure 3.2 we provide the smallest example of a degenerate syndrome that is decodable (in 1 iteration) by the MS.

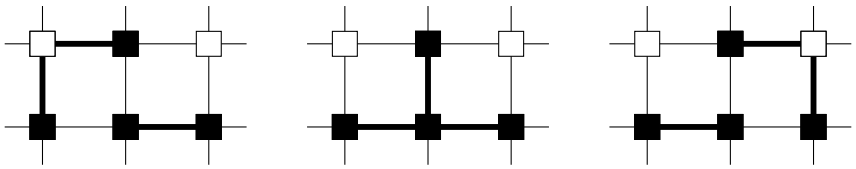


Figure 3.2: Equivalent weight 3 decodable degenerate errors (generating the same degenerate syndrome of weight 4): Although the three errors are all of weight 3, the MS will converge to the middle error in just one iteration (this can be easily checked, by computing "by hand" the *a posteriori* value of each qubit after one iteration).

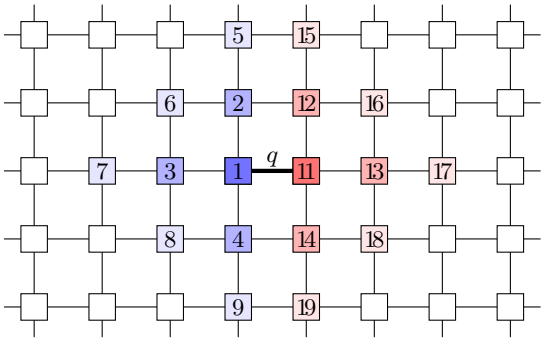
3.3 Some Properties of the Decoding Tree of the Toric Code

We now introduce formally the decoding tree in the context of the toric code, and give several new definitions as well as preliminary lemmas used later in the proofs of Theorem 1 and 2.

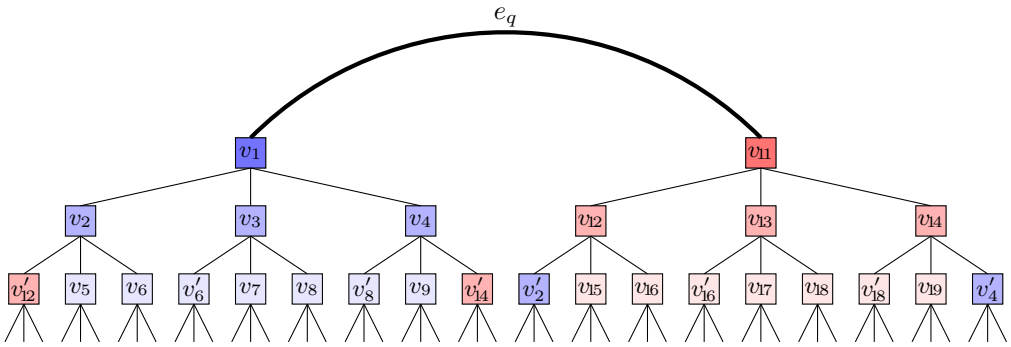
3.3.1 The Decoding Tree of the Toric Code

We represent the toric code as a square tiling of the torus, *i.e.*, a two-dimensional square lattice with periodic boundaries, where the qubits are located on the edges, the Z checks on the vertices, and the X checks or the plaquettes. Note that the definition of X and Z checks is reversed by considering the dual lattice. Thus, we shall only consider here one type of checks, and we assume they are located on the vertices of the lattice. A toric code is depicted in Figure 3.3a (or see [47] for a formal introduction).

As mentioned above, the MS algorithm converges to a localized ML decision if the Tanner graph of the code is a tree [92]. This is not true if the Tanner graph contains cycles, however, there is a neat way to explicitly compute the *a posteriori* value that the MS will output for every qubit at every iteration, using the notion of minimal configurations on decoding trees. Those ideas were first presented in [92], and we quickly review here what will be necessary for the proofs of the next section.



(a) Square tiling of the torus



(b) Decoding tree $\mathcal{T}_{i,q}$ of qubit q at iteration $i = 3$

Figure 3.3: Decoding tree of the toric code. (a) Toric code defined by a square lattice with periodic boundaries. Checks are represented by squares, placed on the lattice vertices, and qubits are represented by edges. (b) The decoding tree of qubit q is shown for three decoding iterations. A check c from the original lattice may have several associated vertices in the decoding tree, denoted as v_c, v'_c. Similarly, each qubit from the original lattice may have several associated edges in the decoding tree (*e.g.*, the edge (v_1, v_2) and one of the dangling edges of v'_2 are both associated with the qubit incident to checks 1 and 2 in the original lattice). We consider the decoding tree as being rooted in the edge e_q, associated with qubit q from the original lattice (see main text).

For a given qubit q, one may recursively trace back through time the computation of its *a posteriori* value, by examining the updates that have occurred. This trace back will form a tree graph rooted at q and consisting of interconnected qubit and checks in the same way as in the original graph, but the same qubit or checks may appear at several places in the tree because of the loopy structure of the decoding graph. For toric codes, checks are represented by vertices of a square lattice, while qubits are represented by edges (Figure 3.3a). Accordingly, the decoding tree of the toric code will have vertices and edges representing, respectively, checks and qubits at different decoding iterations. Decoding trees can be formally defined by using the notion of walk without return, introduced below.

Definition 16. *A* **walk** W *in a graph* $G = (V, E)$ *as a sequence of edges* $W = e_1 e_2 \dots e_k$*, for which there exists a sequence of vertices* $v_1 v_2 \dots v_{k+1}$*, such that*

$$e_i = (v_i, v_{i+1}), \quad \forall i \in \{1, k\}.$$

We say that W *is a* **walk without return (wwr)** *if* $v_i \neq v_{i+2}, \ \forall i \in \{1, \dots, k-1\}$*. The length* $|W|$ *of a walk is the number of edges it contains.*

Definition 17. *A* **path** *is a walk such that all edges are distinct.*

Definition 18. *(For the toric code) The* **decoding tree** $\mathcal{T}_{i,q}$ *is composed of all the wwr of length* i *in the toric code lattice, starting with edge* e_q *(associated to qubit* q*).*

When necessary, we also subscript the vertices with the name of the associated check, so $u_c \in V(\mathcal{T}_{i,q})$ *is a vertex in* $\mathcal{T}_{i,q}$ *associated to check* c*.*

A decoding tree is illustrated in Figure 3.3b. Note that we consider this tree as being rooted in the edge e_q, and that it ends with **dangling edges** (defined below). Technically, we could insert a qubit vertex on each edge, such that the decoding tree would be rooted in a qubit vertex, and terminated with qubit vertices. We will not do this here, and will rather stick to the usual convention for toric codes, where qubits are represented by edges. Remark that since a decoding tree is defined by walks without return, each vertex u has exactly 3 children (see Figure 3.3b).

Definition 19. *A* **dangling edge** *of* $\mathcal{T}_{i,q}$ *is an edge* e *such that* $|\mathcal{N}(e)| = 1$*.*

In the context of decoding trees, we naturally extend the definitions of **walk**, **walk without return** and **path**, that can now eventually start/end with dangling edges.

Definition 20. *The distance between two vertices in* $\mathcal{T}_{i,q}$ *is the length of the path (the number of edges) from one to the other. The distance of a vertex to the bottom is the minimum over the length of all paths that start with that vertex and end with a dangling edge.*

Since the graph representation of the toric code contains loops, several vertices/edges in the decoding tree may be associated to the same check/qubit of the toric code. Therefore, we extend the notation of the syndrome to vertices of $\mathcal{T}_{i,q}$ as: $\mathbf{s}(v_c) = \mathbf{s}(c), \forall v_c \in V(\mathcal{T}_{i,q})$.

Definition 21. *For a decoding tree* $\mathcal{T}_{i,q}$ *and a syndrome* $\mathbf{s}$*, a* **configuration** *is a function* $C : E(\mathcal{T}_{i,q}) \to \{0, 1\}$ *such that:*

$$\sum_{e \in \mathcal{N}(v)} C(e) = \mathbf{s}(v) \bmod 2, \ \ \forall v \in V(\mathcal{T}_{i,q})$$

If $C(e) = 1$*, the edge* e *is said to be* **labeled** *otherwise it is* **unlabeled***.*

Additionally, we are interested to know the labeling of the root edge of the tree, and will call a **root-labeled** *configuration $C_\bullet$ if $C_\bullet(e_q) = 1$ and a* **root-unlabeled** *configuration $C_\circ$ if $C_\circ(e_q) = 0$.*

Definition 22. *We define the* **weight** *of a configuration as:*

$$|C| = \sum_{e \in E(\mathcal{T}_{i,q})} C(e) \quad \in \mathbb{N}$$

Let $\mathfrak{C}$ be the set of all configurations for decoding tree $\mathcal{T}_{i,q}$ and syndrome $\mathbf{s}$. We write

$$\mathfrak{C} = \mathfrak{C}_\bullet \cup \mathfrak{C}_\circ$$

where $\mathfrak{C}_\bullet$ and $\mathfrak{C}_\circ$ are the subset of root-labeled/unlabeled configurations. Theses sets are non-empty since it is always possible to define a configuration starting from the root down to the dangling edges.

Definition 23. *A* **minimal root-labeled/unlabeled configuration** *$C^*_\circ / C^*_\bullet$ is a configuration such that*

$$|C^*_\circ| = \min_{C_\circ \in \mathfrak{C}_\circ} |C| \quad / \quad |C^*_\bullet| = \min_{C_\bullet \in \mathfrak{C}_\bullet} |C|$$

For i.i.d. noise, all qubits have the same *a priori* value, which factors out through all the MS decoding steps, and thus can be set to 1 (which amounts to saying that the maximum likelihood error is the minimum weight one). This will be implicitly assumed in the next theorem and throughout the rest of this work.

Theorem 2 ([92, Section 4.1])**.** *For i.i.d. noise, and given minimal configurations $C^*_\circ$ and $C^*_\bullet$ for the decoding tree $\mathcal{T}_{i,q}$ and syndrome $\mathbf{s}$, the a posteriori value $\mathrm{APP}(q,i)$ computed by the MS for qubit q at iteration i is given by $\mathrm{APP}(q,i) = |C^*_\bullet| - |C^*_\circ|$.*

3.3.2 Links and Alternating Chains

Below, we give a few definitions, and adapt some usual definitions in graph theory to cope with the specificity of decoding trees (taking into account dangling edges).

Definition 24. *A* **link** *is a path in the decoding tree such that its vertex endpoints (if they exist) are associated with unsatisfied checks.*

A link can have dangling edges as endpoints.

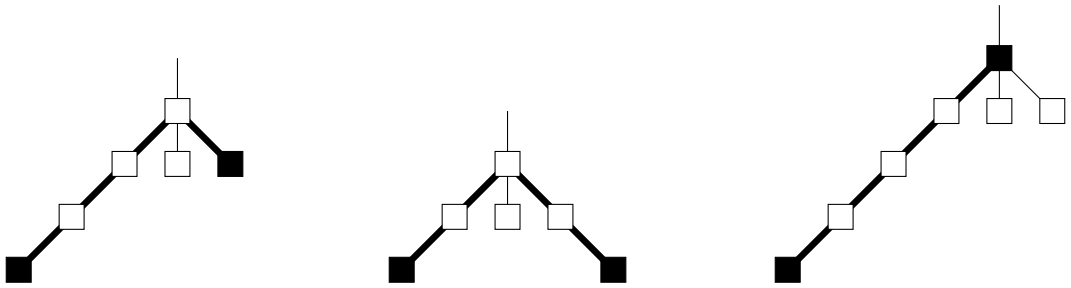
We call a **proper link** *a link with both endpoints associated to unsatisfied checks, and a* **dangling link** *a link where at least one of its endpoints is a dangling edge. For short, we usually only give the vertex endpoints of a link. Giving the endpoints of a proper link is enough to fully*

characterize it, so we write $u \square v$. For a dangling link, we write $u \square \_$. Although this is not enough to uniquely determine a dangling link (there exist many dangling links with endpoint u), whenever this notation is used, it is always considered that this is one of the possible dangling links of minimal length.

Definition 25. *Below we give a name to the 3 possible shapes of proper links of length 4 not containing the root (see Figure 3.4):*

- *The I-link going from an unsatisfied check 4 times down,*
- *The Γ-link going once up then 3 times down,*
- *The Λ-link going twice up then twice down.*

For the I and Γ link, we naturally refer to the upper and lower unsatisfied vertices relative to their depth in the tree.



(a) Γ link (b) Λ link (c) I link

Figure 3.4: The three different shapes of 4-link are depicted in bold. Square vertices represent checks (black for unsatisfied and white for satisfied). Note that in the following the bold notation will only be used to denote labeled links.

Definition 26. *A* **labeled link** *is a link where all edges are labeled. An* **unlabeled link** *is a link where at least one edge is unlabeled. From now on we use the notation* ■ *to denote a labeled link, and* □ *to denote an unlabeled link.*

Definition 27. *An* **alternating chain** *is a walk in the tree consisting of a succession of links alternating between labeled and unlabeled (see Figure 3.5). Subsequent links can have overlapping edges.*

Precisely, an **alternating chain** *is a sequence of links $L_1, L_2, \dots, L_k$ where*

$$\forall L_i = e_1 \dots e_k v_k, \;\; L_{i+1} = v'_1 e'_1 \dots e'_{k'} : \qquad v_k = v'_1$$

$$L_i, L_{i+1} \qquad \text{cannot be both labeled or both unlabeled}$$

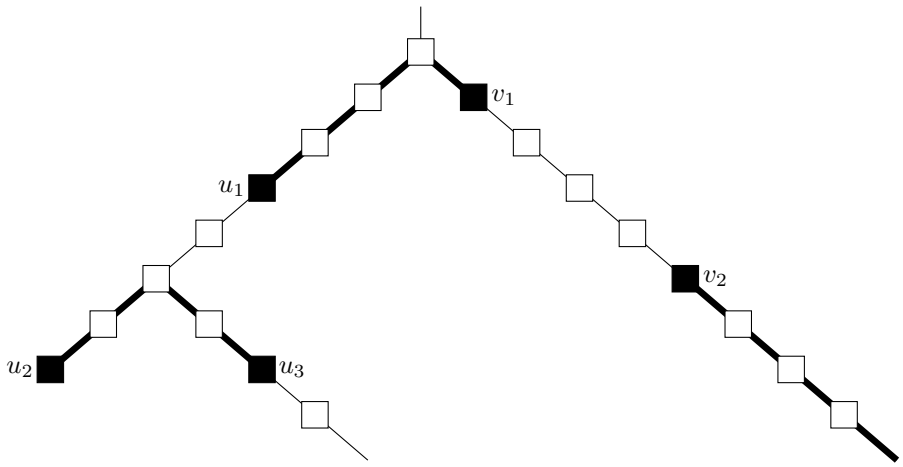


Figure 3.5: An example of an alternating chain $(\_ \square u_3 \overset{\Lambda}{\blacksquare} u_2 \square u_1 \overset{\Gamma}{\blacksquare} v_1 \square v_2 \overset{I}{\blacksquare} \_)$ going from one dangling edge to another (Notice how the two edges above u_2 are used twice, in $u_1 \square u_2$ and $u_2 \blacksquare u_3$). Labeled edges are depicted in bold. All three types of 4-links are depicted here, the Λ and Γ links are **proper**, and the labeled I-link is **dangling**.

Lemma 2 (Walk Inversion Lemma)**.** *Let C be a configuration for a decoding tree $\mathcal{T}_{i,q}$ and syndrome $\mathbf{s}$, and $W = e_1 \dots e_k$ a walk in $\mathcal{T}_{i,q}$ such that both its endpoints are dangling edges. Let C' : $E(\mathcal{T}_{i,q}) \to \{0,1\}$ be the map defined by*

$$\forall e \in E(\mathcal{T}_{i,q}),\ C'(e) = (C(e) + (-1)^{n_e})/2 \text{ where } n_e = |\{e_i = e, e_i \in W\}|$$

Then C' is a configuration for $\mathbf{s}$. Put differently, inverting the label of each edge as many times as it appears in the walk preserves the property of being a configuration for a syndrome.

Proof. Any vertex will touch an even number of edges (counting multiplicities) in W. Hence inverting the labels as many times as they appear along W will keep the parity of all the vertices in the new configuration. □

3.3.3 Minimal Configurations in the MS Decoding Tree

In this section we prove the existence of minimal configurations with special properties, which will be used later in the proof of Theorems 1 and 2.

Lemma 3 (4-links Lemma for root-unlabeled configuration)**.** *Given a syndrome where all unsatisfied checks are at distance 4 or more from one another, there exist a minimal configuration $C_\circ^{\leq 4}$ where all labeled links are of length ≤ 4.*

Proof (Sketch). We give the proof for the existence of $C_\circ^{\leq 4}$ in the form of an algorithm that takes any minimal configuration $C_\circ^*$ and transforms it into a minimal configuration $C_\circ^{\leq 4}$ that satisfies

the condition that all labeled links are of length at most 4. The algorithm works as follows: as long as there is a labeled link of length ≥ 5, we find a path from dangling edge to dangling edge containing it, then use Lemma 2 to remove it. The path is chosen so that the new configuration stays minimal. The details of the proof can be found in Section 3.6.2 □

We now prove a similar statement for $C_\bullet$, but only for decoding trees associated with any of the four qubits neighboring an unsatisfied check[2].

Lemma 4 (4-link Lemma for root-labeled configuration)**.** *Given a syndrome where all unsatisfied checks are at distance 4 or more from one another, for a decoding tree $\mathcal{T}_{i,q}$ where q is neighboring an unsatisfied check c, there exists a minimal configuration $C_\bullet^{\leq 4}$ where all labeled links are of length ≤ 4.*

Proof (Sketch). Starting with a minimal configuration $C_\bullet^*$, and using the algorithm from the proof of Lemma 3, one can remove all the length ≥ 5 labeled links not containing the root. Note that even if the algorithm could be applied on a labeled link containing the root, this would yield an unlabeled configuration which is not what we want. Supposing that there is a labeled link containing the root of length ≥ 5, it will also be removed using the walk inversion Lemma, but the path used will be created differently. The details of the proof can be found in Section 3.6.3. □

Lemma 5. *Given a syndrome where all unsatisfied checks are all at distance at least 5, and a configuration $C^{\leq 4}$ such that all labeled links are of length ≤ 4, then all proper labeled links have endpoints associated with a same check c (i.e., all proper links are of the form $u_c \blacksquare v_c$ for some unsatisfied check c).*

Proof. Since all labeled links are of length ≤ 4 and all unsatisfied checks are at distance ≥ 5, any proper labeled link necessarily joins vertices associated to the same check (by following a path around a plaquette of the toric code). □

3.4 MS Local Blindness

We now prove Theorem 1 that is quite short and intuitive, before conjecturing similar results for NMS and BP.

3.4.1 Proof of Theorem 1

We will prove Theorem 1 by investigating the minimal configurations in the decoding tree that exist thanks to Lemma 3 and 4, and the property of the labeled links from Lemma 5. Here we

[2] It can be shown that the statement does not hold for any other qubit.

restate the theorem for the reader's convenience.

Theorem 1 (MS Local Blindness)**.** *Consider an error syndrome* $\mathbf{s}$ *on a toric code, such that all the unsatisfied checks are at distance at least 5 from each other. Then, under the i.i.d. noise assumption, the MS decoding of* $\mathbf{s}$ *is locally blind in the neighborhood of any unsatisfied check.*

Proof. Let $\mathbf{s}$ be a syndrome such that all the unsatisfied checks are at distance ≥ 5 from each others, c be an unsatisfied check in $\mathbf{s}$, and $q \in \mathcal{N}(c)$. Let $\mathbf{s}^c$ be the fake syndrome (defined in Section 3.1) where only c is unsatisfied , and $\mathbf{s}^{\neq c} = \mathbf{s} - \mathbf{s}^c$ (informally, we can think of $\mathbf{s}$ as the disjoint union of $\mathbf{s}^c$ and $\mathbf{s}^{\neq c}$).

Let also $\mathcal{T}_{i,q}$ be the decoding tree associated to q at iteration i. On $\mathcal{T}_{i,q}$, we now consider some minimal configurations $C_\bullet, C_\circ$ associated to syndrome $\mathbf{s}$, and $C_\bullet^c, C_\circ^c$ associated to $\mathbf{s}^c$, such that all four configurations satisfy Lemmas 3, 4, and 5. Now, considering $C_\bullet$ (the same will hold for $C_\circ$), we define $C_{\bullet|c}, C_{\bullet|\neq c} : E(\mathcal{T}_{i,q}) \to \{0,1\}$ where:

(i) $C_{\bullet|c}$ is defined by the labeled links in $C_\bullet$ that have all their vertex endpoints equal to c (*i.e.*, the proper labeled links of the form $u_c \blacksquare v_c$ and the dangling labeled links of the form $u_c \blacksquare$ _). Formally,

$$C_{\bullet|c}(e) = 1 \iff e \in u_c \blacksquare v_c \text{ or } e \in u_c \blacksquare \text{ \_}$$

Clearly $C_{\bullet|c}$ is a configuration for $\mathbf{s}^c$.

(ii) $C_{\bullet|\neq c}$ is defined by the remaining labeled links (that is, labeled links with endpoint(s) different from c). Alternatively,

$$C_{\bullet|\neq c} = C_\bullet - C_{\bullet|c}$$

Using the fact that all proper links are between vertices associated to the same check, it holds that $C_{\bullet|\neq c}$ is a configuration for $\mathbf{s}^{\neq c}$.

Since $C_\bullet$ is minimal, we conclude that $C_{\bullet|c}$ is a minimal root-labeled configuration for $\mathbf{s}^c$, and $C_{\bullet|\neq c}$ is a minimal root-unlabeled configuration for $\mathbf{s}^{\neq c}$.

Similarly, we can write $C_\circ$ as $C_\circ = C_{\circ|c} + C_{\circ|\neq c}$, where $C_{\circ|c}$ is a minimal root-unlabeled configuration for $\mathbf{s}^c$, and $C_{\circ|\neq c}$ is a minimal root-unlabeled configuration for $\mathbf{s}^{\neq c}$. It follows that:

$$|C_{\bullet|\neq c}| = |C_{\circ|\neq c}|, \quad |C_{\bullet|c}| = |C_\bullet^c|, \text{ and } \quad |C_{\circ|c}| = |C_\circ^c|$$

Therefore,

$$\begin{aligned}
\text{APP}(q,i) &= |C_\circ| - |C_\bullet| \\
&= |C_{\circ|c}| + |C_{\circ|\neq c}| - (|C_{\bullet|c}| + |C_{\bullet|\neq c}|) \\
&= |C_{\circ|c}| - |C_{\bullet|c}| \\
&= |C^c_\circ| - |C^c_\bullet| \\
&= \text{APP}^c(q,i)
\end{aligned}$$

So the blindness property is satisfied for any qubit q neighboring c, which concludes the proof of Theorem 1. □

Corollary 2.1. *Such a syndrome is undecodable by the MS*

Proof. The blindness property for Theorem 1 implies that each of the four qubits around an unsatisfied check has the same same *a posteriori* value (thus the same error estimate), at each iteration, as its horizontal/vertical symmetric. Hence the parity of each unsatisfied check is never satisfied, for any number of iterations, and as such, this syndrome is undecodable by the MS. □

Finally, we note that the blindness property can actually be proven for qubit-to-check messages entering an unsatisfied check, the computation tree of such a message being given by the either left of right half of the decoding tree in Figure 3.3b (with root edge e_q becoming a dangling edge). Consequently, it also applies to check-to-qubit messages going out from an unsatisfied check, since such outgoing messages only depend on the incoming ones.

3.4.2 Conjectures about Normalized MS and BP

For Normalized MS (NMS), we conjecture that Theorems 1 holds for any normalization factor in $]0,1[$. To support our claim, we did some numerical testing for codes with minimum distance $d \in \{11, 24\}$ (giving a "small" and a "larger" code, odd and even size), normalization factor λ between 0.0625 and 1, with a step of 0.0625, and for a maximum number of decoding iterations equal to 100. Note that since the normalization factors are sums of powers of two, we do not have to worry about numerical instability and check equality up to floating point precision. Verifying the statement for all error syndromes with all the unsatisfied checks at distance ≥ 5 even for reasonably small code sizes is not feasible, so we only tested syndromes of weight 2, satisfying the condition that the two unsatisfied checks are at distance ≥ 5. Also to support even more our claim, we tested the syndrome defined by $\mathbf{s}(c_{i,j}) = 1 \iff j = 5i \mod 13$ on toric codes of distance $d = 13$ (Figure 3.6) and $d = 26$. This is a syndrome "packed" with as many unsatisfied

checks as possible, all at distance 5 or 6 from each other. In all our numerical experiments, the blindness property is conserved.

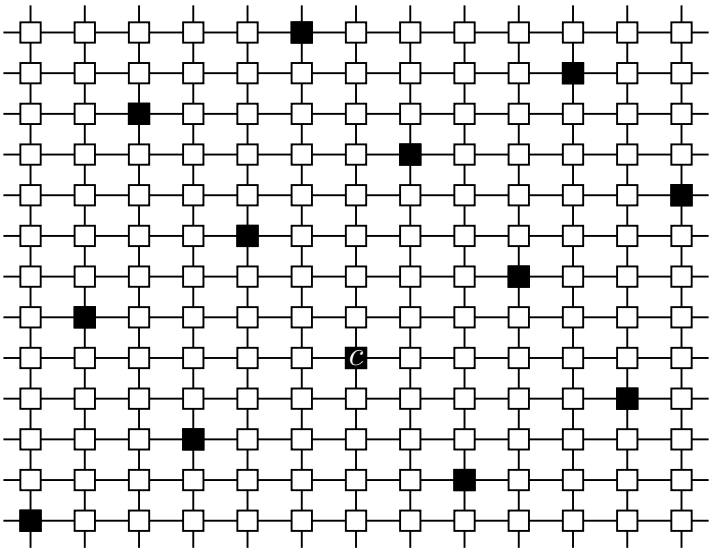


Figure 3.6: Syndrome defined by $\mathbf{s}(c_{i,j}) = 1 \iff j = 5i \mod 13$ on a toric code of distance $d = 13$. Note that this is not a valid error syndrome (as it is of odd weight), but it is only used to verify the blindness property. For $d = 26$ we get a valid error syndrome.

For BP, we know the blindness statement of Theorem 1 to be false, since BP takes into account all the configurations in the decoding tree (for a given syndrome), and not only a minimal one (see [92] for more details). However, the minimal configurations are still somehow dominant in the *a posteriori* reliabilities computed by the BP, so we wonder if a similar statement could be made, perhaps using a relaxed notion of blindness. So far, our preliminary numerical results suggest that syndromes satisfying the hypothesis of Theorem 1 are not decodable by BP, and moreover $\lim_{i\to\infty} (\mathrm{APP}(q,i) - \mathrm{APP}^c(q,i)) = 0$, which is likely explained by the fact that minimal configurations become more dominant as the decoding tree size increases.

3.5 MS Non-Degenerate Decoding Radius

This section gives the proof of Theorem 2. The proof is rather technical, and can be skipped if necessary. Note that the definitions and lemmas shown for the lower bound on Theorem 2 (Section 3.5.3) will be reused in Section 4.4, so feel free to come back to this section at this time.

Here we first restate the Theorem for the reader's convenience.

Theorem 2 (MS Non-Degenerate Decoding Radius)**.** *For any toric code of distance ≥ 9, the non-degenerate decoding radius of the MS is 3.*

We first show an analog of the Lemma 5 in Section 3.5.1 that gives us minimal configurations with good properties where I and Γ links are between copies of the same check. Then, using the results from 3.5.1, we show an upper bound on the non-degenerate decoding radius by showing that $\mathbf{s}^4$ is undecodable. Using a notion of pruning of the decoding tree, we are able to split the decoding tree into a pruned tree where the only remaining unsatisfied vertices are associated to c, and removed subtrees where there might still be vertices associated to c, but with the guarantee (that will be the bulk of the proof) that there are no labeled edges between the removed subtrees and the pruned tree.

Finally, in Section 3.5.3 we show a lower bound on the non-degenerate radius by showing that all non-degenerate errors of weight ≤ 3 are decodable. We do it by a formal argument for codes of larger distance ($d \geq 18$) by introducing the notion of δ-local syndrome, and we verify it numerically for smaller codes ($9 \leq d < 18$).

3.5.1 A Minimal Configuration With I and Γ Links Joining Copies of the Same Check

We first show a preliminary Lemma before showing the analog of Lemma 5.

Lemma 6. *For the syndrome $\mathbf{s}^4$, and for any decoding tree $\mathcal{T}_{i,q}$ associated to qubit q neighboring an unsatisfied check c:*

Let $u \square v$ be a Λ-link in $\mathcal{T}_{i,q}$ and S be the smallest subtree of $\mathcal{T}_{i,q}$ containing it, then for any vertex $w \in V(\mathcal{T}_{i,q}) \setminus V(S)$ associated to an unsatisfied check,

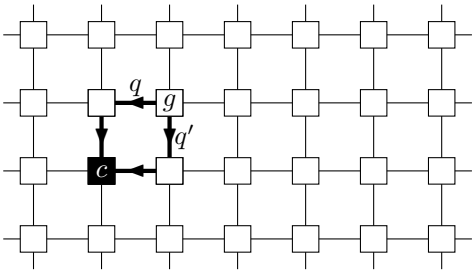
$$d(u, w) = d(v, w) \geq 6$$

Proof. Suppose there is a Λ-link $u_c \square v_{c'}$. We now consider the corresponding walk without return between the checks c and c' on the toric code. There are two possibilities, either $c = c'$ (Figure 3.7a), or $c \neq c'$ (Figure 3.7b). In any case, let g be the toric code check associated with the grandfather of u_c and $v_{c'}$ in $\mathcal{T}_{i,q}$. On the toric code, let q and q' the qubits incident to g on the related walks without return to c and c'. Then any wwr from g to c or c' and starting with $q'' \notin \{q, q'\}$ is of length ≥ 4. Hence, any vertex associated to an unsatisfied check in $V(\mathcal{T}_{i,q}) \setminus S$ must be at distance ≥ 6 from u_c and v'_c. □

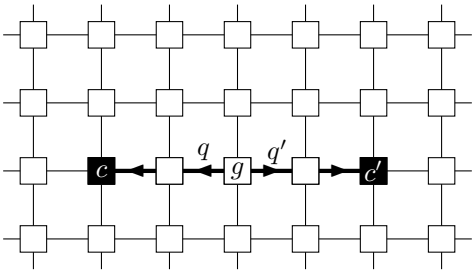
The following corollary is a straight application of Lemma 6.

Corollary 2.2. *Under the hypothesis of Lemma 6, let $u \square v$ be an I-link with v being its the bottom endpoint. Then v cannot belong to a Λ-link.*

The following Lemma is a (weaker) analog of Lemma 5 for the syndrome $\mathbf{s}^4$.



(a) Λ-link between two vertices associated to the same check, visualized on the toric code.



(b) Λ-link between two vertices associated to different checks, visualized on the toric code.

Figure 3.7: The cases of Λ-link

Lemma 7. *For the syndrome $\mathbf{s}^4$, and for any decoding tree $\mathcal{T}_{i,q}$ associated to qubit q neighboring an unsatisfied check c, there exists a minimal configuration such that the endpoints of any I or Γ labeled link are vertices associated to the same check.*

Proof. Take a configuration having all labeled links of length ≤ 4. Suppose there is a labeled link between u_c^0 and $v_{c'}^0$, and take one deepest if there are several. If applying, always consider u^0 always be on a higher level than v^0.

(i) If this link is an I-link or a Γ-link, then there must exist an unlabeled I-link $u_c^0 \square u_c^1$ where u_c^1 is at the same depth as v^0. Then set $\mathcal{A} = \{u_c^1 \square u_c^0, u_c^0 \square v_{c'}^0\}$ The link $u_c^0 \square u_c^1$ must be unlabeled as otherwise the link $u_c^1 \blacksquare v_c^0$ of length ≥ 5 would contradict the assumptions of Lemma 3.

(ii) If the link contains the root edge q (in case of a labeled configuration), then there must also exists an unlabeled link $u_c^0 \square u_c^1$ where u_c^1 is at the same depth as v^0 using the same argument. Then set $\mathcal{A} = \{u_c^1 \square u_c^0, u_c^0 \square v_{c'}^0\}$.

Now complete this into an alternating chain from dangling edge to dangling edge by doing the following iterative algorithm that satisfies the property:

(P) *At each iteration of the algorithm, the chain endpoints are at the same depth on both sides*

At a given iteration, suppose without loss of generality that u^i came from a labeled link and v^i from an unlabeled link. Then pick one labeled link from v^i going down. There must be one and it must be unique since we assumed this is a valid labeling, and by Lemma 6, this cannot be a labeled Λ-link. Also pick an unlabeled link of the same type in the neighborhood of u^0. There must exist one, otherwise this would give a labeled link of length ≥ 5.

This algorithm gives a path from dangling edge to dangling edge. Now inverting this path will remove the link between u_c^0 and $v_{c'}^0$ while maintaining exactly the same weight, hence keeping the minimality of the configuration. □

3.5.2 Upper Bound on the Non-Degenerate Decoding Radius

Here we first introduce formally the notion of pruning, before using it to prove that the decoder is blind on $\mathbf{s}^4$.

Definition 28. Pruning

Given a decoding tree $\mathcal{T}_{i,q}$, we define the pruning of $\mathcal{T}_{i,q}$ as a pair $(\hat{\mathcal{T}}_{i,q}, \mathcal{S})$ where:

- *$\mathcal{S}$ is a set of* **removed subtrees** *(each subtree in $\mathcal{S}$ is rooted in some vertex).*
- *$\hat{\mathcal{T}}_{i,q}$, the* **pruned tree**, *is a connected subgraph of $\mathcal{T}_{i,q}$ containing the root edge e_q, obtained by removing the subtrees in $\mathcal{S}$.*

Since each subtree in $\mathcal{S}$ is routed in a vertex, $\hat{\mathcal{T}}_{i,q}$ has new dangling edges (which are incident in $\mathcal{T}_{i,q}$ to the root of some tree in $\mathcal{S}$). We refer to those as **severed edges**.

We now proceed to show the upper bound on the decoding radius of Theorem 2, by showing that the decoder is blind on the syndrome $\mathbf{s}^4$.

Let c be an unsatisfied check in $\mathbf{s}^4$, the syndrome associated to ϵ^4, and $q \in \mathcal{N}(c)$. Let $\mathcal{T}_{i,q}$ be the decoding tree associated to q at iteration i, and let also $\mathbf{s}^c$ be the fake syndrome where only c is unsatisfied (Section 3.1).

On $\mathcal{T}_{i,q}$, we now consider some minimal configurations $C_\bullet, C_\circ$ associated to syndrome $\mathbf{s}$, and $C_\bullet^c, C_\circ^c$ associated to syndrome $\mathbf{s}^c$ where all labeled links are of length ≤ 4 and for all proper I and Λ links, both endpoints are a copy of the same check (thanks to Lemmas 3,4, and 7).

We now proceed to show that $|C_\bullet| - |C_\circ| = |C_\bullet^c| - |C_\circ^c|$, which implies the blindness property for qubit q by Theorem 2.

Explore $\mathcal{T}_{i,q}$ by doing a breadth-first search. Whenever a vertex $w_{c'}$ associated to a check such that $\mathbf{s}(w_{c'}) = 1$ and $c' \neq c$ is found in $\mathcal{T}_{i,q}$, proceed as follows.

Let g be the grandfather of $w_{c'}$ in $\mathcal{T}_{i,q}$, and prune the subtree rooted in g in $\mathcal{T}_{i,q}$. After doing that for the whole tree, we are left with a pair $(\hat{\mathcal{T}}_{i,q}, \mathcal{S})$.

Note that for any subtree in $\mathcal{S}$, it holds that all unsatisfied vertices are at depth ≥ 2, as otherwise this would give a path in the toric code between unsatisfied checks of length < 4.

Claim 1: In any of the minimal configurations $C_\bullet, C_\circ, C^c_\bullet$ and $C^c_\circ$ all severed edges are unlabeled.

Consider a severed edge, incident (in $\mathcal{T}_{i,q}$) to the root vertex of some removed subtree S. If e is labeled, it must belong to a labeled link. There are 3 cases:

(i) Suppose e belongs to a dangling labeled link v ■ _ with endpoint v in S. Since S was removed, v must be at depth ≥ 2 in S. So v ■ _ must be of length ≥ 7 (2 to go from v to e, plus e, plus at least 4 edges to go back down to a dangling edge in $\hat{\mathcal{T}}_{i,q}$) which is impossible.

(ii) Suppose e belongs to a dangling labeled link v ■ _ with endpoint v in $\hat{\mathcal{T}}_{i,q}$. Since there is an unsatisfied vertex u at depth 2 in S, and that u and v must be at distance ≥ 4, then v is at distance ≥ 5 from the bottom, thus the length of v ■ _ is ≥ 5 which is impossible by assumption.

(iii) Suppose e belongs to a labeled proper link. It must be of the form u_c ■ v_c where $u_c \in \hat{\mathcal{T}}_{i,q}$ and $v_c \in S$ and of length 4. But S was removed because of the unsatisfied vertex $w_{c'}$ with $c' \neq c$, which is at depth 2 in S. Since $d(u_c, v_c) = 4$, v_c is at depth ≥ 2 in S, and $d(u_c, w_{c'}) \geq 4$, we conclude that v_c must be at depth exactly 2 in S, and moreover $d(u_c, v_c) = d(u_c, w_{c'}) = 4$. Thus, $w_{c'}$ □ v_c is a Λ-link with both endpoints at distance 4 from u_c, which yields a contradiction by applying Lemma 6.

Claim 2: Denoting $\hat{C}_\circ, \hat{C}_\bullet, \hat{C}^c_\circ, \hat{C}^c_\bullet$ the restriction of the minimal configurations to the pruned tree $\hat{\mathcal{T}}_{i,q}$, we have that $|\hat{C}_\circ| = |\hat{C}^c_\circ|$ and $|\hat{C}_\bullet| = |\hat{C}^c_\bullet|$.

This is because all the severed edges are unlabeled so the labeling induced by configurations $\hat{C}_\bullet$ and $\hat{C}^c_\bullet$ (resp. $\hat{C}_\circ$ and $\hat{C}^c_\circ$) are interchangeable and of same weight, as otherwise it would contradict the minimality of $C_\bullet$ and $C^c_\bullet$ (resp. $C_\circ$ and $C^c_\circ$).

Claim 3: For every subtree $S \in \mathcal{S}$, since the corresponding severed edge e is unlabeled in both $C_\circ$ and $C_\bullet$, then the restriction of $C_\circ$ and $C_\bullet$ to S have the same weight.

Clearly the minimality of the labeling on such a subtree does not depend on the rest of the tree since e is unlabeled in both cases, so they must have the same weight.

By Claims 1 and 3,

$$|C_\bullet| - |C_\circ| = |\hat{C}_\bullet| - |\hat{C}_\circ| \text{ and } |C^c_\bullet| - |C^c_\circ| = |\hat{C}^c_\bullet| - |\hat{C}^c_\circ|$$

By Claim 2,

$$|\hat{C}_\bullet| - |\hat{C}_\circ| = |\hat{C}^c_\bullet| - |\hat{C}^c_\circ|$$

Hence

$$|C_\bullet| - |C_\circ| = |C^c_\bullet| - |C^c_\circ|$$

3.5.3 Lower Bound on the Non-Degenerate Decoding Radius

We now show that the non-degenerate decoding radius of the MS is ≥ 3, to conclude the proof of Theorem 2. We first give a few definitions and prove two preliminary Lemmas. Note that the notions presented here will be reused in section 4.4 in the proof of Theorem 3.

Given a toric code T, we will refer to a connected planar subgraph τ induced by one or several plaquettes of T as a patch. Precisely, τ is a connected subgraph induced by its vertex set $V(\tau)$, which must satisfy $V(\tau) = \cup_i V(p_i)$, where $V(p_i)$ denotes the set of vertices (checks) of plaquette p_i. It is easily seen that if τ is a patch of T and T' is a toric code of higher minimum distance, then τ can be embedded in T' via a graph monomorphism $\tau \to T'$, and this embedding is unique up to translations, reflections, and rotations in T'. We will use such a patch embedding to embed in T' planar subgraphs of T and syndromes. We note that (i) for any planar subgraph σ of T there exists a patch τ in T such that σ is a subgraph of τ, and (ii) there exists a patch τ with $V(\tau) = V(T)$.

Definition 29. *Let T and T' be toric codes, with T' of larger minimum distance.*

(i) Consider a planar subgraph σ of T, it must be contained in a patch τ in T. We define the embedding of σ in T' as the subgraph σ' of T' induced by embedding τ in T'. This embedding is unique up to translations, reflections, and rotations in T'.

(ii) Consider a syndrome $\mathbf{s}$ on T, and a minimal connected (necessarily planar) subgraph σ of T covering all its unsatisfied checks. We define the embedding of $\mathbf{s}$ in T' as the syndrome $\mathbf{s}'$ on T' induced by embedding σ in T'. This embedding is unique up to translations, reflections, and rotations in T'.

Definition 30. *For a toric code T, the distance between two sets of checks $C, C' \subseteq V(T)$ is defined as $d(C, C') := \min\{\, d(c, c') \mid c \in C,\ c' \in C' \,\}$. We extend this definition to qubits, errors, syndromes, and subgraphs of the toric code in the following way (where the same extension applies to the second argument of $d(-,-)$):*

- *For a set of qubits $Q \subset E(T)$, $d(Q, -) := d(\mathcal{N}(Q), -)$, where $\mathcal{N}(Q) := \cup_{q \in Q} \mathcal{N}(q)$.*
- *For a syndrome $\mathbf{s}$, $d(\mathbf{s}, -) := d(\mathrm{supp}(\mathbf{s}), -)$, where $\mathrm{supp}(\mathbf{s}) := \{c \in V(T) \mid \mathbf{s}(c) = 1\}$*
- *For an error $\mathbf{e}$, $d(\mathbf{e}, -) := d(\mathrm{supp}(\mathbf{e}), -)$, where $\mathrm{supp}(\mathbf{e}) := \{q \in E(T) \mid \mathbf{e}(q) = 1\}$.*

- *For a subgraph σ of T, $d(\sigma, -) := d(V(\sigma), -)$, where $V(\sigma)$ is the vertex set of σ.*

Definition 31. *The* **diameter** *of a syndrome $\mathbf{s}$ is the minimum over the diameters of all the connected subgraphs covering its unsatisfied checks.*

Definition 32. *For a toric code, a δ-***local syndrome** $\mathbf{s}$ *is a syndrome where all unsatisfied checks are contained in a connected subgraph of diameter δ.*

Definition 33. *We say a syndrome $\mathbf{s}$ is* **decoded in k iterations** *if the MS decoder finds an error $\mathbf{e}$ such that $\mathbf{He} = \mathbf{s}$ at iteration k and not before.*

Lemma 8. *Let T be a toric code of distance d, and $\mathbf{s}$ be a δ-local syndrome decoded in k iterations. If $d \geq 2k + \delta$, then for any toric code T' of distance $d' > d$, the embedding of $\mathbf{s}$ in T' will be corrected in k iterations.*

Proof. Let σ be the connected subgraph containing all the edges at distance $\leq \delta$ from $\mathbf{s}$, and their check neighborhood, that is:

$$E(\sigma) = \{e \in E(T) \mid d(e, \mathbf{s}) \leq \delta\} \text{ and } V(\sigma) = \bigcup_{e \in E(\sigma)} \mathcal{N}(e).$$

Note that σ is planar as $\delta < d$ so we can define σ' to be the embedding of σ in T'. For any qubit $q' \in T'$, either $d(q', \mathbf{s}) \geq k$ in which case one can see that its *a posteriori* at each iteration will be $\text{APP}(q', i) = 1 + 2i$, for $i \leq k$ or $d(q', \mathbf{s}) < k$, in which case, $q' \in \sigma'$, and at each iteration, its *a posteriori* is the same as that of $q \in \sigma$, its corresponding qubit in T. This is because since $d \geq 2k + \delta$, there are no qubits in T that have a decoding tree reaching unsatisfied checks of $\mathbf{s}$ by going around the toric code. Formally, this condition can be stated as: given τ a minimal connected covering of $\mathbf{s}$, then for all q in T, $E(\tau) \cup \{e \in E(T),\ d(e, q) \leq k\}$ does not contain a logical operator. Thus at iteration k, the decoder will converge on the embedding of $\mathbf{s}$ on T'. □

Lemma 9. *Let T be any toric code, $\mathbf{s}_1$ a syndrome decoded in k_1 iterations and $\mathbf{s}_2$ a syndrome decoded in k_2 iterations. Further assume that $k_1 \leq k_2$, and that if the decoder were left to run on $\mathbf{s}_1$ for any number $k \in [k_1, k_2]$ of iterations (even after the syndrome is satisfied), the error outputted at each iteration would still satisfy the syndrome. Let δ be the distance between $\mathbf{s}_1$ and $\mathbf{s}_2$. If $\delta \geq 2k_2$ then the syndrome $\mathbf{s}_1 + \mathbf{s}_2$ is decodable in k_2 iterations.*

Proof. For any qubit q, either is true:

(i) $d(\mathbf{s}_1, q) \geq k_2$ and $d(\mathbf{s}_2, q) \geq k_2$. In this case the decoding tree of q does not contain unsatisfied checks for any iteration $k \leq k_2$. So $\text{APP}(q, k) = 1 + 2k$.

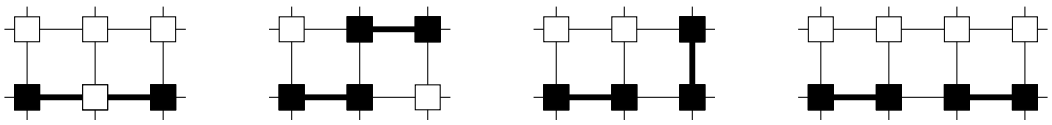


Figure 3.8: The weight 2 errors of diameter 3 (up to symmetries and translation), all are decodable in 2 iterations of the MS.

(ii) There exist i, j with $\{i, j\} = \{1, 2\}$ such that $d(e_i, q) < k_2$ and $d(e_j, q) \geq k_2$

In that case the decoder will take the same decision on q as if the syndrome was only $\mathbf{s}_i$.

At iteration k_2, by (ii) in the k_2-neighborhood of $\mathbf{s}_1$ and $\mathbf{s}_2$, both syndromes are satisfied, and by (i) all the other qubits are not in error, so the syndrome $\mathbf{s}_1 + \mathbf{s}_2$ is satisfied. □

We are now ready to finish the proof of Theorem 2. We do this by treating the first cases numerically, and then use the Lemmas 8, 9 introduced before to prove the result for $d \geq 18$.

Proof. For any toric code of distance $9 \leq d < 18$, all the non-degenerate errors of weight ≤ 3 are decodable by the MS algorithm. This is checked by exhaustive numerical verification.

For toric codes of minimum distance ≥ 18, we do the following :

(i) It can be easily checked that all weight 1 errors are non-degenerate and can be decoded by the MS in 1 iteration, for any toric code of distance $d \geq 3$.

(i') Moreover, it is also true that for a weight 1 error on a toric code of distance ≥ 5, if the decoder were allowed to continue after the first iteration (when it already decodes the error), at the second iteration it would still output the same weight 1 error.

(ii) Any weight 2 non-degenerate error where the two qubits in error are at distance ≥ 2 will be decodable in 1 iteration because of (i) and Lemma 9 for any toric code of distance $d \geq 6$.

(ii') For the remaining non-degenerate errors of weight 2, those where the distance between the two qubits in error is at most 1 (*i.e.*, of diameter 3), there are only 4 types of error up to symmetries and translations (shown in Fig. 3.8). For those, we check numerically that the decoder converges in at most 2 iterations on a toric code of distance 7.

Using Lemma 8, since $2k + \delta = 2 \times 2 + 3 = 7$, we conclude that all the errors of weight 2 and diameter 3 are corrected on a toric code of any distance $d \geq 7$.

(iii) Any weight 3 error where the distance between any two pair of qubits is ≥ 2 is correctable in 1 iteration by Lemma 9 for any toric code of distance ≥ 3.

(iii') Remaining errors can be written as the sum of an error $\mathbf{e}_1$ of weight 1 (of diameter 1) and an error $\mathbf{e}_2$ of weight 2 and of diameter ≤ 3 (depicted in Figure 3.8). Suppose $d(\mathbf{e}_1, \mathbf{e}_2) \geq 4$. By (i), $\mathbf{e}_1$ is decoded in 1 iterations. and by (ii'), $\mathbf{e}_2$ is decoded in ≤ 2 iterations. We can thus use Lemma 9 on the syndromes associated to $\mathbf{e}_1$ and $\mathbf{e}_2$ to conclude that $\mathbf{e}_1 + \mathbf{e}_2$ is decoded in 2 iterations. Note that this argument is true for any error of weight 3 satisfying the condition $d(\mathbf{e}_1, \mathbf{e}_2) \geq 4$ irrespective of the distance of the code.

(iii'') For the remaining finite number of weight 3 errors of diameter $\delta \leq 8 = 4 + 1 + 3$ (where $d(\mathbf{e}_2, \mathbf{e}_2) \leq 4$, the diameter of $\mathbf{e}_1$ is 1 and the diameter of $\mathbf{e}_2$ is at most 3). We check numerically that these errors are decodable in ≤ 5 iterations on a toric code of distance 18. By applying Lemma 8, and since $2k + \delta = 2 \times 5 + 8 \leq 18$, we conclude that for any toric code of distance $d \geq 18$, all weight 3 non-degenerate errors are decodable.

□

3.6 Proofs of Lemmas 3 and 4

Here we provide the full proof of Lemmas 3 and 4. We first start by introducing a notion a maximal links and prove some properties on them in 3.6.1.

3.6.1 Properties of Maximal Links

Definition 34. *A labeled link is said* **maximal** *if it cannot be extended into a longer labeled link. It is said* **robustly maximal** *if there is no other labeled link sharing at least one edge with it that is of greater length.*

Lemma 10. *Given a configuration C, and a maximal proper labeled link $u \blacksquare v$. Then at least 2 of the 3 edges going down from v are unlabeled.*

Proof. Since v is the endpoint of a maximal labeled link, it is either true that:

(i) u belongs to the subtree rooted in v, in which case the only labeled edge going down from v is the one belonging to the labeled link $u \blacksquare v$, as otherwise $u \blacksquare v$ could be extended and would not be maximal.

(ii) On the other hand, if u does not belong to the subtree rooted in v, all three edges below v are unlabeled as otherwise the labeled link $u \blacksquare v$ could be extended and would not be maximal.

□

Lemma 11. *Given a configuration C for a syndrome where all unsatisfied checks are at distance ≥ 4, and a maximal proper labeled link $u \blacksquare v_c$, where v_c is at distance ≥ 5 from the bottom. Then there exists an unlabeled I-link $v_c \square v_c'$ such that v_c is its upper endpoint.*

Proof. By Lemma 10, there must be one unlabeled edge e_q going down from v_c. In the toric code, there exists a path of length 4 that starts from c, goes through qubit q and loops back to c. Hence, since e_q is unlabeled, there exist an unlabeled I-link $v_c \square v_c'$. □

Lemma 12. *Given a minimal configuration C for a syndrome where all unsatisfied checks are at distance ≥ 4, and a maximal proper labeled link $u \blacksquare v$ of length ≥ 4, where v is a distance $d \leq 4$ from the bottom. Then any dangling link $v \square \_$ is of length d and unlabeled.*

Proof. We note that the subtree rooted in v does not contain any unsatisfied vertex (since all vertices in the subtree are at distance ≤ 3 from v). In particular, the proper link $u \blacksquare v$ must come from above. Since $u \blacksquare v$ is maximal, it cannot be extended inside the subtree rooted in v, and since C is minimal it follows that all the edges in this subtree are unlabeled. Thus, any dangling link $v \square \_$ is of length d and unlabeled. □

3.6.2 Proof of Lemma 3

Let $C_\circ^*$ be any minimal root-unlabeled configuration on the decoding tree $\mathcal{T}_{i,q}$.

Case 1 (Proper labeled link $u^0 \blacksquare v^0$): Suppose there is a proper labeled link $u^0 \blacksquare v^0$ of length ≥ 5. If there are several choices, it is always possible to take one that is robustly maximal and as deep as possible (*i.e.*, that in the subtrees rooted in u^0 and v^0, there are no labeled links of length ≥ 5, except possibly $u^0 \blacksquare v^0$).

Now recursively create an alternating chain $\mathcal{A}$ containing the labeled link $u^0 \blacksquare v^0$ that goes from one dangling edge to another. This alternating chain will have the property:

$$P : \text{all labeled links in } \mathcal{A} \text{ are maximal}$$

We start with $\mathcal{A} = \{u^0 \blacksquare v^0\}$. Set $i = 0$ and do recursively starting from u^0 (and later do the same for v^0):

(i) i even: an unlabeled link must be appended. If u^i is at distance ≥ 5 from the bottom, then there exists an unlabeled I link $u^i \square u^{i+1}$ such that u^i is its upper endpoint (by Lemma 11). Set $\mathcal{A} = \mathcal{A} \cup \{u^i \square u^{i+1}\}$[3].

[3]Strictly speaking, an alternating chain is an ordered sequence of links so the union should be seen as preppending/appending the link to the ordered list.

Otherwise, u^i is at distance $d \leq 4$ from the bottom, and there exists a dangling unlabeled link $u^i \square \_$ of length d (by Lemma 12). Set $\mathcal{A} = \mathcal{A} \cup \{u^i \square \_\}$ and terminate the algorithm on this side.

(ii) i odd: a labeled link must be appended. Take any labeled link having u^i as endpoint (we show below that is necessarily maximal). If it is a proper labeled link $u^i \blacksquare u^{i+1}$, set $\mathcal{A} = \mathcal{A} \cup \{u^i \blacksquare u^{i+1}\}$.

Otherwise, it is a dangling link $u^i \blacksquare \_$. Set $\mathcal{A} = \mathcal{A} \cup \{u^i \blacksquare \_\}$ and terminate the algorithm on this side.

Since $u^0 \blacksquare v^0$ was taken to be as deep as possible, any time a labeled link is appended, it must be of length ≤ 4, and if it is a proper link, since all vertices associated to unsatisfied checks are at distance ≥ 4, it must be of length exactly 4, and thus maximal, so property P is conserved throughout the algorithm.

This algorithm will always terminate because, one of the following is always true:

(i) When adding an unlabeled link, either the algorithm terminates on this side or an I-link was appended, in which case the depth increases by 4.

(ii) When adding a labeled link, either the algorithm terminates on this side, or a proper labeled link $u^i \blacksquare u^{i+1}$ of length 4 has been appended, in which case the depth decreases by at most 3 as u^i is already the bottom endpoint of an unlabeled I link so it cannot also be the bottom endpoint of a labeled I link.

Thus the algorithm must end.

We now have an algorithm to construct an alternating chain that goes from dangling edge to dangling edge. This is not enough to use the inversion Lemma directly, so we further modify it to ensure that there is no dangling unlabeled link of length ≥ 3.

Suppose there is one unlabeled dangling link of length ≥ 3, without loss of generality write it $u^k \square \_$. Then since u^k it at distance ≥ 3 from the bottom, u^k must be the upper endpoint of a Γ-link $u^k \square u^{k+1}$. This Γ-link must be unlabeled because :

(i) If $k = 0$, $u^{k-1} = v^0$ and $u^{k-1} \blacksquare u^{k+1}$ would be longer than $u^0 \blacksquare v^0$ and this would give a contradiction to the fact that $u^0 \blacksquare v^0$ is robustly maximal.

(ii) If $k \neq 0$, then $u^{k-1} \blacksquare u^{k+1}$ would be a labeled link of length ≥ 6 which would contradict the fact that $u^0 \blacksquare v^0$ was taken as deep as possible.

Now set $\mathcal{A} = \Big(\mathcal{A} \setminus \{u^k \square \_\}\Big) \cup \{u^k \square u^{k+1}\}$.

At this point, we arrive at unsatisfied check u^{k+1} at distance ≤ 2 of the bottom, and we need to add a labeled dangling link to finish our alternating chain. Since u^{k+1} is unsatisfied, it is must be the endpoint of a labeled link. It is impossible that u^{k+1} is the bottom endpoint of a labeled I or Γ link as it would contradict the fact that $u^k \square u^{k+1}$ is unlabeled. So it must either be a labeled dangling link, in which case we are done, or it is a labeled Λ-link. In that case we append it and then finish the alternating chain by appending a length 2 unlabeled dangling link that must exist according to Lemma 12. We now have the alternating chain with all the required properties to finish the proof.

We look at the walk given by the alternating chain (with possible back-and-forth). We do the path inversion according to Lemma 2. This gets rid of the labeled link $u^0 \blacksquare v^0$ of length ≥ 5.

To conclude the proof, we now argue that the configuration obtained by applying the walk inversion Lemma is minimal.

The added cost of labeling each previously unlabeled proper 4-link will be compensated by unlabeling the next proper labeled 4-link, if it exists. As they are both of length 4 and there will be as many edges being unlabeled as being relabeled[4]. We only need to deal with the dangling links at both ends of the alternating chain. We show that whatever weight will be added at both end is compensated by the unlabeling of $u^0 \blacksquare v^0$, thus keeping the minimality of the configuration.

There are several cases to consider. On both sides, the chain can end with either an unlabeled link of length 1 or 2, or a labeled link of length 1, 2, 3 or 4, preceeded by an unlabeled I link of length 4. Consider one end of the alternating chain, and denote these cases as U_1, U_2 and $L_1, \ldots, L_4$ respectively.

- U_1,U_2: adds 1 or 2 to the weight
- L_1: adds 3 to the weight.
- L_2,L_3,L_4: adds respectively 2, 1 or 0 to the weight

The only troublesome case is L_1, since if that were to happen on both ends of the alternating chain, the added weight of 6 might not be compensated by unlabeling of a weight 5 labeled link $u^0 \blacksquare v^0$. A simple parity argument takes care of this corner case: in case the $u^0 \blacksquare v^0$ is of length 5, u^0 and v^0 must be at a depth of different parity in the tree[5]. Since at each step adding a 4-link conserves the parity of the depth, it cannot be that the case L_1 happens on both ends.

[4] It should be easy to convince oneself that this is also true when there are overlapping edges.

[5] The only case where this is not true is when the link contains the root but this case is purposefully avoided in the Theorem

Case 2 (Dangling labeled link u^0 ■ _): Suppose there is a dangling labeled link with endpoint u^0 of length ≥ 5. The same algorithm can be applied from u^0, except the argument for the minimality has to be (easily) adapted from Case 1.

Recursive Application of the Algorithm: By applying the algorithm as long as there are labeled links of length ≥ 5 (except possibly if it contains the root), this algorithm transforms any minimal root-unlabeled configuration in a minimal root-unlabeled configuration where all labeled links not containing the root are of length ≤ 4.

3.6.3 Proof of Lemma 4

Starting with a minimal configuration $C^*_\bullet$, and using the algorithm from the proof of Lemma 3, one can remove all the length ≥ 5 labeled links not containing the root.

Now suppose there is a labeled link of length ≥ 5 containing the root e_q. Since q is neighboring an unsatisfied check c, we know that there is an unsatisfied vertex u_c just next to e_q.

Note that the edges going down from u_c must be unlabeled. This is because, since e_q is labeled, to keep the parity of u_c, it must be that two edges below u_c are labeled, hence, that there are two labeled links u_c ■ u' and u_c ■ u'' below it. But then the labeled link u' ■ u'' does not contains the root and must be of length ≥ 8 which is a contradiction. So we conclude that the longest labeled link starting from u_c and going through the root e_q is robustly maximal.

Case 1 (proper labeled link u_c ■ v): Suppose the labeled link containing the root is of length ≥ 5 of the form u_c ■ v. Let u_c and w be the vertices adjacent to e_q. Since all the labeled links not containing the root are of length ≤ 4, it follows that from the three edges going down from w, only the one contained in the path going to v is labeled. Then, we can take an unlabeled edge e going down from w, such that there exists a proper unlabeled 4-link u_c □ v'_c containing e_q and e.

(i) Suppose v'_c is contained in a labeled proper 4-link v'_c ■ v''_c. In that case, we use a "trick" and run the algorithm from the proof of Lemma 3, as if v''_c □ v was a proper labeled link (we can do this since the edges adjacent to v''_c and v on the v''_c □ v link are labeled; also, note that the link v''_c □ v contains at least 5 labeled edges, which guarantees the minimality of the new configuration, obtained after applying the algorithm). This will invert the labels of the edges on the v''_c □ v, giving us a configuration where the link u_c ■ v'_c is labeled and contains the root e_q, and where all the labeled links are of length 4.

(ii) Suppose v'_c is not contained in a proper link, it must then contained in a dangling labeled

link v'_c ■ _. This dangling labeled link cannot be of length 1 since we know v is deeper than v'_c. So it can be of length 2, 3 or 4. In which case v is the vertex endpoint of an unlabeled dangling link v □ _, of length respectively 1, 2 or 3. In that case inverting the alternating chain $\{v'_c$ ■ _, v'_c □ v, v □ _$\}$ gives a minimal configuration where all labeled links are of length at most 4, and this concludes the proof.

Case 2 (dangling labeled link u_c ■ _): Suppose the labeled link containing the root is a dangling link of length ≥ 5. The same argument as above can be adapted in combination with Case 2 in the proof of Lemma 3 to conclude the proof.

3.7 Conclusion

Using the powerful framework of decoding tree, we were able to show that there is an inherent limitation to the error-correction capability of the MS on the toric code. This was done by introducing the notion of blindness, which not only emphasize the fact that the decoder is unable to converge, but also prove that the spreading of the messages in the code is inherently limited. We also showed that by using a low cost syndrome-aware pre-processing we were able to make the decoding radius and the non-degenerate decoding radius of the MS match. In the vein of what had been done recently on the limitations of the renormalization decoder on the toric code in [77], to the best of our knowledge this is the first formal result explaining why the MS decoder does not perform well on the toric code, and we also conjecture that similar results could be obtained for other iterative decodings (NMS and BP) that have similar decoding processes, and can also be explained using the decoding tree, although in a different manner that do not allow the proofs to transfer directly.

The blindness property we proved here comes from a strong imbalance between the girth and the diameter of the graph. This can be seen as a *graph degeneracy*, complementing the standard notion of *code degeneracy* (*i.e.*, classical minimum distance smaller that the quantum minimum distance), the latter being independent of the Tanner graph representation of the code. It is worth noticing that for well-constructed classical LDPC codes, *e.g.*, constructed by the progressive edge growth algorithm [45], such a graph degeneracy does not happen: the length of the shortest cycle passing through a given check is greater than or equal to the maximum distance between that check and all the others. This ensures fast spreading of messages in the Tanner graph. We suspect that the graph degeneracy could have blindness effects for more general quantum LDPC codes, although the imbalance between girth and diameter would be reduced for finite-length Tanner graphs with increased connectivity degree.

Chapter 4

Stabiliser-Blowup Pre-Processing

This chapter is based on the second part of the preprint [13] where we show that by using a light preprocessing, we are able to get the decoding radius of the Min-Sum to match with its non-degenerate decoding radius.

4.1 Introduction

In a direct followup with the ideas developed in Chapter 3, here we propose a method to remove low-weight degeneracy on the toric code. Although the toric code is know to be the hypergraph product code of two classical repetition codes [86], on which the MS decoding radius is equal to $\lfloor \frac{d-1}{2} \rfloor$, the decoding radius of the MS on the toric code is only equal to 1, and the essence of Theorem 2 is that even the **non-degenerate** decoding radius is no greater than 3. We propose a new pre-processing that runs in linear time and allows correcting all (degenerate) errors of weight up to 3 thus improving the decoding radius of the MS to 3. We refer to this pre-processing step as stabiliser-blowup (SB), and it amounts to applying a change of variable that locally removes the degeneracy to allow the decoder to converge. We formally prove the following Theorem.

Theorem 3 (Stabilizer Blowup Pre-Processing)**.** *SB+MS is able to correct all errors of weight up to 3 on a toric code of distance* ≥ 7*.*

It follows that the SB+MS yields a quadratic improvement in the logical error rate performance, as compared to MS only. In particular, it can be applied to quadratically reduce the number of calls to a possible post-processing step, which remains a must[1] for the MS-based decoding of toric codes of minimum distance $d \geq 9$.

[1] According to the limited performances of state of the art MS improvements, of which the poor performances are likely being explained by the local blindness property from Theorem 1.

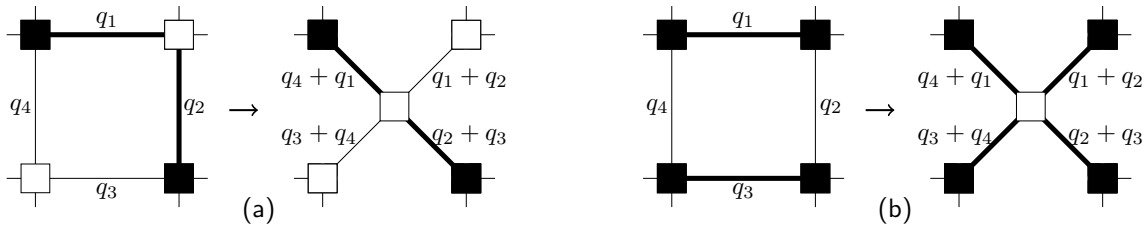


Figure 4.1: Stabiliser-blowup for degenerate errors of weight 2: the four edges around the original plaquette are replaced by four new edges, connecting the checks around the plaquette to a new check in the middle of the plaquette. Labels q_i's on the edges of the original plaquette are interpreted here as binary random variables, equal to 1 if the corresponding qubit is in error, and 0 otherwise. Each new edge is labeled by the sum of the q_i's incident to the same check. (a) The diagonal weight-2 syndrome, shown on the original plaquette together with one of its possible explanations ($q_1 = q_2 = 1$) on the left. After the blowup, the syndrome is uniquely explained as $q_4 + q_1 = q_2 + q_3 = 1$. (b) The square weight-4 syndrome, shown on the original plaquette together with one of its possible explanations ($q_1 = q_3 = 1$). After the blowup, the syndrome is uniquely explained as $q_4 + q_1 = q_1 + q_2 = q_2 + q_3 = q_3 + q_4 = 1$.

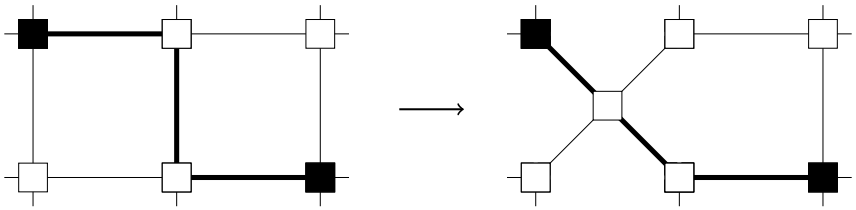


Figure 4.2: Stabiliser-blowup for degenerate errors of weight 3: an example of a weight-3 degenerate error on the original toric code (generating a weight-2 syndrome) becoming a weight-3 non-degenerate error after the blowup.

4.2 The Stabiliser-Blowup Graph Modification

The stabiliser-blowup is a decoding graph modification that allows to locally remove the degeneracy of the code around a plaquette. In the following, we use it as a preprocessing to remove the degeneracy of degenerate weight 2 errors (depicted in Figure 4.1) and degenerate weight 3 errors (depicted in Figure 4.2).

We remove the four edges associated to the qubits around a given plaquette and replace them by four new edges connecting the four corner checks of the plaquette to an additional (satisfied) check in the middle. Each new edge corresponds to the parity of the two removed edges around a given check. Thus, the sum of the four new edges must be even, hence the addition of the satisfied middle check connecting them. We later show that any solution satisfying the syndrome

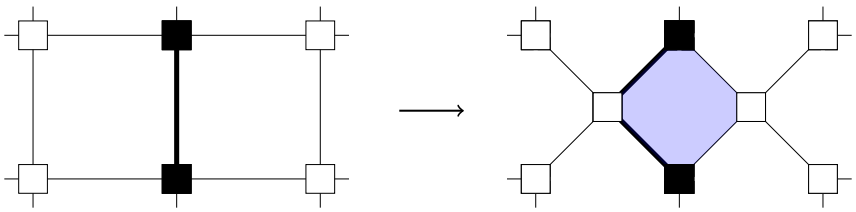


Figure 4.3: Stabiliser-blowup on adjacent plaquettes leads to undecodable weight-1 errors: the weight-1 error on the original toric code becomes a weight-2 degenerate error after the blowup.

on the modified graph will easily yield a valid solution for the original graph, and this solution is unique up to stabiliser equivalence. As a remark, this code change should not be applied on two neighboring plaquettes, as this would create new degeneracy issues between them (see Figure 4.3). Therefore, we use the stabiliser-blowup as a pre-processing, and only apply the blowup where the error syndrome suggests that degeneracy issues might occur. One could ask why those local degeneracy issues are not directly solved "by hand" as a pre-processing, by assigning a fixed value to one of the qubits thus removing the degeneracy. But this could lead to a degradation of the decoding by systematically increasing the weight of the outputted error, thus increasing the likelihood of logical errors, so we instead rely on the MS to decide on the error pattern.

4.3 The Algorithm

In order to correct all errors of weight ≤ 3, it is enough to consider the syndromes of *local* degenerate errors of weight 2 and 3. Whenever one of those patterns appears around a plaquette, we do the blowup, except if an adjacent plaquette (sharing an edge) is already blown-up.

To remove all errors of weight ≤ 3, we use the following **greedy heuristic**: We go through all plaquettes in the lexicographical order of their coordinates. We do this three times, corresponding to steps (i) (ii) (iii) below, each time checking for a different set of patterns. If one of the patterns looked for at this step is found in the neighborhood of a plaquette p, we apply the blowup on p if it has no adjacent plaquette that has been blown-up. The local neighborhood of each plaquette is labeled in Figure 4.4.

Each check is given a number, and in the following we associate to each check a true/false value depending on whether it is unsatisfied/satisfied.

(i) Weight 2 degenerate errors with condition on diagonal

- $\mathbf{s}(c_1) \wedge \mathbf{s}(c_2) \wedge \mathbf{s}(c_3) \wedge \mathbf{s}(c_4)$
- $\overline{\mathbf{s}(c_1)} \wedge \mathbf{s}(c_2) \wedge \overline{\mathbf{s}(c_3)} \wedge \mathbf{s}(c_4) \wedge \Big(\mathbf{s}(c_9) \oplus \mathbf{s}(c_{11}) \oplus \mathbf{s}(c_5) \oplus \mathbf{s}(c_{15}) = 0\Big)$

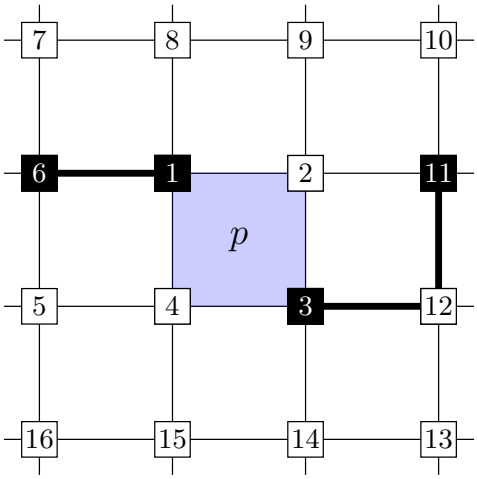


Figure 4.4: The neighborhood of plaquette p, corresponding to the 16 checks in its surrounding. In bold the corner case of the heuristic in (i) is depicted.

- $\mathbf{s}(c_1) \wedge \overline{\mathbf{s}(c_2)} \wedge \mathbf{s}(c_3) \wedge \overline{\mathbf{s}(c_4)} \wedge \Big(\mathbf{s}(c_6) \oplus \mathbf{s}(c_8) \oplus \mathbf{s}(c_{12}) \oplus \mathbf{s}(c_{14}) = 0\Big)$

(ii) Weight 2 diagonal degenerate errors without condition

- $\overline{\mathbf{s}(c_1)} \wedge \mathbf{s}(c_2) \wedge \overline{\mathbf{s}(c_3)} \wedge \mathbf{s}(c_4)$
- $\mathbf{s}(c_1) \wedge \overline{\mathbf{s}(c_2)} \wedge \mathbf{s}(c_3) \wedge \overline{\mathbf{s}(c_4)}$

(iii) Weight 3 degenerate errors - patterns not yet catched by (i) and (ii)

- $\mathbf{s}(c_1) \wedge \overline{\mathbf{s}(c_2)} \wedge \overline{\mathbf{s}(c_{11})} \wedge \overline{\mathbf{s}(c_4)} \wedge \overline{\mathbf{s}(c_3)} \wedge \mathbf{s}(c_{12})$
- $\overline{\mathbf{s}(c_1)} \wedge \overline{\mathbf{s}(c_2)} \wedge \mathbf{s}(c_{11}) \wedge \mathbf{s}(c_4) \wedge \overline{\mathbf{s}(c_3)} \wedge \overline{\mathbf{s}(c_{12})}$
- $\mathbf{s}(c_1) \wedge \overline{\mathbf{s}(c_2)} \wedge \overline{\mathbf{s}(c_4)} \wedge \overline{\mathbf{s}(c_3)} \wedge \overline{\mathbf{s}(c_{15})} \wedge \mathbf{s}(c_{14})$
- $\overline{\mathbf{s}(c_1)} \wedge \mathbf{s}(c_2) \wedge \overline{\mathbf{s}(c_4)} \wedge \overline{\mathbf{s}(c_3)} \wedge \mathbf{s}(c_{15}) \wedge \overline{\mathbf{s}(c_{14})}$

The diagonal condition of (i) comes from a corner case that would make the heuristic fail if not property taken into account. This is illustrated in Figure 4.4, where, if the plaquette p is blown-up, then the plaquette on the right could not be blown-up in its turn, and the MS would fail to converge. Clearly (i) and (ii) are enough to take care of all weight 2 degenerate errors. There are some weight 3 errors which are also taken into account by (i) and (ii), the remaining ones being taken care of by (iii).

Numerical Results

Here we plot the error correction performance of the MS versus the SB+MS. One can see that the slopes match the expected values. Indeed, the MS is able to converge on all errors of weight

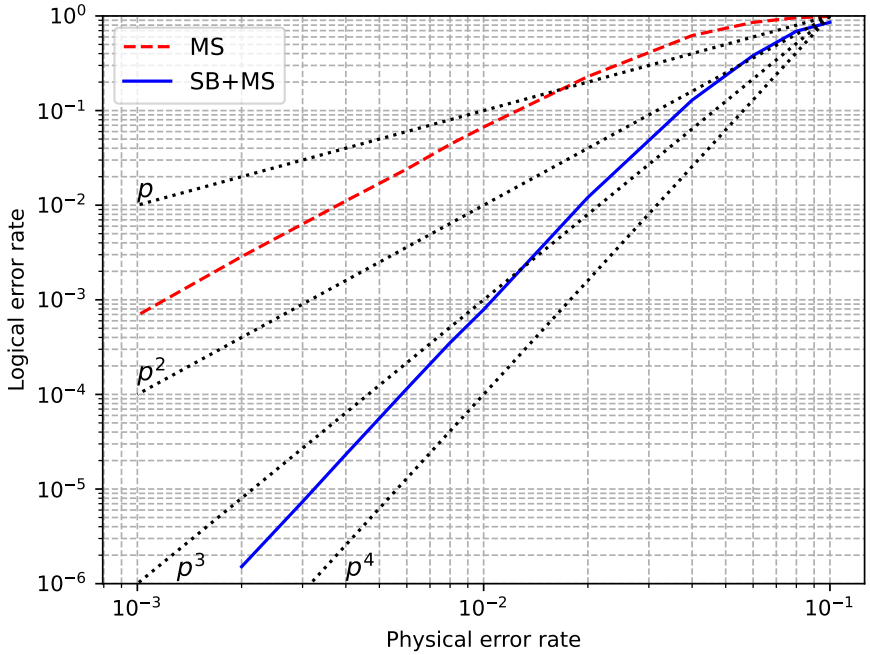


Figure 4.5: Numerical results for MS and SB+MS decoding on the toric code of distance 11. A maximum number of 15 decoding iterations are used for the MS decoding. The slope of the MS is p^2 since the smallest undecodable errors are of weight 2, and the slope of the SB+MS is p^4 since the smallest undecodable errors are of weight 4.

1 but fails on some errors of weight 2. The dominant term is thus p^2 and the slope matches it for small p. For the SB+MS since the smallest undecodable error is of weight 4, and the slope matches the p^4 curve.

SB+MS+OSD

One of the most used post-processing is Ordered Statistic Decoding (OSD), proposed in [68]. Numerically, MS+OSD is a good approximation of the classical ML for the toric code (see for example [76]). However since the OSD post-processing is very costly, namely $\mathcal{O}(n^3)$,[2] it is interesting to run the best possible decoder beforehand to avoid unnecessary use of OSD. Looking at the curves from Figure 4.5, this yields a quadratic improvement in the number of calls of OSD, only needing to call OSD with probability $\approx p^4$ compared to $\approx p^2$. Since our post-processing is linear, this implies a significant improvement on the average decoding complexity for small p, going from $\mathcal{O}(n\log(n) + p^2n^3)$ to $\mathcal{O}(n\log(n) + p^4n^3)$.

[2] We acknowledge that the theoretical complexity of Gaussian elimination is $\mathcal{O}(n^{2.38})$ (see for example [30]), but such algorithms are not practical as they suffer from huge constant costs, and for real life use cases, the Gaussian elimination algorithm is used.

4.4 Proof of Theorem 3

We now prove Theorem 3, using the tools introduced in Section 3.5.3 in the proof of the second part of Theorem 2.

For all toric codes of distance $7 \leq d \leq 28$ we enumerate all errors of weight ≤ 3 and check that SB+MS is able to correct them. There are two cases:

(i) the diameter of the error is ≤ 8.

(ii) there must be one qubit in error that is at distance ≥ 4 from the others.

For case (i), we know that all errors of weight ≤ 3 take at most 10 iterations to converge, hence using Lemma 8, we conclude that this holds for any toric code of distance $\geq 2 \times 10 + 8 = 28$. Case (ii) can be taken care of using Lemma 9 using the same argument as above, so we conclude this is true for any toric code of distance ≥ 28.

4.5 Conclusion

In this Chapter, we showed that by using a low cost linear-time syndrome-aware pre-processing we were able to make the decoding radius and the non-degenerate decoding radius of the MS match. Although the decoding radius of SB+MS is still a constant, which is unsatisfactory as the size of the toric code grows, is is especially interesting to use it when a costly post-processing like OSD, as the average complexity of the decoder drastically decreases because of it. Additionally, it might be interesting to explore the stabiliser-blowup pre-processing on other classes of codes. There should be a straightforward generalisation for all planar codes, but we do believe that the pre-processing could be extended to generic quantum LDPC codes. It could offer a low-cost decoding improvement, especially if they also suffer from low-weight degenerate errors.

Chapter 5

Layered decompositions for qLDPC codes

This chapter is based on the article [21] presented at ISTC24. To the best of our knowledge, this work was the first in the quantum decoding literature to discuss at length the topic of layered decoding of quantum code.

5.1 Introduction

As it was explained in the preliminaries (see 1.3.4 for a reminder), layered decoding is well known in the classical message passing decoding community. It has been subject to extensive research, and it has been shown that the MP scheduling may significantly impact the convergence speed [96], the error rate (*e.g.*, in case of adaptive scheduling strategies [10, 59, 79]), or the hardware performance [6] (*e.g.*, latency, area, power-consumption). The vast majority of hardware designs are based on partly-parallel architectures, implementing a layered decoding scheduling, which can be considered as a *de facto* standard solution, able to provide relevant complexity and performance advantages in most applications [6].

To the best of my knowledge, in the quantum community however, this was never properly investigated. As a result, for most of the codes used in the community today, either a layered decomposition is not known, or it has too many layers to be of practical interest.

For quantum error-correction applied to computing, using a parallel architecture for decoding is is not only important for latency constraints, but also for practical reasons. Because the decoding chip will need to be close to the actual quantum computer (once again for latency purpose), and since most quantum architectures require the qubits to be kept at a very low temperature, the power consumption of the decoding chip must be low to limit the heat losses. Parallel chips like

FPGAs or ASICs lend themselves perfectly for this job as they consume much less power than regular computers. So it is very important to optimise the decoding of qLDPC codes for parallel architectures.

Most of the work that has been done in order to improve the decoding of qLDPC codes based on MP decoders rely on the use of post-processing techniques [23, 68, 73], whose feasibility is still to be demonstrated on parallel hardware.

Throughout this thesis, many evidences will be given that in quantum, the difference between flooded and serial scheduling is not only in term of decoding speed, but also in terms of error rate, a fact that can be most likely attributed to the code degeneracy (a subject that will be discussed further in Chapter 7). Also, some post-processing techniques may also be highly dependent on the MP decoding scheduling. For instance, the OSD post-processing has been shown to provide very good performance when a layered scheduling is used, but its performance may be drastically degraded using a flooded scheduling (see also Chapter 7).

The first result that will be presented in this chapter is a generic construction of layer decomposition that works for the widely used class of HP codes. We further show the optimality of this construction. Secondly, we go on to describe a generalisation of the layer decomposition called a t-cover that allows efficient parallelization of those codes that do not admit a "good" layer decomposition. This is of practical interest, as most of the codes used today within the community were not designed to allow layered decomposition, and as a result they suffer from a high number of layers that render layered decoding impractical. Using t-covers instead of the usual layer decomposition improves the average number of layers processed at each iteration. Thirdly, there will be a discussion on the actual scheduling used (the order in which the layers are processed). Finally, we provide numerical results validating the usefulness of layer decomposition in practice for decoding, both in term of speedup and improved correction performance.

5.2 Layered Scheduling

Here we recall the definition of the layered decoding that was introduced in Section 1.3.4 for classical codes. We here give a slightly modified and more generic definition of a layer, that will be useful later on to develop the generalisation of layered decomposition called t-covering.

Definition 35. *A* **layer** *is a collection of check-nodes such that any two check-nodes have no neighbouring variable-node in common.*

Definition 36. *A* **layer decomposition** $\mathbf{L}_0 \sqcup \cdots \sqcup \mathbf{L}_{k-1}$ *is a partition of the set of check-nodes into* k *layers.*

Definition 37. *A decomposition is said to be* **minimal** *if it is impossible to find a decomposition in less layers.*

A simple density argument is enough to state the following fundamental inequality:

Lemma 13. *Any k layers decomposition of a $(\_, \delta)$-regular*[1] *(m, n)-parity-check matrix satisfies $k \geq \frac{\delta m}{n}$.*

Definition 38. *A decomposition is γ-***balanced** *if*

$$\frac{|\mathbf{L}_i|}{|\mathbf{L}_j|} \leq \gamma, \quad \forall \mathbf{L}_i, \mathbf{L}_j$$

A decomposition will be said to be **balanced** *if it is 1-balanced. Balanced decompositions ensure an efficient use of hardware resources (check-node processing units).*

5.3 Hardware Requirements

In contrast to classical LDPC decoders, which prioritize optimizing throughput, qLDPC ones must satisfy highly constrained values of latency to avoid the backlog problem [43], which would lead to an exponential slowdown of the quantum processor making the error correction impractical.

Table 5.1: Latency approximation for the different architectures

Parallel	Serial	Layered
$T_{min}^{(P)} \times 2 \times it_{max}$	$T_{min}^{(S)} \times (it_{max}/2) \times m$	$T_{min}^{(L)} \times 2 \times (it_{max}/2) \times k$

To illustrate the behavior of the decoder, the B1 code from [68] is taken as an example (defined in Section 5.4). An MP decoder for the B1 code can achieve with this architecture [2], a clock period between 8 and 10ns which derives a latency between 480ns and 600ns at 30 iterations, that is close to the most constrained technology. Taking this into account, the clock period usually can be reduced to 70% and 80% of the clock period obtained with the parallel version. The question is that with this schedule and the derived architecture only one check node is updated in a clock cycle, because of the sequential update of the messages. Due to this, at least m clock-cycles are required [3] to complete just one iteration of the MP algorithm. Following the example of

[1] A matrix is said to be (c, d)-regular if every column is of weight c and every row of weight d.

[2] All the latencies reported here come from implementations of an either fully parallel [89] or serial min-sum decoder architecture, with exchanged messages quantized on 6 bits, on a Xilinx FPGA xcv095 board. Those simulations were done by our coauthor Francisco Garcia-Herrero.

[3] Assuming that due to the reduced complexity of the units both the check node and the connected variable nodes can be updated in parallel.

code B1, the clock period should be between 5.6 and 7ns, but the total latency at 30 iterations would be between 74.26μs and 92.82μs. Even assuming a reduction of the number of iterations to get similar performance to the flooded schedule, the range of latencies would be out of the time budget of supercomputing qubits and transmons. To meet the timing requirements, the clock period should be equal to $\frac{5.6}{m/2}$=0.025ns, which is a maximum clock frequency of 40GHz. This frequency cannot be achieved by any FPGA or ASIC, and on the other hand, it would require a large power consumption that will cause another problem with the power budget and the refrigeration system [88]. With the previous examples, it is easy to conclude that the implementation of serial scheduling, even if it has a better performance than the flooded one, it is not a realistic solution when it comes to implementation. A trade-off solution between both serial and flooded may be the layered schedule. If the number of layers is small enough, the number of clock cycles per iteration will not grow too much and the number of iterations will be usually reduced by two. Going back to the B1 code example, assuming that in the worst case, the clock period will be similar to the parallel architecture, and with a distribution of 3.5 layers[4] the total latency for 30 iterations will be between 8x3.5x2x30=1.68μs and 10x3.5x2x30=2.10μs; and between 8x3.5x2x15=840ns and 10x3.5x2x15=1.05μs with 15 iterations, which is fairly close to the constraints of superconducting qubits and meets the requirements of other technologies.

As we will see in the following sections, the layered schedule will also benefit from some non-negligible performance improvements, apart from the reduction in the number of iterations, compared to the flooded schedule.

In table 5.1, we can find a summary of the total latency for different architectures, where $T_{min}^{(P)}$, $T_{min}^{(S)}$ and $T_{min}^{(L)}$ are the minimum clock periods achievable by the parallel, serial and layered architectures respectively, and it_{max} is the maximum number of iterations configured in the parallel decoder. Note that we assume that the number of iterations of serial and layered is usually half the number of iterations of the parallel architecture [96].

5.4 Generic Constructions

5.4.1 Layered Construction for Hypergraph Product Codes

Consider an hypergraph product code defined by two matrices $\mathbf{A}$ and $\mathbf{B}$, such that $\mathbf{H}_X = [\mathbf{A}\otimes\mathbf{I}, \mathbf{I}\otimes \mathbf{B}^\mathsf{T}]$ and $\mathbf{H}_Z = [\mathbf{I}\otimes\mathbf{B}, \mathbf{A}^\mathsf{T}\otimes\mathbf{I}]$ [86]. There exist a layer construction from a layer decomposition of $\mathbf{A}, \mathbf{B}, \mathbf{A}^\mathsf{T}, \mathbf{B}^\mathsf{T}$.

[4] See Section 5.4 for the formal definition of fractional layer number.

Theorem 3. *Given minimal decompositions* $\mathbf{A} = \mathbf{A}_0 \sqcup \cdots \sqcup \mathbf{A}_{k_\mathbf{A}-1}, \mathbf{B} = \mathbf{B}_0^\mathsf{T} \sqcup \cdots \sqcup \mathbf{B}_{k_{\mathbf{B}^\mathsf{T}}-1}^\mathsf{T}$*, one can construct a minimal decomposition of* $\mathbf{H}_X$ *in* $k = \max(k_\mathbf{A}, k_{\mathbf{B}^\mathsf{T}})$ *layers.*

A similar theorem can be stated for $\mathbf{A}^\mathsf{T}, \mathbf{B}$ and $\mathbf{H}_Z$ with the same proof techniques.

Construction If $k_\mathbf{A} \neq k_\mathbf{B}$, without loss of generality, suppose $k_\mathbf{A} < k_\mathbf{B}$. The first step is to add empty layers to $\mathbf{A}$ so that $k'_\mathbf{A} = k'_\mathbf{B} = k$. That is let $\mathbf{A} = \mathbf{A}_0 \sqcup \cdots \sqcup \mathbf{A}_{k_\mathbf{A}-1} \sqcup \mathbf{A}_{k_\mathbf{A}} \cdots \sqcup \mathbf{A}_k$ where $\mathbf{A}_{k_\mathbf{A}} = \cdots = \mathbf{A}_k = \emptyset$. Let's label each row of $\mathbf{H}_X$ as

$$a \circledast b := [a \otimes e_b, e_a \otimes b], \quad a \in \mathsf{rows}(\mathbf{A}), b \in \mathsf{rows}(\mathbf{B}^\mathsf{T})$$

In the following we will denote by *left* the sub-matrix $[\mathbf{A} \otimes \mathbf{I}]$ and *right* the sub-matrix $[\mathbf{I} \otimes \mathbf{B}^\mathsf{T}]$. Create layers $\mathbf{L}_0 \ldots \mathbf{L}_{k-1}$ such that

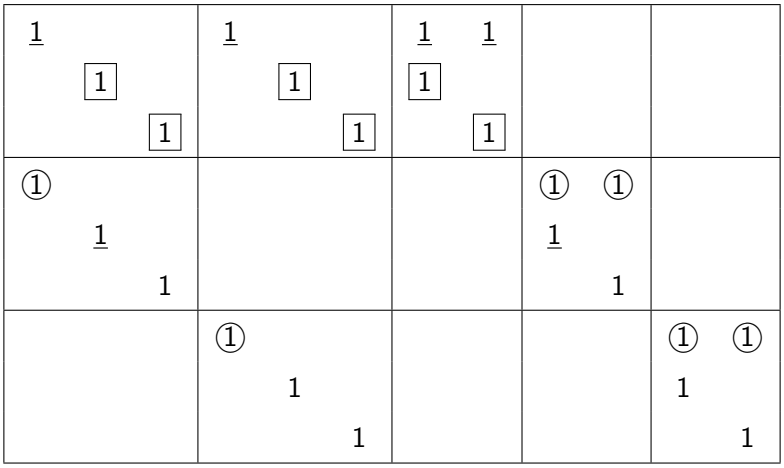
$$(a \circledast b) \in \mathbf{L}_i \Leftrightarrow \exists j \quad a \in \mathbf{A}_j, b \in \mathbf{B}_{j+i \mod k}^\mathsf{T} \tag{5.1}$$

By definition, all checks belong to some layer, we now have to check that any two checks in a given layer have disjoint variable nodes support. Suppose that two checks $a \circledast b, a' \circledast b'$ belong to $\mathbf{L}_i$. Case A : $a = a'$. They do not touch on the left thanks to the tensor product with the identity. Furthermore, it means that $b \neq b'$ but then both belong to $\mathbf{B}_l^\mathsf{T}$ for some l, so they have disjoint support on the right. Case B,C : $a \neq a'$. They have disjoint support on the right because of the tensor product with the identity. To show that they do not intersect on the left, there are two cases : If a and a' belong to some A_l (case B), then by definition they have disjoint support on the left. If a and a' belong respectively to $A_l, A_{l'}$ with $l \neq l'$ (case C), then it means that $b \in \mathbf{B}_{l+i \mod k}^\mathsf{T}, b' \in \mathbf{B}_{l'+i \mod k}^\mathsf{T}$, two distinct classes. Hence even though a and a' might share variable nodes in $\mathbf{A}$, they do not intersect in the tensored version $\mathbf{A} \otimes \mathbf{I}$. Fig. 5.1 depicts a simple example of the 3 cases.

Minimality The proof is by contradiction. Assume that there is a decomposition in less than $k_{\mathbf{B}^\mathsf{T}}$ layers, then one could recover a decomposition for $\mathbf{B}^\mathsf{T}$ in less than $k_{\mathbf{B}^\mathsf{T}}$ from a restriction to the $\{a \circledast b, \quad \forall b\}$ positions for any given a. Any decomposition in less than $k_\mathbf{A}$ layers would similarly give a decomposition for $\mathbf{A}$ from the restriction of $\mathbf{H}_X$ to any $\{a \circledast b, \quad \forall a\}$ for a given b. Hence the decomposition in $\max(k_\mathbf{A}, k_\mathbf{B}^\mathsf{T})$ is minimal for $\mathbf{H}_X$.

Note that the construction is not unique, for example, Equation 1 can be replaced by the following equation where σ is any k-permutation, although this is still not the most generic formula:

$$(a \circledast b) \in \mathbf{L}_i \Leftrightarrow \exists j, \quad a \in \mathbf{A}_{\sigma(j)}, b \in \mathbf{B}_{j+i \mod k}^\mathsf{T} \tag{5.2}$$



Case $\boxed{\text{A}}$: $a = a' \implies b \neq b', \exists l, \quad b, b' \in \mathbf{B}_l^{\mathsf{T}}$

Case Ⓑ : $a \neq a', \quad a, a' \in \mathbf{A}_l$

Case $\underline{\text{C}}$: $a \neq a', a \in \mathbf{A}_l, a' \in \mathbf{A}_{l'}$

Figure 5.1: Small visualization example of proof cases where $\mathbf{A} = \mathbf{B}^{\mathsf{T}}$

Theorem 4. *Given k-layerings for $\mathbf{A}$ and $\mathbf{B}^{\mathsf{T}}$, respectively α and β-balanced. Then $\mathbf{H}_X$ is γ-balanced with :*

(i) $\gamma < min(\alpha, \beta)$ *if* $\alpha, \beta > 1$

(ii) $\gamma = 1$ *otherwise.*

Proof. (i) Let $a_0 ... a_{k-1}$ be the sizes of layers $\mathbf{A}_0 ... \mathbf{A}_{k-1}$, and $b_0 ... b_{k-1}$ the sizes of the layers $\mathbf{B}_0^{\mathsf{T}} ... \mathbf{B}_{k-1}^{\mathsf{T}}$. Then each layer $\mathbf{L}_l$ of $\mathbf{H}_X$ has size $\sum a_i b_{i+l}$. We also suppose layers of $\mathbf{A}$ and $\mathbf{B}^{\mathsf{T}}$ are ordered from biggest to smallest hence $a_0 = \alpha a_{k-1}$ and $b_0 = \beta b_{k-1}$.
The layer $\mathbf{L}_0$ of size $\sum a_i b_i$ is the biggest layer, a classical proof of that is by contradiction, using the fact that $\forall a \geq c, b \geq d, \quad ab + cd \geq ad + bc$. The ratio between any other layer $\mathbf{L}_j$ and $\mathbf{L}_0$ is smaller than β since $\beta \sum a_i b_{i+j} > \sum a_i b_0 > \sum a_i b_i$ using the fact that $\forall b_j, \quad \beta b_j \geq b_0$ (and similarly for α).

(ii) Suppose $\mathbf{A}$ is perfectly balanced. In that case, for any $b \in \mathbf{B}^{\mathsf{T}}$, it will appear the same number of times in each layer $\mathbf{L}_i$ since the $a \circledast b, \forall a \in \mathbf{A}$ will be equally balanced in the layers. Hence the code will be balanced. The same argument holds if $\mathbf{B}^{\mathsf{T}}$ is perfectly balanced.

□

Corollary 4.1. *Given a $k_{\mathbf{A}}$-layering or $\mathbf{A}$, and a $k_{\mathbf{B}} > k_{\mathbf{A}}$ layering for $\mathbf{B}^{\mathsf{T}}$ β-balanced. Then $\mathbf{H}_X$ is γ-balanced with :*

$$\gamma \leq \beta$$

Proof. Same as above, considering a $k_{\mathbf{B}}$ layering for $\mathbf{A}$ by adding empty layers. This new layering is ∞-balanced. □

5.4.2 t-Covering of Layers

For many codes, the theoretical bound on k given by a density argument is not tight. However, since for the quantum codes the number of layers is fixed due to latency constraints, it is important to stay as close as possible to the theoretical bound. We introduce a generalization of the layer decomposition called a t-**covering of (k) layers**. We drop the requirement that the layers should be disjoint, and only require that their union taken with multiplicities should cover each check exactly t times. In the following, the parameters of a t-covers will be specified as (t, k, γ), giving the cover parameters and the balance of the layers. Note that when using t-covers, the usual term of "iteration" becomes ambiguous because it might be the case that the decoder stops while all the checks have not been seen the same number of times. Since by pipelining the process, the syndrome satisfaction could be checked after each layer application adding very low latency, in the following we will often refer to the number of iterated layers (but always specify it when we do so). To quickly compare a t-cover with another or with a layer decomposition, it is useful to introduce the **fractional layer number** as $\frac{k}{t}$, intuitively it captures the "average" number of layers the decoder has to process to see each check once. Finally, by concatenating t times the matrix $\mathbf{H}$, it is clear that the density bound of lemma 1 applies to the fractional layer number. As a simple application, for the code B1 given below, we found a (2,7,1)-cover, $\frac{k}{t} = 3.5$. We could also find a (1,4,2) layer decomposition, $\frac{k}{t} = 4$, and the density bound gives us $\frac{k}{t} \geq 3$, since no decomposition in 3 layers is known for the B1 code, our 2-cover sets a new standard in decoding efficiency.

5.5 Applications on Particular Quantum Codes

5.5.1 A (1,5,1.1)-cover for the C2 Code

The C2 code is a hypergraph product code generated from a single cyclic matrix ($\mathbf{A} = \mathbf{B}$) of generator polynomial $p(x) = 1 + x^2 + x^5$ and length $l = 31$ (The C2 code was described at length in Section 2.2). Since this cyclic matrix (and its transpose) accepts a decomposition in 5-layers, using the technique from theorem 1, we can construct a 5 layer decomposition for the C2 code. As said earlier about the balancing effect of the procedure, the decompositions used for $\mathbf{A}, \mathbf{B}, \mathbf{A}^{\mathsf{T}}, \mathbf{B}^{\mathsf{T}}$

are $(1, 5, 2)$-covers but they yields a $(1, 5, 1.1)$-cover for C2. This shows the balancing effect of the procedure, as we go from $\gamma = 2$ to $\gamma = 1.1$. Here are the layers used for the quasicyclic matrix:

$\mathbf{A}_0$	0 1 7 8 14 15 21 22
$\mathbf{A}_1$	2 3 9 10 16 17 23 24
$\mathbf{A}_2$	4 11 18 25 29
$\mathbf{A}_3$	5 12 19 26 30
$\mathbf{A}_4$	6 13 20 27 28

It should be noted that in order to improve the latency (at the cost of a more complex construction), we were also able to create a $(224, 961, 1)$-cover of C2, achieving a fractional layer number of 4.29 and giving close numerical results.

5.5.2 A (2,7,1)-cover for the B1 Code

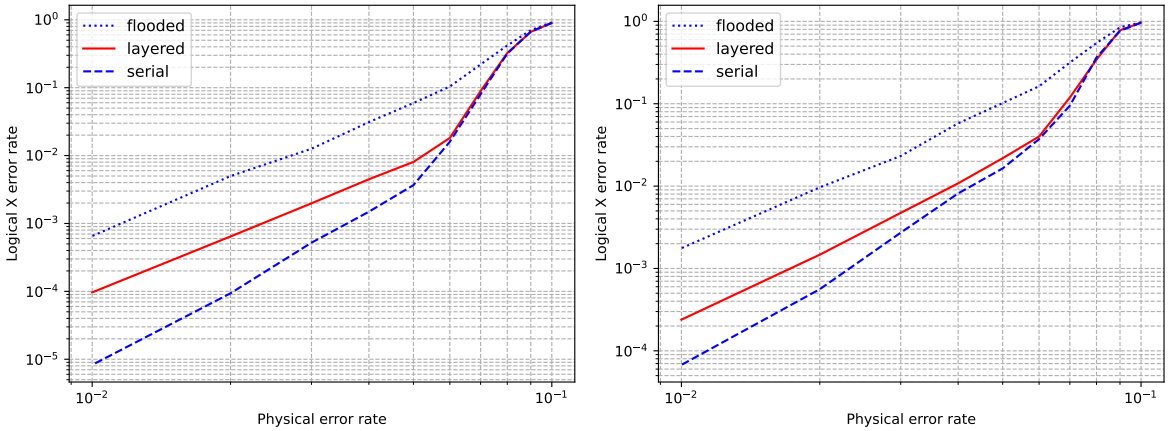
The B1 code is a Generalized Hypergraph Product code (the construction is described at length in 2.2). As such it shares similarities with the Hypergraph Product Codes. Although we do not have a generic decomposition for this family of codes, some ideas from the hypergraph product theorem apply when creating a cover for the B1 code. The B1 code accepts a 2-cover in 7 layers, hence giving a fractional layer number of 3.5. The layers are as follows :

$$\mathbf{L}_i = \{i + 7j \quad \forall j\} \cup \{3 + i + 7j \quad \forall j\}$$

This 2-cover comes from a decomposition of the quasi-cyclic matrix defined by polynomial $p(x) = 1 + x + x^6$ in 7 layers $\mathbf{S}_0 \dots \mathbf{S}_6$ such that $\mathbf{S}_i = \{i + 7j \quad \forall j\}$ which is an obvious decomposition (albeit not minimal) given the generating polynomial. Those layers have the property that the union of any two layers $\mathbf{S}_i, \mathbf{S}_{i+3 \mod 7}$ is still a valid layer which gives the basis for the layers of the code B1. In fact, those layers are extended to the matrices of $\mathbf{H}_X, \mathbf{H}_Z$ much in the fashion of what is done with the hypergraph product but with a "twist" as the blocks are quasi-cyclic shifts of identities instead of all identities so one has to be more careful and cannot use the generic formula.

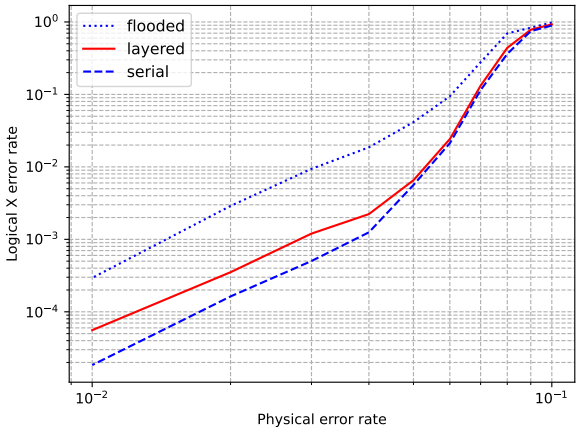
5.6 Numerical Results

Fig. 5.2 compares the different decoding techniques proposed for the B1 and C2 codes under Z-noise, showing both the usefulness of the HP decomposition (for C2) and the t-cover (for B1). We consider Sum-Product (SP) and Normalized Min-Sum (NMS) decoders, with serial, layered and flooded scheduling [80]. When decoding classical codes, using serial scheduling yields a factor two

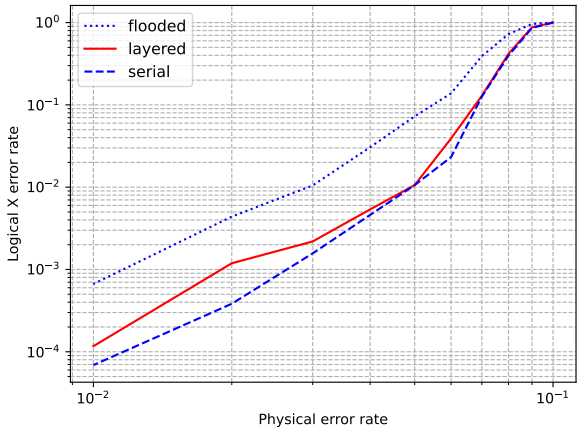


(a) $B1[[882, 24]]$ SP

(b) $C2[[1922, 50, 16]]$ SP



(c) $B1[[882, 24]]$ NMS



(d) $C2[[1922, 50, 16]]$ NMS

Figure 5.2: Comparison of different decoders and scheduling on B1 and C2 codes under Z-noise. In the simulations, we use a perturbated NMS, where each check node message is multiplied by a normalization factor uniformly chosen at random in $\{0.875, 0.9275\}$ at each iteration. This perturbation is important to avoid an error-floor degradation.

improvement in convergence speed over flooded scheduling. This is not the case in our numerics on quantum codes, as the serial scheduling suffers from a high error floor[5]. For the flooded scheduling, the number of iterations used is $i = 128$, for serial $i = 64$. For the layered scheduling of a $(t, k, \_)$-cover, the layer iteration number $i_{lay} = \lfloor 64 \times \frac{k}{t} \rfloor$. Although we argued before that checking the syndrome after each layer is essentially costless, to do a "fairer" comparison with serial scheduling under random order scheduling we also tried checking the syndrome only after $\lceil \frac{k}{t} \rceil$ layer iterations. We did not include the numerics as the two curves match almost perfectly,

[5]This error floor can be virtually eliminated by using a random ordering scheduling, that will be discussed in Chapter 6

making it a non-issue.

On the B1 code, because it is a t-cover and not a layer decomposition, we alter the random ordering scheduling a little bit to boost the performances by requiring that the permutation is not chosen uniformly at random, and must satisfy the additional constraint that two successive layers should not share any check. These additional constraints help the decoder to converge faster as processing the same check twice in a row in different layers would not change its soft information.

5.7 Conclusion

In this chapter, we showed how to implement a layered scheduling for qLDPC codes to meet with the hardware latency limitations. In our numerics, this decoder was more efficient than what could be achieved using similar resources in flooded scheduling which might make it the go to hardware option in the future. We also show that the random order scheduling is a result interesting on its own, as it can be applied to both serial and layered scheduling to alleviate the high error floor of some codes without the need of a post-processing. It should be noted that presently, the best decoders for those codes use some kind of post-processing after message passing, something that will be discussed in further chapters, where we devise hardware friendly post-processing. Knowing that our serial scheduling with random ordering already achieves the performances of the Ordered Statistic Decoding (OSD) post-processing on those codes[6].

[6] See Chapter 7[Sec 4, Fig. 7.2], error probability should be multiplied by 2/3 to compare the two since the error model is depolarizing noise there.

Chapter 6

Orderings

This chapter is based on material from the articles [21, 22], as well as additional unpublished content[1].

6.1 Introduction

When using serial or layered decoding, one aspect of the decoding that is usually overlooked in quantum is the processing order of the checks/layers, although it is known for classical codes that a smarter ordering can yield improvements (see for example [96] for the impact on convergence speed, and [10, 59, 79] for decoding performances). In the following, we give numerical results hinting that ordering might be an interesting and low-cost improvement in term of decoding performance. In particular, we showed in [21] and [22] that the order in which each check/layer is processed at each iteration can drastically improve the decoding performance. The intuition behind that is that contrary to flooded scheduling where the decoder will systematically get stuck with degenerate errors and many other small weight errors, **the sequential update** used in serial and layered can help break symmetries and eventually decode (see for example [69] for other techniques to break the symmetries induced by degeneracy).

6.2 Random Ordering

The random-ordering (RO) is a simple scheme to test if the ordering has a meaningful impact on decoding, by trying random ordering instead of a fixed one. There are two ways to do it. The first one that can be used to test if there exists a fixed ordering such that the decoding performances are improved is to randomise the ordering before decoding starts, and keep the randomized order

[1] Preliminary work done with Davide Orsucci from DLR during a visit to CEA

throughout the computation. We refer to this one as $\mathsf{RO}_{\mathsf{dec}}$. For the second one, referred to as $\mathsf{RO}_{\mathsf{iter}}$, the permutation is taken at random at each iteration during the decoding.

We first recall the algorithm of the standard layered decoding before introducing $\mathsf{RO}_{\mathsf{dec}}$ and $\mathsf{RO}_{\mathsf{iter}}$.

Algorithm 1: $MP_{\mathsf{lay}}(\mathbf{H}, \mathbf{s}, \mathcal{L})$

while $i < i_{\max}$ **do**
  for $L \in \mathcal{L}$ **do**
    Process checks of L;
  if $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ **then**
    return $\tilde{\mathbf{e}}$;
  $i = i + 1$;
return $\tilde{\mathbf{e}}$;

Algorithm 2: $\mathsf{RO}_{\mathsf{dec}}$ $(\mathbf{H}, \mathbf{s}, \mathcal{L})$

Take σ a random permutation of $\mathcal{L}$;
while $i < i_{\max}$ **do**
  for $L \in \sigma(\mathcal{L})$ **do**
    Process checks of L;
  if $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ **then**
    return $\tilde{\mathbf{e}}$;
  $i = i + 1$;
return $\tilde{\mathbf{e}}$;

6.3 Numerical Results on B1 code

Here we provide numerical evidence that the ordering is an important factor when decoding using serial or layered decoding. For example, on Figure 6.1, we where able to improve the decoding performances on the error floor using $\mathsf{RO}_{\mathsf{dec}}$. Moreover, the curve of $\mathsf{RO}_{\mathsf{iter}}$ is very interesting as it also alleviate the error floor and suggests that the usual ordering (the one given by the lines of the matrix) might not always be efficient, and that for practical uses, it might be interesting to optimise on the different orderings to find one that gives the best results for the decoding of a given code.

In addition, further simulations showed us that one does not even have to use a "good"

Algorithm 3: RO_{iter}

Data: syndrome $\mathbf{s}$, parity-check matrix $\mathbf{H}$, layers $\mathcal{L}$, maximum number of iterations $i_{\max}$

Result: $\tilde{\mathbf{e}}$

while $i < i_{\max}$ **do**

  Take σ a random permutation of $\mathcal{L}$;

  for $L \in \sigma(\mathcal{L})$ **do**

    Process checks of L;

  if $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ **then**

    return $\tilde{\mathbf{e}}$;

  $i = i + 1$;

return $\tilde{\mathbf{e}}$;

pseudo-random generator to generate the permutation, and that the same results can be found at virtually no cost using a simple congruent generator, a solution that is hardware friendly.

6.4 Numerical Results on the Toric code

In Chapter 3, we highlighted that some small errors are undecodable by the min-sum with flooded scheduling on the toric code. In particular, we showed that the smallest degenerate undecodable errors are of weight 2, and the smallest non-degenerate undecodable error is of weight 4. There are two undecodable degenerate errors of weight 2, and one undecodable non-degenerate error of weight 4 (up to translations and rotations), consisting of 4 horizontal qubits in a row, all depicted in Figure 6.2. It is natural to ask ourselves the same questions for serial and layered decoding and different orderings, so in the following, we discuss the decoding of those 3 errors (2-L,2-parallel, and 4-in-row) in serial/layered scheduling.

6.4.1 Serial Decoding with Random Ordering

As mentionned earlier, the most natural way to test if the ordering has an impact on decoding is to try the two versions of random ordering on serial decoding. In Table 6.1, we provide such results for the 3 errors mentionned earlier.

Using RO_{dec}, one can see that already the natural choice of ordering might not always be the best for decoding. In fact, although we were not able to find an ordering able to correct the diag-error, there are a non-negligible fraction of orderings that allow to correct the other two. The results for RO_{iter} are better (as expected), and we see that the 3 errors are decodable with some probability.

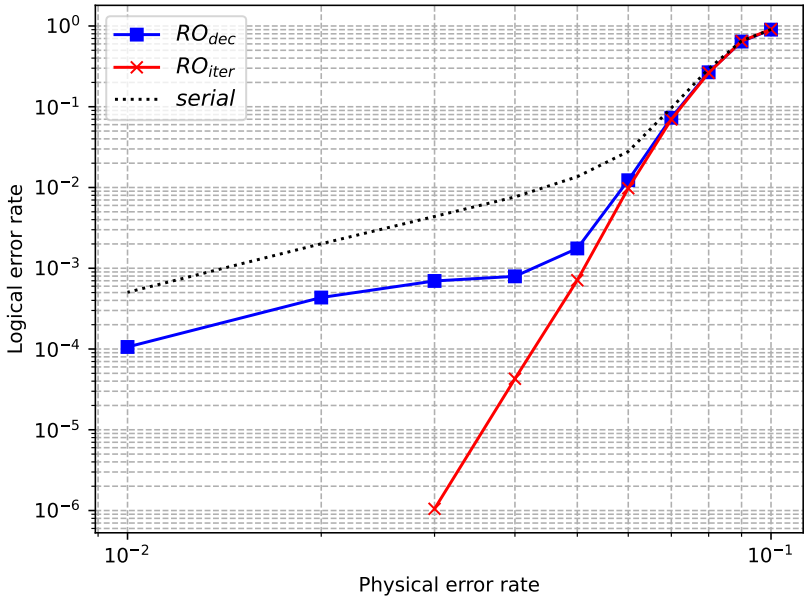
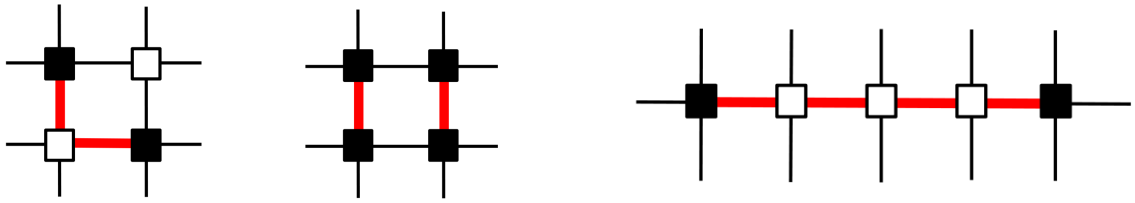


Figure 6.1: 50 iteration SP on the B1 code



(a) 2-L (b) 2-parallel (c) 4-in-row

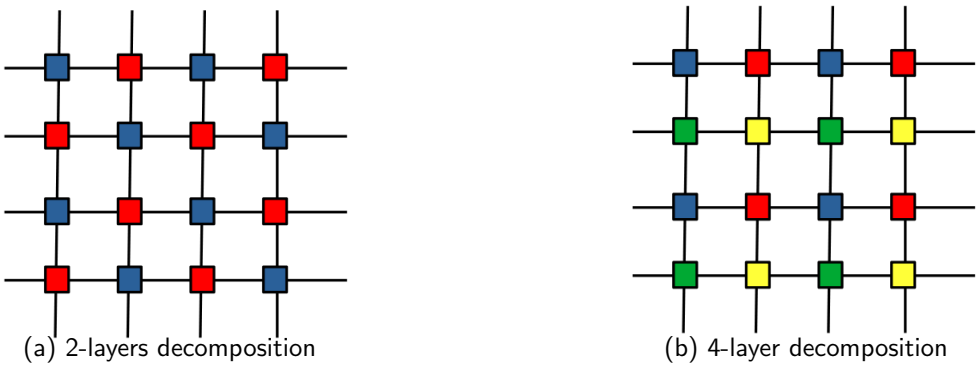
Figure 6.2: The 3 low-weight errors used to test the random ordering layered decoding of the toric code : (a) and (b) are degenerate errors (the smallest, of weight 2), (c) is non-degenerate.

Table 6.1: Effect of the random-ordering on 3 errors on the toric code, 1000000 simulations for each probability

	2-L	2-parallel	4-in-row
RO_{dec}	1.00e+00	7.09e-01	9.93e-01
RO_{iter}	8.50e-01	5.14e-01	9.63e-01

6.4.2 Layered Decoding with Fixed Ordering

Here we do a 4-layering of the toric code. In fact, the toric code admits a 2-layer decomposition, but in that case there is only a single possible ordering (both layers are depicted in Figure 6.3. In order for us to discuss the topic of orderings, we chose a natural 4-layers decompositions, and show that by using alternate orderings for even/odd iterations, we are able to improve on the



(a) 2-layers decomposition (b) 4-layer decomposition

Figure 6.3: Layerings for the toric code

decoding of some low-weight errors. We give several possible layers orders and see if they are able to decode the 3 errors mentionned earlier.

Table 6.2: Effect of the layer order on 3 errors on the toric code

ordering		2-L	2-parallel	4-in-row
cycle	0,1,2,3	no	no	no
zig-zag	0,2,1,3	no	no	no
back-and-forth cycle	0,1,2,3,2,1,0,3	yes	no	no
rotated zig-zag	0,1,3,2,0,3,1,2	yes	no	yes

As can be seen in Table 6.2, a fixed ordering for the layers is not able to correct any of the 3 error patterns. Notice than in the two last examples, back-and-forth cycle and rotated zig-zag, the order used for the layers spans 2 iterations. It turns out that although no ordering on one iteration (up to symmetries and rotations) can solve any of the 3 errors patterns, by using alternate orderings on even/odd iterations, we are able to correct up to two of the error patterns. Note that doing an exhaustive search on all possible ordering on 2 iterations for those layers, we were not able to find an ordering solving all 3 patterns.

6.5 Conclusion

In this section, we showed that when using a serial or layered scheduling, the ordering has a real impact on the decoding performances of quantum codes. We offer a simple tool to improve the decoding performances by randomizing the order, which can be made very low-cost using a simple linear congruent random generator, and make a case that when trying to optimise sequential decoding, one should always look to optimise the order used, as this usually results in improved decoding performances at no cost by just reordering the columns of the matrix.

Chapter 7

Stabiliser Inactivation Post-Processing

This chapter is based on the article [23], presented at ITW22. At the time of its publication, it was the only known post-processing to yield similar or better results than OSD for a worstcase complexity of $\mathcal{O}(\lambda n log(n))$ where λ is a hyperparameter that can scale from a constant to n (compared to $\mathcal{O}(n^3)$ for OSD). In this Chapter, we also provide additional unpublished content relative to the decoding of depolarizing noise using correlations, which builds on the works of [18, 75], and is linked to the recent work of [71].

7.1 Introduction

Many techniques have been proposed to (partially) overcome the degeneracy of qLDPC codes (a non exhaustive list would include the heuristics of [69], Reweighted belief-propagation decoding from [1], and more recently, neural belief-propagation decoding [55]). However, as a general purpose decoder for qLDPC codes, to date the most successful approach has been the Ordered-Statistics-Decoding (OSD) post-processing proposed in 2021 by Panteleev et Kalachev [68], which applies generally across a large spectrum of qLDPC codes [76]. Although the decoding performances of OSD are impressive, its cost ($\mathcal{O}(n^3)$) is prohibitive, especially when one realizes that many times it is used to solve very straightforward and quite "simple" degeneracy issues.

With this idea in mind, we here discuss a new post-processing step *stabilizer inactivation* (SI) designed to tackle degeneracy more efficiently. Assuming we are only decoding X-errors, the idea of the SI post-processing step is to *inactivate* the qubits in the support of some X-check, and then run the MP decoding again. Inactivating these qubits means that we take them out from the MP decoding process (which happens if we set to zero the corresponding log-likelihood ratio *a priori* values). If the inactivated X-check[1] is carefully chosen, the MP decoding converges outside its

[1]By a slight abuse of language, we shall sometimes say that the X-check is inactivated (instead of the qubits in

support. All that remains is to determine the error on the support of the inactivated X-check, by solving a small linear system. We were able to show that MP+SI outperforms the MP+OSD decoder for different qLDPC code constructions, different MP decoding algorithms and scheduling strategies, all while having a significantly reduced complexity.

7.2 Stabilizer Inactivation Post-Processing

In the following, we will only discuss the correction of X errors, but similar arguments apply to Z-errors. Thus, in the rest of this chapter except in Section 7.4 where we decode both X and Z errors, whenever we refer to a matrix $\mathbf{H}$, it refers to a matrix $\mathbf{H}_Z$ used to correct X errors, and whenever we refer to a stabilizer, it belongs to $\mathcal{C}_X$. We first start by discussing the specific degenerate errors that the SI is targeted at.

7.2.1 Stabilizer-Splitting Errors

Here we formally define the notion of stabilizer-splitting errors that will be used later to describe the algorithm, but it should be noted that it is very similar to the notion of Quantum Trapping Sets developed independently in [73], as well as mentioned earlier in the example in [69, Fig. 2]. Note that stabilizer-splitting errors are a subclass of degenerate errors as defined in 2.4.1.

Definition 39. *Let* $\mathbf{e}$ *be an error of minimum weight satisfying syndrome* $\mathbf{s} = \mathbf{He}$.

$\mathbf{e}$ *is a* **stabiliser-splitting error** *if there is a stabiliser* $\mathbf{r}$ *such that* $|\mathbf{e}| = |\mathbf{e} + \mathbf{r}|$[2].

If an MP decoder is run with input syndrome $\mathbf{s}$ generated by a stabiliser-splitting error $\mathbf{e}$ associated to stabiliser $\mathbf{r}$, it is drawn by $\mathbf{e}$ and $\mathbf{e} + \mathbf{r}$ in two different directions, with similar intensity. This may cause the MP decoder getting lost, while trying to find its way to a valid correction of the syndrome.

Recall $\tilde{\gamma}$ is the *a posteriori* soft information of the MP decoder, and $\tilde{\mathbf{e}}$ the corresponding hard decision estimate. Let us define the following quantity :

$$\kappa(\mathbf{r}) := \sum_{q \in \mathrm{supp}(\tilde{\mathbf{e}}+\mathbf{e})} |\tilde{\gamma}(q)| \quad - \sum_{q \in \mathrm{supp}(\tilde{\mathbf{e}}+\mathbf{e}+\mathbf{r})} |\tilde{\gamma}(q)|. \tag{7.1}$$

Since flipping a hard-decision estimate value requires the corresponding soft information to change its sign, the two sums in Equation (7.1) indicate the necessary change in the soft information, so that the corresponding hard decision estimate moves from $\tilde{\mathbf{e}}$ to either $\mathbf{e}$ (left sum), or $\mathbf{e}'$ (right sum). In case the MP decoder fails, that is $\mathbf{H}\tilde{\mathbf{e}} \neq \mathbf{s}$, we expect that this is due to the decoder being

its support).

[2] In particular this implies that $\mathbf{e}$ is a degenerate error as defined in 2.4.1

attracted to a similar degree towards both $\mathbf{e}$ and $\mathbf{e}'$ ("lost in-between" behavior). Put differently, we expect this quantity to be small.

Using $|\tilde{\gamma}(q)| = (-1)^{\tilde{\mathbf{e}}(q)}\tilde{\gamma}(q)$ and $\mathbf{e}' = \mathbf{e} + \mathbf{r}$, one can easily verify that $\kappa(\mathbf{r})$ rewrites as $\sum_{q\in\mathrm{supp}(\mathbf{e})}\tilde{\gamma}(q) - \sum_{q\in\mathrm{supp}(\mathbf{e}')}\tilde{\gamma}(q) = \sum_{q\in\mathrm{supp}(\mathbf{r})}(-1)^{\mathbf{e}'(q)}\tilde{\gamma}(q)$, and therefore,

$$|\kappa(\mathbf{r})| \leq \nu(\mathbf{r}) := \sum_{q\in\mathrm{supp}(\mathbf{r})} |\tilde{\gamma}(q)| \tag{7.2}$$

We refer to $\nu(\mathbf{r})$ as the reliability of the X-check r. Hence, we may use X-check reliability values to determine which X-checks are possibly responsible of a "lost in-between" behavior. To illustrate this, let us define

$$\mathrm{rank}(\mathbf{r}) := |\{\mathbf{r}' \mid \nu_{\mathbf{r}'} < \nu_{\mathbf{r}}\}|. \tag{7.3}$$

We generate random errors $\mathbf{e}$, where each error divides the support of a random X-check $\mathbf{r}$ in two equal parts, and compute the $\mathrm{rank}(\mathbf{r})$ values. Fig. 7.1 shows the estimated probability distribution (histogram format) of $\mathrm{rank}(\mathbf{r})$ values. As expected, it can be observed that the $\mathrm{rank}(\mathbf{r})$ value is small with high probability.

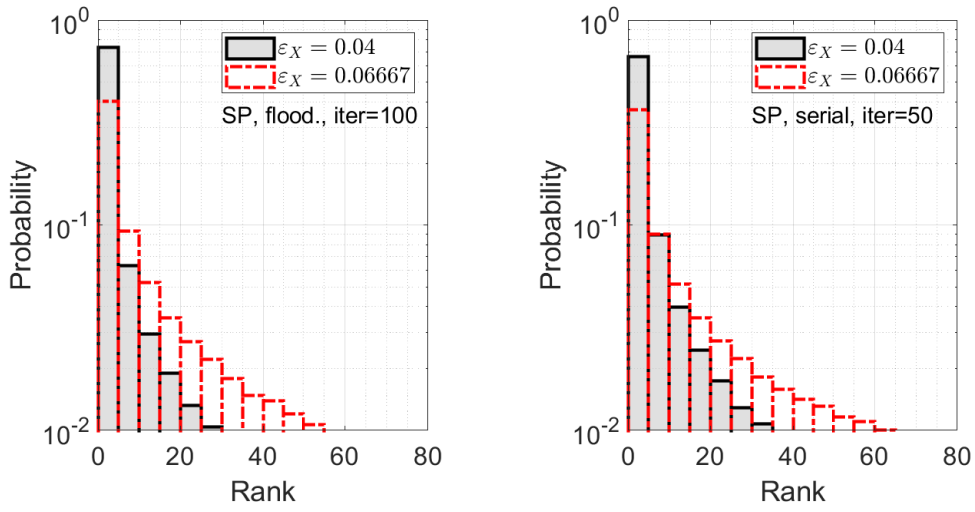


Figure 7.1: Probability distribution of $\mathrm{rank}(\mathbf{r})$ values for the lifted product code B1$[882, 24]$, decoded by the SP algorithm, with both flooded and serial schedulings (we use 50 decoding iterations for serial scheduling, and 100 decoding iterations for flooded scheduling). We assume a depolarizing channel with $p_X = p_Y = p_Z = p/3$, where $p = 0.1$ or $p = 0.06$, giving $\varepsilon = 0.0667$ or $\varepsilon_X = 0.04$.

7.2.2 SI Post-Processing

We consider the situation where the MP decoder fails to find a hard decision estimate $\tilde{\mathbf{e}}$ satisfying the given syndrome $\mathbf{s}$. The idea of the SI post-processing (Algorithm 4) is to *inactivate* the qubits in the support of the less reliable X-checks. By a slight abuse of language we shall say that we inactivate the X-check itself (rather than its support). We start by inactivating the least reliable X-check, and let this be $\mathbf{r}$. This means that we discard the qubits checked by $\mathbf{r}$ from the MP decoding process. Precisely, let

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{|\mathbf{r}} & \mathbf{A} \\ 0 & \mathbf{H}_{|\bar{\mathbf{r}}} \end{bmatrix}, \tag{7.4}$$

where $\mathbf{H}_{|\mathbf{r}}$ is the submatrix determined by the columns corresponding to the qubits in $\mathrm{supp}(\mathbf{r})$, and the rows having at least one non-zero entry in any one of these columns[3] (for the sake of illustration, in (7.4) we assumed that $\mathbf{H}_{|\mathbf{r}}$ is a leading submatrix). Then we rerun MP decoding on the matrix $\mathbf{H}_{|\bar{\mathbf{r}}}$, with input syndrome $\mathbf{s}_{|\bar{\mathbf{r}}} = [0\ \mathbf{H}_{|\bar{\mathbf{r}}}]\mathbf{e}$ (that is, syndrome $\mathbf{s}$ is restricted to the rows of the $\mathbf{H}_{|\bar{\mathbf{r}}}$ submatrix only). If MP decoding succeeds, it provides an estimate $\tilde{\mathbf{e}}_{|\bar{\mathbf{r}}}$ of the error outside $\mathrm{supp}(\mathbf{r})$. To estimate the error on $\mathrm{supp}(\mathbf{r})$, we solve the linear system $\mathbf{H}_{|\mathbf{r}}\tilde{\mathbf{e}}_{|\mathbf{r}} = \mathbf{s}_{|\mathbf{r}} + \mathbf{A}\tilde{\mathbf{e}}_{|\bar{\mathbf{r}}}$. If the system has a solution[4], say $\tilde{\mathbf{e}}_{|\mathbf{r}}$, the decoding process stops and outputs $\tilde{\mathbf{e}} := (\tilde{\mathbf{e}}_{|\mathbf{r}}, \tilde{\mathbf{e}}_{|\bar{\mathbf{r}}})$. In case that either the system has no solution, or the MP decoding fails, SI post-processing continues by inactivating the next least reliable X-check, until a maximum (fixed) number of X-check inactivations, denoted by λ_{max}, is reached.

Two observations are in place here:

(a) X-check reliability values are *computed only once, after the initial MP decoding attempt* (run on the whole $\mathbf{H}$ matrix).

(b) X-checks are sorted in increasing order of reliability, and inactivated, *one at a time*, in this order. In our experiments, run on qLDPC codes of length less than 2000 qubits, inactivating several X-checks at the same time did not improve the decoding performance.

Finally, as mentioned above, the following two situations may occur, causing SI post-processing to continue:

(i) the MP decoder does not converge to a solution on the reduced matrix $\mathbf{H}_{|\bar{\mathbf{r}}}$.

(ii) The MP decoder converges, but the system on the remaining qubits cannot be solved.

[3] Note that $\mathbf{r} \in \ker \mathbf{H}$, hence any row of $\mathbf{H}$ has an even number (≥ 0) of non-zero entries in the columns corresponding to $\mathrm{supp}(\mathbf{r})$.

[4] Note that in that case the system will have at least two solutions. We pick any of them.

Algorithm 4: SI[$\lambda_{\max}$] post-processing for X-error correction

$\tilde{\mathbf{e}} \leftarrow \mathrm{MP}(\mathbf{H}, \mathbf{s})$

if $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ **then**

  Return $\tilde{\mathbf{e}}$

else

  Compute reliabilities: $\nu_{\mathbf{r}} = \sum_{q \in \mathrm{supp}(\mathbf{r})} |\tilde{\gamma}(q)|\,,\ \forall \mathbf{r} \in \mathcal{C}_X$

  Sort checks in ascending reliability: $\mathbf{r}_1 \dots \mathbf{r}_m = \mathrm{sorted}_{\nu(\mathbf{r})}(\{\mathbf{r} \in \mathcal{C}_X\})$

  for $1 \leq \lambda \leq \lambda_{\max}$ **do**

    $\mathbf{r} \leftarrow \mathbf{r}_\lambda$

    $\tilde{\mathbf{e}}_{|\bar{\mathbf{r}}} \leftarrow \mathrm{MP}\left(\mathbf{H}_{|\bar{\mathbf{r}}}, \mathbf{s}_{|\bar{\mathbf{r}}}\right)$

    if $\mathbf{H}_{|\bar{\mathbf{r}}}\tilde{\mathbf{e}}_{|\bar{\mathbf{r}}} \neq \mathbf{s}_{|\bar{\mathbf{r}}}$ **then**

      continue

    else

      if $\mathbf{H}_{|\mathbf{r}}\mathbf{e}_{|\mathbf{r}} = \mathbf{s}_{|\mathbf{r}} + \mathbf{A}\tilde{\mathbf{e}}_{|\bar{\mathbf{r}}}$ *has a solution* $\tilde{\mathbf{e}}_{|\mathbf{r}}$ **then**

        return $\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_{|\mathbf{r}}, \tilde{\mathbf{e}}_{|\bar{\mathbf{r}}})$

      else

        continue

Return decoding failure

Case (i) typically happens when the inactivated X-check was not one that implied the failure in the first place (initial MP decoding attempt on the full $\mathbf{H}$ matrix). Case (ii) happens when the MP decoding converged to a wrong solution on the reduced matrix $\mathbf{H}_{|\bar{\mathbf{r}}}$. This may happen when some of the active qubits become "dead ends" (degree 1 is the reduced matrix $\mathbf{H}_{|\bar{\mathbf{r}}}$), thus "absorbing" the error around the Z-checks they are connected to.

7.2.3 Complexity

The SI[$\lambda_{\max}$] post-processing step has worst-case complexity $\mathcal{O}(\lambda_{\max} n \log n)$, where $\lambda_{\max}$ is the maximum number of inactivated X-checks and $\mathcal{O}(n \log n)$ is the complexity of the MP decoding (assuming the number of decoding iterations increases as $\log n$ [80]). Solving the linear system in the SI post-processing step has constant complexity, since the size of the system does not exceed the maximum X-check weight. In case $\lambda_{\max}$ is not a constant, but scales linearly with n, the worst-case complexity becomes $\mathcal{O}(n^2 \log n)$. In any case, the average-case complexity is $\mathcal{O}(\lambda_{\text{ave}} n \log n)$, where λ_{ave} is the average number of inactivated X-checks during SI post-processing.

7.3 Numerical Results

In this section, we start by benchmarking the SI post-processing against the state-of-the-art OSD post-processing for different schedulings and algorithms, and then provide evidence that SI is able to achieve a threshold phenomenon on a family of codes, for $\lambda_{\max}$ scaling as a small constant fraction of n.

We argue that the average complexity approaches $n \log n$, making the post-processing very efficient in practice.

7.3.1 Logical Error Rate Performance

In Fig. 7.2, we provide numerical results for the codes B1$[881, 24, \leq 24]$ and C2$[1922, 50, 16]$. We note that for both B1 and C2 codes, the constituent classical LDPC codes have column weight at least 3, and no cycles of length 4. These characteristics are well suited to the proposed SI post-processing method, since except for the resolution of a small, constant size, linear system, it entirely relies on MP decoding.

We consider the depolarizing channel, with physical error rate p (shown on the abscissa), and $p_X = p_Y = p_Z = p/3$. Fig. 7.2 shows the logical-X error rate, for three different MP algorithms, and two different schedulings, using either SI or OSD-0 post-processing. It can be observed that SI$[\lambda_{\max} = 10]$ significantly outperforms the OSD-0 post-processing[5] in case of either MS or SP decoding, for both scheduling strategies. For NMS decoding[6], the performance of SI$[\lambda_{\max} = 10]$ is only slightly better than that of OSD-0, but comes however at the cost of a significantly reduced complexity.

The SI[all] curves in Fig. 7.2 correspond to the case when the $\lambda_{\max}$ value is set to the total number of stabilisers ($|\mathcal{C}_X|$), thus providing the best achievable performance by the proposed SI post-processing. (We note that for the code B1, the SI[all] performance is slightly better using serial scheduling than using flooding scheduling, while the opposite occurs for the code C2 – not shown in the figure). Finally, in order to evaluate the average-case complexity, we also show in Fig. 7.2 the λ_{ave} values for the SI[all] post-processing, in the lower part of the waterfall region (logical error rates less than 10^{-3}). In order to avoid a bias due to MP decoding performance,

[5] We note that we perform OSD-0 post-processing on binary MP decoding (of X-errors), which explains the slight shift of the curves with respect to [68], where non-binary MP decoding (of both X and Z errors) is performed.

[6] For the NMS decoding, the normalization factor is chosen so as to ensure the best performance of the corresponding post-processing step. NMS+OSD exhibits very good performance using normalization factor 0.625 (the same normalisation factor was used in [68], for non-binary NMS decoding). However, its performance would have been severely degraded, if the normalization factor had been set to 0.9. Conversely, NMS+SI performs well when the normalization factor is set to 0.9, but rather poor when it is set to 0.625.

the average is computed only over the cases when the SI post-processing is used (MP decoding failed), thus $1 \leq \lambda_{\mathsf{ave}} \leq \lambda_{\mathsf{max}}$. It can be seen that $\leq \lambda_{\mathsf{ave}}$ approaches 1 for low error probabilities, giving an average-case complexity slightly higher than the MP decoding complexity.

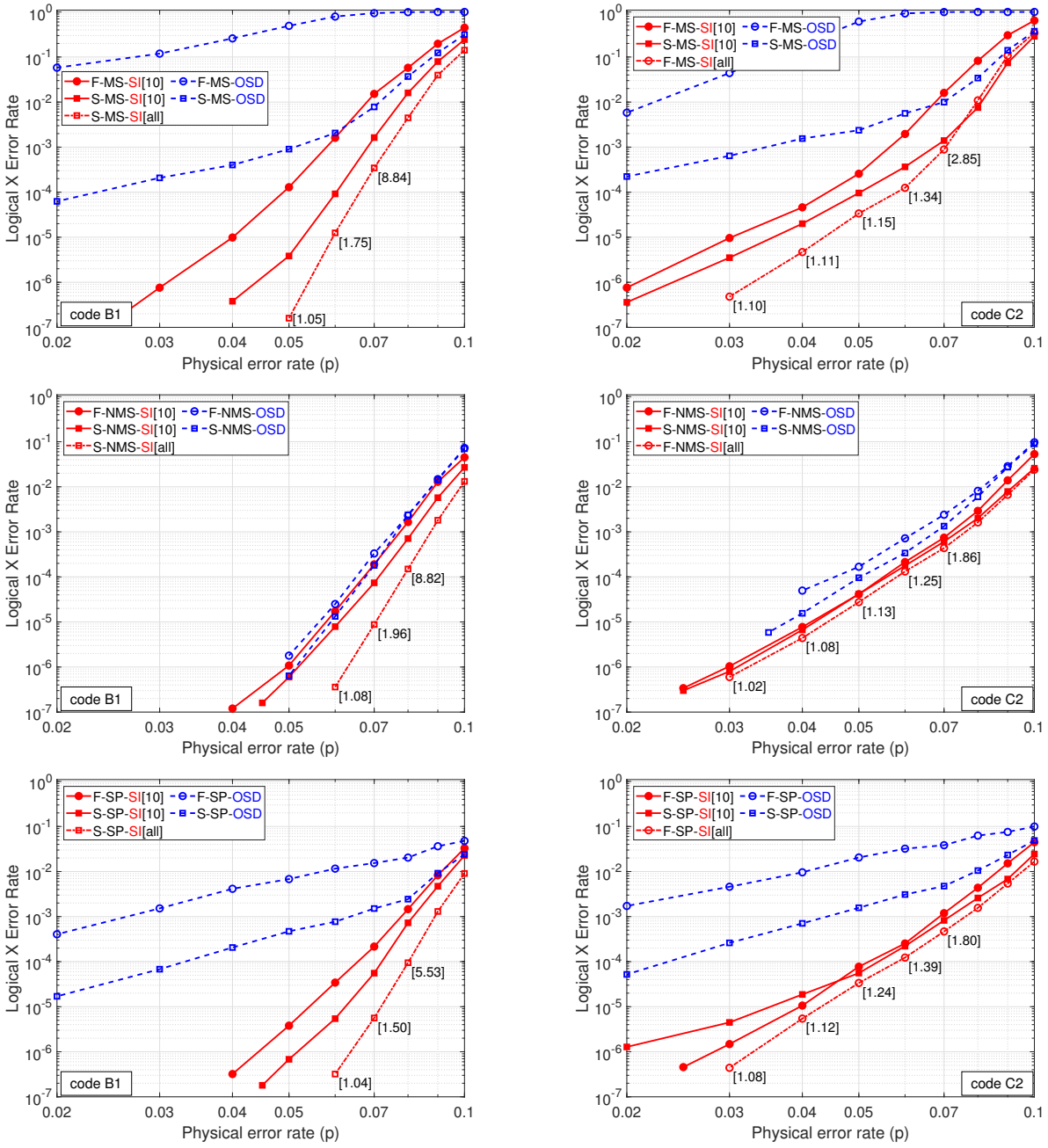


Figure 7.2: Comparison of SI and OSD on B1$[882, 24, \leq 24]$ code (left) and C2$[1922, 50, 16]$ code (right) with MS (top), NMS (middle) and SP (bottom) decoders and flooded ('F', circle markers) and serial ('S', square markers) scheduling. OSD post-processing is of order 0. For NMS, we use normalization factor 0.625 for OSD (following [68]) and 0.9 for SI. All MP decoders use 50 decoding iterations for serial scheduling, and 100 decoding iterations for flooded scheduling.

7.3.2 Threshold

In Figure 7.3, we provide numerical evidence that SI can achieve a threshold for a family of generalized bicycle codes[7] proposed in [68, Appendix C, Fig. 9]. Generalized bicycle codes (first studied in [48] and described in Section 2.2 are constructed by using two commuting square matrices $\mathbf{A}$ and $\mathbf{B}$, with $\mathbf{H}_X = [\mathbf{A}, \mathbf{B}], \quad \mathbf{H}_Z = [\mathbf{B}^\mathsf{T}, \mathbf{A}^\mathsf{T}]$. For the family of codes considered here, $\mathbf{A}$ and $\mathbf{B}$ are circulant matrices, thus they commute, and code parameters are given by $[[n, k]] = [[2^{i+1} - 2, 2i]]$, for $i = 6, 7, 8, 9, 12$.

We estimate the threshold under MP decoding, with both SI and OSD post-processing, for the depolarizing channel (same channel model as before). For a fair comparison between SI and OSD, we use NMS with appropriate normalization factors. The scheduling chosen is serial, as it allows faster convergence and better results in most cases.

For the SI post-processing, we choose $\lambda_{\text{max}} = 10$ for the code of length 1022 qubits, representing 2% of the number of X-checks, denoted by $m = |\mathcal{C}_X|$. Note that $m = (n - k)/2$. Then, we keep the same ratio between the maximum number of inactivated X-checks, and the total number of X-checks, that is, we consider $\lambda_{\text{max}} = 0.02\, m$, for the five simulated codes[8]. Since $m \approx n/2$, the worst case complexity of the MP+SI decoder scales as $0.01n^2 \log(n)$. However, we note that the average case complexity approaches $O(n \log n)$ in the low error rate region (that is, λ_{ave} approaches 1 for low error probabilities, similar to the observation made in Section 7.3.1).

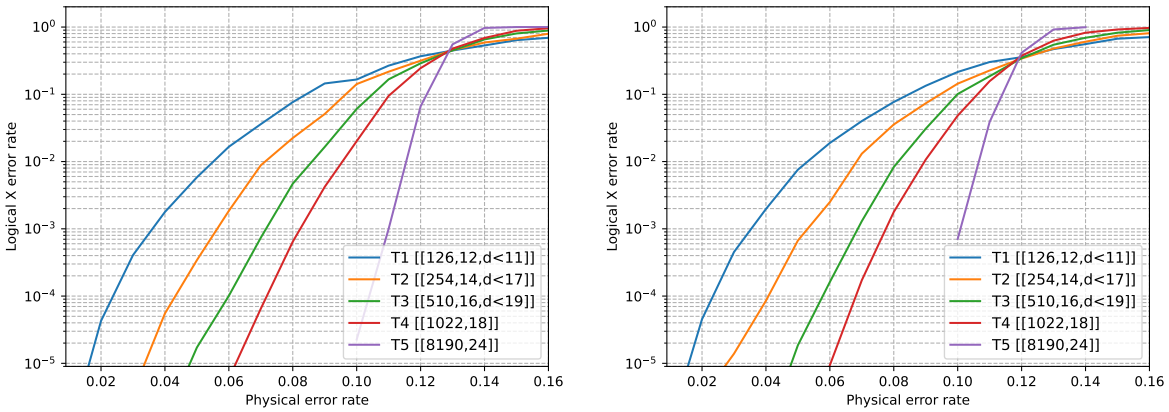
The threshold given by SI is around 13%, while the one of OSD-0 is only 12%. Apart from the numerical results, the computational advantage of SI becomes clear for codes above a thousand qubits. The observed threshold phenomenon shows that the SI post-processing scales well with longer codes, in spite of the fact that it inactivates only one check at a time, and the inactivation is tested for quite a small (2%) fraction of checks.

7.4 Adjusted Decoding With Post-Processing

Although in the rest of the thesis, we are always discussing the decoding of only X (or Z) errors, here we give a few results concerning the use of SI for decoding depolarizing noise using correlations (to better take into account Y errors). One natural way to combine two decoders for X and Z noise to use them for decoding depolarizing noise is to do them sequentially and feed correlation information from the first to the second. This was first proposed in [18], and the ideas were expanded first in [75] under the name Adjusted Binary Decoder, and more recently in [71] under

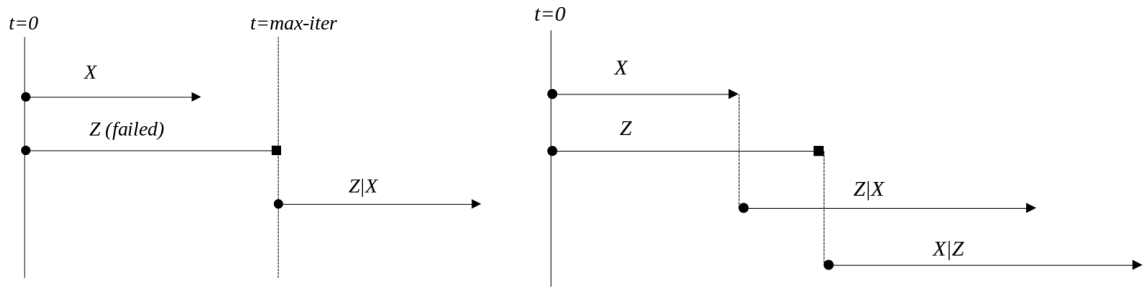
[7] These codes can be found in the alist format at https://gricad-gitlab.univ-grenoble-alpes.fr/ducrestj/qldpc-codes

[8] Precisely, $\lambda_{\text{max}} = 2, 3, 5, 10, 82$, for code length $n = 126, 254, 510, 1022, 8190$, respectively.



(a) NMS+SI[$\lambda_{\max} = 0.02\,m$] (b) NMS+OSD-0

Figure 7.3: Threshold for SI[$\lambda_{\max} = 0.02\,m$] and OSD-0 on a family of generalized bicycle codes with NMS and serial scheduling. For NMS, we use normalization factor 0.625 for OSD (following [68]) and 0.9 for SI. All MP decoders use serial scheduling with at most 100 decoding iterations.



(a) ADJ_α decoder (b) ADJ_β decoder

Figure 7.4: The ADJ decoder, in the vanilla (α) form and the more refined (β) form, that should be used whenever a post-processing is applied.

the name of turbo-XZ decoder. Here we focus on applying those ideas together with a post-processing step. In the following, we follow the notations of [75] (which our work builds on) and describe our decoder as **adjusted**. This decoding technique is described in [75] as depicted in Figure 7.4 (a), which we refer to as ADJ_α . In that case, both the X and Z decoders are run in parallel, and if one of them fails (say Z for example), then we run the decoder again with modified *a priori* taking into account the X error pattern that was decoded by the X decoder. We refer to this run of the algorithm with modified *a priori* as $Z|X$ (pronounced "Z knowing X").

We first briefly explain how the correlations are taken into account in those setting. Suppose there is a depolarizing noise of probability p, then X, Z and Y errors each happen with probability $p/3$. We can write the error vector as the sum of and X error vector and a Z error vector. If one

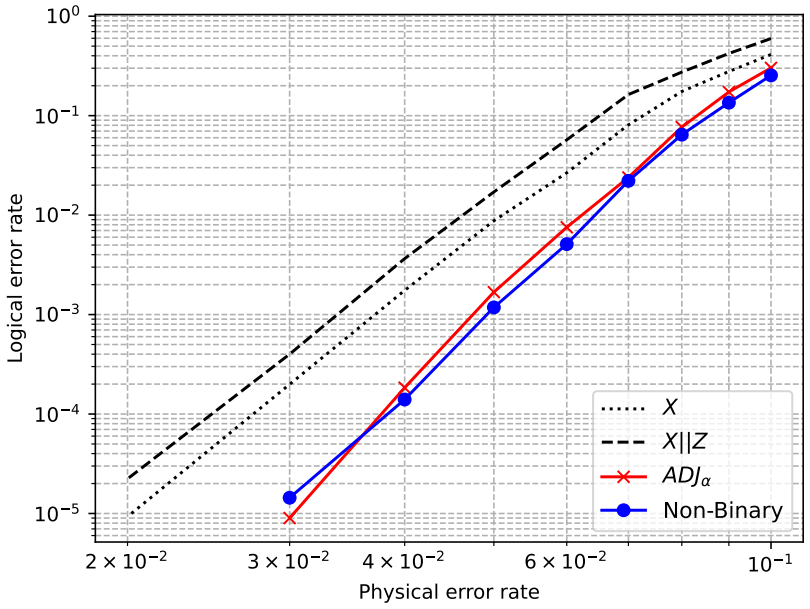


Figure 7.5: Non Binary Decoding vs ADJ_{α} vs $X||Z$ on A5 code using 32 iterations serial NMS (sf=0.90625).

were to decode independantly each error vector, each qubit would have *a priori* $\frac{2p}{3}$. Suppose one is provided with the X error vector, and that qubit q is decoded as being in X-error. Then the *a priori* probability of Z error of q knowing that there is an X error on q is now $1/2$. Likewise, if q' does not appear in the X error vector, then the probability of a Y error on q becomes $\frac{p}{3-2p}$.

Supposing both X and Z codes and decoders have roughly the same decoding performances, we denote $f = \Pr(X \text{ fails}) \approx \Pr(Z \text{ fails})$ for a given set of parameters (code, decoder, and noise p).

First, we discuss the advantages of such a method (Figure 7.5). The curve denoted X is just a standard run of the X MP decoder. The curve $X||Z$ is the result of running the X and Z decoder in parallel without sharing any correlation information. As it can be seen, the error rate curve matches the expected probability of error of $2 \times f - f^2$. The scheme ADJ_{α}, is when both X and Z decoders are run in parallel, and should one fail (e.g. X), it is rerun a second time with the *a priori* of the other (e.g. $X|Z$). This is explained visually in 7.4 (a). As can be seen the performances increase, and quite surprisingly match the curve of the non-binary decoding.

Now we discuss an interesting phenomenon that has to the best of our knowledge not been discussed elsewhere. On Figure 7.6, we exhibit a behavior that might seem strange at first glance. Looking at the dashed curves, when using the ADJ_{α} scheme based on MP+SI, the use of the "better" decoder SI[$\lambda_{\max}$=all] actually decreases the performances of the compared to the "worse" decoder SI[$\lambda_{\max}$=2]. In fact, there is a straightforward explanation for that. To understand that,

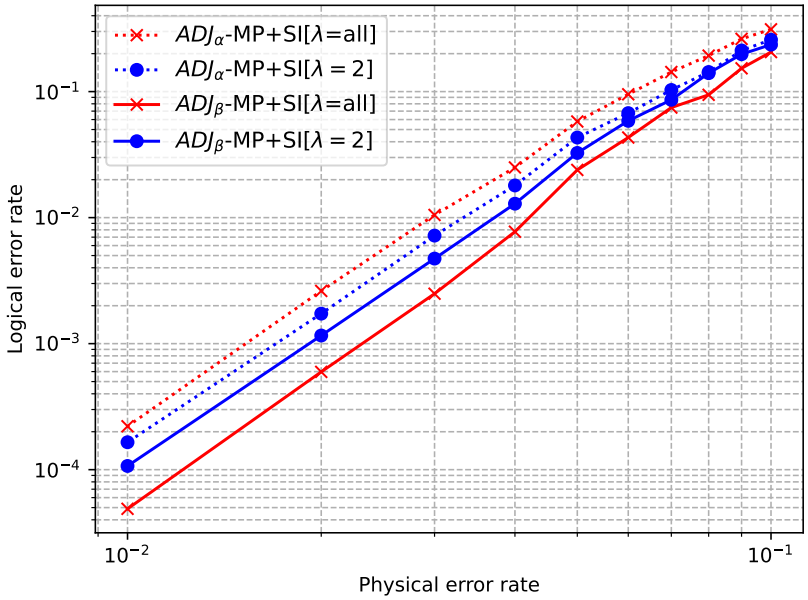


Figure 7.6: Comparison of the ADJ_α and ADJ_β decoder on the A3 code

we first need to discuss the **error typology** associated to a decoder. Whenever an MP decoder fails, it can result in a logical error where the decoder converged to an error that does not belong to the equivalence class of the real error, or it can fail to converge to an error satisfying the syndrome. We refer to those cases as **logical failure** and **decoding failure**. In a sense, decoding failures give more information than logical failure because the decoder warns us that it failed. This is exemplified in the comparison between MP+OSD, which always outputs an error satisfying the syndrome, hence the error typology is only logical failures, whereas MP+SI lies more on the side of logical failures (but this also depends on the code, noise, etc...). Going back to our example of Figure 7.6, on the small code A3 that we used, the decoder MP+SI[$\lambda_{\max}$=all] is trying to converge at all cost, yielding roughly 90% logical failures, while, the decoder MP+SI[$\lambda_{\max}$=2] only yields about 50% logical failures. Hence when used in the ADJ_α decoder, the second step $X|Z$ or $Z|X$ is never almost never run for the MP+SI[$\lambda_{\max}$=all] decoder, leading to worse decoding performances. To cope with that, we now present the ADJ_β scheme, that allows to go over those problems, even with decoders whose error typology lies more on the logical failure side, which could be applied for example for OSD. In the ADJ_β scheme, where both X and Z are run in parallel, and $X|Z$ and $Z|X$ are always run later. This gives us 4 combinations of X and Z error patterns to chose from, which can be decided by a) **an error weight** metric where the likelihood of the 4 proposed errors are computed, and the most likely is outputted, or b) **a speed metric** where the output of a decoder is considered more reliable than an other if it converged faster[9].

[9]Note that this metric cannot be applied on MP+OSD, but other metrics (on the qubit reliability for example)

The numerical results of the ADJ_β scheme using the speed metric can be seen on the solid lines in Figure 7.6, where one can see a decoding improvement where the MP+SI[$\lambda_{\max}$=all] yields better results than MP+SI[$\lambda_{\max} = 2$] as expected.

7.5 Conclusion

To cope with degeneracy issues, the SI post-processing method inactivates a set of qubits, supporting a check in the dual code. Except for the resolution of a small, constant size, linear system, the proposed MP+SI approach entirely relies on MP decoding. Its low complexity and compatibility with various MP decoding algorithms and scheduling strategies make it an attractive and practical decoding solution. Our numerical simulations showed that MP+SI provides effective results, especially when the constituent classical LDPC codes have column weight at least 3, and no cycles of length 4. We have also shown that SI post-processing can achieve a threshold on a family of generalized bicycle codes, outperforming the one achieved by OSD. We also highlighted the fact that when doing sequential decoding of X and Z errors to decode depolarizing noise, the choice of the post-processing and the tuning of the parameters is especially important, and that SI makes a good choice if such a sequential decoding were to be implemented because of its easily tunable parameters.

could be devised.

Chapter 8

Check-Agnosia Post Processing

This chapter is based on the article [22], that was published in the Quantum journal. It is a direct followup of the previous work on SI, trying to optimise the ideas used in SI to make it even more hardware friendly by reusing as much as possible the informations computed during the iterative process and providing a solution to get rid of the small system solving from SI in most of the cases.

8.1 Introduction

Before large-scale quantum technology becomes available, two important problems need to be addressed from the qLDPC decoding perspective: i) devising new decoding algorithms that overcome or mitigate the effect of degeneracy [73], thus providing increased error correction capabilities, and ii) developing hardware designs that meet latency and power constraints imposed by the quantum system (*e.g.*, latency values within the decoherence time of the qubits to be protected, or power limitations for qubit technologies requiring cryogenic cooling, when the decoder is implemented within the low-temperature layers), a topic that only got attention recently [2, 17, 71].

To achieve the first objective, several approaches building upon classical message-passing (MP) decoding algorithms have been recently proposed in the literature, where the degeneracy issue is dealt with by either incorporating neural network techniques in the MP decoder [55], or adding a post-processing step, taking advantage of the soft information delivered by the MP decoder [23,68].

Neural-network-based decoders are bound to the noise models used to train them and do not scale well with the number of qubits [63]. Moreover, as shown in [11], there are not only different sources and noise models, but also the noise may be different depending on the area of the layout of the quantum processor, the environmental conditions, and the evolution of errors with time since the last calibration (space and time drift of the errors [61]). In that sense, more

generalized solutions are required, at least at the moment of writing these lines, when there is no standardized or predominant technology or architecture for future large-scale quantum devices. Hence, post-processing techniques may become an interesting choice.

However, additional hardware-oriented analysis and optimization are required to ensure the hardware design meets the constraints required to provide real-time support to a quantum processor. Here we present the a hardware friendly post-processing called check-agnosia (CA).

First, we describe the post-processing, inspired by SI. The algorithm takes into account the architectural properties of MP decoders in order to reduce the computational load and the required hardware resources. It also limits the amount of information required from the code, eliminating the need to know the stabilizer structure (dual code) and just treating both parity-check matrices as independent. Similar to the stabilizer inactivation, the algorithm identifies a small set of unreliable qubits (which are however not inactivated, in the sense described above). The information considered to identify such a set of qubits is based on the check-node reliability. When the MP decoder fails, the *a priori* information for the qubits connected to the least reliable check nodes is erased, and the post-processor will then try to learn again the reliability of these qubits based on the information from the rest. For this reason, we call the post-processing technique check-agnosia. We also suggest several approaches to perform the selection of unreliable check-nodes, to reduce power consumption and latency, which are the constraints that limit the implementation of decoders in real systems [85, 88].

Second, a non-agnostic hardware perspective is described to help to meet the constraints of future large-scale quantum devices. Aligned with this, a functional description in terms of performance and hardware results of the proposed solution for the two main schedules employed for MP decoders (flooded and layered [58, 82]) is introduced. We carry out a detailed analysis of different corner cases, which is then illustrated for a specific qLPDC code, by providing latency and power consumption values of the check-agnosia solution implemented on an FPGA board.

8.2 Check-Agnosia Decoder

We introduce in this section the Check-Agnosia (CA) post-processing. We first describe the generic post-processing technique (Algorithm 1) and then discuss possible modifications.

8.2.1 Generic Check-Agnosia Decoder

The error vector $\tilde{\mathbf{e}}$ is estimated first using a soft-output MP decoding algorithm. If the error estimate $\tilde{\mathbf{e}}$ satisfies the syndrome, *i.e.*, $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$, then no post-processing is applied.

If the initial MP decoding fails, a metric on the exchanged soft information is used to find the $\lambda_{\max}$ checks $\{c_\lambda\}_{\lambda\in[\lambda_{\max}]}$ whose supports are the most likely to be involved in the decoding failure (this metric will be discussed later). The post-processing will consist of rerunning the MP decoder at most $\lambda_{\max}$ times with new *a priori* qubit reliabilities γ' and a modified stopping criterion.

For the k-th decoder, the input reliability will be set to $\gamma'(q) = \begin{cases} 0, & \text{if } q \in \mathcal{N}(c_\lambda), \\ \gamma_q, & \text{otherwise.} \end{cases}$.

Putting the input reliability to 0 can be considered as an erasure in the MP decoder [56], ensuring that these qubits are deprived of any *a priori* information that may interfere in the decoding attempt of the more reliable qubits. We further define $\mathfrak{N}_k = \cup_{q\in\mathcal{N}(c_\lambda)}\mathcal{N}(q)$, the set of checks that share a neighbor qubit-node with c_λ (note that $c_\lambda \in \mathfrak{N}_k$). This allows us to define $\mathbf{s}_{|\overline{\mathfrak{N}_k}}$ the *partial syndrome* vector containing only the checks that have no neighbor qubit-node in $\mathcal{N}(c_\lambda)$, and $\mathbf{s}_{|\mathfrak{N}_k}$ the *residual syndrome*. We then run a MP decoder with a modified stopping criterion that only tries to match the partial syndrome $\mathbf{s}_{|\overline{\mathfrak{N}_k}}$. In Algorithm 1, we denote this decoder by $\text{MP}^\star(\mathbf{H}, \mathbf{s}, \gamma', \overline{\mathfrak{N}_k})$. We emphasize that MP* applies exactly the same decoding rules on the same Tanner graph as MP, except that MP* is initialized with qubit reliabilities γ', and it stops when the partial syndrome $\mathbf{s}_{|\overline{\mathfrak{N}_k}}$ is satisfied (no matter whether the residual syndrome $\mathbf{s}_{|\mathfrak{N}_k}$ is satisfied or not). If the MP* succeeds in matching the partial syndrome, the decoder then attempts to match the residual syndrome by brute-forcing the error pattern on $\mathcal{N}(c_\lambda)$. Note that sometimes the MP* can actually match the full syndrome, in which case no brute-forcing is needed (will be discussed in more detail later). In Algorithm 1, $\mathbf{H}_{|\overline{\mathfrak{N}_k}}$ denotes the submatrix of $\mathbf{H}$ whose rows correspond to check-nodes $c \notin \mathfrak{N}_k$. Likewise, $\mathbf{H}_{|\mathfrak{N}_k}$ denotes the submatrix of $\mathbf{H}$ whose rows correspond to check-nodes $c \in \mathfrak{N}_k$. Similar to SI, if $\tilde{\mathbf{e}}_{|\overline{\mathcal{N}(c_\lambda)}}$ matches the partial syndrome $\mathbf{s}_{|\overline{\mathcal{N}(c_\lambda)}}$, we keep its value and bruteforce $\tilde{\mathbf{e}}_{|\mathcal{N}(c_\lambda)}$ to match also the residual syndrome.

The intuition behind the post-processing is that the presence of quantum trapping sets [73] in the Tanner graph causes the *a posteriori* reliability values of trapped qubit-nodes to oscillate. This prevents the decoder from converging, regardless of the number of decoding iterations (for oscillating trapping sets see also [49]). Taking into account the oscillation effect, it is reasonable to think that the messages associated with the untrapped qubits will grow with each iteration while the trapped ones will keep relatively low reliability. This effect will help to identify possible trapped qubits. To this end, we define a reliability metric on checks to decide (the support of) which checks should be *erased*. A natural approach to define such a reliability metric is to consider the reliability (*i.e.*, absolute value) of either incoming messages $\{\mu_{q\to c} \mid q \in \mathcal{N}(c)\}$ or outgoing messages $\{\mu_{c\to q} \mid q \in \mathcal{N}(c)\}$. However, for MS-based decoders, the absolute value of outgoing messages is equal to either the first or the second minimum of the absolute values of incoming

Algorithm 5: Generic Check-Agnosia Decoder for X-errors

$\tilde{\mathbf{e}} \leftarrow \mathsf{MP}(\mathbf{H}, \mathbf{s}, \gamma)$

if $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ **then**

  return $\tilde{\mathbf{e}}$

else

  Compute reliabilities: $\nu_c = \min_{q \in \mathcal{N}(c)} |\mu_{q \to c}| + \min_{q \in \mathcal{N}(c)}{}_2 |\mu_{q \to c}|, \ \forall c \in \mathcal{C}_Z$

  Sort checks in ascending reliability: $c_1 \dots c_m = \mathsf{sorted}_{\nu_c}(\mathcal{C}_Z)$

  for $1 \leq \lambda \leq \lambda_{\max}$ **do**

    $\forall q \in \mathcal{Q}$, set $\gamma'(q) = \begin{cases} 0, & \text{if } q \in \mathcal{N}(c_\lambda), \\ \gamma_q, & \text{otherwise.} \end{cases}$

    Determine $\mathfrak{N}_k = \cup_{q \in \mathcal{N}(c_\lambda)} \mathcal{N}(q)$

    $\tilde{\mathbf{e}} \leftarrow \mathsf{MP}^\star(\mathbf{H}, \mathbf{s}, \gamma', \overline{\mathfrak{N}_k})$

    if $(\mathbf{H}_{|\overline{\mathfrak{N}_k}} \tilde{\mathbf{e}}_{|\overline{\mathcal{N}(c_\lambda)}} \neq \mathbf{s}_{|\overline{\mathfrak{N}_k}})$ **then**

      continue

    else

      Try to solve $\mathbf{H}_{|\mathfrak{N}_k} \tilde{\mathbf{e}}_{|\mathcal{N}(c_\lambda)} = \mathbf{s}_{|\mathfrak{N}_k}$ by bruteforcing.

      if *successful* **then**

        return $\tilde{\mathbf{e}}$

return decoding failure

messages, denoted by $\min_{q \in \mathcal{N}(c)} |\mu_{q \to c}|$ and $\min_{2\, q \in \mathcal{N}(c)} |\mu_{q \to c}|$, respectively. This motivates the reliability metric[1] ν_c considered in Algorithm 1.

The cost of computing this metric is nearly none, as the two minima are already computed by the MS decoder. Also, this metric is computed for all the check-nodes, based on the $\{\mu_{q \to c}\}$ messages at some specific (predetermined) iteration of the MS decoder (as discussed below). This allows the sorting of all the checks according to the proposed metric, after which the post-processing can be applied to the $\lambda_{\max}$ most unreliable checks.

To reduce the overall latency (initial MP decoding and post-processing), one may compute the check reliability values ν_c at an early iteration, *i.e.*, before the initial MP reaches the maximum number of decoding iterations.

This allows the post-processing to start running in parallel before the initial MP has ended. If the initial MP succeeds later on, the post-processing will stop and the decoder will output the error

[1] While different variations of this metric are possible (*e.g.*, the sum of the absolute values of all incoming messages) we have not observed any significant difference in terms of error correction performance. Also, similar reliability metrics can be obtained for other MP decoding algorithms, *e.g.*, sum-product.

found by the initial decoder. However, if the initial MP decoder fails, the post-processing will have already started, reducing the total latency. As it will be shown in Section 8.4, the error correction performance obtained by determining the list of least reliable checks using the soft information from either the last or an early iteration is almost the same, but the speedup is considerably higher in the latter case. Moreover, the reliability metric computed after a few iterations may be more accurate than the one computed at the last iteration, as the oscillation effects (also combined with saturation effects of the finite precision arithmetic) might alter quite considerably the accuracy of the reliability metric computed after a large number of iterations.

We discuss now the brute-forcing of $\tilde{\mathbf{e}}_{|\mathcal{N}(c_\lambda)}$ in Algorithm 1. To solve the system $\mathbf{H}_{|\mathfrak{N}_k}\tilde{\mathbf{e}} = \mathbf{s}_{|\mathfrak{N}_k}$ there are several possible methods, including Gaussian elimination. However, since the system to solve is small, brute-forcing, *i.e.*, trying all the possible combinations, hopefully finding one that satisfies the system[2], is a more efficient solution for hardware implementation. Moreover, it is not too difficult to see that the brute force approach can be simplified by taking into account the local structure of the code, eliminating a lot of computation. For instance, a check-node $c \in \mathfrak{N}_k \setminus \{c_\lambda\}$ that has exactly one qubit-node in common with c_λ, uniquely determines the value of that qubit.

8.2.2 Check-Agnosia Decoder Without System Solver

One alternative to determine $\tilde{\mathbf{e}}_{|\mathcal{N}(c_\lambda)}$, described in Algorithm 2, is to use a regular MP decoder that stops only if the full syndrome is matched. Precisely, the MP$^\star(\mathbf{H}, \mathbf{s}, \{\gamma'(q)\})$ in Algorithm 2 is a regular MP decoder, initialized with qubit reliabilities $\{\gamma'(q)\}$, and which stops when the full syndrome is satisfied. We keep the MP* notation in the post-processing step only to distinguish it from the initial MP decoder (will be needed later on Section 8.3). To justify Algorithm 2, let us consider the case when the graph induced by any subset $\mathcal{S} \subseteq \mathcal{N}(c_\lambda)$ contains at least a check-node of degree one. Then, assuming the MP decoder has converged on $\tilde{\mathbf{e}}_{|\overline{\mathcal{N}(c_\lambda)}}$, it will converge on the remaining $\tilde{\mathbf{e}}_{|\mathcal{N}(c_\lambda)}$ at the cost of a few more iterations. The above condition is the same as requiring $\mathcal{N}(c_\lambda)$ contains no stopping subset[3], and running the MP for a few more iterations amounts to running a peeling decoding [57] on the erased qubits. For instance, if the Tanner graph contains no cycles of length four, then $\mathcal{N}(c_\lambda)$ satisfies the no-stopping subset condition, and one extra iteration is enough to determine $\tilde{\mathbf{e}}_{|\mathcal{N}(c_\lambda)}$. The no-stopping subset condition may also be satisfied for graphs containing cycles of length four, but in such a case more than one

[2]Note that there is not guaranteed that the system has a solution, as such, the algorithm can fail at this step.

[3]A set of qubit-nodes is said to be a stopping set, if the induced subgraph contains no check-nodes of degree 1. If the qubit-nodes in a stopping set are erased, they can get no information during the MP decoding, that is, incoming and outgoing messages to and from these qubit-nodes remain equal to zero during the entire iterative decoding process.

Algorithm 6: Check-Agnosia Decoder Without System Solver for X-errors

$\tilde{\mathbf{e}} \leftarrow \mathrm{MP}(\mathbf{H}, \mathbf{s}, \gamma(q))$

if $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ **then**

  return $\tilde{\mathbf{e}}$

else

  Compute reliabilities: $\nu_c = \min_{q \in \mathcal{N}(c)} |\mu_{q \to c}| + \min_{q \in \mathcal{N}(c)_2} |\mu_{q \to c}|, \ \forall c \in \mathcal{C}_Z$

  Sort checks in ascending reliability: $c_1 \dots c_m = \mathrm{sorted}_{\nu_c}(\mathcal{C}_Z)$

  for $1 \leq \lambda \leq \lambda_{max}$ **do**

    $\forall q \in \mathcal{Q}$, set $\gamma'(q) = \begin{cases} 0, & \text{if } q \in \mathcal{N}(c_\lambda), \\ \gamma_q, & \text{otherwise.} \end{cases}$

    $\tilde{\mathbf{e}} \leftarrow \mathrm{MP}(\mathbf{H}, \mathbf{s}, \gamma'(q))$

    if $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ **then**

      return $\tilde{\mathbf{e}}$

return decoding failure

extra iteration may be needed.

For a given Tanner graph the above no-stopping subset condition can easily be verified, and then we may use Algorithm 2 instead of Algorithm 1 (numerical simulations also confirmed that those two approaches give similar performance). For the simulation results shown later (Section 8.4), we always use Algorithm 2.

The presumably only meaningful case in which the no-stopping subset condition is not verified is when the code is auto-dual (*i.e.*, $\mathbf{H}_x = \mathbf{H}_z$), since in such a case $\tilde{\mathbf{e}}_{|\mathcal{N}(c_\lambda)}$ is the support of a codeword, hence a stopping set. It is worth noticing that for auto-dual codes, the check-agnosia (Algorithm 1) and stabilizer-inactivation [23] decoders are the same, up to the reliability metric used to select the $\lambda_{\max}$ least reliable check-nodes. However, for codes that are not auto-dual, the check-agnosia decoder, implemented as in Algorithm 2, presents several advantages, including the use of a simpler, hardware-friendly check-node reliability metric (and not requiring the use of the dual matrix), as well as the fact that it relies solely on MP decoding, eliminating the need of brute-forcing or other system solving methods.

A final remark is that all MP and MP* decoders can implement a flooded or a layered schedule, as discussed in Section 8.2, to cope with the hardware constraints.

8.3 Hardware Architectures

This section aims to analyze the impact of the post-processing algorithm on the hardware implementation, considering architectures with different schedules and varying degrees of parallelism. We carry out a detailed analysis of different corner cases, providing latency and power bounds to assist future hardware decoder designers.

8.3.1 MP Decoder Architecture

We consider first a single MP decoder, without any post-processing. To implement the MP decoder in hardware [4], one can use a fully parallel architecture, implementing a flooded schedule, referred to as flooded decoder, or a partly parallel architecture, implementing a layered schedule, referred to as layered decoder.

We will make standard assumptions[5] regarding the two above architectures [6]. For the flooded decoder, the Tanner graph is instantiated in hardware, where messages are exchanged through wires between processing units, corresponding to qubit- and check nodes. Each decoding iteration is performed in two clock cycles, with one clock cycle for qubit-node messages and *a posteriori* values, and a second one for check-node messages. Thus, the worst case (maximum) latency of the flooded decoder is equal to $(1+2I_F)/f_F$ (s), where we count one clock-cycle for data loading, I_F is the maximum number of decoding iterations of the flooded decoder, and f_F is the clock frequency.

For the layered decoder, the number of processing units instantiated in hardware is given the size of the largest layer[6], messages are exchanged through shared memory, and each processing unit is reused η_L times for each decoding iteration, where η_L denotes the number of layers per iteration. The worst case latency of the layered decoder is equal to $(1+\eta_L I_L)/f_L$ (s), where we count again one clock-cycle for data loading, I_L is the maximum number of decoding iterations of the layered decoder, and f_L is the clock frequency.

Two observations are in place here. First, the flooded architecture may lead to a large number of connections among processing units, causing routing congestion in case of large codes. Due to the large interconnect network, the operating clock frequency of the flooded architecture (f_F) is usually smaller than twice[7] that of the layered architecture (f_L). Second, as discussed in

[4] Serial schedule is not considered due to its extremely large latency, not suitable for real-time implementations.

[5] The hardware implementation reported later on Section 8.3.4 is consistent with the assumptions made here.

[6] Usually all layers have the same size, although this condition is more difficult to satisfy for qLDPC codes [21].

[7] Note that in the flooded architecture, qubit and check-node messages are computed in two different clock cycles, by different processing units, while in the layered architecture they are computed in the same clock cycle, by a processing unit that merges the qubit and check node processing.

Section 8.2, the layered schedule propagates information about twice faster than the flooded one, thus the maximum number of iterations of the layered architecture (I_L) is usually smaller than that of the flooded architecture (I_F). Overall, this can make the layered architecture comparably fast to the flooded one, despite the fact that it employs a reduced degree of parallelism (of course, the number of layers per iteration has to be sufficiently small).

Finally, one possible approach to further increase the clock frequency of the layered decoder is to pipeline the design (*i.e.*, perform each layer in a number of pipelined clock cycles). However, this may lead to delayed message write-backs in memories, and thus, to pipeline related hazards [4]. Solving such hazards (without relying on pipeline stalls, introducing extra latency) can be done for classical LDPC codes at the code construction stage [5]. However such solutions are not generic (need a specific code construction) and may not apply to qLDPC codes. Therefore, to keep the analysis as generic as possible, we do not consider pipelined designs in this work.

8.3.2 Post-Processing Elements

For the check-agnosia scheme, the first step after the MP decoder is the computation of the check reliability values, as outlined in Algorithms 1 and 2. The metric used to calculate the check reliability, denoted as ν_c, involves adding the two least reliable messages. These values are computed during the tree finder process employed to calculate check-node messages in the min-sum decoder. Thus, the only additional hardware required is an adder per check-node to compute ν_c. These values are updated on-the-fly during each iteration, eliminating the need for extra clock cycles after the MP decoder. As described earlier in Section 8.3, one does not have to wait until the end of the initial MP decoder to start the post-processing (the impact of utilizing the ν_c information from early iterations will be evaluated in Section 8.4). In the proposed architecture, the ν_c values can be stored in the registers of the sorting unit (see below) before the first MP decoder completes, without any additional hardware. This allows absorbing some additional latency and initiating the post-processing MP* decoders before the initial MP decoder completes.

After the ν_c values are available, a sort of the checks in order of reliability is computed. To sort the checks in order of increasing reliability $|\mathcal{C}| - 1$ comparators are required to implement a tree structure, which should be pipelined to avoid increasing the critical path of the decoder. The number of clock cycles needed to obtain the complete sorted list is $\lceil \lambda/2 \rceil \times \lceil \log_2 |\mathcal{C}| \rceil$.

8.3.3 Overall Check-Agnosia Architecture

In this section, we detail the check-agnosia architecture corresponding to Algorithm 2 (that relies on MP decoding only, without brute-forcing). After the list of λ_{max} least reliable checks is obtained, the λ_{max} MP* decoders are performed. Depending on the time constraints and/or power budget, we may consider two different approaches, illustrated in Figure 8.1.

The first approach consists of performing the $\lambda_{\max}$ MP* decoders sequentially, reusing the same hardware as the one used for MP. Only $|\mathcal{Q}|$ extra multiplexors are required to choose between $\gamma'(q) = \gamma(q)$ or $\gamma'(q) = 0$, and $|\mathcal{C}|$ extra multiplexors are required to decide which syndromes belong to $\mathbf{s}_{|\mathfrak{N}_k}$, depending on the check c_λ. This approach of reusing hardware yields higher latency, but maybe interesting for a quantum computer with time constraints close to microseconds, *e.g.*, based on trapped ion technology [9].

For the second approach, the $\lambda_{\max}$ MP* decoders are performed in parallel, by using dedicated hardware. Moreover, the $\lambda_{\max}$ MP* decoders may start before the initial MP completes, using check-reliability values computed at an early iteration, that we will denote in the sequel by I_{ν_c}. This approach may be interesting for quantum technologies with more restrictive latency constraints, but having in mind that power can be also a limitation, as happens with superconducting qubits in which the decoder needs to reduce its power budget when it operates close to the quantum chip at cryogenic temperatures.

To illustrate the degree of complexity in hardware implementations and measure the gap between the proposed solutions to latency/power constraints, we analyze below the Pareto designs for the two approaches above, where the MP decoder uses either a flooded or a layered schedule. We provide the worst-case latency (simply referred to as latency), as well as the power consumption as a function of the nominal power consumption of the MP decoder, denoted by P_F or P_L, with a subscript indicating the flooded or layered architecture (we may reasonably assume that MP and MP* yield the same power consumption). For the latency value, we take into account the latency induced by sorting the check nodes according to their reliability (Section 8.3.2). The corresponding power consumption is not accounted for, we will assume it is negligible with respect to the power consumption of the MP decoder.

Flooded MP/MP* decoders:

1. Hardware reuse (sequential post-processing)

   MP flooded decoder + check reliability unit + one MP* flooded decoder running $\lambda_{\max}$ rounds:

   - Latency: $\left[(1+2I_F) \; + \; \lceil \lambda_{\max}/2 \rceil \, \lceil \log_2 |\mathcal{C}| \rceil \; + \lambda(1+2I_F)\right]/f_F$

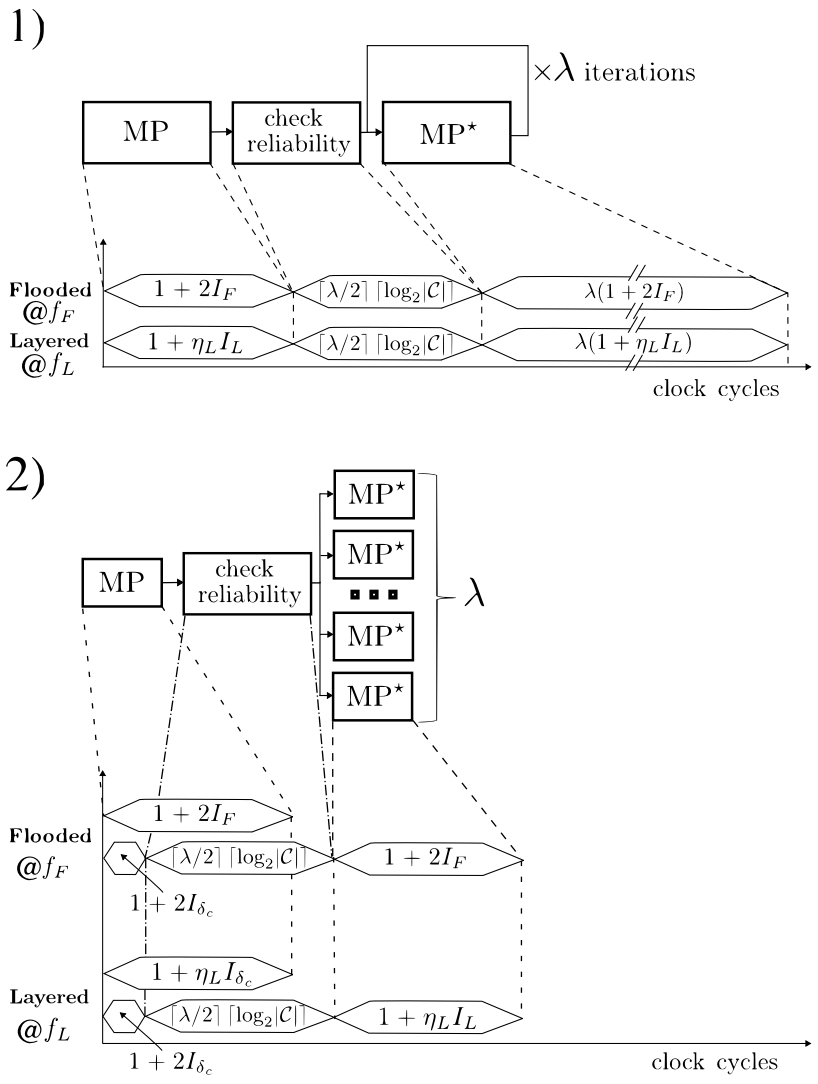


Figure 8.1: Comparison of different architectures for the check-agnosia decoder. The clock cycle diagram is included for the different proposals (Warning: drawing is not to scale). In case 1), MP and MP* use the same hardware. Note that in the figure, λ is used instead of $\lambda_{\max}$.

- Power: P_F

2. Dedicated hardware (parallel post-processing)

   MP flooded decoder + check reliability unit starting after iteration $I_{\nu_c} + \lambda_{\max}$ MP* flooded decoders running in parallel

   - Latency: $\left[(1 + 2I_{\nu_c}) \; + \; \lceil \lambda_{\max}/2 \rceil \, \lceil \log_2 |\mathcal{C}| \rceil \; + (1 + 2I_F)\right] / f_F$
   - Power: $(\lambda + 1) P_F$

Layered MP/MP* decoders:

1. Hardware reuse (sequential post-processing)

   MP layered decoder + check reliability unit + one MP* layered decoder running $\lambda_{\max}$ rounds:

   - Latency: $\left[(1 + \eta_L I_L) \; + \; \lceil \lambda_{\max}/2 \rceil \, \lceil \log_2 |\mathcal{C}| \rceil \; + \lambda_{\max}(1 + \eta_L I_L)\right] / f_L$
   - Power: P_L

2. Dedicated hardware (parallel post-processing)

   MP layered decoder + check reliability unit starting after iteration $I_{\nu_c} + \lambda_{\max}$ MP* layered decoders running in parallel

   - Latency: $\left[(1 + \eta_L I_{\nu_c}) \; + \; \lceil \lambda_{\max}/2 \rceil \, \lceil \log_2 |\mathcal{C}| \rceil \; + (1 + \eta_L I_L)\right] / f_L$
   - Power: $(\lambda_{\max} + 1) P_L$

8.3.4 Implementation Results

To illustrate the analysis from the previous section, we have implemented both flooded and layered NMS decoders on a Xilinx FPGA xcv095 board, for the B1[[882, 24]] code from [68]. The implemented decoders use finite precision arithmetic, with exchanged messages quantized on 6 bits, and *a posteriori* values quantized on 8 bits. The parity-check matrix (for both X and Z errors) is of size 441×882 (check-nodes $\times$ qubit-nodes) and has no four-cycles (thus, it satisfies the no-stopping subset condition, and we may safely apply Algorithm 2).

The flooded NMS / NMS* decoders achieve a maximum operating frequency $f_F = 100\,\text{MHz}$ (corresponding to a critical path of 10 ns), with 62% of the hardware resources of the device utilized, and a total power consumption $P_F = 5.5\,\text{W}$.

To implement the layered decoders, we use the 2-covering approach from [21], where 7 overlapping layers are used to cover 2 iterations, yielding a fractional number of layers per iteration $\eta_L = 3.5$. The layered NMS / NMS* decoders achieve a maximum operating frequency

Table 8.1: Latency (L) and power consumption (P) values for the Pareto designs in Section 8.3.3 ($I_{\max}$ is the table stands for I_F for flooded architectures, or for I_L for layered ones).

	Flooded	Layered
HW reuse	$L = 7.2\,\mu$s $P = 5.5\,$W	$L = 7.9\,\mu$s $P = 2.03\,$W
Dedicated HW $I_{\nu_c} = I_{\max}$	$L = 1.7\,\mu$s $P = 60.5\,$W	$L = 1.9\,\mu$s $P = 22.3\,$W
Dedicated HW $I_{\nu_c} = 3$	$L = 1.1\,\mu$s $P = 60.5\,$W	$L = 1.4\,\mu$s $P = 22.3\,$W

$F_L = 80\,$MHz (corresponding to a critical path of 12.5 ns), where about 25% is due to the logic depth of the operations and 75% is due to the routing limitations of the FPGA device. The decoder uses only 13% of the hardware resources of the device, and the total power consumption is around $P_L = 2.03\,$W.

We consider a maximum number of decoding iterations $I_F = 30$ for the flooded decoders, and $I_L = 15$ for the layered decoders (due to faster convergence). For the post-processing step, we consider a list of $\lambda_{\max} = 10$ least reliable checks (these parameters will be evaluated from the error correction perspective in Section 8.4). Latency and power consumption values are summarized in Table 8.1, for the Pareto designs considered in the previous section. Note that we consider two cases for the dedicated hardware scenario, in which the iteration I_{ν_c} (used to compute the check-node reliability values) is chosen to be either the last or the third iteration of the NMS decoder.

It can be observed that the layered architecture achieves latency values close to the flooded one, despite the fact it employs a degree of parallelism 3.5 times lower, while considerably reducing the power consumption. It is also worth noticing that the part of the latency due to the sorting unit is 0.45 μs for the flooded architectures, and 0.56 μs for the layered ones. To reduce the latency of the sorting unit further optimizations are possible (*i.e.*, carefully balancing the pipeline stages of the sorting unit by taking into account the maximum critical path latency of the MP decoder, splitting the sorting unit into layers in case of a layered schedule, or using a different clock domain for the sorting unit), which are however behind the scope of this work. We mention that the maximum frequency that can be reached for the sorting unit (implemented alone) is 230 MHz, which gives a lower bound on the achievable latency of 0.2 μs[8].

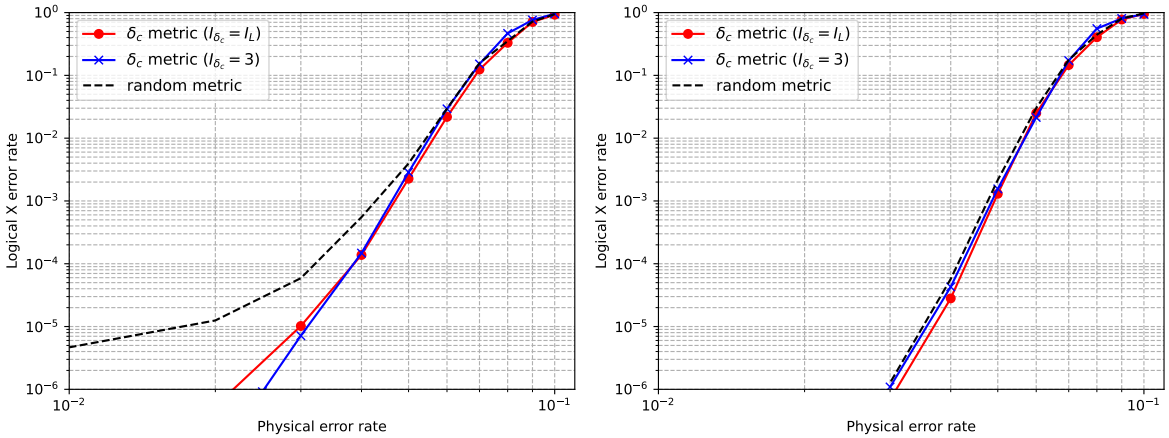
[8] At an operating frequency of 230 MHz, the power consumption for the sorting unit is 0.66 W, versus 0.26 W at at 100 MHz.

Moreover, as will be shown in Section 8.4, because of the highly degenerate structure of the codes, the layered schedule provides better error correction performance than the flooded one, even if the number of decoding iterations of the latter exceeds significantly the number of decoding iterations of the former (in fact, to get a flooded decoder that approaches the layered decoder, albeit not closely, one would have to go for at least 60 iterations, see Section 8.4). One last advantage of the layered architecture, reported in [21], is that the logical error rate can be considerably improved by processing layers in random order at each iteration. Such a random layer order can be implemented at a very low cost, as it only requires modifying the ROM memory that stores the layers' control sequence and including a deeper memory with a pseudo-random sequence of layers.

From the results presented before, it can be concluded that timing constraints can be in the range of the requirements reported in [2] for transmons and ion trap technology, between microseconds and milliseconds. However, these implementations do not meet the highly restrictive conditions of superconducting qubits in both time and power which are around 400 ns and 1W, see [88]. The difference compared to the fastest solution in Table 8.1 exceeds 3 times the time budget and it is more than one order of magnitude far in terms of power consumption. For these scenarios, it is important to remark that other approaches to implementation like ASICs or more advanced FPGA devices based on 16nm CMOS process or below (note that the xcvu095 belongs to the previous generation of 20nm) need to be explored in future work. Moreover, exploiting a ping-pong architecture that takes benefit of the pipeline registers to reduce the number of MP* decoders to half for the parallel implementation of flooded schedule can be a good proposal to reduce power consumption to almost half.

Extrapolating from state-of-the-art ASIC implementations of classical LDPC decoders, a clock frequency of 151 MHz is reported in [62] for a 65-nm CMOS ASIC implementation of a min-sum decoder using the layered architecture described in Section 8.3.1, for a regular LDPC code with characteristics similar to those of the B1 code investigated here[9]. The operating frequency is expected to further increase for the B1 code, given that both the parity check matrix and the layer size are smaller than that of the LDPC code in [62]. For $f_L = 151$ MHz, the latency of the layered architecture with parallel post-processing (dedicated hardware) is equal to $1\,\mu$s if $I_{\nu_c} = I_L$ (last iteration), and $0.73\,\mu$s if $I_{\nu_c} = 3$. Since the operating frequency increases with decreasing technology node, and assuming an inverse-linear frequency scaling [39], we may conclude that a latency constraint around 400 ns or below can be easily achieved for more advanced technology

[9] In [62], the parity check matrix is of size 648×1296, with column weight 3 and rows weight 6. Each layer consists of 216 checks (referred to as full-layer therein). The parity check matrix of the B1 code is of size 441×882, with column weight 3 and rows weight 6, and each layer consists of 126 checks.



(a) $B1[[882, 24]]$ flooded

(b) $B1[[882, 24]]$ layered

Figure 8.2: Performance of the check-agnosia decoder with different reliability metrics (B1 code)

nodes, *e.g.*, below 22 nm (today technology scaling is actually much lower).

Finally, we note that all the previous results assume that the check-agnosia decoder is implemented as in Algorithm 2. For the codes where Algorithm 2 cannot be applied, the latency will be a little bit worse than what was computed here, due to the brute-forcing step in Algorithm 1. We also provide in Appendix 8.4.3 arguments for why OSD post-processing (widely used today in the community for decoding of small to medium LDPC codes) is not a viable solution going forward, if trying to cope with the hardware implementation constraints.

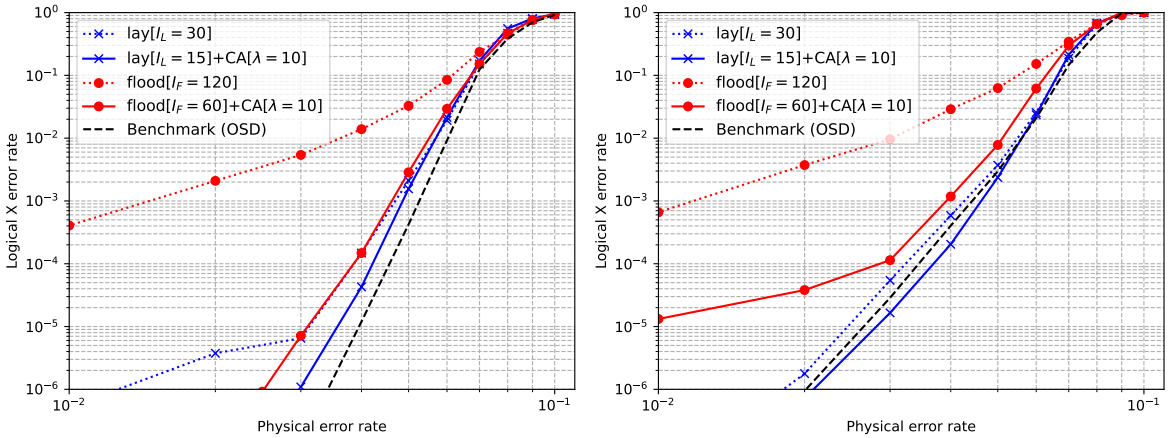
8.4 Error Correction Performance

In this section, we give numerical results for Check-Agnosia post-processing. We first provide numerical results on benchmark codes B1 and C2. We then show that we are able to get a threshold using CA post-processing for a reasonable value of $\lambda_{\max}$. Finally, we give a comparison in term of latency between CA and OSD, showing that for realistic use-cases, OSD cannot be used on parallel hardware.

8.4.1 Numerical Results of Check-Agnosia on Benchmark B1 and C2 Codes

In this section, we evaluate the error correction performance of the proposed check-agnosia post-processing. The codes used are B1[[882, 24]] and C2[[1922, 50]] from [68]. For both codes, the no-stopping subset condition from Section 8.2.2 is satisfied, hence in the following all simulations are performed using Algorithm 2 (without brute-forcing the system).

As our post-processing is targeted at decoding X and Z errors separately, we use an X noise



(a) $B1[[882, 24]]$ (b) $C2[[1922, 50, 16]]$

Figure 8.3: Analysis of the check-agnosia post-processing on codes B1 and C2 (ν_c metric, with $I_{\nu_c} = 3$).

model, and thus "physical error rate" does actually refer to the physical X error rate.

Our numerical simulations are consistent with the parameters used in Section 8.3.4. Precisely, we consider a finite-precision NMS decoder, using 6 bits for the exchanged messages, and 8 bits for the *a posteriori* values. Although in floating point precision the *a priori* values of the NMS decoder can be scaled to 1, in finite precision the *a priori* values have a non-negligible impact on convergence. In the simulations, we use the following parameters, optimized by extensive search. For the flooded decoder we set the *a priori* values to $\gamma = 12$ and the NMS scaling factor is set to $s_{\text{NMS}} = 0.875$. For the layered decoder we use $\gamma = 8$ and NMS scaling factor $s_{\text{NMS}} = 0.9375$. Note that scaling factors are a sum of powers of 2 and as such the scaling operation can be implemented efficiently in hardware, using only SHIFT and ADD operations.

For the maximum number of decoding iterations, we use $I_F = 60$ for the flooded decoder, and $I_L = 15$ for the layered decoder. The maximum number of decoding iterations for the flooded decoder is the only deviation with respect to the parameters used in Section 8.3.4 (where $I_F = 30$ was used). In fact, our goal here is to demonstrate the advantage in terms of error correction performance of the layered architecture as compared to the flooded one, even when the latter employs a significantly higher number of decoding iterations. For the post-processing part, we use $\lambda_{\max} = 10$.

Whenever the layered decoding is used, we add the random ordering perturbation introduced in Chapter 6 that was shown to significantly improve the decoding convergence.

Figure 8.2 shows the impact of the iteration I_{ν_c} used to select the checks in the post-processing, all simulations are done on the B1 code. For flooded simulations (Figure 8.2(a)), it is actually

beneficial to use the 3rd iteration for the metric instead of the last (60th iteration). The most probable explanation is that the relatively high number of iterations combined with the finite precision algorithm makes the metric less reliable after a larger number of iterations. For comparison purposes, we also consider a random metric, corresponding to a random choice of the $\lambda_{\max}$ checks in the post-processing. As it can be seen, the random metric exhibits a bad error floor, validating the metric used in that case.

For layered simulation (Figure 8.2(b)), all three curves are close by, since the layered NMS decoder with random layer ordering performs already very well.

In fact, the three metrics yield virtually the same performance of the check-agnosia decoder, but which is better than the layered NMS with 30 iterations and without post-processing in Figure 8.3(a). Although the metric is less important in this case, this shows that the perturbation introduced by the post-processing step in the input reliabilities has an impact on the decoding, and that it is better to run multiple decoders in parallel with perturbed inputs and fewer iterations rather than running a single decoder for a long time. As a whole, this validates the fact that the post-processing can be done efficiently using the dedicated hardware approach, increasing the post-processing parallelism and improving the latency.

We would also like to make a case for the choice $I_{\nu_c} = 3$. This hyperparameter can be optimized to get the best numerical results for a given code. However, the value 3 here was chosen for a different reason. Since both codes have girth 6, choosing I_{ν_c} to be equal to 3 guarantees that when the aposteriories are extracted, the decoder got access to the information of the biggest neighbourhood of each variable nodes *without loopy information*. This ensures that although it is very local, this information is also less noisy than information coming from later rounds.

In Figure 8.3, the post-processing is applied to the codes B1 and C2, with both flooded and layered schedules. For a comparison with the state of the art, in both figures, we added a dashed black curve of an optimized NMS-OSD decoder using 100 iterations, floating point NMS with a scaling factor of 0.625 [68]. Keep in mind that this decoder is not at all hardware-friendly, in terms of complexity, latency and power consumption, and it only serves as a reference. As it can be seen from both simulations, our results are matching closely the performance of the OSD post-processing, concretely showing the effectiveness of our hardware-friendly approach. In both figures, the results in red are the curves for flooded and in blue for layered. Each time, the dotted curves show the performance of the decoder without post-processing.

On the B1-code for the flooded schedule, the impact of the post-processing is clear, and the check-agnosia flooded decoder exhibits good performance while keeping a latency around $1.7\,\mu$s (taking into account $I_F = 60$). For the layered schedule, the use of the post-processing increases

the steepness of the waterfall. The check-agnosia layered decoder keeps the latency at around $1.4\,\mu$s. (Latency values above correspond to our FPGA implementation from Section 8.3.4.)

On the C2 Code, the performance gains for flooded are clear even if the post-processing suffers from a relatively high error floor. For layered scheduling, once again check-agnosia achieves better results in the error floor compared to no post-processing, closely matching the NMS-OSD curve.

Further numerical results are provided in Appendix 8.4.2, where we evaluate the error correction performance of the check-agnosia decoder on the family of T-codes from [76], showing a threshold phenomenon with hyperparameter $\lambda_{\max} = 0.02 \times |C|$.

8.4.2 Hyperparameter λ and Decoding Threshold

In Figure 8.4, we include additional numerical results giving a threshold for a constant rate family of LDPC codes, namely the T codes family from [76]. Since we are not aware of a simple way to build layers for this family of codes, we ran the simulations using a serial decoder, which is a fair approximation of the numerical results one would get with layered decoding. We use check-agnosia with hyperparameter $\lambda_{\max} = 0.02 \times |\mathcal{C}| = 0.01 \times |\mathcal{Q}|$, since for all the codes of the T

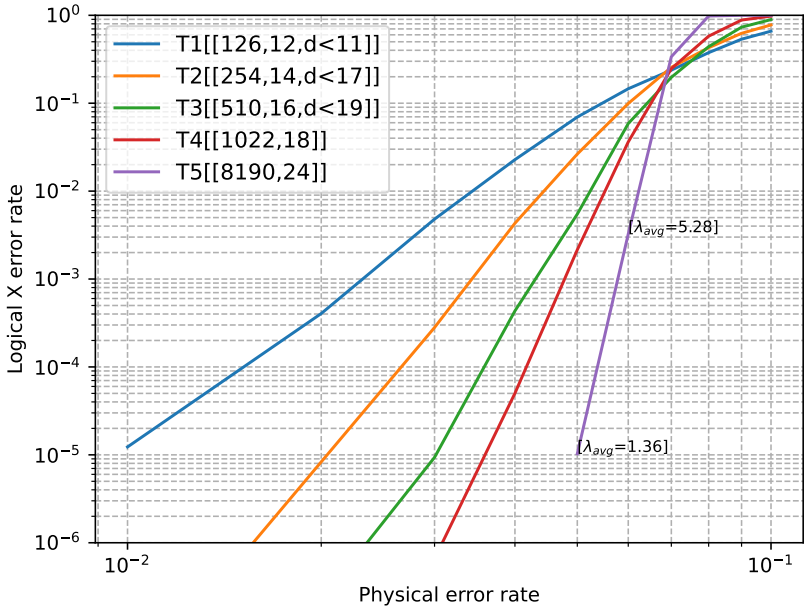


Figure 8.4: Check-agnosia threshold for the T codes family from [76]. Serial scheduling with random ordering and 15 iterations. Normalized min-sum with scaling factor 0.9375 and finite precision arithmetic, with 6-bit quantization for the *a priori* values and exchanged messages, and 8-bit quantization for the *a posteriori* values. For the post-processing, check-agnosia is used with $I_{\nu_c} = 3$ and $\lambda_{\max} = 0.02 \times |\mathcal{C}|$.

family, $|\mathcal{C}| = |\mathcal{Q}|/2$. The computational complexity of the algorithm hence is $(0.01\times)n^2 \times \log n$, where $n = |\mathcal{Q}|$ is the number of qubits, and this complexity can be spread between time and energy consumption depending on the architecture needs (see Fig 8.1). Furthermore, we make a case that the average complexity of the decoder is actually much better than that. On the figure, we also included the average number of inactivations (denoted λ_{avg}) for physical error rates 0.6 and 0.5, where the average values get very close to one (meaning only one inactivation might usually be necessary). Since this number goes close to one for low error-rates, it means that in practice the cost of the post-processing could only add a constant multiplicative overhead. This lambda average is particularly meaningful in a sequential architecture where we stop the post-processing as soon as the first post-processing converges. In the parallel architecture, it should still be possible to optimize the actual number of parallel runs if we have access to some prior information on the noise level, *e.g.*, by considering a pool of MP* decoders that serve for the post-processing of several logical qubits and are dynamically allocated between them.

8.4.3 Latency Comparison of CA and OSD

We provide below a comparison, in terms of latency, between the check agnosia proposal and the OSD post-processing solution. This comparison is similar to the method presented in [89], and is intended to clarify the differences between the two solutions with respect to hardware VLSI implementations.

We consider a layered MP decoder, with check-agnosia implemented through "Dedicated hardware" (that is, the $\lambda_{\max}$ MP* decoders are executed in parallel). Since the layered MP* decoders achieve a maximum operating frequency $F_L = 80\,\text{MHz}$ (corresponding to a critical path of 12.5 ns), the total latency of the check-agnosia post-processing is $(12.5 \times 3.5 \times 15) = 656.25\,\text{ns}$. We have omitted here the latency of the sorting unit, required to sort the check-nodes according to their reliability.

Considering now the OSD post-processing, we will omit again the latency of the sorting unit, required this time to sort the qubit-nodes according to their reliability. We will actually consider only the latency of the Gaussian elimination step required by OSD post-processing (and omit the latency of any other steps). Refs. [42, 78] below provide the two main highly parallel architectures known in the literature to perform Gaussian elimination over finite fields. However, in both cases, the number of clock cycles required to perform Gaussian elimination is equal to $(M^2 + M)/2$, where M is the number of rows of the parity-check matrix. Thus, the latency of the Gaussian elimination implementation is determined by $T_{\text{OSD}} = (M^2 + M)/2/f_{\text{OSD}}$, where f_{OSD} is the operating frequency. So the frequency required to achieve the same time budget as our proposal

is $f_{\text{OSD}} = (441^2 + 441)/2/656.25\,\text{ns} = 148.5\,\text{GHz}$. Such a frequency is completely unrealistic, and would certainly lead to timing violations in the design (note that it is $148.5/0.08 = 1856$ times larger than the one of the layered MP decoder). Besides, it would also translate into an extremely large power consumption, which typically increases linearly with the operating frequency.

8.5 Conclusion

This work introduced the check-agnosia algorithm, a new post-processing method improving on the syndrome-inactivation algorithm from a hardware-oriented viewpoint. Interestingly, although in the general case brute-forcing a small linear system may still be needed, for a large class of qLDPC codes the check-agnosia post-processing relies only on MP decoding, eliminating the need for any system solver. The proposed solution is flexible and it allows devising different hardware architectures, in order to meet the latency or the power constraints of the quantum system. The analysis carried out in the document, along with the hardware implementation results for MP decoders (our own implementation on an FPGA board, or results extrapolated from state-of-the-art ASIC implementations), showed that our solution can meet latency constraints of a wide range of quantum technologies, while providing state of the art error-correction performance, with hardware-accurate, finite-precision arithmetic. To the best of our knowledge, there is no prior work on the hardware architecture and implementation of a post-processing enhanced MP decoder for qLDPC codes. An interesting open question going forward would be to look at space-time decoding and see if the underlying graph structure lends itself well to the use of the check-Agnosia post-processing without system-solving.

Conclusion

As the perspective of building a quantum computer with meaningful power approaches, where the use of qLDPC codes for error correction will likely be mandatory, the results presented in this thesis are one step in a better comprehension of the message passing decoding for qLDPC codes, as well as offering practical solutions for possible near term applications in decoding of those codes.

Our work on the capacity of the toric code is a first step in the rigorous analysis of the decoding capabilities of message-passing decoders in the context of quantum codes. We hope that the notion of blindness used in conjunction with the decoding tree formalism can be used to prove similar results for other scheduling (serial decoding) and also other families of qLDPC codes. We also conjectured that there should exist a blindness property for the NMS based on preliminary numerical evidences, and we conjecture that by relaxing the notion of blindness, a similar statement could be shown for BP. Doing efficient layered decoding is an important step in performing fast decoding of quantum codes. As the comprehension of classical LDPC codes improved, most classical LDPC codes today come with a natural and efficient layer decomposition. However, because of the additional constraints of orthogonality, the problem of creating good CSS codes with a natural and good layering have been overlooked. In this thesis we show ways to create efficient layerings for existing codes, but in the future, it would be very interesting to find constructions with a natural layer decomposition. We also quickly touched on the subject of how the ordering can be an important tool in alleviating the effects of degeneracy. We proposed a simple randomization ordering procedure to test if the ordering has a meaningful impact on the error rate, and gave numerical evidences that often, the naive order given by the rows of the matrix is not the best one. This is a broad subject where much remains to be done, and that will hopefully be more looked at in the future. Finally, we proposed two new post-processings of practical interest for the decoding of qLDPC codes. One natural question is how message-passing decoders and related post-processing can be adapted to perform well in the circuit level noise, which is a noise model closest to the actual implementations of error-correction for quantum computing. As the first implementations of larger scale error correction are bound to arrive soon, this question will be of pressing concern in the near future.

Bibliography

[1] Zunaira Babar, Panagiotis Botsinis, Dimitrios Alanis, Soon Xin Ng, and Lajos Hanzo. Fifteen years of quantum LDPC coding and improved decoding strategies. *IEEE Access*, 3:2492–2519, 2015.

[2] Francesco Battistel, Christopher Chamberland, Kauser Johar, Ramon W. J. Overwater, Fabio Sebastiano, Luka Skoric, Yosuke Ueno, and Muhammad Usman. Real-Time Decoding for Fault-Tolerant Quantum Computing: Progress, Challenges and Outlook, 2023.

[3] Noah Berthusen, Dhruv Devulapalli, Eddie Schoute, Andrew M. Childs, Michael J. Gullans, Alexey V. Gorshkov, and Daniel Gottesman. Toward a 2D local implementation of quantum LDPC codes. *arXiv preprint arXiv:2404.17676*, 2024.

[4] Oana Boncalo, Gyorgy Kolumban-Antal, Alexandru Amaricai, Valentin Savin, and David Declercq. Layered LDPC decoders with efficient memory access scheduling and mapping and built-in support for pipeline hazards mitigation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(4):1643–1656, 2018.

[5] Oana Boncalo, Gyorgy Kolumban-Antal, David Declercq, and Valentin Savin. Code-design for efficient pipelined layered ldpc decoders with bank memory organization. *Microprocessors and Microsystems*, 63:216–225, 2018.

[6] Emmanuel Boutillon and Guido Masera. Hardware design and realization for iteratively decodable codes. In D. Declercq, Marc Fossorier, and E. Biglieri, editors, *Channel coding: Theory, algorithms, and applications*, pages 583–642. Elsevier, 2014.

[7] Sergey Bravyi, Martin Suchara, and Alexander Vargo. Efficient algorithms for maximum likelihood decoding in the surface code. *Physical Review A*, 90(3):032326, 2014.

[8] Nikolas P. Breuckmann and Jens N. Eberhardt. Balanced product quantum codes. *arXiv:2012.09271*, 2020.

[9] Nikolas P. Breuckmann and Jens N. Eberhardt. Quantum Low-Density Parity-Check Codes. *PRX Quantum*, 2:040101, Oct 2021.

[10] Andres I. Vila Casado, Miguel Griot, and Richard D. Wesel. LDPC decoders with informed dynamic scheduling. *IEEE Trans. on Communications*, 58(12):3470–3479, 2010.

[11] Christopher Chamberland and Pooya Ronagh. Deep neural decoders for near term fault-tolerant experiments. *Quantum Science and Technology*, 3(4):044002, jul 2018.

[12] Bin Cheng, Xiu-Hao Deng, Xiu Gu, Yu He, Guangchong Hu, Peihao Huang, Jun Li, Ben-Chuan Lin, Dawei Lu, Yao Lu, Chudan Qiu, Hui Wang, Tao Xin, Shi Yu, Man-Hong Yung, Junkai Zeng, Song Zhang, Youpeng Zhong, Xinhua Peng, Franco Nori, and Dapeng Yu. Noisy intermediate-scale quantum computers. *Frontiers of Physics*, 18(2), March 2023.

[13] Julien Du Crest, Mehdi Mhalla, and Valentin Savin. A blindness property of the min-sum decoding for the toric code, 2024. arXiv.2406.14968.

[14] Nicolas Delfosse. Decoding color codes by projection onto surface codes. *Physical Review A*, 89(1), 2014.

[15] Nicolas Delfosse, Michael E. Beverland, and Maxime A. Tremblay. Bounds on stabilizer measurement circuits and obstructions to local implementations of quantum LDPC codes. *arXiv preprint arXiv:2109.14599*, 2021.

[16] Nicolas Delfosse and Naomi H Nickerson. Almost-linear time decoding algorithm for topological codes. *arXiv:1709.06218*, 2017.

[17] Nicolas Delfosse and Naomi H. Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 5:595, December 2021.

[18] Nicolas Delfosse and Jean-Pierre Tillich. A decoding algorithm for css codes using the X/Z correlations. In *2014 IEEE International Symposium on Information Theory*, pages 1071–1075. IEEE, 2014.

[19] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[20] Irit Dinur, Min-Hsiu Hsieh, Ting-Chun Lin, and Thomas Vidick. Good quantum LDPC codes with linear time decoders. *arXiv preprint arXiv:2206.07750*, 2022.

[21] Julien du Crest, Francisco Garcia-Herrero, Mehdi Mhalla, Valentin Savin, and Javier Valls. Layered decoding of quantum LDPC codes. In *International Symposium on Topics in Coding*, 2023. arXiv:2308.13377.

[22] Julien du Crest, Francisco Garcia-Herrero, Mehdi Mhalla, Valentin Savin, and Javier Valls. Check-agnosia based post-processor for message-passing decoding of quantum LDPC codes. *Quantum 8, 1334*, 2024.

[23] 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, 2022.

[24] Frédéric Dupuis, Ashutosh Goswami, Mehdi Mhalla, and Valentin Savin. Purely quantum polar codes. In *2019 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2019.

[25] Shai Evra, Tali Kaufman, and Gilles Zémor. Decodable quantum ldpc codes beyond the n distance barrier using high-dimensional expanders. *SIAM Journal on Computing*, (0):FOCS20–276, 2022.

[26] Marc 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.

[27] Austin G Fowler. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $o(1)$ parallel time. *arXiv preprint arXiv:1307.1740*, 2013.

[28] Robert Gallager. *Low Density Parity Check Codes*. PhD thesis, MIT, Cambridge, Mass., September 1960.

[29] Konstantinos Georgopoulos, Clive Emary, and Paolo Zuliani. Modeling and simulating the noisy behavior of near-term quantum computers. *Phys. Rev. A*, 104:062432, Dec 2021.

[30] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. JHU press, 2013.

[31] Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, 2023. arXiv:2207.06431.

[32] Daniel Gottesman. Stabilizer codes and quantum error correction, 1997.

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

[34] Antoine Grospellier, Lucien Grouès, Anirudh Krishna, and Anthony Leverrier. Combining hard and soft decoders for hypergraph product codes. *Quantum*, 5:432, 2021.

[35] Shouzhen Gu, Christopher A Pattison, and Eugene Tang. An efficient decoder for a linear distance quantum LDPC code. *arXiv preprint arXiv:2206.06557*, 2022.

[36] Arnaud Guyader and Eric Fabre. Dealing with short cycles in graphical codes. In *2000 IEEE International Symposium on Information Theory (Cat. No. 00CH37060)*, page 10. IEEE, 2000.

[37] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal, 29:147–160*, 1950.

[38] Matthew B. Hastings, Jeongwan Haah, and Ryan O'Donnell. Fiber bundle codes: Breaking the $N^{1/2}$ polylog(N) barrier for quantum LDPC codes. *arXiv:2009.03921*, 2020.

[39] John Hauser. MOSFET device scaling. In *Handbook of Semiconductor Manufacturing Technology*. Boca Raton, FL: CRC Press, 2008.

[40] Timo Hillmann, Lucas Berent, Armanda O. Quintavalle, Jens Eisert, Robert Wille, and Joschka Roffe. Localized statistics decoding: A parallel decoding algorithm for quantum low-density parity-check codes, 2024.

[41] Dale E. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. In *IEEE Workshop on Signal Processing Systems (SIPS)*, pages 107–112, 2004.

[42] Bertrand Hochet, Patrice Quinton, and Yves Robert. Systolic solution of linear systems over gf(p) with partial pivoting. In *1987 IEEE 8th Symposium on Computer Arithmetic (ARITH)*, pages 161–168, 1987.

[43] Adam Holmes, Mohammad Reza Jokar, Ghasem Pasandi, Yongshan Ding, Massoud Pedram, and Frederic T. Chong. NISQ+: Boosting Quantum Computing Power by Approximating Quantum Error Correction. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, page 556–569. IEEE Press, 2020.

[44] Min-Hsiu Hsieh, Todd A. Brun, and Igor Devetak. Entanglement-assisted quantum quasi-cyclic low-density parity-check codes, 2009.

[45] Xiao-Yu Hu, Evangelos Eleftheriou, and Dieter-Michael Arnold. Regular and irregular progressive edge-growth Tanner graphs. *IEEE Transactions on Information Theory*, 52(51):386–398, 2005.

[46] IEEE-802.11n. Standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 5: Enhancements for higher throughput, 2009.

[47] Alexei Y. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1), 2003.

[48] Alexey A. Kovalev and Leonid P. Pryadko. Quantum kronecker sum-product low-density parity-check codes with finite rate. *Phys. Rev. A*, 88:012311, Jul 2013.

[49] A. Dinesh Kumar and Ambedkar Dukkipati. A two stage selective averaging LDPC decoding. In *2012 IEEE International Symposium on Information Theory Proceedings*, pages 2866–2870, 2012.

[50] Kao-Yueh Kuo and Ching-Yi Lai. Exploiting degeneracy in belief propagation decoding of quantum codes. *npj Quantum Information*, 8(1):111, 2022. arXiv:2104.13659.

[51] Anthony Leverrier, Jean-Pierre Tillich, and Gilles Zémor. Quantum expander codes. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 810–824. IEEE, 2015.

[52] Anthony Leverrier and Gilles Zémor. Efficient decoding up to a constant fraction of the code length for asymptotically good quantum codes. *arXiv preprint arXiv:2206.07571*, 2022.

[53] Anthony Leverrier and Gilles Zémor. Quantum Tanner codes. *arXiv:2202.13641*, 2022.

[54] Ning Liu, Xiaohang Ren, Yunhe Liu, and Xingchen Liu. Research of ldpc coding and encoding for satellite communication. In *2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, pages 63–68, 2018.

[55] Ye-Hua Liu and David Poulin. Neural belief-propagation decoders for quantum error-correcting codes. *Physical review letters*, 122(20):200501, 2019.

[56] Gianluigi Liva, Enrico Paolini, Balazs Matuz, and Marco Chiani. A decoding algorithm for LDPC codes over erasure channels with sporadic errors. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 458–465, 2010.

[57] Michael G. Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.

[58] Mohammad M. Mansour and Naresh R. Shanbhag. VLSI architectures for SISO-APP decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(4):627–650, 2003.

[59] Yongyi Mao and Amir H. Banihashemi. Decoding low-density parity-check codes with probabilistic schedule. In *2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (IEEE Cat. No. 01CH37233)*, volume 1, pages 119–123. IEEE, 2001.

[60] Sisi Miao, Alexander Schnerring, Haizheng Li, and Laurent Schmalen. Neural belief propagation decoding of quantum LDPC codes using overcomplete check matrices. In *2023 IEEE Information Theory Workshop (ITW)*, pages 215–220. IEEE, 2023.

[61] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 527–540, New York, NY, USA, 2019. Association for Computing Machinery.

[62] Thien Truong Nguyen-Ly, Valentin Savin, Khoa Le, David Declercq, Fakhreddine Ghaffari, and Oana Boncalo. Analysis and design of cost-effective, high-throughput LDPC decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(3):508–521, 2017.

[63] Xiaotong Ni. Neural Network Decoders for Large-Distance 2D Toric Codes. *Quantum*, 4:310, Aug 2020.

[64] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[65] Josias Old and Manuel Rispler. Generalized belief propagation algorithms for decoding of surface codes. *Quantum*, 7:1037, 2023. arXiv:2212.03214 .

[66] Dimiter Ostrev, Davide Orsucci, Francisco Lázaro, and Balazs Matuz. Classical product code constructions for quantum calderbank-shor-steane codes, 2022.

[67] Pavel Panteleev and Gleb Kalachev. Asymptotically good quantum and locally testable classical LDPC codes. *arXiv:2111.03654*, 2021.

[68] Pavel Panteleev and Gleb Kalachev. Degenerate Quantum LDPC Codes With Good Finite Length Performance. *Quantum*, 5:585, nov 2021.

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

[70] Aiden Price and Joanne Hall. A survey on trapping sets and stopping sets. *arXiv preprint arXiv:1705.05996*, 2017.

[71] Nithin Raveendran, Emmanuel Boutillon, and Bane Vasić. Turbo-XZ algorithm: Low-latency decoders for quantum LDPC codes. In *2023 12th International Symposium on Topics in Coding (ISTC)*, pages 1–5. IEEE, 2023.

[72] Nithin Raveendran and Bane Vasic. Trapping set analysis of horizontal layered decoder. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[73] Nithin Raveendran and Bane Vasić. Trapping sets of quantum LDPC codes. *Quantum*, 5:562, October 2021.

[74] Tom Richardson and Shrinivas Kudekar. Design of low-density parity check codes for 5g new radio. *IEEE Communications Magazine*, 56(3):28–34, 2018.

[75] Alex Rigby, JC Olivier, and Peter Jarvis. Modified belief propagation decoders for quantum low-density parity-check codes. *Physical Review A*, 100(1):012330, 2019. arXiv:1903.07404.

[76] 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. arXiv:2005.07016.

[77] Wouter Rozendaal and Gilles Zémor. Analysis of the error-correcting radius of a renormalisation decoder for Kitaev's toric code. *IEEE Transactions on Information Theory*, 2024. arXiv:2309.12165.

[78] Andreas Rupp, Jan Pelzl, Christof Paar, MC Mertens, and Andrey Bogdanov. A parallel hardware architecture for fast gaussian elimination over GF(2). In *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 237–248. IEEE, 2006.

[79] Valentin Savin. Iterative LDPC decoding using neighborhood reliabilities. In *IEEE Int. Symp. on Inf. Theory (ISIT)*, pages 221–225, 2007.

[80] Valentin Savin. LDPC decoders. In *Channel coding: Theory, algorithms, and applications*, pages 211–260. Elsevier, 2014.

[81] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3-4):379–423 and 623–656, 1948.

[82] Eran Sharon, Simon Litsyn, and Jacob Goldberger. An efficient message-passing schedule for LDPC decoding. In *2004 23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, pages 223–226. IEEE, 2004.

[83] Michael Sipser and Daniel A Spielman. Expander codes. *IEEE transactions on Information Theory*, 42(6):1710–1722, 1996.

[84] Robert M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.

[85] Swamit S. Tannu, Zachary A. Myers, Prashant J. Nair, Douglas M. Carmean, and Moinuddin K. Qureshi. Taming the Instruction Bandwidth of Quantum Computers via Hardware-Managed Error Correction. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, page 679–691, New York, NY, USA, 2017. Association for Computing Machinery.

[86] 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.

[87] Maxime A. Tremblay, Nicolas Delfosse, and Michael E. Beverland. Constant-overhead quantum error correction with thin planar connectivity. *Physical Review Letters*, 129(5):050504, 2022.

[88] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. QECOOL: On-line quantum error correction with a superconducting decoder for surface code. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 451–456. IEEE, 2021.

[89] Javier Valls, Francisco Garcia-Herrero, Nithin Raveendran, and Bane Vasić. Syndrome-based min-sum vs OSD-0 decoders: FPGA implementation and analysis for quantum LDPC codes. *IEEE Access*, 9:138734–138743, 2021.

[90] Savvas Varsamopoulos, Koen Bertels, and Carmen Garcia Almudever. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, 69(2):300–311, 2019.

[91] Yun-Jiang Wang, Barry C Sanders, Bao-Ming Bai, and Xin-Mei Wang. Enhanced feedback iterative decoding of sparse quantum codes. *IEEE Transactions on Information Theory*, 58(2):1231–1241, 2012. arXiv:0912.4546.

[92] Niclas Wiberg. *Codes and decoding on general graphs*. PhD thesis, Likoping University, Sweden, 1996.

[93] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. SIAM, 2024.

[94] Stasiu Wolanski and Ben Barber. Ambiguity clustering: an accurate and efficient decoder for qldpc codes, 2024. arXiv:2406.14527.

[95] Hanwen Yao, Waleed Abu Laban, Christian Häger, Henry D. Pfister, et al. Belief propagation decoding of quantum LDPC codes with guided decimation, 2023. preprint, arXiv:2312.10950.

[96] Juntan Zhang, Yige Wang, Marc Fossorier, and Jonathan S. Yedidia. Iterative decoding with replicas. *IEEE Transactions on Information Theory*, 53(5):1644–1663, 2007.