Karlsruhe Institute of Technology
Communications Engineering Lab
Prof. Dr.-Ing. Laurent Schmalen

Decoding of Quantum LDPC Codes with Modified Belief Propagation Decoder

Master Thesis

Alexander Schnerring

Advisor	:	Prof. Dr.-Ing. Laurent Schmalen
Supervisor	:	M.Sc. Sisi Miao

Start date	:	01.11.2021
End date	:	16.05.2022

Declaration

With this statement I declare that I have independently completed the above master's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

Karlsruhe, 16.05.2022

Alexander Schnerring

Abstract

Quantum low-density parity-check (QLDPC) codes are promising candidates to perform quantum error correction. By treating quantum codes as additive codes over GF(4), they can be decoded efficiently using a quaternary belief propagation (BP) decoder. However, the decoding performance of BP suffers from unavoidable short cycles in the underlying Tanner graph, as well as quantum degeneracy, a phenomenon with no classical counterpart. To alleviate these issues, several modifications have been proposed, including update scheduling, message normalization and the application of post-processing procedures. While these modifications result in increased decoding performance, they often come at the cost of increased computational complexity. This is especially problematic in quantum computing, where the fast decoherence of quantum states necessitates low-complexity decoders.
In this thesis, we investigate the effect of several modifications on the performance of quaternary BP when decoding hypergraph product (HP) codes, a special class of QLDPC codes constructed from two classical codes. We propose a low-complexity post-processing procedure that incorporates knowledge of the code structure and demonstrate that our approach closes the performance gap between BP and an optimal decoder in the low error rate region. Furthermore, we compare the performances of BP decoding with different update schedules. We demonstrate that the performance of quaternary BP on HP codes is related to the performance of binary BP on the corresponding classical codes and propose a method to choose a good classical code. We show that message normalization can help to tackle message overestimation and demonstrate that current approaches of constant message normalization can be outperformed by varying the normalization scalar during decoding. We propose a parameterized function specifying this scalar and perform an optimization of the according parameters.

Acknowledgment

First of all, I would like to thank my parents, for always supporting me throughout my studies and beyond. Thank you to my supervisor, Sisi Miao, for her patience, advice and support. Thank you to Prof. Dr.-Ing. Laurent Schmalen, for giving insightful hints and providing me helpful code. Thank you to Dr.-Ing Holger Jäkel for introducing hundreds of students (including myself) to the fascinating field of communication engineering and accompanying us throughout the years as a fantastic lecturer. Thank you to the staff of the CEL and fellow students for creating a pleasant working environment and relaxing lunch breaks. And finally, thanks to my friends and family, for showing me that there is more to life than studying.

Contents

1. Introduction

1.1. Motivation

Quantum computers are devices that use quantum mechanical principles to perform computation. It is believed that certain problems can be solved more efficiently on a quantum computer than on a classical computer [NC10]. A particularly prominent example is the task of finding prime factors of an integer. While no efficient classical algorithm is known, Shor proposed a quantum algorithm that solves this task in sub-exponential time [Sho94]. Once quantum computers can be built on a large scale, public-key cryptography systems such as the RSA scheme could be broken, as they rely on the intractability of prime factorization. Grover demonstrated that a quantum algorithm can significantly speed up the search through some unstructured database [Gro96]. Other fields of application involve the simulation of quantum mechanical systems, as no efficient simulation on classical computers is known [NC10].
There are still some major challenges to be addressed, before quantum computers can compete with classical computers. "Any quantum computer requires a system with long-lived quantum states and a way to interact with them", as stated by Gottesman [Got97]. However, quantum states are very delicate, as small interactions of a quantum system with it's environment causes it to *decohere* [Got97]. A reliable quantum computer thus has to account for this high susceptibility to noise. The delicacy of quantum states is met with the application of quantum error correction (QEC). Just like in classical coding, quantum information is protected against the effects of noise by adding redundant information. The original information can then be recovered from the (potentially errorneous) encoded information, given that the noise keeps within limits.
While QEC is closely related to classical error correction, techniques used in classical settings rely on assumptions which are not met in quantum systems. For example, a continuum of errors may occur on a single qubit, since states of qubits can take on a continuum of values. Moreover, the direct measurement of qubits destroys information, making decoding based on direct observation of the quantum codeword impossible. [NC10, Got97]
Despite these challenges, a theory of QEC has been developed over the past decades. It can be shown that it suffices to correct the Pauli errors in order to correct the most general possible noise. This rather surprising result is known as the *discretization of noise* and stems from the fact that the Pauli matrices X, Y and Z form a complete set: They can be used to describe any transformation on qubits and thus, can express general noise [NC10]. Another key insight is that, although a direct measurement of a quantum codeword is not possible, information about the error can still be obtained: For a measurement not to disturb the actual state of the code, this measurement has to yield the same result for every valid codeword in the codespace [Sho95]. This fact ultimately led to the development of the stabilizer formalism, which *defines* the code space as all states that are not corrupted by such a *syndrome measurement.* [Got97]

The stabilizer formalism allows us to treat quantum stabilizer codes (QSCs) as additive codes over GF(4). It establishes a connection between quantum codes and the well-studied field of classical codes [CRSS97]. However, fundamental properties of quantum mechanics

translate to a duality constraint on these codes. This constraint represents a major challenge in the design of QSCs. Besides theoretical aspects, a practical QEC system requires the availability of efficient decoders. A fast decoding algorithm is of particular interest in quantum computation, where errors have to be identified faster than they accumulate [TZ14]. In addition, it is desirable to have a small number of interactions per qubit due to the delicacy of quantum states [MMM04].
The aforementioned aspects inspired MacKay et al. to construct sparse QSCs, also referred to as quantum low-density parity-check (QLDPC) codes [MMM04]. QLDPC codes draw inspiration from classical low-density parity-check (LDPC) codes, a class of linear codes introduced by Gallager in 1962 [Gal62]. LDPC codes are widely used in classical settings, as they show outstanding error correction performance [MN02] close to the theoretical limit derived by Shannon [Sha48]. The iterative belief propagation (BP) algorithm, originally proposed by Pearl [Pea82], constitutes a satisfying solution to the decoding problem, as it's complexity scales only linear in the blocklength [KFL01]. During BP decoding, probabilistic messages are exchanged over the *Tanner* graph of a code, a bipartite graph defined by the parity-check matrix (PCM) associated to the code. "Decoded with a BP algorithm, sparse codes are amongst the best known classical error correction codes. For this reason, it is desirable to generalize them to the quantum setting", as stated by Poulin and Chung [PC08].

1.2. Existing Work

BP decoding of QLDPC codes comes with several decoding issues: The duality constraint leads to unavoidable cycles of length four [PC08]. Unfortunately, short cycles are known to impair the decoding performance of BP, which is only guaranteed to produce the exact solution when the graph is a tree (i.e., free of cycles) [KFL01]. A property of quantum codes with no classical analogue is referred to as *degeneracy*. DiVincenzo et al. describe degeneracy as the fact that "two errors may be indistinguishable by their error syndromes, but may nevertheless be both correctable" [DSS98]. While degeneracy can improve the performance of a quantum code, the performance of BP decoding is typically impaired by this property [PC08].
A common approach to alleviate these issues is to apply a post-processing procedure subsequent to BP decoding. Poulin and Chung propose several methods to help overcome the aforementioned issues [PC08]. Among them, the technique of "random perturbation" showed to be most effective: If the syndrome is not matched by the decoder after a maximum number of iterations, a frustrated check node is chosen randomly. The error probabilities of all neighboring variable nodes are perturbed randomly and another round of BP is invoked. This process is repeated, until all checks are satisfied or a maximum number of rounds is reached. However, Raveendran and Vasić demonstrated that random perturbation requires many iterations to achieve significant improvements over BP without this post-processing procedure [RV21].
Another post-processing approach was proposed by Panteleev and Kalachev: They apply ordered statistics decoding (OSD) (originally introduced in [FL95]) posterior to BP decoding in order to force the decoder to match the observed syndrome [PK21]. The authors report significant improvements in the decoding performance when applying OSD. However, as OSD scales cubically with the blocklength of the code [RV21], the high complexity may be

prohibitive for the application in quantum computing, as quantum states decohere quickly [KL20, RV21].
A common approach of low complexity to mitigate the effect of message overestimation is to normalize the messages by a scalar [CF02]. As the conventional quaternary BP exchanges multi-dimensional messages, message normalization cannot be directly applied. To overcome this obstacle, Kuo and Lai proposed a refined BP decoder, exchanging scalar messages [KL20], hence enabling the application of message normalization to BP of quantum codes. The authors show that message normalization increases the decoding performance of BP on different quantum codes. In addition, Kuo and Lai compare the performance of different update schedules: The update schedule determines the order in which nodes in the Tanner graph are updated during BP decoding. Theoretical [SLG07] and experimental [CGW10] results indicate that the update schedule affects the convergence speed of BP in terms of iterations. Experimental results suggest that a serial update schedule may outperform a parallel update schedule in terms of decoding performance [KL20].

1.3. Outline of this Thesis

This thesis is organized in the following way. In Chapter 2, we introduce the *qubit*, which is the most most basic unit in quantum computation. We introduce the stabilizer formalism and establish the connection between quantum codes and additive classical codes over GF(4). We show how quantum mechanical properties translate to a duality constraint of these codes and provide a short description of Calderbank-Shor-Steane (CSS) codes. Chapter 2 closes with a summary of the quaternary BP decoder in the log-domain, as proposed by Lai and Kuo [LK21].
In Chapter 3, we introduce the hypergraph product (HP) code construction. HP codes were proposed by Tillich and Zemor [TZ14] and provide a means to convert any two classical codes to a valid quantum code. We propose a new notation for quantum errors, particularly useful to describe patterns in the context of HP codes. Subsequently, the error analysis of an exemplary HP code is performed. We then establish a connection between pairs of errors for *symmetric* HP codes and describe some decoding issues associated to quaternary BP on HP codes in more detail.
In Chapter 4, we compare the decoding performance of parallel and serial BP_4 on HP codes. We demonstrate how the choice of the classical code used to construct symmetric HP codes affects the decoding performance and, based on this observation, propose a method to find a classical code that results in a HP code with good performance.
Chapter 5 addresses the effect of message overestimation. We demonstrate that fixed initialization and message normalization result in an improved decoding performance on HP codes. In contrast to current literature on message normalization for quantum codes [KL20], we propose to choose a variable scalar when normalizing the exchanged messages. We show that, using our variable message normalization (VMN) approach, the performance of a decoder with constant message normalization can be slightly outperformed.
In Chapter 6, we propose a low-complexity post-processing procedure, that incorporates knowledge of the structure of decoding failure causing errors, obtained from the analysis in Chapter 3.
In Chapter 7, we conclude the thesis with some final remarks.

1.4. Notation

We use bold letters to denote non-scalar quantities. Matrices are represented by upper case letters. We denote the row m of a matrix $\boldsymbol{X}$ by $\boldsymbol{X}_m$. To refer to the column n of $\boldsymbol{X}$, we write $\boldsymbol{X}_{:,n}$. A single entry in row m and column n is denoted by X_{mn}. Vectors are represented by lower case letters. To refer to the n-th entry of a vector $\boldsymbol{x}$, we write $\boldsymbol{x}_n$. Since this notation may interfere with subscripts of matrices or vectors, we use brackets $[\cdot]$ to avoid confusion in this case. As an example, the n-th column of the matrix $\boldsymbol{X}_\mathrm{a}$ is referred to as $[\boldsymbol{X}_\mathrm{a}]_{:,n}$. Similarly, the n-th entry of a vector $\boldsymbol{x}_\mathrm{a}$ is written as $[x_\mathrm{a}]_n$. We start to count at the index 0, e.g., the first column is indexed by $n = 0$.

While the four Pauli operators $\{\boldsymbol{I}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}\}$ are matrices, we use non-bold letters to refer to them in the context of the stabilizer formalism and BP decoding. This (1) serves the purpose of emphasizing that we think of these operators as entries of matrices or vectors rather than matrices themselves, and (2) is consistent with the convention largely used in the literature. Furthermore, we follow the convention in the literature to denote quantum errors by upper case letters, despite the fact that they can be thought of as vectors containing Pauli matrices.
The Galois field with two elements is denoted by $\mathbb{F}_2$ and the Galois field with four elements is denoted by GF(4). The set of complex numbers is denoted by $\mathbb{C}$.

2. Theoretical Basis

2.1. Quantum Information Theory

We start with a brief introduction on the basics of quantum computation. We mainly focus on the mathematical description of qubits and operations acting on qubits. For a more detailed description, we refer the reader to the excellent text book by Nielsen and Chuang [NC10].

2.1.1. Qubits

Classical computation revolves around the most basic unit of information, the *bit*. Analogously, the *quantum bit* (or *qubit*) constitutes the most basic unit in quantum computation. A qubit can be described as an abstract mathematical object. More precisely, the *state* of a qubit is a vector in a two-dimensional complex vector space. Similar to the states of a classical bit, 0 and 1, we denote the *computational basis states* of this vector space as $|0\rangle$ and $|1\rangle$. Unlike a classical bit, which can only have the state 0 or 1, the state $|\psi\rangle$ of a qubit can be a linear combination of these computational basis states [NC10]:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,,$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Note that a measurement of a quantum system can only reveal partial information on the state of this system. When the state of a qubit $|\psi\rangle$ is measured, the outcome is 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. Furthermore, the state collapses into the measured state, i.e., the measurement disturbs the state of the measured quantum system. We elaborate on mathematical properties of quantum measurements in the following section.

Qubits can be used for *quantum computation*. Similar to classical computing, *quantum circuits* are composed of *quantum gates* (or *operators*), processing quantum information. Since the state of a qubit can be described as a complex vector in a two-dimensional vector space, they are frequently represented in a vector notation in quantum computation. This allows for a convenient description of operators as matrices. The computational basis states $|0\rangle$ and $|1\rangle$ form an orthonormal basis and are assigned the vectors

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad\qquad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Thus, a superposed state $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ has the representation

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

An operator can be described as a linear transform $\boldsymbol{U}$, acting on the state of a qubit. Suppose a qubit has the state $|\psi'\rangle = \alpha'\,|0\rangle + \beta'\,|1\rangle = \boldsymbol{U}\,|\psi\rangle$ after the operator acted on it. The normalization constraint requires $\boldsymbol{U}$ to be length-preserving, i.e., $|\alpha'|^2 + |\beta'|^2 = 1$

still has to be satisfied after the qubit was processed by the gate. Matrices satisfying this property are called *unitary.* A unitary matrix $\boldsymbol{U}$ satisfies $\boldsymbol{U}\boldsymbol{U}^\dagger = \boldsymbol{I}$, where $\boldsymbol{U}^\dagger$ denotes the *adjoint* (or complex conjugate transpose) of $\boldsymbol{U}$. It turns out that this condition is necessary and sufficient: Any quantum gate can be described by a unitary matrix and any unitary matrix describes a valid quantum gate. [NC10]

2.1.2. Pauli Matrices

Four operators that play a particularly important role in the field of QEC are the *Pauli* matrices $\boldsymbol{I}, \boldsymbol{X}, \boldsymbol{Y}$ and $\boldsymbol{Z}$. The identity matrix $\boldsymbol{I}$ acts trivial, i.e., it leaves the state of a qubit unchanged. The matrix $\boldsymbol{X}$ (also referred to as the quantum NOT gate) exchanges the computational basis states, i.e., $\boldsymbol{X}|0\rangle = |1\rangle$ and $\boldsymbol{X}|1\rangle = |0\rangle$. In matrix notation, the $\boldsymbol{X}$ gate has the representation

$$\boldsymbol{X} \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

In contrast to a classical bit, the phase of qubit can also be flipped. The corresponding gate is referred to as the $\boldsymbol{Z}$ gate. It leaves $|0\rangle$ unchanged and flips the sign of $|1\rangle$:

$$\boldsymbol{Z} \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The $\boldsymbol{Y}$ gate can be viewed as a simultaneous bit and phase flip and is defined as

$$\boldsymbol{Y} \equiv i\boldsymbol{X}\boldsymbol{Z} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}.$$

If a state $|\psi\rangle$ satisfies the equation $\boldsymbol{U}|\psi\rangle = \lambda|\psi\rangle$, we refer to it as an *eigenstate* of $\boldsymbol{U}$, i.e., the operator only scales eigenstates by a factor of λ. In quantum mechanics, operators are associated to measurements: The measurement of a quantum state projects that state onto an eigenstate of the corresponding operator. The measurement outcome will then be the eigenvalue corresponding to the eigenstate onto which the measured state was projected. This projection happens in a probabilistic manner, where the probability of projecting the measured state onto a certain eigenstate (and hence obtaining the corresponding eigenvalue as the measurement outcome) depends on the inner product of the measured state and this eigenstate.
In this sense, we can understand the measurement described in the section above as measuring the eigenvalue of the operator $\boldsymbol{Z}$: $\boldsymbol{Z}$ has two eigenstates, $|0\rangle$ and $|1\rangle$ with eigenvalues $(+1)$ and (-1) respectively. A measurement of $\boldsymbol{Z}$ projects a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ onto $|0\rangle$ with probability $|\alpha|^2$, in which case the measurement yields the result $(+1)$. In the other case, $|\psi\rangle$ is projected onto $|1\rangle$ with probability $|\beta|^2$, yielding the measurement outcome (-1).

The physical properties of a quantum measurement translate to mathematical properties of the operators associated to these measurements. Consider two operators, $\boldsymbol{A}$ and $\boldsymbol{B}$. We can simultaneously measure their eigenvalues, if the associated matrices commute, i.e., if

$\boldsymbol{AB} = \boldsymbol{BA}$. This stems from the fact that two commuting matrices can be simultaneously diagonalized, i.e., they share the same eigenvectors. On the other hand, if $\boldsymbol{AB} = -\boldsymbol{BA}$, the matrices are said to *anti-commute.*
As we will see further below, we are interested in the commutation relation of the Pauli matrices: Two Pauli matrices commute if at least one of them is $\boldsymbol{I}$ or if both matrices are identical. In other words, if both matrices are non-identity and different, they anti-commute.

2.1.3. Multiple Qubit Systems

A system composed of N qubits is characterized by a unit vector in the complex Hilbert space $\mathbb{C}^{2^N}$. Be aware that any vector describing a valid quantum state has to be of length one due to the normalization constraint. The basis vectors used to describe an N qubit system are obtained as $|\boldsymbol{z} : \boldsymbol{z} \in \mathbb{F}_2^N\rangle$, where $\mathbb{F}_2^N$ is the Galois field consisting of two elements. This means that we require 2^N basis vectors, as $|\mathbb{F}_2^N| = 2^N$.

Example 1. The quantum state of a system with $N = 2$ qubits is described as a superposition of the computational basis states $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, i.e.,

$$|\psi\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle\,,$$

where $\alpha, \beta, \gamma, \delta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$.

In this sense, the state $|0...0\rangle$ does *not* describe the zero-vector but the basis vector where each qubit is in the state $|0\rangle$. Tensor products provide a mathematical framework to construct larger vector spaces from basis vector spaces, enabling us to combine states of single qubit systems into more complex, larger systems. The tensor product of two matrices (also known as *Kronecker* product) has the matrix representation

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} A_{11} \cdot \boldsymbol{B} & A_{12} \cdot \boldsymbol{B} & \dots & A_{1N_\mathrm{A}} \cdot \boldsymbol{B} \\ A_{21} \cdot \boldsymbol{B} & A_{22} \cdot \boldsymbol{B} & \dots & A_{2N_\mathrm{A}} \cdot \boldsymbol{B} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M_\mathrm{A}1} \cdot \boldsymbol{B} & A_{M_\mathrm{A}2} \cdot \boldsymbol{B} & \dots & A_{M_\mathrm{A}N_\mathrm{A}} \cdot \boldsymbol{B} \end{pmatrix},$$

where $\dim(\boldsymbol{A} \otimes \boldsymbol{B}) = M_\mathrm{A}M_\mathrm{B} \times N_\mathrm{A}N_\mathrm{B}$ if $\dim \boldsymbol{A} = M_\mathrm{A} \times N_\mathrm{A}$ and $\dim \boldsymbol{B} = M_\mathrm{B} \times N_\mathrm{B}$. Applied to the computational basis states of single qubit systems, it becomes clear how the tensor product relates to the description of multiple qubit systems from above.

Example 2. In the case $N = 2$, we obtain

$$|0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 0 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle\,.$$

To summarize, the computational basis states of a two qubit systems have the representation

$$\begin{aligned}
|00\rangle &= |0\rangle \otimes |0\rangle = \begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}^T,\\
|01\rangle &= |0\rangle \otimes |1\rangle = \begin{pmatrix}0 & 1 & 0 & 0\end{pmatrix}^T,\\
|10\rangle &= |1\rangle \otimes |0\rangle = \begin{pmatrix}0 & 0 & 1 & 0\end{pmatrix}^T,\\
|11\rangle &= |1\rangle \otimes |1\rangle = \begin{pmatrix}0 & 0 & 0 & 1\end{pmatrix}^T.
\end{aligned}$$

The tensor product ensures that the computational basis states are still orthonormal. The tensor product can also be used to conveniently describe quantum gates for multiple qubit systems. Suppose we want to construct a gate for a two qubit system which leaves the first qubit unchanged and performs a bit flip on the second qubit. This gate has the matrix representation

$$\boldsymbol{I} \otimes \boldsymbol{X} = \begin{pmatrix} 1 \cdot \boldsymbol{X} & 0 \cdot \boldsymbol{X} \\ 0 \cdot \boldsymbol{X} & 1 \cdot \boldsymbol{X} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

For example, the state $|01\rangle$ is transformed into the state $|00\rangle$, as can be easily seen from

$$(\boldsymbol{I} \otimes \boldsymbol{X}) \cdot |01\rangle = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle .$$

In the context of QEC, we are interested in tensor products of Pauli matrices. When used to describe QLDPC codes, these tensor products often contain many identity entries, resulting in cumbersome terms when using the "$\otimes$" notation. There exists an index-based notation, which facilitates a concise representation of sparse tensor products: All $\otimes$ symbols are omitted, as it is clear from the context that the Pauli matrices are connected via tensor products. Additionally, all identity-entries are omitted. The indices of non-identity entries are written in the subscript.

Example 3. The tensor product $\boldsymbol{X} \otimes \boldsymbol{I} \otimes \boldsymbol{I} \otimes \boldsymbol{Y} \otimes \boldsymbol{Z}$ has the index-based notation $\boldsymbol{X}_0 \boldsymbol{Y}_3 \boldsymbol{Z}_4$. Note that we start to count from $n = 0$.

2.1.4. Quantum Error Correction

In order to make use of quantum mechanical principles, the qubits in a quantum computer have to be implemented by single particles or a small number of particles [Got97]. In contrast to classical computers, where a large number of electrons are used to implement a bit, qubits are highly susceptible to noise. Nevertheless, QEC uses the same fundamental idea as classical error correction to overcome noise: The quantum information is *encoded*, i.e., redundant information is added. This ensures that all information of the original message can be recovered (i.e., *decoded*) from the original message, given that the effect of noise keeps within limits. However, classical error-correction techniques rely on assumptions which are not met in quantum systems.

For example, the *no-cloning theorem* prohibits duplicating a quantum state, preventing the application of codes such as the repetition code, which is commonly used in classical error-correction. In addition, since the states of qubits can take on a continuum of values, a continuum of errors may occur on a single qubit. Moreover, the measurement of qubits would destroy information, making decoding based on direct observation of the quantum codeword impossible. [NC10, Got97]
Despite these challenges, a theory of QEC has been developed over the past decades. While the no-cloning theorem prohibits to duplicate superpositions of quantum states (as this would require to measure the state and hence, destroy it's superposition), it is possible to duplicate states within a single basis. As an example, the encoding $\alpha |0\rangle + \beta |1\rangle \rightarrow \alpha |000\rangle + \beta |111\rangle$ doesn't violate the no-cloning theorem, as the superposition is spread out but not repeated [Got97]. One key insight is that, although direct measurement of a quantum codeword is not possible, information about the error can still be obtained [Sho95]: For a measurement not to disturb the actual state of the code, this measurement has to yield the same result for every valid codeword. Such a measurement is referred to as a *syndrome measurement.* Furthermore, it can be shown that it suffices to correct the Pauli errors in order to correct the most general possible noise. This rather surprising result is known as the *digitization of noise* and stems from the fact that the Pauli matrices $\boldsymbol{I}$, $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ form a complete set: They can be used to describe any transformation on qubits and thus can express general noise [NC10].

Figure 2.1 shows a schematic of a QEC system: An encoding circuit maps logical qubits $|\psi\rangle_L$ into codewords $|\psi\rangle$. These codewords are exposed to a quantum channel, which applies an error transformation $\boldsymbol{E}$ to $|\psi\rangle$. A syndrome measurement obtains information on the error, which is then fed to the syndrome decoder. The decoder determines an operation $\boldsymbol{F}$, such that $\boldsymbol{FE} |\psi\rangle = |\psi\rangle$ with high probability.

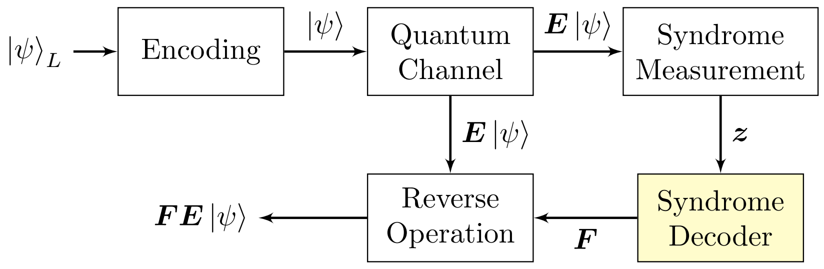


Figure 2.1: Overview of a quantum error correction system.

In this framework, QEC is the task of choosing a quantum code such that there exists a syndrome decoder (theoretically and practically) which finds a suitable operation $\boldsymbol{F}$ with maximum probability, given a quantum channel. Among the multitude of quantum codes, QLDPC codes have been shown to be promising candidates to perform QEC due to their good performance and the availability of efficient decoders. They are based on the stabilizer formalism, which is briefly described in the next section.

2.2. Stabilizer Formalism

Recall from the section above that a quantum measurement of an operator projects the measured state $|\psi\rangle$ onto an eigenstate of the operator and yields as a result the eigenvalue of that eigenstate. While such a measurement usually destroys the superposition of $|\psi\rangle$, a quantum state that is an eigenstate of the measured operator will not be affected by the corresponding measurement. QSCs use this fact to *define* a quantum code in terms of special operators, obtained as tensor products of Pauli matrices. The codespace of such a code consists of all quantum states $|\psi\rangle$ that are eigenstates of these operators. This way, a measurement will not reveal any information on the codeword itself. Hence, the superposition of valid codewords stays intact when measured w.r.t. these operators.
We refer to these operators as *stabilizers*, since they stabilize the codespace. In other words, all valid quantum codewords are $(+1)$-eigenstates of the stabilizers. We can think of stabilizers as the quantum analogue of parity checks used in classical linear codes. Stabilizer codes are closely related to classical codes over GF(4), as described further below. This enables the application of classical coding theory in order to perform QEC. [Got97]

2.2.1. An Introductory Example

Let us first introduce the stabilizer formalism by means of an example and proceed to a more formal description from there. Recall from Sec. 2.1.1 that the $\boldsymbol{Z}$ quantum gate has eigenstates $|0\rangle$ and $|1\rangle$ with eigenvalues $(+1)$ and (-1), respectively. The use of eigenstates extends to multiple qubit systems, where quantum gates are represented as tensor products of single qubit gates: Consider the stabilizer $\boldsymbol{Z}_0 = \boldsymbol{Z} \otimes \boldsymbol{I} \otimes \boldsymbol{I}$, which flips the sign of the computational basis state if the qubit at the first digit is $|1\rangle$. Analogously, the $\boldsymbol{Z}_1 = \boldsymbol{I} \otimes \boldsymbol{Z} \otimes \boldsymbol{I}$ gate flips the sign of the computational basis state, if the qubit at the second digit is $|1\rangle$. Thus, the gate $\boldsymbol{Z}_0\boldsymbol{Z}_1 = \boldsymbol{Z} \otimes \boldsymbol{Z} \otimes \boldsymbol{I}$ flips the phase twice when the qubit at the first and second digit are $|1\rangle$, resulting in no overall phase flip. Due to linearity, the effect on any superposed quantum state is characterized by the effect on the computational basis states, given in Tab. 2.1.

Table 2.1: Effect of the $\boldsymbol{Z}_0\boldsymbol{Z}_1$ gate on the eight computational basis states of a three qubit system.

$\lvert\psi\rangle$	$\boldsymbol{Z}_0\boldsymbol{Z}_1 \lvert\psi\rangle$
$\lvert 000\rangle$	$+\lvert 000\rangle$
$\lvert 001\rangle$	$+\lvert 001\rangle$
$\lvert 010\rangle$	$-\lvert 010\rangle$
$\lvert 011\rangle$	$-\lvert 011\rangle$
$\lvert 100\rangle$	$-\lvert 100\rangle$
$\lvert 101\rangle$	$-\lvert 101\rangle$
$\lvert 110\rangle$	$+\lvert 110\rangle$
$\lvert 111\rangle$	$+\lvert 111\rangle$

While the space of all possible three qubit states $|\psi\rangle$ is of dimension $2^3 = 8$, the $(+1)$ and (-1) eigenstates are of dimension $2^{3-1} = 4$. The operator $\boldsymbol{Z}_0\boldsymbol{Z}_1$ can thus be seen as a quaternary parity check, imposing a constraint which is satisfied only for valid codewords.

The $(+1)$ eigenspace corresponding to $\boldsymbol{Z}_0\boldsymbol{Z}_1$ can be further constrained by introducing a second operator, e.g., $\boldsymbol{Z}_1\boldsymbol{Z}_2$. The intersection of both eigenspaces results in a 2^{3-1}-dimensional subspace with the basis $\{|000\rangle, |111\rangle\}$. These two basis states can now be used to encode a logical qubit, i.e., $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$.

We can now associate a syndrome measurement to the stabilizers $\boldsymbol{Z}_0\boldsymbol{Z}_1$ and $\boldsymbol{Z}_1\boldsymbol{Z}_2$. Such a measurement only reveals information about the error that occured on quantum codeword, without revealing information on the codeword itself: To understand this, suppose that a quantum channel flips the second qubit (i.e., indexed by $n = 1$ since we start at $n = 0$). The operator corresponding to this error thus has the form $\boldsymbol{E} = \boldsymbol{I} \otimes \boldsymbol{X} \otimes \boldsymbol{I} = \boldsymbol{X}_1$. The resulting errorneous state is $\boldsymbol{E}|\psi\rangle = \boldsymbol{X}_1(\alpha|000\rangle + \beta|111\rangle) = \alpha|010\rangle + \beta|101\rangle$. Since a measurement yields the eigenvalue associated to the eigenstate onto which the measured state is projected, the measurement of the operator $\boldsymbol{Z}_0\boldsymbol{Z}_1$ results in (-1), as $\boldsymbol{Z}_0\boldsymbol{Z}_1(\alpha|010\rangle + \beta|101\rangle) = -(\alpha|010\rangle + \beta|101\rangle)$ (which follows from Tab. 2.1 and linearity). Analogously, the measurement of the operator $\boldsymbol{Z}_1\boldsymbol{Z}_2$ results in (-1). We can thus perform a measurement of the errorneous quantum state $\boldsymbol{E}|\psi\rangle$ which reveals information on the error $\boldsymbol{E}$.

2.2.2. Quantum Stablizer Codes

The stabilizers for a quantum code of blocklength N are elements of the *Pauli* group $\mathcal{P}_N$, which consists of N-fold tensor products of the Pauli matrices, together with multiplicative factors $\{\pm 1, \pm i\}$:

$$\mathcal{P}_N = \{\pm 1, \pm i\} \times \{\boldsymbol{I}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}\}^{\otimes N}$$

Just like in the single qubit case, two elements from $\mathcal{P}_N$ either commute or anti-commute. A QSC is defined by a subgroup $\mathcal{S} \subset \mathcal{P}_N$. We say that $\mathcal{S}$ is *generated* from a set of stabilizers $\boldsymbol{S}_0, ..., \boldsymbol{S}_{M-1}$ if every element in $\mathcal{S}$ can be written as the product of these stabilizers. If $\mathcal{S}$ is generated by a set of operators $\boldsymbol{S}_0, ..., \boldsymbol{S}_{M-1}$, we also write $\mathcal{S} = \langle \boldsymbol{S}_0, ..., \boldsymbol{S}_{M-1}\rangle$. It suffices to consider M operators to describe the stabilizer group $\mathcal{S}$. With this, we can state the definition of a QSC. [NC10]

Defintion 1 (Quantum stabilizer code)**.** *An $[[N, K]]$ stabilizer code is a 2^K-dimensional subspace of the N-qubit space $\mathbb{C}^{2^n}$, defined as the common $(+1)$-eigenspace of $M = N - K$ stabilizers $\boldsymbol{S}_0, ..., \boldsymbol{S}_{M-1}$:*

$$\mathcal{C} = \{\, |\psi\rangle \in \mathbb{C}^{2^N} \ : \ \boldsymbol{S}_m|\psi\rangle = |\psi\rangle, \forall m \in 0, ..., M-1\}.$$

We say that the codespace is stabilized by the stabilizers $\boldsymbol{S}_m$.

Remark 1. *Not any set of elements $\boldsymbol{S}_0, ..., \boldsymbol{S}_{M-1} \in \mathcal{P}_N$ can be used to define a valid quantum code. $\mathcal{S}$ has to form a commutative subgroup of $\mathcal{P}_N$ and $\mathcal{S}$ is not allowed to contain $-\boldsymbol{I}^{\otimes N}$ in order to stabilize a non-trivial codespace [CRSS97]. To understand these condition, assume that two elements $\boldsymbol{S}_m, \boldsymbol{S}_{m'} \in \mathcal{S}$ anti-commute (note that two elements in $\mathcal{P}_N$ either commute or anti-commute). Since all elements in $\mathcal{S}$ stabilize the codespace, we have $\boldsymbol{S}_m\boldsymbol{S}_{m'}|\psi\rangle = \boldsymbol{S}_m|\psi\rangle = |\psi\rangle$. However, if $\boldsymbol{S}_m$ and $\boldsymbol{S}_{m'}$ anti-commute, we also have $\boldsymbol{S}_m\boldsymbol{S}_{m'}|\psi\rangle = -\boldsymbol{S}_{m'}\boldsymbol{S}_m|\psi\rangle = -|\psi\rangle$. Hence, our assumption leads to $|\psi\rangle = -|\psi\rangle$, which is only satisfied by the zero vector (i.e., a trivial codespace). The same contradiction is created if $\mathcal{S}$ contains $-\boldsymbol{I}^{\otimes N}$. [NC10]*

A convenient way to describe a QSC code is by arranging the M stabilizers into a matrix. Since each stabilizer $\boldsymbol{S}_m$ is the N-fold tensor product of the four Pauli matrices and we require M such stabilizers, the resulting *stabilizer matrix* is of the form $\boldsymbol{S} \in \{I, X, Y, Z\}^{M \times N}$. The stabilizer matrix can be seen as the quantum analogue of the classical PCM.

Remark 2. *While we use bold letters to denote non-scalar quantities such as vectors or matrices in this thesis, from now on we use non-bold letters to denote the Pauli matrices, when they occur in the context of stabilizer matrices or quantum errors. This emphasizes that Pauli matrices can be seen as entries of a matrix, analogous to the classical PCM, whose entries are scalar. With this convention, we hope to clarify whether we refer to a collection of entries (such as a stabilizer matrix or a row of a stabilizer matrix) or to single entries (such as a Pauli matrix).*

When the QSC is defined in terms of the matrix $\boldsymbol{S}$, it's stabilizers are obtained as the rows of $\boldsymbol{S}$. Thus, to define a valid quantum code, each pair of rows $\boldsymbol{S}_m, \boldsymbol{S}_{m'}$ has to commute. Recall that each row is associated to an element in $\mathcal{P}_N$, which in turn is obtained as the tensor product of Pauli matrices. To assess whether two tensor products of Pauli matrices (i.e., two rows of $\boldsymbol{S}$) commute or anti-commute, we have to consider the commutation relation element-wise: Each position with anti-commuting Pauli matrices introduces a negative sign in the result. If the elements anti-commute in an even number of positions, the negative signs cancel, causing the two rows to commute. On the other hand, an odd number of anti-commuting Pauli matrices causes the rows to anti-commute.

2.2.3. Quantum Errors and Syndrome

As already described in the introductory example, we can describe quantum errors as operators $\boldsymbol{E}$, acting on quantum codewords $|\psi\rangle$. The *error discretization theorem* states that a quantum error correcting code is able to correct a continuum of errors by only correcting the discrete set of Pauli errors. This is due to the fact that the syndrome measurement forces a general noisy state to choose among the Pauli errors. This chosen error can then be reversed, using the information obtained from the measurement [NC10]. We can thus think of quantum errors $\boldsymbol{E}$ as "vectors of Pauli matrices", i.e., $\boldsymbol{E} \in \{I, X, Y, Z\}^N$. Just like in classical coding, the stabilizer matrix ("quantum PCM") $\boldsymbol{S}$ constitutes a mapping between the error space and the syndrome space. Each row of $\boldsymbol{S}$ performs a projective measurement of the error, resulting in the M-dimensional binary syndrome $\boldsymbol{z}(\boldsymbol{E}) = \langle \boldsymbol{S}, \boldsymbol{E} \rangle = (z_0, ..., z_{M-1}) \in \{0, 1\}^M$. The m-th entry of $\boldsymbol{z}$ indicates whether $\boldsymbol{E}$ commutes or anti-commutes with the m-th row of $\boldsymbol{S}$, i.e.,

$$z_m = \begin{cases} 0 & \text{if} \quad \boldsymbol{E}\boldsymbol{S}_m = \boldsymbol{S}_m\boldsymbol{E} \\ 1 & \text{if} \quad \boldsymbol{E}\boldsymbol{S}_m = -\boldsymbol{S}_m\boldsymbol{E} \end{cases} .$$

It is the task of the syndrome decoder to find a reverse operation $\boldsymbol{F}$, used to recover the original quantum codeword from the errorneous version, i.e., $\boldsymbol{F}\boldsymbol{E}\,|\psi\rangle \overset{!}{=} |\psi\rangle$.

Remark 3. *While the syndrome measurement yields the eigenvalues of the eigenstates onto which the measured state is projected (i.e., vectors of the form $\{+1, -1\}^N$), we choose* *to represent the syndrome as an element in* $\mathbb{F}_2^N = \{0,1\}^N$. *These representations are connected by an element-wise mapping, where* $0 \leftrightarrow (+1)$ *and* $1 \leftrightarrow (-1)$. *While* $\{0, 1\}$ *forms a commutative group under addition,* $\{1, -1\}$ *forms a commutative group under multiplication. We elaborate on this isomorphism in Sec. 2.4, where we describe the connection of QSCs and additive codes over* GF(4).

2.3. Quantum Degeneracy

The normalizer $N(\mathcal{S})$ is defined as the set of errors that induce the trivial syndrome $\boldsymbol{z} = \mathbf{0}$, i.e., it consists of all errors that commute with every stabilizer. Like in classical coding, the error space $\{I, X, Y, Z\}^N$ can be partitioned into cosets w.r.t. $N(\mathcal{S})$. If two errors $\boldsymbol{E}_i$ and $\boldsymbol{E}_j$ are connected by an element in $N(\mathcal{S})$ (i.e., $\boldsymbol{E}_i\boldsymbol{E}_j \in N(\mathcal{S})$), they result in an identical syndrome. A syndrome decoder cannot distinguish between two such errors.
However, this indistinguishability might not be a problem due to an interesting property of quantum codes with no classical analogue: Since the codespace is stabilized by every element in $\mathcal{S}$, it suffices to find $\boldsymbol{F}$ such that $\boldsymbol{FE} \in \mathcal{S}$. This means that, in contrast to classical error correction, an error can be corrected by more than one reverse operation $\boldsymbol{F}$. Framed differently, an operation $\boldsymbol{F}$ may correct more than one error. This intrinsic property of quantum codes is referred to as *quantum degeneracy.* [Got97]
Note that $\mathcal{S}$ is a normal subgroup of $N(\mathcal{S})$. This means that we can further partition $N(\mathcal{S})$ into cosets w.r.t. to the stabilizer. The group structure of the *quotient group* $N(\mathcal{S})/\mathcal{S}$ acting on these cosets can be seen as logical operations, which move codewords around within the code $\mathcal{C}$. In this context, errors in the stabilizer correspond to the special case of moving codewords within $\mathcal{C}$ in a trivial way, i.e., fixing the codespace. [Got97]

Example 4. Consider the introductory three-qubit code from above with $\mathcal{S} = \langle Z_0Z_1, Z_1Z_2\rangle$. The errors $\boldsymbol{E}_i = X_2$ and $\boldsymbol{E}_j = Z_0Z_1X_2$ both have the same effect on a quantum codeword since $\boldsymbol{E}_i(\alpha\,|000\rangle + \beta\,|111\rangle) = \boldsymbol{E}_j(\alpha\,|000\rangle + \beta\,|111\rangle) = \alpha\,|001\rangle + \beta\,|110\rangle$. Thus, the error $\boldsymbol{E}_i$ may be used to correct the error $\boldsymbol{E}_j$, and vice versa.

Let us now discuss how the error correction capabilities of a QSCs are related to $N(\mathcal{S})$ and $\mathcal{S}$. In the classical setting, a set of errors is correctable iff all elements can be uniquely identified by their syndrome. In other words, the difference of any pair of errors from a correctable is not allowed to lie in $\mathcal{C} \setminus \{\mathbf{0}\}$. Note that the difference is allowed to lie in $\{\mathbf{0}\}$, as the all-zero codeword acts trivially on the codespace. For a QSC, errors in a set are uniquely identified by their syndromes if for any pair of errors $\boldsymbol{E}_i$ and $\boldsymbol{E}_j$, we have $\boldsymbol{E}_i\boldsymbol{E}_j \notin N(\mathcal{S})$. However, due to degeneracy $\boldsymbol{E}_i\boldsymbol{E}_j$ is allowed to lie in $\mathcal{S}$, as elements in $\mathcal{S}$ act trivially on the codespace [CRSS97]. Thus, to assess the error correction capabilities of a code, we have to consider the set $N(\mathcal{S})\setminus\mathcal{S}$, as summarized in the following Lemma.

Lemma 1. *Let* $\mathcal{S}$ *be a commutative subgroup of the* N*-fold Pauli group* $\mathcal{P}_N$*, generated by* $M = N - K$ *stabilizers* $\boldsymbol{S}_m \in \mathcal{P}_N$*, and suppose that* $-\boldsymbol{I} \notin \mathcal{S}$*. Let* D *be the minimum weight of* $N(\mathcal{S}) \setminus \mathcal{S}$*. Then the space stabilized by all elements of* $\mathcal{S}$ *is an* $[[N, K, D]]$ *quantum code. [CRSS97]*

Remark 4. *The minimum weight D is defined as $D = \min_{\boldsymbol{E} \in N(\mathcal{S}) \setminus \mathcal{S}} w(\boldsymbol{E})$, where the weight of an error $w(\boldsymbol{E})$ is defined as the number of it's non-identity entries $\{X, Y, Z\}$. Just like in classical coding, a QSC with minimum weight D can correct all errors of weight up to $\lfloor (D-1)/2 \rfloor$ [CRSS97]. If the stabilizer of this code contains elements of weight less than D, the code is referred to as degenerate [Got97].*

We conclude this section by refining the setup used to analyse quantum codes in the scope of this thesis, as shown in Fig. 2.2: Since QEC is based on the syndrome, it suffices to sample errors $\boldsymbol{E}$ according to a quantum channel model, as the syndrome doesn't depend on the quantum codeword. The syndrome $\boldsymbol{z}$ is measured w.r.t. the underlying quantum code, represented by the stabilizer matrix $\boldsymbol{S}$. The syndrome decoder is given the measured syndrome as well as $\boldsymbol{S}$ in order to deduce a reverse operation $\boldsymbol{F}$. Due to quantum degeneracy, it suffices to find an $\boldsymbol{F}$ such that $\boldsymbol{F}\boldsymbol{E} \in \mathcal{S}$.

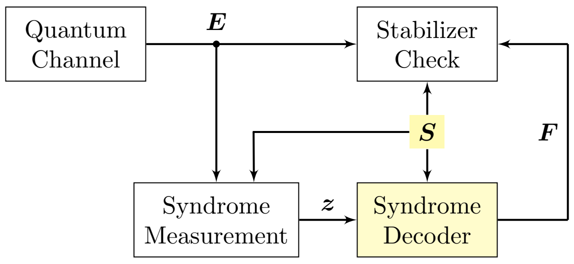


Figure 2.2: Overview of the refined setup, used to analyze quantum codes.

Throughout this thesis, we assume that quantum errors are generated by a *depolarizing channel*. This channel can be seen as the quantum analogue to a binary symmetric channel (BSC): The errors on each qubit are i.i.d. and the probability for a non-identity error on each qubit is $\epsilon/3$, i.e.,

$$P_{\text{ch}}(E_n = W) = \begin{cases} 1-\epsilon & \text{if} \quad W = I \\ \epsilon/3 & \text{if} \quad W \in \{X, Y, Z\}. \end{cases}$$

As a performance measure for decoders, we use the logical error rate (LER): For a given error rate ϵ, we sample and decode a large number of errors according to the depolarizing channel model. After a maximum number of iterations, we count the number of samples that result in a logical error. For a sampled error $\boldsymbol{E}$, a logical error occurs if the decoder fails to output a reverse operation $\boldsymbol{F}$, such that $\boldsymbol{F}\boldsymbol{E} \in \mathcal{S}$. The LER is approximated as the total number of errors resulting in a logical error, divided by the total number of sampled errors.

2.4. Connection to Classical Codes

QSCs can be treated as additive codes over GF(4). The commutative property of the stabilizer formalism requires these codes to be self-orthogonal with respect to the trace inner product [CRSS97]. Furthermore, we can establish a connection to codes over $\mathbb{F}_2$, enabling us to access classical binary codes in the construction of quantum codes.

2.4.1. Codes Over GF(4)

The connection between QSCs and additive codes over GF(4) is rooted in the following isomorphism, mapping the Pauli matrices to GF(4):

$$\boldsymbol{I} \to 0, \qquad \boldsymbol{X} \to 1, \qquad \boldsymbol{Z} \to \omega, \qquad \boldsymbol{Y} \to \omega^2 = \bar{\omega}.$$

"The multiplication of Pauli operators is transformed into the addition of the corresponding elements in GF(4)", as stated by Babar et al. [BBA+15]. Let us first familiarize with GF(4) and then derive an expression for the commutation relation in this representation. Addition and multiplication rules in GF(4) are depicted in Tab. 2.2.

Table 2.2: Addition table and multiplication table for GF(4).

$+$	0	1	ω	ω^2
0	0	1	ω	ω^2
1	1	0	ω^2	ω
ω	ω	ω^2	0	1
ω^2	ω^2	ω	1	0

$\cdot$	0	1	ω	ω^2
0	0	0	0	0
1	0	1	ω	ω^2
ω	0	ω	ω^2	1
ω^2	0	ω^2	1	ω

The multiplication of two elements $\boldsymbol{S}_m$ and $\boldsymbol{S}_{m'}$ in the Pauli domain maps to the addition of the corresponding elements $\hat{\boldsymbol{S}}_m$ and $\hat{\boldsymbol{S}}_{m'}$ in GF(4). The commutation relation of two elements translates to the *trace inner product*:

$$\mathrm{Tr}\,\langle \hat{\boldsymbol{S}}_m, \hat{\boldsymbol{S}}_{m'} \rangle = \mathrm{Tr}\bigg(\sum_n \hat{S}_{mn} \cdot \bar{\hat{S}}_{m'n} \bigg).$$

In GF(4), the trace is defined as $\mathrm{Tr}(0) = \mathrm{Tr}(1) = 0$, $\mathrm{Tr}(\omega) = \mathrm{Tr}(\omega^2) = 1$. The conjugate $\bar{x}$ swaps the elements ω and ω^2 while preserving 0 and 1.

Example 5. Consider the stabilizers $\boldsymbol{S}_m = \boldsymbol{X}_0\boldsymbol{Z}_1\boldsymbol{Y}_3$ and $\boldsymbol{S}_{m'} = \boldsymbol{X}_0\boldsymbol{X}_1\boldsymbol{Y}_2\boldsymbol{Z}_3$. These stabilizers commute, since they have anti-commuting Pauli matrices at indices one and three (i.e., an even number of indices). Their corresponding elements in GF(4) are $\hat{\boldsymbol{S}}_m = (1 \quad \omega \quad 0 \quad \omega^2)$ and $\hat{\boldsymbol{S}}_{m'} = (1 \quad 1 \quad \omega^2 \quad \omega)$. Their trace inner product then reads

$$\begin{aligned}\mathrm{Tr}\,\langle \hat{\boldsymbol{S}}_m, \hat{\boldsymbol{S}}_{m'} \rangle &= \mathrm{Tr}\big(1 \cdot \bar{1} + \omega \cdot \bar{1} + 0 \cdot \bar{\omega^2} + \omega^2 \cdot \bar{\omega}\big) \\ &= \mathrm{Tr}\big(1 + \omega + 0 + \omega\big) = 0\,,\end{aligned}$$

confirming that the two elements commute.

2.4.2. Binary Representation of Quantum Codes

The Galois Field GF(4) can also be represented as the set of all binary tuples, $\mathbb{F}_2^2$. This allows us to represent quantum codes using a binary PCM notation. The underlying isomorphism is

$$\boldsymbol{I} \to (0,0), \qquad \boldsymbol{X} \to (1,0), \qquad \boldsymbol{Z} \to (0,1), \qquad \boldsymbol{Y} \to (1,1).$$

The commutation relation of two tuples $a = (a_1, a_2)$ and $b = (b_1, b_2)$ is defined as the twisted product

$$(a_1, a_2) * (b_1, b_2) = a_1 b_2 + a_2 b_1 \mod 2.$$

Table 2.3 demonstrates the equivalence of the the commutation relation in the Pauli domain, the trace inner product for codes over GF(4) and the twisted product for codes over $\mathbb{F}_2$.

Table 2.3: The commutation relation in the Pauli domain (left) may be implemented in GF(4) using the trace inner product (middle) and in $\mathbb{F}_2$ using the twisted product (right).

$[a,b]$	I	X	Z	Y
I	$+$	$+$	$+$	$+$
X	$+$	$+$	$-$	$-$
Z	$+$	$-$	$+$	$-$
Y	$+$	$-$	$-$	$+$

$\mathrm{Tr}(a \cdot \bar{b})$	0	1	ω	ω^2
0	0	0	0	0
1	0	0	1	1
ω	0	1	0	1
ω^2	0	1	1	0

$(a * b)$	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(0,0)$	0	0	0	0
$(0,1)$	0	0	1	1
$(1,0)$	0	1	0	1
$(1,1)$	0	1	1	0

A stabilizer $\boldsymbol{S}_m$ of length N can be represented by a binary vector $\boldsymbol{S}_m^{\mathrm{b}}$ of length $2N$ by splitting it into two components $\boldsymbol{S}_{m,X}^{\mathrm{b}}$ and $\boldsymbol{S}_{m,Z}^{\mathrm{b}}$, both of length N, and concatenating them horizontally, i.e., $\boldsymbol{S}_m^{\mathrm{b}} = (\boldsymbol{S}_{m,X}^{\mathrm{b}} \,|\, \boldsymbol{S}_{m,Z}^{\mathrm{b}})$: The first component contains a one where $\boldsymbol{S}_m$ has an X entry, the second component contains a one where $\boldsymbol{S}_m$ has a Z entry. A one at the same position in both vectors indicates that $\boldsymbol{S}_m$ has a Y entry at that position.

The twisted product extends to the *symplectic product* when we consider vectors of binary tuples of the form $\boldsymbol{S}_m^{\mathrm{b}}$. It can be thought of as the analogue of the commutation relation in the binary domain: The symplectic product is zero if the number of tuples with a twisted product of one is even. For two vectors $\boldsymbol{S}_m^{\mathrm{b}}$ and $\boldsymbol{S}_{m'}^{\mathrm{b}}$ this is compactly represented by the dot product

$$\boldsymbol{S}_m^{\mathrm{b}} * \boldsymbol{S}_{m'}^{\mathrm{b}} = \boldsymbol{S}_{m,X}^{\mathrm{b}} \cdot \boldsymbol{S}_{m',Z}^{\mathrm{b}} + \boldsymbol{S}_{m,Z}^{\mathrm{b}} \cdot \boldsymbol{S}_{m',X}^{\mathrm{b}} \mod 2.$$

Example 6. For the example from above, we have $\boldsymbol{S}_m^{\mathrm{b}} = (1001\,|\,0101)$ and $\boldsymbol{S}_{m'}^{\mathrm{b}} = (1110\,|\,0011)$. The symplectic product then yields

$$\boldsymbol{S}_m^{\mathrm{b}} * \boldsymbol{S}_{m'}^{\mathrm{b}} = (1001\,|\,0101) * (1110\,|\,0011) = (1001) \cdot (0011) + (0101) \cdot (1110) = 0 \mod 2,$$

just like in the Pauli domain and in the quaternary domain.

We can apply this construction to the stabilizer matrix $\boldsymbol{S}$ by writing each row as a binary vector. This results in a binary matrix $\boldsymbol{S}^{\mathrm{b}}$ of dimension $M \times 2N$, consisting of two binary matrices $\boldsymbol{H}_X$ and $\boldsymbol{H}_Z$, i.e., $\boldsymbol{S}^{\mathrm{b}} = (\boldsymbol{H}_X \,|\, \boldsymbol{H}_Z)$. These matrices can be interpreted as PCMs of classical codes over $\mathbb{F}_2$.
The commutation relation imposes a constraint on these matrices: Since all elements in $\mathcal{S}$ have to commute with each other, the symplectic product of each pair of rows of $\boldsymbol{S}^{\mathrm{b}}$ has to be zero. This can be summarized in the *symplectic criterion*

$$\boldsymbol{H}_X \boldsymbol{H}_Z^T + \boldsymbol{H}_Z \boldsymbol{H}_X^T = \boldsymbol{0}. \tag{2.1}$$

Any two classical codes satisfying this criterion may be used to construct a valid quantum code.

2.5. Calderbank-Shor-Steane Codes

CSS codes [CRSS97] constitute an important subclass of QSCs. They provide a means to construct quantum codes from two classical codes, $\mathcal{C}_1$ and $\mathcal{C}_2$, where $\mathcal{C}_2^\perp \subseteq \mathcal{C}_1$. The binary stabilizer matrix $\boldsymbol{S}^{\mathrm{b}}$ of a CSS can be written in separable form, i.e.,

$$\boldsymbol{S}^{\mathrm{b}} = \left(\begin{array}{c|c} \boldsymbol{H}'_X & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{H}'_Z \end{array} \right)$$

This simplifies the symplectic criterion in Eq. 2.1 to $\boldsymbol{H}'_X \boldsymbol{H}'^T_Z = \boldsymbol{0}$. Thus, designing a CSS code comes down to finding dual codes with good error correcting properties. Error correction can then be performed separately for X and Z errors, as briefly described in the section below.

Remark 5. *The stabilizer matrix $\boldsymbol{S}$ is related to $\boldsymbol{S}^{\mathrm{b}}$ via*

$$\boldsymbol{S} = \begin{pmatrix} X \cdot \boldsymbol{H}'_X \\ Z \cdot \boldsymbol{H}'_Z \end{pmatrix} \tag{2.2}$$

2.6. Belief Propagation for Quantum LDPC Codes

2.6.1. Syndrome Decoding

Recall that for QSCs, decoding is based on the syndrome $\boldsymbol{z}$, which is obtained from a stabilizer measurement of the error $\boldsymbol{E}$,

$$z_m = \langle \boldsymbol{S}_m, \boldsymbol{E} \rangle \in \{0, 1\}.$$

It is the task of the syndrome decoder to find the a reverse operation $\boldsymbol{F}$, such that $\boldsymbol{F}\boldsymbol{E} \in \mathcal{S}$. One decoding technique is to identify the most likely error that produced the measured syndrome $\boldsymbol{z}$, i.e.,

$$\boldsymbol{F} = \underset{\boldsymbol{E} \in \{I,X,Y,Z\}^N,\, \langle \boldsymbol{S}, \boldsymbol{E} \rangle = \boldsymbol{z}}{\operatorname{argmax}} P(\boldsymbol{E} | \boldsymbol{z}).$$

While this decoding problem defines an NP-complete problem [PC08], an approximate solution can be found as the element-wise optimum rather than the global optimum. BP constitutes an efficient algorithm to find this local optimum. A common approach to solve this task is to view the depolarizing channel as two independent BSCs, one for bit flips and the other one for phase flips [BBA$^+$15]. BP can then be run over the binary Tanner graph of the binary check matrix $\boldsymbol{S}^{\mathrm{b}} = (\boldsymbol{H}_X \,|\, \boldsymbol{H}_Z)$. However, this approach ignores correlations between X and Z errors, resulting in degraded performance [BBA$^+$15].
In contrast, a quaternary BP decoder takes into account these correlations by operating on the Tanner graph of the stabilizer matrix $\boldsymbol{S}$. In the scope of this thesis, we only consider the quaternary BP decoder, which we will denote by BP$_4$. We give a brief overview of the algorithm in the following sections. Our notation is largely based on the notation used in [KL20] and [LK21], which we also recommend for an in-depth description of BP$_4$ decoding.

2.6.2. Tanner Graph

During BP decoding, probabilistic messages are exchanged along the edges of the *Tanner* graph $\mathcal{T}(\boldsymbol{S})$. The Tanner graph is obtained by interpreting $\boldsymbol{S}$ as the adjacency matrix of a bipartite graph, consisting of two sets of nodes, N variable nodes (VNs) and M check nodes (CNs). A VN v_n is connected to a CN c_m if $S_{mn} \in \{X, Y, Z\}$. Hence, $\mathcal{T}(\boldsymbol{S})$ can contain edges of three types, corresponding to the entries X, Y and Z.

Example 7. The stabilizer matrix $\boldsymbol{S} = \left(\begin{smallmatrix} X & Y & I \\ Z & Z & Y \end{smallmatrix}\right)$ corresponds to the Tanner graph depicted in Fig. 2.3.

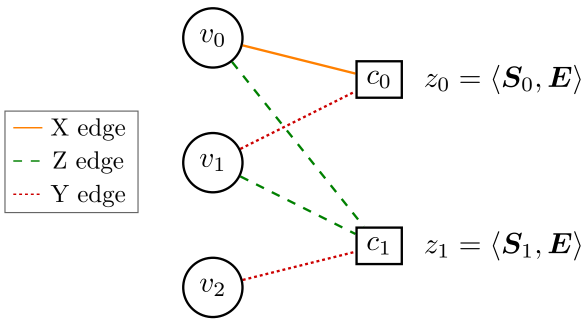

X edge
Z edge
Y edge
v_0
v_1
v_2
c_0
c_1
$z_0 = \langle \boldsymbol{S}_0, \boldsymbol{E} \rangle$
$z_1 = \langle \boldsymbol{S}_1, \boldsymbol{E} \rangle$

Figure 2.3: Tanner graph $\mathcal{T}(\boldsymbol{S})$. A Tanner graph associated to a QSC potentially has three edge types.

During a node update, outgoing messages from a node to it's neighbors are computed, based on incoming messages from the neighbors to this node. Let us denote the set of VNs that participate in a check m by $\mathcal{N}(m) = \{n : S_{mn} \neq I\}$. Analogously, the set of checks connected to a qubit n is denoted by $\mathcal{M}(n) = \{m : S_{mn} \neq I\}$. An important concept of BP is to only use *extrinsic information* to update an outgoing message: This means that, when updating the outgoing message from VN n to CN m, we only consider the incoming messages from neighbors $m' \in \mathcal{M}(n) \setminus m$. Analogously, when updating the message from CN m to VN n, only incoming messages from neighbors $n' \in \mathcal{N}(m) \setminus n$ are considered.

2.6.3. Log-Domain Implementation

For non-binary BP over GF(q), it can be shown that the complexity for generating check-to-variable messages is $\mathcal{O}(q^2)$ [DM98]. This leads to a 16-fold increase in the CN complexity of conventional BP$_4$ decoding, compared to binary BP decoding, where scalar messages can be exchanged[1] [KL20].
Kuo and Lai suggest a refined quaternary BP algorithm, exchanging only scalar messages [KL20]. The authors show that it is sufficient to exchange information on whether an entry E_n commutes or anti-commutes with the type S_{mn} of the edge along which the corresponding messages is passed. Just like in binary BP, this information can be expressed as a scalar value. In the log-domain (or Λ-rule based) implementation of BP$_4$, we express probability distributions in terms of log-likelihood ratios (LLRs).

[1] The complexity for variable-to-check messages is $\mathcal{O}(q)$ and hence omitted in the complexity analysis.

Let $\boldsymbol{p}_n = (p_n^I, p_n^X, p_n^Y, p_n^Z)$ be the prior distribution for each qubit. The prior beliefs corresponding to $\boldsymbol{p}_n$ are obtained as

$$\Lambda_n^W = \ln \frac{p_n^I}{p_n^W}$$

for $W \in \{X, Y, Z\}$. To represent a four-dimensional prior distribution, it suffices to use three-dimensional vectors of LLRs

$$\boldsymbol{\Lambda}_n = (\Lambda_n^X,\, \Lambda_n^Y,\, \Lambda_n^Z) \in \mathbb{R}^3.$$

Assuming the depolarizing channel model with error rate ϵ, we have $p_n^I = 1 - \epsilon$ and $p_n^W = \epsilon/3$ for $W \in \{X, Y, Z\}$. Hence, the prior beliefs are initialized as

$$\Lambda_n^W = \ln \frac{3(1-\epsilon)}{\epsilon}.$$

While the initial beliefs $\{\boldsymbol{\Lambda}_n\}_{n=0}^{N-1}$ are being held constant, the running beliefs $\boldsymbol{\Gamma}_n = (\Gamma_n^X,\, \Gamma_n^Z,\, \Gamma_n^Y)$ are updated during decoding, based on messages passed from VNs to CNs and vice versa. The entries Γ_n^W seek to estimate

$$\ln \frac{P(E_n = I|\boldsymbol{z})}{P(E_n = W|\boldsymbol{z})} = \frac{\sum_{\boldsymbol{U} \in \{I,X,Y,Z\}^N :\, U_n = I} \mathbb{1}\{\langle \boldsymbol{S}, \boldsymbol{U} \rangle = \boldsymbol{z}\} P(\boldsymbol{E} = \boldsymbol{U})}{\sum_{\boldsymbol{U} \in \{I,X,Y,Z\}^N :\, U_n = W} \mathbb{1}\{\langle \boldsymbol{S}, \boldsymbol{U} \rangle = \boldsymbol{z}\} P(\boldsymbol{E} = \boldsymbol{U})}.$$

While this expression is very similar to the binary case, an efficient computation is aggravated by the fact that $\boldsymbol{U}$ is a vector in $\mathrm{GF}(4)^N$ rather than $\mathbb{F}_2^N$. Kuo and Lai propose to circumvent this obstacle by introducing an auxiliary binary random variable (RV), describing "the likelihood of commutation and anti-commutation" [LK21] of E_n and edge S_{mn}.
In the log-domain, this is expressed by the function $\lambda_\eta : \mathbb{R}^3 \to \mathbb{R}$ for $\eta \in \{X, Y, Z\}$, mapping LLR vectors $\boldsymbol{L} = (L^X, L^Y, L^Z)$ representing a four-dimensional distribution of an entry E to scalar a LLR, representing the likelihood that E commutes with η, where

$$\lambda_\eta(\boldsymbol{L}) = \ln \frac{P(\langle \eta,\, E \rangle = 0)}{P(\langle \eta,\, E \rangle = 1)} = \ln \frac{1 + \sum_{V \in \{X,Y,Z\} :\, \langle \eta,\, V \rangle = 0} e^{-L^V}}{\sum_{V \in \{X,Y,Z\} :\, \langle \eta,\, V \rangle = 1} e^{-L^V}}. \tag{2.3}$$

Sending the scalar $\lambda_{S_{mn}}(\boldsymbol{\Gamma}_{n\to m})$ along the edge S_{mn}, every CN obtains $|\mathcal{N}(m)|$ scalar values and computes it's outgoing messages as

$$\Delta_{m\to n} = (-1)^{z_m} \boxplus_{n' \in \mathcal{N}(m) \setminus n} \lambda_{S_{mn'}}(\boldsymbol{\Gamma}_{n'\to m}).$$

During the VN update step, check-to-variable messages arriving along the edge S_{mn} only add to beliefs Γ_n^W, if W and S_{mn} anti-commute (hence the term $\mathbb{1}\{\langle S_{m'n}, W \rangle = 1\}$):

$$\Gamma_{n\to m}^W = \Lambda_n^W + \sum_{m' \in \mathcal{M}(n) \setminus m} \mathbb{1}\{\langle S_{m'n}, W \rangle = 1\}\, \Delta_{m'\to n}$$

for $W \in \{X, Y, Z\}$. The same rule applies to the hard decision step, where

$$\Gamma_n^W = \Lambda_n^W + \sum_{m \in \mathcal{M}(n)} \mathbb{1}\{\langle S_{mn}, W \rangle = 1\}\, \Delta_{m\to n}$$

is computed for $W \in \{X, Y, Z\}$. Readers interested in more details on these update rules are referred to [LK21], where a derivations of these expressions are provided.

Remark 6. *The LLR of a binary RV* $\mathtt{v} = \mathtt{u}_1 \oplus \ldots \oplus \mathtt{u}_n$ *with LLRs* $L(\mathtt{u}_1), \ldots, L(\mathtt{u}_n)$ *has the form [HOP96]*

$$L(\mathtt{v}) = L(\mathtt{u}_1 \oplus \ldots \oplus \mathtt{u}_n) = \boxplus_{i=1}^{n} L(\mathtt{u}_i) = 2\tanh^{-1}\big(\prod_{i=1}^{n} \tanh(L(\mathtt{u}_i)/2)\big),$$

where $\oplus$ *denotes addition in* $\mathbb{F}_2$.

Algorithm 1 Parallel Λ-rule BP$_4$ [LK21]

```
    Input:
1:  $\boldsymbol{S} \in \{I, X, Y, Z\}^{M\times N}$, $\boldsymbol{z} \in \{0,1\}^M$, LLR vectors $\{\boldsymbol{\Lambda}_n \in \mathbb{R}^3\}_{n=0}^{N-1}$, $\ell_{\max}$
    Initialization:
2:  $\ell = 0$
3:  for $n = 0, \ldots, N-1$ and $m \in \mathcal{M}(n)$ do
4:      $\boldsymbol{\Gamma}_{n\to m} = \boldsymbol{\Lambda}_n$
5:      Compute $\lambda_{S_{mn}}(\boldsymbol{\Gamma}_{n\to m})$
    Horizontal Step:
6:  for $m = 0, \ldots, M-1$ and $n \in \mathcal{N}(m)$ do
7:      $\Delta_{m\to n} = (-1)^{z_m} \boxplus_{n'\in\mathcal{N}(m)\setminus n} \lambda_{S_{mn'}}(\boldsymbol{\Gamma}_{n'\to m})$
    Vertical Step:
8:  for $n = 0, \ldots, N-1$ and $m \in \mathcal{M}(n)$ do
9:      $\Gamma^W_{n\to m} = \Lambda^W_n + \sum_{m'\in\mathcal{M}(n)\setminus m} \mathbb{1}\{\langle S_{m'n}, W\rangle = 1\}\,\Delta_{m'\to n}$   for $W \in \{X, Y, Z\}$
10:     Compute $\lambda_{S_{mn}}(\boldsymbol{\Gamma}_{n\to m})$
    Hard Decision:
11: for $n = 0, \ldots, N-1$ do
12:     $\Gamma^W_n = \Lambda^W_n + \sum_{m\in\mathcal{M}(n)} \mathbb{1}\{\langle S_{mn}, W\rangle = 1\}\,\Delta_{m\to n}$   for $W \in \{X, Y, Z\}$
13: Let $\boldsymbol{F} = (F_0, \ldots, F_{N-1})$, where $F_n = \begin{cases} I & \text{if } \Gamma^W_n > 0 \ \forall\, W \in \{X,Y,Z\} \\ \operatorname{argmin}_{W\in\{X,Y,Z\}} \Gamma^W_n & \text{otherwise} \end{cases}$
14: $\ell \leftarrow \ell + 1$
15: if $\langle \boldsymbol{S}, \boldsymbol{F}\rangle = \boldsymbol{z}$ then
16:     return SUCCESS
17: else
18:     if $\ell > \ell_{\max}$ then
19:         return FAILURE
20:     else
21:         goto Horizontal Step
```

2.6.4. Practical Considerations

Efficient Message Computation

It can be shown [HOP96] that $x \boxplus y$ has the representation

$$x \boxplus y = \ln \frac{1 + e^{x+y}}{e^x + e^y},$$

Using the *Jacobian* logarithm $\ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|})$, $x \boxplus y$ can be efficiently implemented using a lookup-table [HEAD01]. This way, computationally expensive multiplications can be avoided. Furthermore, we can rewrite Eq. 2.3 as

$$\lambda_\eta(\boldsymbol{L}) = \ln(1 + e^{-L^\eta}) - \ln(e^{-L^A} + e^{-L^B}),$$

where $\{A, B\} = \{X, Y, Z\} \setminus \eta$. Each of these terms can be efficiently calculated using the Jacobian logarithm and a lookup-table, as described above [LK21].
Additionally, the computation in the vertical step in Algorithm 1 may be simplified. It can be shown [LK21] that

$$\lambda_{S_{mn}}(\boldsymbol{\Gamma}_{n\to m}) = \lambda_{S_{mn}}(\boldsymbol{\Gamma}_n) - \Delta_{m\to n}. \tag{2.4}$$

It can thus be beneficial to omit line 9 in Algorithm 1 and first compute $\boldsymbol{\Gamma}_n$ according to line 12. Then, line 10 can be replaced by Eq. 2.4.

Further simplifications known from binary BP decoding such as the forward-backward algorithm in the vertical step may further reduce the complexity [HEAD01].

Implementation

We implemented and evaluated the BP_4 decoder using the Python programming language. In particular, we used the package Numpy [HMvdW+20] for matrix computations and the package galois [Hos20] to perform operations in GF(4).

3. Error Analysis for Hypergraph Product Codes

3.1. Bounded Distance Decoding

Consider a decoder which is able to correctly infer all errors up to weight w from a given syndrome. Such a decoder is referred to as a bounded distance decoder (BDD). The LER of a BDD is given by

$$P_{\mathrm{BDD}}(N, w) = 1 - (\sum_{j=0}^{w} \binom{N}{j} \epsilon^j (1-\epsilon)^{N-j}).$$

BDD performance can be easily evaluated and serves as a benchmark for the performance of a given decoder. This becomes particularly expressive in the low error rate domain, where decoding performance is mainly determined by the decoding behavior for low weight errors.
A BDD assumes that all errors up to a certain weight are decoded successfully. This assumption may overestimate the theoretical optimum decoding performance, since multiple errors within the correction radius w may share a syndrome. In this case, an optimal decoder has to decide for the most likely error that induces the given syndrome, leading to a potential decoding failure for other errors with the same syndrome within the correction radius. However, this might not always be the case for QSCs, since the (more likely) lower weight error might still correct the higher weight error due to degeneracy.
A decoder that considers such information is referred to as a generalized bounded distance decoder (GBDD). For a depolarizing channel with error rate ϵ, it has the logical error rate [LK21]

$$P_{\mathrm{GBDD}}(N, w, \gamma) = 1 - (\sum_{j=0}^{w} \gamma_j \binom{N}{j} \epsilon^j (1-\epsilon)^{N-j}), \tag{3.1}$$

where w is the correction radius and γ_j is the fraction of correctable weight-j errors. This term can be evaluated by counting all non-correctable weight-j errors within a radius of weight w. This aggravates the analysis of a GBDD, but results in a more precise estimate of the optimal decoding performance.
Recall from Sec. 2.3 that two errors $\boldsymbol{E}_m$ and $\boldsymbol{E}_{m'}$ induce the same syndrome if they are connected via an element $\boldsymbol{N} \in N(\mathcal{S})$. As described above, analysis of correctable errors also has to account for degeneracy: If $\boldsymbol{N} \in \mathcal{S}$, $\boldsymbol{E}_m$ corrects $\boldsymbol{E}_{m'}$ and vice versa. Thus, when counting non-correctable errors we only consider errors that are connected via elements $\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}$.

3.1.1. Type-1 Errors

Defintion 2 (Type-1 Error)**.** *An error $\boldsymbol{E}$ of weight $w(\boldsymbol{E})$ is a type-1 error, if the lowest weight of all errors related to $\boldsymbol{E}$ via an element $\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}$ is larger than $w(\boldsymbol{E})$, i.e., if*

$$w(\boldsymbol{E}) < \min_{\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}} w(\boldsymbol{N}\boldsymbol{E}).$$

Type-1 errors are correctable by an optimal decoder: Any other error with the same syndrome is of larger weight. The type-1 error thus constitutes the most likely error to induce the corresponding syndrome.

3.1.2. Type-2 Errors

Defintion 3 (Type-2 Error)**.** *An error $\boldsymbol{E}$ of weight $w(\boldsymbol{E})$ is a type-2 error, if the lowest weight of all errors related to $\boldsymbol{E}$ via an element $\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}$ is equal to $w(\boldsymbol{E})$, i.e., if*

$$w(\boldsymbol{E}) = \min_{\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}} w(\boldsymbol{N}\boldsymbol{E}).$$

Type-2 errors are partly correctable by an optimal decoder: Note that the equality in the definition above indicates that for syndromes of type-2 errors, multiple errors of minimum weight exist. Since all errors in the set of minimum weight errors occur with equal probability, an optimal decoder can arbitrarily choose any error from this set.

3.1.3. Type-3 Errors

Defintion 4 (Type-3 Error)**.** *An error $\boldsymbol{E}$ of weight $w(\boldsymbol{E})$ is a type-3 error, if the lowest weight of all errors related to $\boldsymbol{E}$ via an element $\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}$ is smaller than $w(\boldsymbol{E})$, i.e., if*

$$w(\boldsymbol{E}) > \min_{\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}} w(\boldsymbol{N}\boldsymbol{E}).$$

Type-3 errors are non-correctable: For a syndrome of a type-3 error, at least one error of lower weight exists. An optimal decoder thus chooses the (more likely) lower weight error, resulting in a decoding failure if the syndrome was induced by a type-3 error.

3.2. Hypergraph Product Codes

The HP code construction, first proposed by Tillich and Zemor [TZ14], provides a means to construct a CSS code from any two classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$ with binary PCMs $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$. The stabilizer matrix is of the form described in Eq. 2.2 and we choose [TZ14]

$$\begin{aligned} \boldsymbol{H}_X &= [\boldsymbol{H}_1 \otimes \boldsymbol{I}_{N_2} \mid \boldsymbol{I}_{M_1} \otimes \boldsymbol{H}_2^T] \\ \boldsymbol{H}_Z &= [\boldsymbol{I}_{N_1} \otimes \boldsymbol{H}_2 \mid \boldsymbol{H}_1^T \otimes \boldsymbol{I}_{M_2}], \end{aligned} \tag{3.2}$$

where $\dim \boldsymbol{H}_1 = M_1 \times N_1$ and $\dim \boldsymbol{H}_2 = M_2 \times N_2$. The resulting matrices are of dimension $\dim \boldsymbol{H}_X = M_1N_2 \times (N_1N_2 + M_1M_2)$ and $\dim \boldsymbol{H}_Z = M_2N_1 \times (N_1N_2 + M_1M_2)$.
Suppose that the codes $\mathcal{C}_1$ and $\mathcal{C}_2$ have minimum distances d_1 and d_2, respectively. Tillich and Zemor showed that the minimum distance of the resulting HP is lower bounded by the term

$$D \geq \min(d_1, d_2, d_1^T, d_2^T),$$

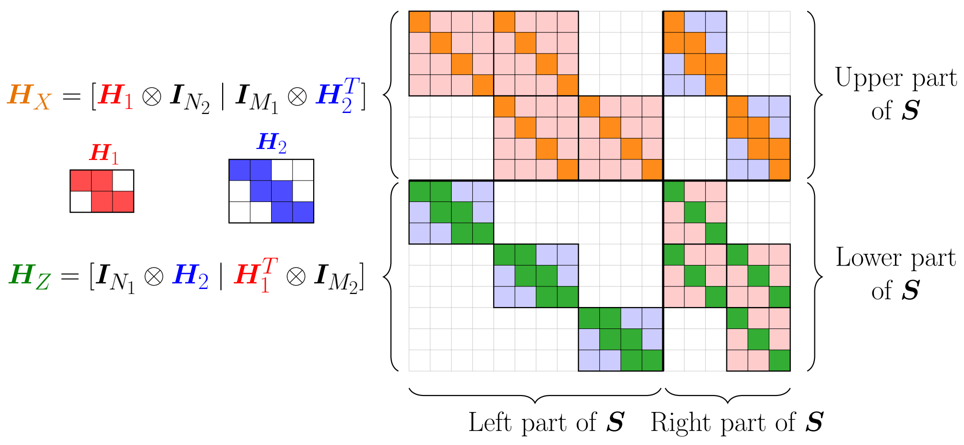
where d_1^T and d_2^T denote the minimum distances of the transpose codes $\mathcal{C}_1^T$ and $\mathcal{C}_2^T$ with PCMs $\boldsymbol{H}_1^T$ and $\boldsymbol{H}_2^T$ [TZ14].
There are no restriction on the classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$. As Tillich and Zemor pointed out, "any family of classical asymptotically good LDPC codes of fixed rate yields a family of quantum LDPC codes of fixed rate and minimum distance proportional to a square root of the block length" [TZ14]. We provide further details on quantum codes constructed from LDPC codes in Sec. 3.4 and in Sec. 4.3.

3.2.1. Notation for Hypergraph Product Codes

Equation 3.2 suggests that $\boldsymbol{S}$ can be partitioned into a left part $\boldsymbol{S}_\mathrm{l}$ (consisting of columns with index $n < N_1N_2$, starting from $n = 0$) and a right part $\boldsymbol{S}_\mathrm{r}$ (consisting of columns with index $N_1N_2 \leq n < N_1N_2 + M_1M_2$). Alternatively, $\boldsymbol{S}$ may be represented as an upper part $\boldsymbol{S}^X$ and a lower part $\boldsymbol{S}^Z$. These two representations can be combined by introducing submatrices for the upper-left ($\boldsymbol{S}_\mathrm{l}^X$), upper-right ($\boldsymbol{S}_\mathrm{r}^X$), lower-left ($\boldsymbol{S}_\mathrm{l}^Z$) and lower-right ($\boldsymbol{S}_\mathrm{r}^Z$) part, such that

$$\boldsymbol{S} = \begin{pmatrix} \boldsymbol{S}_\mathrm{l}^X & \boldsymbol{S}_\mathrm{r}^X \\ \boldsymbol{S}_\mathrm{l}^Z & \boldsymbol{S}_\mathrm{r}^Z \end{pmatrix} = \begin{pmatrix} X \cdot (\boldsymbol{H}_1 \otimes \boldsymbol{I}_{N_2}) & X \cdot (\boldsymbol{I}_{M_1} \otimes \boldsymbol{H}_2^T) \\ Z \cdot (\boldsymbol{I}_{N_1} \otimes \boldsymbol{H}_2) & Z \cdot (\boldsymbol{H}_1^T \otimes \boldsymbol{I}_{M_2}) \end{pmatrix} = \begin{pmatrix} \boldsymbol{S}_\mathrm{l} & \boldsymbol{S}_\mathrm{r} \end{pmatrix} = \begin{pmatrix} \boldsymbol{S}^X \\ \boldsymbol{S}^Z \end{pmatrix}.$$

Example 8. Suppose we construct a HP code, from two repetition codes, $\mathcal{C}_1$ ($[3, 1, 3]$ code) and $\mathcal{C}_2$ ($[4, 1, 4]$ code). The resulting HP will have the dimensions $N = 18$, $M = 17$. Figure 3.1 shows the stabilizer matrix $\boldsymbol{S}$ as well as the underlying classical PCMs $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$.



$\boldsymbol{H}_X = [\boldsymbol{H}_1 \otimes \boldsymbol{I}_{N_2} \mid \boldsymbol{I}_{M_1} \otimes \boldsymbol{H}_2^T]$
$\boldsymbol{H}_1$
$\boldsymbol{H}_2$
$\boldsymbol{H}_Z = [\boldsymbol{I}_{N_1} \otimes \boldsymbol{H}_2 \mid \boldsymbol{H}_1^T \otimes \boldsymbol{I}_{M_2}]$
Upper part of $\boldsymbol{S}$
Lower part of $\boldsymbol{S}$
Left part of $\boldsymbol{S}$
Right part of $\boldsymbol{S}$

Figure 3.1: Stabilizer matrix $\boldsymbol{S}$, constructed from a $[3, 1, 3]$ repetition code and a $[4, 1, 4]$ repetition code. $\boldsymbol{S}$ can be partitioned into four parts.

The Kronecker product in the stabilizer matrix $\boldsymbol{S}$ translates to a grid-structure of the nodes in the Tanner graph $\mathcal{T}(\boldsymbol{S})$. Note that in the scope of this thesis, we mainly focus on the left part of $\boldsymbol{S}$, as most of our observations are related to patterns in $\boldsymbol{S}_l$. However, the same considerations regarding the structure of the code apply for the right part of $\boldsymbol{S}$.
We first consider the upper-left part $\boldsymbol{S}_l^X$: The Tanner graph $\mathcal{T}(\boldsymbol{S}_l^X) = \mathcal{T}(X \cdot (\boldsymbol{H}_1 \otimes \boldsymbol{I}_{N_2}))$ is obtained as the row-wise concatenation of N_2 Tanner graphs $\mathcal{T}(\boldsymbol{H}_1)$, where each Tanner graph contains N_1 VNs and M_1 CNs. Thus, the VNs are arranged in an $N_2 \times N_1$ grid, whereas the CN are arranged in an $N_2 \times M_1$ grid. VNs in row i are connected to CNs in row i according to the Tanner graph $\mathcal{T}(\boldsymbol{H}_1)$. The grid-structure of the Tanner graph corresponding to $\boldsymbol{S}_l^X$ is shown in Figure 3.2. Note that all edges in $\mathcal{T}(\boldsymbol{S}_l^X)$ are of type X, as the term $\boldsymbol{H}_1 \otimes \boldsymbol{I}_{N_2}$ is multiplied by an X.

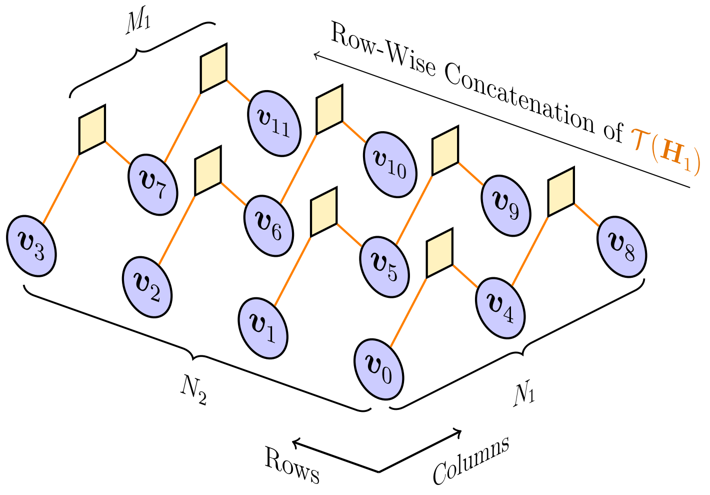


Figure 3.2: The term $\boldsymbol{H}_1 \otimes \boldsymbol{I}_{N_2}$ gives rise to the row-wise concatenation of the Tanner graph $\mathcal{T}(\boldsymbol{H}_2)$. Thus, the VN can be arranged on an $N_2 \times N_1$. Analogously, the CNs can be arranged on an $N_2 \times M_1$ grid.

Analogously, the Tanner graph $\mathcal{T}(\boldsymbol{S}_l^Z) = \mathcal{T}(Z \cdot (\boldsymbol{I}_{N_1} \otimes \boldsymbol{H}_2))$ is obtained as the column-wise concatenation of N_1 Tanner graphs $\mathcal{T}(\boldsymbol{H}_2)$, where each Tanner graph contains N_2 VNs and M_2 CNs. Again, we can think of the VNs as arranged in a $N_2 \times N_1$ grid. In fact, since $\boldsymbol{S}_l^X$ and $\boldsymbol{S}_l^Z$ are stacked vertically in $\boldsymbol{S}_l$, CNs corresponding to the upper part and to the lower part are connected to the same grid of VNs. CN corresponding to the lower part are arranged in an $M_2 \times N_1$ grid. VN in each column o are connected to CN in each column o according to the Tanner graph $\mathcal{T}(\boldsymbol{H}_2)$. The corresponding Tanner graph is depicted in Fig. 3.2.

We propose to use a new notation, which emphasizes the grid-structure of VNs and CNs in the left and the right part of $\boldsymbol{S}$, respectively. This notation is particularly useful to describe error patterns that are related to the classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$, as we will see in later sections. To differentiate between the left and the right part, let us denote VNs in the left part by vl and VNs in the right part by vr. Analogously, CNs in the upper part are denoted by cx and CNs in the lower part by cz, in accordance with the respective edge types.

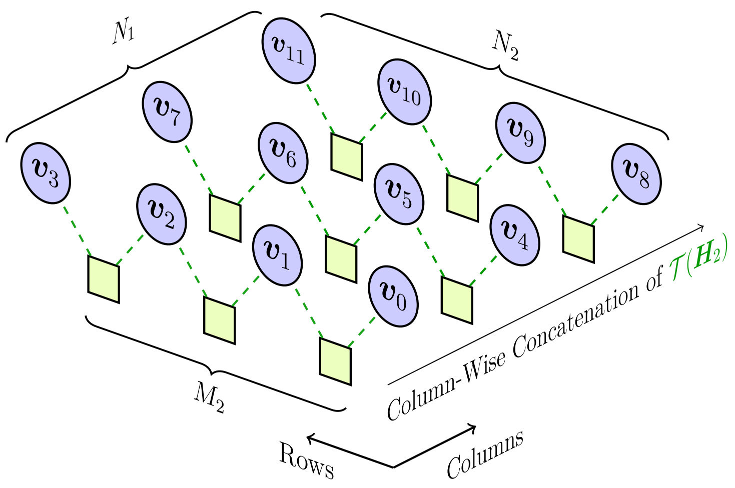


Figure 3.3: The term $\boldsymbol{I}_{N_1} \otimes \boldsymbol{H}_2$ gives rise to the column-wise concatenation of the Tanner graph $\mathcal{T}(\boldsymbol{H}_2)$. Thus, the VNs can be arranged on an $N_2 \times N_1$. Analogously, the CNs can be arranged on an $M_2 \times N_1$ grid.

We associate every VN index n with the row index (or "inner index") i and the column index (or "outer index") o corresponding to the position of the VN n in the grid. We choose the names "inner" and "outer" to emphasize the connection to the stabilizer matrix $\boldsymbol{S}$: The left part of $\boldsymbol{S}$ can be organized into N_1 "blocks" (i.e., collections of columns) of length N_2, as shown in Fig. 3.1. The outer index o describes the block index in which the index n lies. The inner index i describes the position of index n inside block o. For the left part of $\boldsymbol{S}$, the indices are related via

$$n = N_2 \cdot o + i, \qquad o = \left\lceil \frac{n}{N_2} \right\rceil, \qquad i = n \mod N_2.$$

To establish a connection between the indices in the right part of $\boldsymbol{S}$, we have to account for the fact that the index of the first VN in the right part of $\boldsymbol{S}$ is $n = N_1 N_2$. Since the block length in the right part is M_1, we obtain

$$n = N_1 N_2 + M_1 \cdot o + i, \qquad o = \left\lceil \frac{n - N_1 N_2}{M_1} \right\rceil, \qquad i = n - N_1 N_2 \mod M_1$$

to convert between n and (o, i) for VNs corresponding to the right part of $\boldsymbol{S}$.
VNs in the left part of $\boldsymbol{S}$ are then described as vl_o^i in our notation, whereas VNs in the right part of $\boldsymbol{S}$ are described as vr_o^i. As we are mostly interested in VNs in the left part of $\boldsymbol{S}$, we refer to these VNs as v_o^i instead of vl_o^i, if not stated differently.
We describe subsets of variable nodes $\{v_{n_0}, v_{n_1}, ...\}$ by simply concatenating the corresponding single terms. Errors with multiple non-identity entries are described using the same convention, just like in Sec. 2.1.3. As we will see later, the error analysis is facilitated if patterns that are characteristic to the classical codes are reflected in the representation.

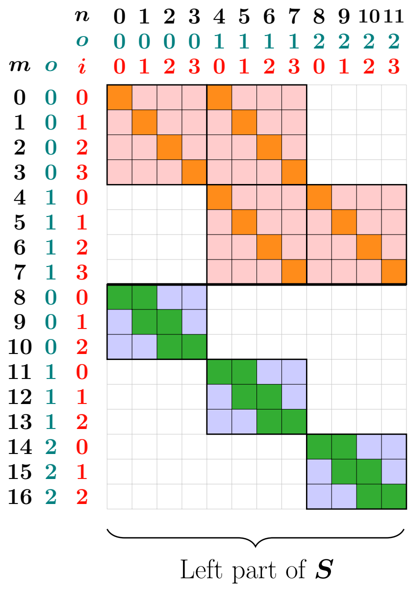
Just like we introduced outer and inner indices to account for the grid structure of VNs in the left part $\boldsymbol{S}_{\mathrm{l}}$ and the right part $\boldsymbol{S}_{\mathrm{r}}$, we an use outer and inner indices to account for the grid structure of CNs in the upper part $\boldsymbol{S}^X$ and in the lower part $\boldsymbol{S}^Z$. Recall from above that for CNs in the upper part of $\boldsymbol{S}$, the dimension of the grid is $N_2 \times M_1$. Accordingly, the CN indices m are related by

$$m = N_2 \cdot o + i, \qquad o = \left\lceil \frac{m}{N_2} \right\rceil, \qquad i = m \mod N_2.$$

In the lower part of $\boldsymbol{S}$, the CN grid has the dimension $M_2 \times N_1$. Taking into account the shift by $M_1 N_2$, we obtain

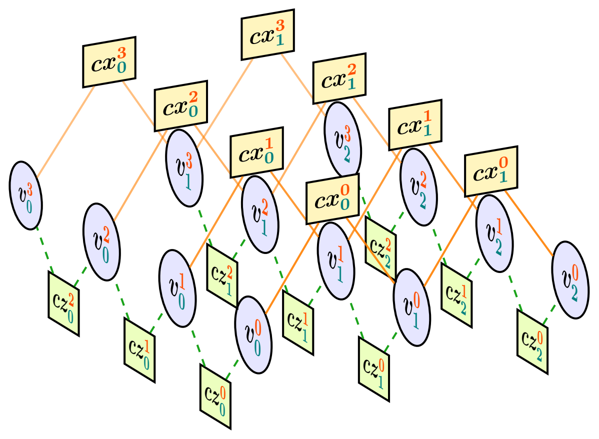
$$m = M_1 N_2 + M_1 \cdot o + i, \qquad o = \left\lceil \frac{m - M_1 M_2}{M_1} \right\rceil, \qquad i = m - M_1 N_2 \mod M_1.$$

Example 9. We conclude this section by illustrating our proposed notation for the left part of the HP described in Example 8. Figure 3.4a shows the left part of the stabilizer matrix $\boldsymbol{S}_{\mathrm{l}}$ along with the indices n and m for VNs and CNs and the corresponding outer and inner indices. Each column of $\boldsymbol{S}_{\mathrm{l}}$ corresponds to a VN, while each row corresponds to a CN. With our notation, it becomes easy to see connections to properties of the classical codes.



n 0 1 2 3 4 5 6 7 8 9 10 11
o 0 0 0 0 1 1 1 1 2 2 2 2
m o i 0 1 2 3 0 1 2 3 0 1 2 3
0 0 0
1 0 1
2 0 2
3 0 3
4 1 0
5 1 1
6 1 2
7 1 3
8 0 0
9 0 1
10 0 2
11 1 0
12 1 1
13 1 2
14 2 0
15 2 1
16 2 2
Left part of $\boldsymbol{S}$

(a) Stabilizer matrix $\boldsymbol{S}_{\mathrm{l}}$, with indices n (VNs) and m (CNs) and the corresponding outer and inner indices.



(b) The Tanner graph $\mathcal{T}(\boldsymbol{S}_{\mathrm{l}})$ shows a grid structure in both, VNs and CNs.

Figure 3.4: Stabilizer matrix and Tanner graph of the left part of the HP code constructed from a $[3, 1, 3]$ repetition code and a $[4, 1, 4]$ repetition code.

3.3. Error Analysis for the [[129, 28, 3]] Hypergraph Product Code

The $[[129, 28, 3]]$ HP code is used in several papers on BP decoding of quantum codes [KL20, LK21, LP19]. It serves as a good introductory example, since results are available in the literature for comparison and error analysis is feasible due to a rich algebraic structure and a comparably short blocklength. In the following, we describe the classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$ used in it's construction and demonstrate the connection between properties of the classical codes and error correction properties of the resulting HP code.

3.3.1. Code Construction

The $[[129, 28, 3]]$ HP code is constructed from the $[7, 4, 3]$ Bose–Chaudhuri–Hocquenghem (BCH) code $\mathcal{C}_1$ and the $[15, 7, 5]$ BCH codes $\mathcal{C}_2$ with the generator polynomials $g_1(x) = x^3 + x + 1$ and $g_2(x) = x^8 + x^7 + x^6 + x^4 + 1$, respectively. The resulting stabilizer matrix $\boldsymbol{S}$ is depicted in Fig. 3.5.

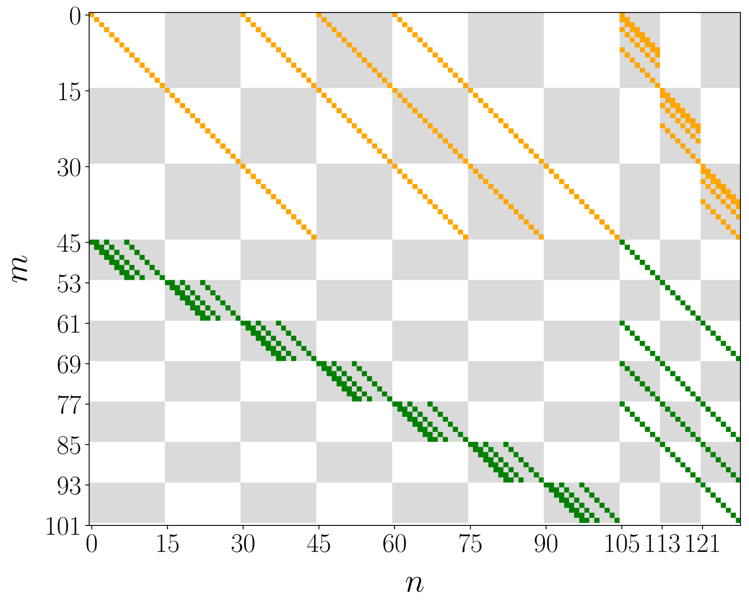


Figure 3.5: Stabilizer matrix $\boldsymbol{S}$ of the $[[129, 28, 3]]$ HP code. The check pattern is provided to emphasize the dimensions for each of the four regions.

For the $[[129, 28]]$ HP code, the analysis of the GBDD for $w = 1, 2$ is feasible and insightful, since it provides us with a good estimate on the optimal decoding performance in the low error rate region. To better understand this, consider the weight distribution of errors, sampled according to the depolarizing channel model. The weight of an error is determined by the number of non-identity entries. Since each entry is in $\{X, Y, Z\}$ with probability ϵ or I with probability $1 - \epsilon$, the weight of an error of length n follows the binomial distribution:

$$P(N, w) = \binom{N}{w} (1 - \epsilon)^{N-w} \epsilon^w.$$

Figure 3.6 shows this distribution for different error rates ϵ. As ϵ becomes small, the weight distribution is dominated by low weight errors. For $\epsilon = 10^{-2.5}$, roughly 70% of all errors have weight zero and only 5% of all errors have weight larger than one.

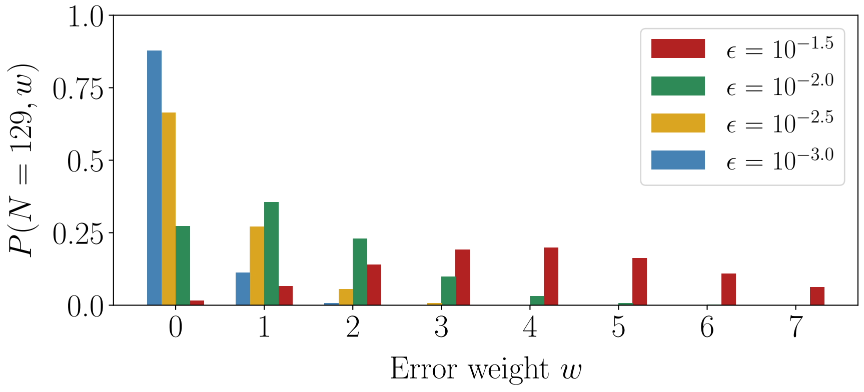


Figure 3.6: Weight distribution of errors, sampled according to a depolarizing channel model with different values for ϵ.

In the low error rate region it thus suffices to consider errors of weight one and two, since the a-priori probability for errors of weight larger than two becomes negligible.

3.3.2. Weight-1 Errors

For a code of length N, there exist $\binom{N}{j} \cdot 3^j$ weight-j errors, since each qubit can be corrupted by an entry $E_n \in \{X, Y, Z\}$. This amounts to $3 \cdot 129 = 387$ weight-1 errors for the $[[129, 28]]$ HP. The same number of unique syndromes can be counted. This means that every weight-1 error is uniquely identified by it's syndrome and thus correctable by the GBDD.

3.3.3. Weight-2 Errors

While every weight-1 error is uniquely identified by it's syndrome, some weight-2 errors share their syndrome with errors of weight one or two. This can be attributed to the presence of weight-3 errors in $N(\mathcal{S}) \setminus \mathcal{S}$. These weight-3 errors have their origin in the codespace of the classical $[7, 4, 3]$ BCH code $\mathcal{C}_1$, used to construct the $[[129, 28]]$ HP code. Recall from Section 3.2.1 that due to the HP code construction, VNs in the left part of $\boldsymbol{S}$ with identical inner indices are connected to checks in the upper half of $\boldsymbol{S}$ via X edges, according to the Tanner graph $\mathcal{T}(\boldsymbol{H}_1)$.
Suppose an error consists only of identity entries, except for VNs in one row i of the VN grid, where Z entries may occur. As these VN are connected to X checks according to $\mathcal{T}(\boldsymbol{H}_1)$, all checks in row i will be satisfied iff the outer indices o of the Z entries match a valid codeword of $\mathcal{C}_1$. Such an error will also commute with all other checks in the upper half, as all remaining entries are I entries. It will also commute with all checks in the lower half, since all of it's non-identity entries commute with the Z checks in the lower half. Thus, weight-w errors of the form $\boldsymbol{N} = Z^i_{o_0} \dots Z^i_{o_{w-1}}$ are elements of the set $N(\mathcal{S})$ if the outer indices $\{o_0, \dots, o_{w-1}\}$ match the indices of non-zero entries of valid weight-w codewords in $\mathcal{C}_1$. For the $[[129, 28]]$ HP code, we are particularly interested in weight-3

codewords in $\mathcal{C}_1$, as they constitute the codewords of minimum weight. Since all stabilizers of the $[[129, 28]]$ HP code have weight larger than three [KCL21], these weight-3 errors are also elements of the set $N(\mathcal{S}) \setminus \mathcal{S}$.

Remark 7. *The same line of reasoning applies for codewords in $\mathcal{C}_2$. Errors of the form $\boldsymbol{N} = X_o^{i_0} \dots X_o^{i_{w-1}}$ will satisfy every check in the lower half of $\boldsymbol{S}$ if the indices $\{i_0, \dots, i_{w-1}\}$ match the indices of non-zero entries of a valid weight-w codeword in $\mathcal{C}_2$. Such errors will also commute with all checks in the upper half, since these only use X edges.*
However, we omit these errors in the error analysis, since $\mathcal{C}_2$ has a minimum distance of $d_{\min} = 5$. In the low error rate domain, decoding failures connected to elements in $N(\mathcal{S}) \setminus \mathcal{S}$ will hence be dominated by the code $\mathcal{C}_1$, which has minimum distance $d_{\min} = 3$.

Type-3 Errors

In the following, we describe how type-3 errors follow from weight-3 errors in $\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}$. Recall from above that such errors have the form $\boldsymbol{N} = Z^i_{o_0} Z^i_{o_1} Z^i_{o_2}$. We can now partition $\boldsymbol{N}$ into three pairs of the form $(\boldsymbol{E}_{w1}, \boldsymbol{E}_{w2})$, i.e., $(Z^i_{o_0}, Z^i_{o_1} Z^i_{o_2})$, $(Z^i_{o_1}, Z^i_{o_0} Z^i_{o_2})$ and $(Z^i_{o_2}, Z^i_{o_0} Z^i_{o_1})$. For each pair, we have $\boldsymbol{N} = \boldsymbol{E}_{w1}\boldsymbol{E}_{w2}$. Since $\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}$, $\boldsymbol{E}_{w2}$ is a type-3 error of weight-2, as defined in Definition 4.
We now count the number of type-3 errors arising due to a weight-3 error in $N(\mathcal{S}) \setminus \mathcal{S}$. The weight enumerator $A(W)$ of $\mathcal{C}_1$ is given as

$$A(W) = 7W^3 + 7W^4 + W^7.$$

Each of the seven weight-3 codewords in $\mathcal{C}_1$ gives rise to three type-3 errors. This behaviour is consistent across $N_2 = 15$ inner indices. Thus, the number of type-3 errors multiplies accordingly, resulting in a total of 15 (inner indices) $\cdot$ 7 (weight-3 errors) $\cdot$ 3 (type-3 errors per weight-3 error) $= 315$ type-3 errors.

Type-2 Errors

Similar to the case above, errors of the form $\boldsymbol{N} = Z^i_{o_0} Z^i_{o_1} Z^i_{o_2}$ connect two errors $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ of weight-2. Both $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ follow Definition 3. Note that these pairs are of the form $(\boldsymbol{E}_1, \boldsymbol{E}_2) = (Z^i_{o_0} Y^i_{o_1}, X^i_{o_1} Z^i_{o_2})$. An optimal decoder chooses one of the two type-2 errors from the corresponding type-2 syndrome. This results in decoding success in half of all cases where a type-2 error occurs (assuming a depolarizing channel model where both errors in a pair a equally likely).
To count the number of type-2 error pairs, we can use similar arguments as for the type-3 errors. Recall that we can find seven errors of the form $\boldsymbol{N} = Z^i_{o_0} Z^i_{o_1} Z^i_{o_2}$. For each of these errors, we can place the Y entry at any index $\{o_0, o_1, o_2\}$. For the Z entry, two indices remain, resulting in a total number of $3 \cdot 2 = 6$ possible errors $\boldsymbol{E}_1$ for each of the seven $\boldsymbol{N}$. The resulting combinations are listed in Tab. 3.1. Again, this pattern repeats $N_2 = 15$ times, resulting in a total number of $15 \cdot 7 \cdot 6 = 630$ type-2 error pairs. For each pair, one error is correctable by an optimal decoder.

Table 3.1: There are six combinations to choose a Y and a Z entry in two out of three positions.

$\boldsymbol{E}_1$	YZI	YIZ	IYZ	ZYI	ZIY	IZY
$\boldsymbol{NE}_2$	XIZ	XZI	ZXI	IXZ	IZX	ZIX

Type-1 Errors

For a QSC of blocklength N, there exist $3^2 \cdot \binom{N}{2}$ errors of weight two. For the $[[129, 28]]$ HP, this gives a total 74304 weight-2 errors. Due to weight-3 errors $\boldsymbol{N} \in N(\mathcal{S}) \setminus \mathcal{S}$, 315 weight-2 errors are of type-1 and 1260 weight-2 errors are of type-2. Numerical analysis shows that the remaining $74304 - 315 - 1260 = 72729$ errors are of type-1, i.e., they have a unique syndrome among all weight-1 and weight-2 errors. Recall from Sec. 3.1.1 that all type-1 errors are correctable by an optimal decoder.

3.3.4. Theoretical Error Correction Performance

Table 3.2 summarizes the distribution of both, weight-1 errors and weight-2 errors, as well as the error correction capabilities of an optimal decoder for these errors.

Table 3.2: Number of corrected weight-1 and weight-2 errors for parallel and serial BP_4 decoding with different initial values ϵ_0.

	Weight-1	Weight-2			
		Type-1	Type-2	Type-3	$\sum$
Number of Errors	387	72729	1260	315	74304
Number of Correctable Errors	387	72729	630	0	73359

We can use Tab. 3.2 to determine the value of γ_2, used in Eq. 3.1. While it is not possible to correct any type-3 error, half of the type-2 errors can be corrected. This gives a fraction of $\gamma_2 = (74304 - 315 - 630)/(74304) \approx 98.73\%$ correctable weight-2 errors. In Chapter 4, we compare the performances of BP_4 decoding and the GBDD for $w = 2$.

3.4. Error Analysis for Symmetric Hypergraph Product Codes

Next, we will study properties of errors for the family of *symmetric* HP code. For this class of quantum codes, Roffe et al. report a considerable performance improvement to BP when OSD is applied in a post-processing step [RWBC20]. This shows that BP decoding performs far from optimal, leaving room for improvement. As stated by Raveendran and Vasić, the knowledge of structures in the Tanner graph may help to improve the decoding performance without the need of (computationally expensive) post-processing [RV21]. In the following sections, we strive to shed some light on error configurations that cause decoding failures in order to improve BP decoding for this special class of QLDPC codes.

3.4.1. Symmetric Hypergraph Product Codes

Symmetric HP codes are obtained from the HP code construction when $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}$. Suppose that $\mathcal{C}$ is a linear code with parameters $[n, k, d]$ and PCM $\boldsymbol{H}$. The transpose code $\mathcal{C}^T$ has the parameters $[m, k^T, d^T]$ with $m = n - k$, where k^T and d^T describe the number of logical encoded qubits and the minimum distance of the transpose code, respectively. The resulting quantum code has the parameters $[[n^2 + m^2, k^2 + (k^T)^2, \min(d, d^T)]]$ [RWBC20]. Symmetric HP codes have a stabilizer matrix with row weights $i + j$, where i and j denote the row and column weights of the classical PCM $\boldsymbol{H}$ respectively [TZ14].

3.4.2. Using Regular LDPC Codes in the Code Construction

Suppose we use a regular (d_v, d_c)-LDPC code $\mathcal{C}_\text{C}$ of dimension $[n, k, d]$ for both, $\mathcal{C}_1$ and $\mathcal{C}_2$ in the HP code construction. The resulting quantum code will have a constant CN degree $d_\text{v} + d_\text{c}$. The VN degree is $2 \cdot d_\text{v}$ in the left part of $\boldsymbol{S}_\text{l}$ and $2 \cdot d_\text{c}$ in $\boldsymbol{S}_\text{r}$. [RWBC20]
Roffe et al. choose classical codes of rate $r_\text{C} = 0.25$, leading to HP codes of rate $r_\text{HP} = 0.04$. The parameters of both classical codes and the resulting HP codes are listed in Tab. 3.3.

Table 3.3: Parameters of the classical code $\mathcal{C}_\text{C}$ and the resulting HP code $\text{HP}(\mathcal{C}_\text{C})$.

$\mathcal{C}_\text{C}$	r_C	$\mathcal{C}_\text{HP}$	r_HP
$[16, 4, 6]$	0.25	$[[400, 16, 6]]$	0.04
$[20, 5, 8]$	0.25	$[[625, 25, 8]]$	0.04
$[24, 6, 10]$	0.25	$[[900, 36, 10]]$	0.04

3.4.3. Error Pairs with Equivalent Decoding Behaviour

We now use our grid-based notation and the symmetric property of symmetric HP codes to establish a connection between pairs of errors on which parallel BP$_4$ decoding shows equivalent decoding behaviour.

Theorem 1. *Let $f(\cdot)$ be a mapping which interchanges X entries with Z entries and leaves I entries and Y unchanged. For a collection of Pauli matrices (such as vectors, matrices or graphs), it acts elementwise. Let π_c be a permutation which maps an index $[\,\cdot\,]_o^i$ to $[\,\cdot\,]_i^o$, i.e., π_c interchanges outer and inner indices. For two errors $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ that are connected via*

$$\boldsymbol{E}_2 = f(\pi_\text{c}(\boldsymbol{E}_1)), \tag{3.3}$$

parallel BP$_4$ decoding shows equivalent behaviour (up to $f(\pi_\text{c}(\cdot))$. This means that the belief histories for both errors will be closely related: If the qubits are reordered according to π_c and beliefs on X entries and Z entries are interchanged, the belief histories will be identical.

Proof. Let $\boldsymbol{P}_\text{c}$ denote a matrix that performs column permutations of $\boldsymbol{S}$, according to π_c. A permutation of columns of $\boldsymbol{S}$ translates to a rearrangement of VNs in the corresponding

Tanner graph $\mathcal{T}$. While this changes the naming conventions of VNs, the permuted Tanner graph $\mathcal{T}_\mathrm{c}$ is isomorph to $\mathcal{T}$. As long as all VNs are updated in parallel, an error whose entries are permuted with the same π_c as the VNs in the Tanner graph will result in identical messages (up to π_c).

The key insight lies in a property of the stabilizer matrix $\boldsymbol{S}$ of symmetric HP codes: Recall from Eq. 3.2 that $\boldsymbol{S}$ is of the form

$$\boldsymbol{S} = \begin{pmatrix} X \cdot \boldsymbol{H}_X \\ Z \cdot \boldsymbol{H}_Z \end{pmatrix},$$

where $\boldsymbol{H}_X = [\boldsymbol{H} \otimes I_{N_\mathrm{C}} | I_{M_\mathrm{C}} \otimes \boldsymbol{H}^T]$, and $\boldsymbol{H}_Z = [I_{N_\mathrm{C}} \otimes \boldsymbol{H} | \boldsymbol{H}^T \otimes I_{M_\mathrm{C}}]$ and $\dim \boldsymbol{H} = M_\mathrm{C} \times N_\mathrm{C}$. The edge types of the lower and upper part of $\boldsymbol{S}$ can be interchanged by permuting rows and columns, i.e., we can obtain $\boldsymbol{S}' = f(\boldsymbol{S})$ as

$$\boldsymbol{S}' = \begin{pmatrix} Z \cdot \boldsymbol{H}_X \\ X \cdot \boldsymbol{H}_Z \end{pmatrix} = \boldsymbol{P}_\mathrm{r} \cdot \begin{pmatrix} X \cdot \boldsymbol{H}_X \\ Z \cdot \boldsymbol{H}_Z \end{pmatrix} \cdot \boldsymbol{P}_\mathrm{c} = \boldsymbol{P}_\mathrm{r} \cdot \boldsymbol{S} \cdot \boldsymbol{P}_\mathrm{c}. \tag{3.4}$$

A proof of this relation is provided in Appendix B. The permutation π_c rearranges columns by interchanging outer and inner indices, i.e., a VN vl_o^i is interchanged with VN vl_i^o and a VN vr_o^i is interchanged with VN vr_i^o. We elaborate on the construction of $\boldsymbol{P}_\mathrm{c}$ in Remark 8. Note that the subsequent row permutation π_r doesn't change the code, since it only rearranges CNs. It serves the sole purpose of emphasizing the connection $\boldsymbol{S}' = f(\boldsymbol{S}) = \pi_\mathrm{r}(\pi_\mathrm{c}(\boldsymbol{S}))$ and is thus not needed in the following consideration.

Since applying $f(\cdot)$ to $\boldsymbol{S}$ has the effect of interchanging X and Z entries, the Tanner graph $\mathcal{T}(\boldsymbol{S}')$ can be obtained from both, rearrangement of VNs in $\mathcal{T}(\boldsymbol{S})$ as well as applying f to $\mathcal{T}(\boldsymbol{S})$ (up to a permutation of rows π_r, which doesn't affect the code space and/or the decoding behaviour). This relation is summarized in Fig. 3.7.
Now suppose an error $\boldsymbol{E}_1$ shows a certain decoding behaviour on Tanner graph $\mathcal{T}(\boldsymbol{S})$. Thus, $\boldsymbol{E}_1' = \pi_\mathrm{c}(\boldsymbol{E}_1)$ shows equivalent decoding behaviour on $\mathcal{T}(\boldsymbol{S}')$ (up to permutation of belief histories according to π_c). Since $f(\mathcal{T}(\boldsymbol{S}')) = \mathcal{T}(\boldsymbol{S})$ and due to the symmetry in the VN update rule of BP$_4$, the error $\boldsymbol{E}_2 = f(\boldsymbol{E}_1')$ shows the same decoding behaviour on $\mathcal{T}(\boldsymbol{S})$ than $\boldsymbol{E}_1'$ shows on $\mathcal{T}(\boldsymbol{S}')$, if X and Z beliefs interchanged. Thus, $\boldsymbol{E}_2 = f(\pi_\mathrm{c}(\boldsymbol{E}_1))$ shows equivalent decoding behaviour on $\mathcal{T}(\boldsymbol{S})$ to the decoding behaviour of $\boldsymbol{E}_1$ on $\mathcal{T}(\boldsymbol{S})$ (up to $f(\pi_\mathrm{c}(\cdot))$). $\square$

$$\begin{tikzcd}
& \mathcal{T}(\boldsymbol{S}_\mathrm{c}) \arrow[dr, "\pi_\mathrm{r}"] & \\
\mathcal{T}(\boldsymbol{S}) \arrow[ur, "\pi_\mathrm{c}"] \arrow[rr, bend left, "f"'] & & \mathcal{T}(\boldsymbol{S}') \arrow[ll, bend left]
\end{tikzcd}$$

Figure 3.7: The Tanner graph $\mathcal{T}(\boldsymbol{S})$ can be obtained from both, column and row permutation as well as exchanging X and Z connections.

Example 10. Consider the stabilizer matrix $\boldsymbol{S}$ of a symmetric HP code, where the classical PCM $\boldsymbol{H} = \left(\begin{smallmatrix}1&1&0\\0&1&1\end{smallmatrix}\right)$ is chosen according to a $[3,1,3]$ repetition code. $\boldsymbol{S}$ is depicted in Fig. 3.8. Since $N_{\mathrm{C}} = 3$ and $M_{\mathrm{C}} = 2$, we have $N_{\mathrm{C}}^2 + M_{\mathrm{C}}^2 = 13$ VNs. Outer and indices within each part of $\boldsymbol{S}$ are swapped by multiplication with the matrix

$$\boldsymbol{P}_{\mathrm{c}} = \left(\begin{array}{ccc|ccc|ccc||cc|cc} \boldsymbol{e}_0 & \boldsymbol{e}_3 & \boldsymbol{e}_6 & \boldsymbol{e}_1 & \boldsymbol{e}_4 & \boldsymbol{e}_7 & \boldsymbol{e}_2 & \boldsymbol{e}_5 & \boldsymbol{e}_8 & \boldsymbol{e}_9 & \boldsymbol{e}_{11} & \boldsymbol{e}_{10} & \boldsymbol{e}_{12} \end{array} \right).$$

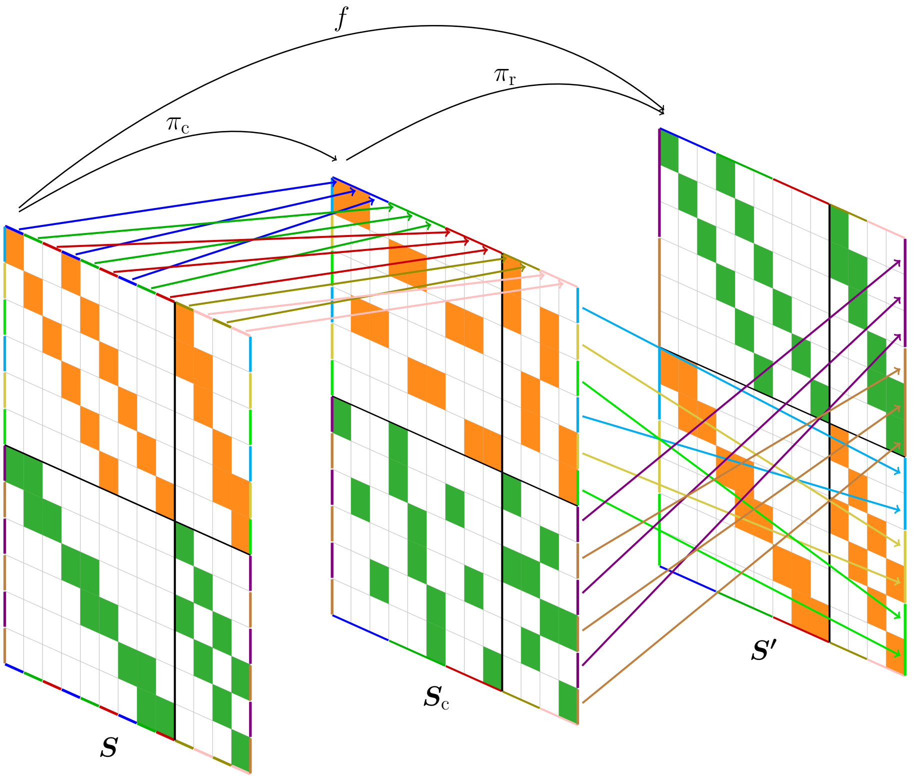
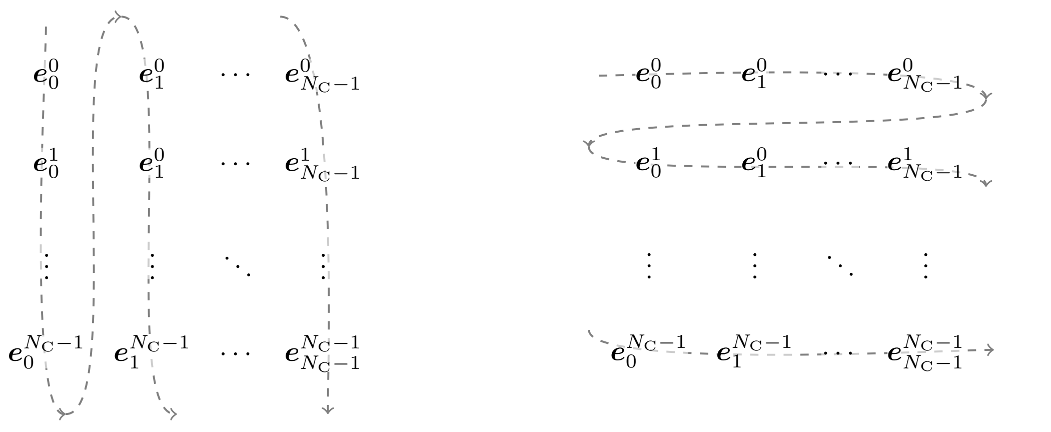


Figure 3.8: Successive permutation of columns and rows of $\boldsymbol{S}$ has the same effect as exchanging X and Z connections in $\boldsymbol{S}$.

Remark 8. *Since $\boldsymbol{P}_{\mathrm{c}}$ only permutes columns within each part of $\boldsymbol{S}$, it can be represented as $\boldsymbol{P}_{\mathrm{c}} = (\boldsymbol{P}_{\mathrm{c}}^{\mathrm{l}} \ \boldsymbol{P}_{\mathrm{c}}^{\mathrm{r}})$, where $\boldsymbol{P}_{\mathrm{c}}^{\mathrm{l}}$ only permutes columns with index $n < N_{\mathrm{C}}^2$ and $\boldsymbol{P}_{\mathrm{c}}^{\mathrm{r}}$ only permutes columns with index $N_{\mathrm{C}}^2 \leq n < N_{\mathrm{C}}^2 + M_{\mathrm{C}}^2$. A convenient way to obtain $\boldsymbol{P}_{\mathrm{c}}^{\mathrm{l}}$ is depicted in Fig. 3.9. The first N_{C}^2 unit vectors of an $(N_{\mathrm{C}}^2 + M_{\mathrm{C}}^2)$-dimensional space are arranged in an $N_{\mathrm{C}} \times N_{\mathrm{C}}$ grid in a column-wise order (Fig. 3.9a). This arrangement is analogue to the VN grid associated to the symmetric HP code. We then concatenate the unit vectors into the permutation matrix $\boldsymbol{P}_{\mathrm{c}}^{\mathrm{l}}$. The order of this concatenation is obtained from reading the grid in a row-wise order (Fig. 3.9b) This yields the permutation matrix that interchanges outer and inner indices in the left part of $\boldsymbol{S}$, i.e.,*

$$\boldsymbol{P}_{\mathrm{c}}^{\mathrm{l}} = \left(\begin{array}{cccc|cccc|c|cccc} \boldsymbol{e}_0^0 & \boldsymbol{e}_1^0 & \cdots & \boldsymbol{e}_{N_{\mathrm{C}}-1}^0 & \boldsymbol{e}_0^1 & \boldsymbol{e}_1^1 & \cdots & \boldsymbol{e}_{N_{\mathrm{C}}-1}^1 & \cdots & \boldsymbol{e}_0^{N_{\mathrm{C}}-1} & \boldsymbol{e}_1^{N_{\mathrm{C}}-1} & \cdots & \boldsymbol{e}_{N_{\mathrm{C}}-1}^{N_{\mathrm{C}}-1} \end{array} \right).$$



(a) We start by writing the first N_C^2 unit vectors in an $N_\mathrm{C} \times N_\mathrm{C}$ grid in a column-wise order.

(b) The order of unit vectors in $\boldsymbol{P}_\mathrm{c}^\mathrm{l}$ is obtained by reading the grid in a row-wise order.

Figure 3.9: Construction of the permutation matrix $\boldsymbol{P}_\mathrm{c}^\mathrm{l}$.

Analogously, $\boldsymbol{P}_\mathrm{c}^\mathrm{r}$ is obtained by writing the last M_C^2 unit vectors of an $(N_\mathrm{C}^2 + M_\mathrm{C}^2)$-dimensional space in an $M_\mathrm{C} \times M_\mathrm{C}$ grid in a column-wise order and concatenating these unit vectors into $\boldsymbol{P}_\mathrm{c}^\mathrm{r}$ by reading the grid row-wise.

3.5. Decoding Issues of Quaternary Belief Propagation for Hypergraph Product Codes

3.5.1. Message Overestimation

It is known that BP only produces exact results if the Tanner graph is a tree (or in other words, if it is free of cycles) [KFL01]. While BP can produce good approximations even in the presence of cycles, the girth (i.e., the length of the shortest cycle in the Tanner graph) should be sufficiently large [BBA+15]. Especially short cycles can be detrimental to the decoding performance, as they can quickly build unwanted dependency between passed messages, leading to "mistakenly higher reliability", as stated by Yazdani et al. in [YHB04]. Unfortunately, the symplectic criterion leads to unavoidable cycles of length four as shown in the following proof (largely based on [BBA+15]).

Proof. Recall from Sec. 2.2.2 that the commutation relation requires each pair of rows $\boldsymbol{S}_m, \boldsymbol{S}_{m'}$ of the stabilizer matrix $\boldsymbol{S}$ to commute. In other words, each pair of rows must have different non-identity entries for an even number of indices n (corresponding to the VNs). As a first instinct, one might only assign a single non-identity type W to a VN n, in order to avoid that this VN contributes anti-commuting entries for any pair of stabilizers. However, the resulting code wouldn't be able to detect the weight-1 error W_n and hence, would have a minimum distance of one, which is undesirable [BBA+15]. Since it is necessary to have two different non-identity entries on index n for at least one pair of stabilizers, at least one pair of stabilizers anti-commutes on the corresponding entry n. As the total number of anti-commuting entries has to be an even number, the corresponding stabilizers are required to anti-commute on another entry, leading to a cycle of length four in the underlying Tanner graph. □

As a consequence, BP_4 on quantum codes often suffers from message overestimation, resulting in a degraded decoding performance [KL20].

3.5.2. Split Beliefs

One complication of BP decoding is that it seeks to estimate the single most likely error in a qubit-wise fashion [ROJ19]. Hence, symmetries in the code are reflected in symmetric update rules of the decoder. However, this symmetry is broken by the channel, leading to symmetric errors [PC08]. Consider two errors of equal weight $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ with identical syndrome $\boldsymbol{z}$, i.e., $\boldsymbol{E}_1\boldsymbol{E}_2 \in N(\mathcal{S})$. Suppose that both errors are minimum-weight solutions to the syndrome equation. An optimal decoder has to account for this symmetry by adding the probabilities of both errors and assign the sum to either one of them [PC08]. However, a standard BP decoder doesn't include this fact in it's update rules and instead attempts to converge to both (equally likely) errors simultaneously. As the superposition of these errors doesn't match the syndrome, decoding fails to converge. Such a situation is referred to as a *split belief* [RWBC20].

3.5.3. Type-A and Type-B Errors

Recall from Sec. 3.2.1 that the Tanner graph $\mathcal{T}(\boldsymbol{S})$ of a HP code contains the Tanner graphs of the classical codes $\mathcal{T}(\boldsymbol{H}_1)$ and $\mathcal{T}(\boldsymbol{H}_2)$ as subgraphs. We observed that BP_4 decoding often fails for errors whose structure is related to these subgraphs. For such errors, the decoding behaviour of BP_4 is closely related to the decoding behaviour of BP_2 on the classical code. We first define the form of these errors and then discuss, why their structure results in a connection between BP_4 and BP_2.

Defintion 5 (Type-A error:)**.** *An error $\boldsymbol{E}$ is a type-A error, if it has the representation $\boldsymbol{E} = W_{n_0} \dots W_{n_{w-1}} = W_o^{i_0} \dots W_o^{i_{w-1}}$, and all $n_0, \dots, n_{w-1} < N_1 N_2$. In other words, the indices of all non-identity entries (1) have to be associated to columns in the left part of $\boldsymbol{S}$ and (2) have to have the same outer index o.*

Note that, as all non-identity entries of a type-A error share the same outer index, they all lie in the same column of the $N_2 \times N_1$ grid described in Sec. 3.2.1. Now suppose that all non-identity entries of a type-A error $\boldsymbol{E}$ are X entries, i.e., $\boldsymbol{E}$ has the representation $\boldsymbol{E} = X_o^{i_0} \dots X_o^{i_{w-1}}$. Let us refer to errors of this form as *homogeneous* type-A errors. $\boldsymbol{E}$ then commutes with every check in the upper half of $\boldsymbol{S}$. Since all non-identity entries fall into the same column o in the VN grid, such an error only induces a non-trivial syndrome in Z checks with outer index o.

Example 11. Consider the left part of the stabilizer matrix $\boldsymbol{S}$ of the symmetric HP code used in Example 10. The Tanner graph $\mathcal{T}(\boldsymbol{S}_l)$ is shown in Fig. 3.10: The VNs can be arranged into an $N_\mathrm{C} \times N_\mathrm{C} = 3 \times 3$ grid and the Z checks can be arranged into an $M_\mathrm{C} \times C_\mathrm{C} = 2 \times 3$ grid. Figure 3.10 shows a homogeneous type-A error, i.e., all (potential) non-identity entries are X entries and lie in column $o = 0$. It can be easily seen that such an error only induces a non-trivial syndrome for Z checks in the corresponding column.

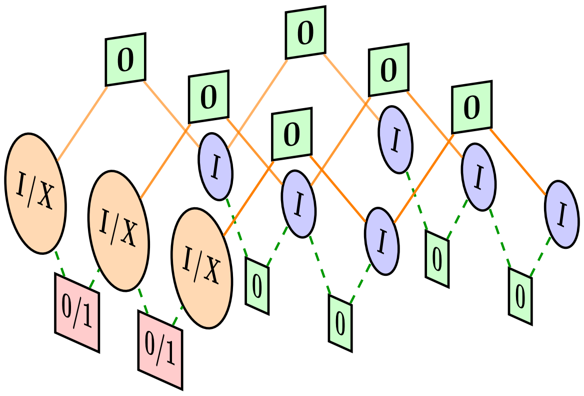


Figure 3.10: A type-A error with X entries only in one column o only induces a non-trivial syndrome in the corresponding CN column in the lower half of $\boldsymbol{S}$.

Defintion 6 (Type-B error)**.** *An error $\boldsymbol{E}$ is a type-B error, if it has the representation $\boldsymbol{E} = W_{n_0} \dots W_{n_{w-1}} = W^i_{o_0} \dots W^i_{o_{w-1}}$, and all $n_0, \dots, n_{w-1} < N_1 N_2$. In other words, the indices of all non-identity entries (1) have to be associated to columns in the left part of $\boldsymbol{S}$ and (2) have to have the same inner index i.*

All non-identity entries of a type-B error lie in the same row i of the VN grid, since they all share the same inner index. Suppose that all non-identity entries of a type-B error $\boldsymbol{E}$ are Z entries, i.e., $\boldsymbol{E} = Z^{o_0}_i \dots Z^{o_{w-1}}_i$. Such an error commutes with the lower half of $\boldsymbol{S}$ and only induces a non-trivial syndrome in X checks positioned in row i of the CN grid.

Example 12. Again, consider the symmetric HP code from Example 11. Figure 3.11 shows a homogeneous type-B error, where all (potential) non-identity entries are Z entries and lie in row $i = 0$. It can be easily seen that such a type-B error only induces a non-trivial syndrome for X checks in the corresponding row (in the case of Fig. 3.11, row $i = 0$).

Connection to Binary Belief Propagation on the Classical Code

There exists a connection between the decoding behaviour of BP$_4$ on homogeneous type-A and type-B errors and the decoding behaviour of BP$_2$ on the classical code. To understand this, consider a homogeneous type-A error whose X entries share the same outer index o. The syndrome in the Z checks with outer index o is identical to the syndrome that a binary error $\boldsymbol{e} = (e_0, \dots, e_{N_2-1})$ with

$$e_i = \begin{cases} 1 & \text{if} \quad E^i_o = X \\ 0 & \text{if} \quad E^i_o = I \end{cases}$$

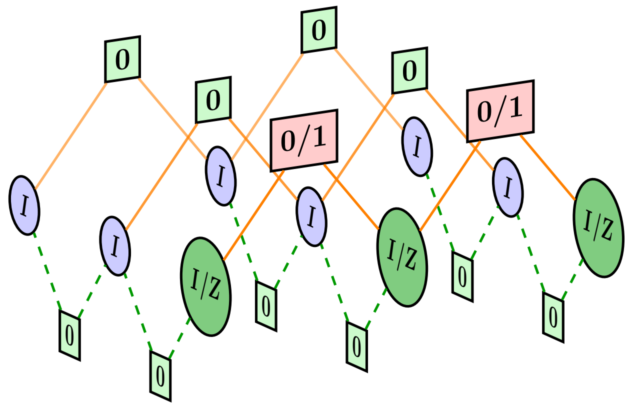
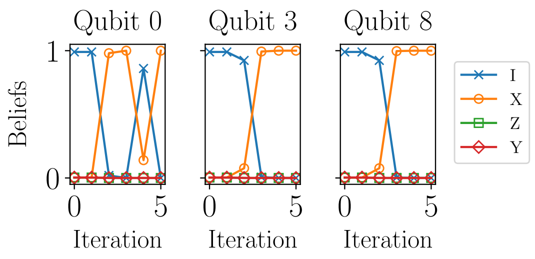


Figure 3.11: A type-B error with Z entries only in one row i only induces a non-trivial syndrome in the corresponding CN row in the upper half of $\boldsymbol{S}$.

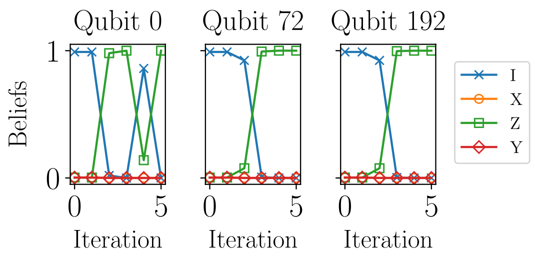
yields when multiplied with the PCM $\boldsymbol{H}_2$. This is true since VNs and CNs in column o are connected according to the Tanner graph $\mathcal{T}(\boldsymbol{H}_2)$. Hence, the exchanged messages within the subgraph corresponding to column o will have a strong resemblance to the messages exchanged in a BP$_2$ decoder, operating on the Tanner graph $\mathcal{T}(\boldsymbol{H}_2)$ and initialized with the syndrome $\boldsymbol{H}_2\,\boldsymbol{e}$.
While the exchanged messages will not be identical (as the subgraph is not isolated from extrinsic messages, e.g., from X checks in column o in the upper half), incoming messages to VNs in column o are likely signal to these VNs that they commute with X (i.e., I entries and X entries are assigned high probabilities by the upper half). This is true since all checks outside of the subgraph are satisfied and hence likely to agree on I entries for all VNs not in column o. As a consequence, the upper X checks will likely "vote" for I or X entries on VNs in column o. As X checks cannot differentiate between I or X (since they both commute with X), the "binary" choice between I and X is determined by messages exchanged in the subgraph $\mathcal{T}(\boldsymbol{H}_2)$ in the lower half.
Since the decoding behaviour of BP$_4$ on the HP code is related to BP$_2$ on the classical code, it seems natural to choose a classical code were BP$_2$ performs well. We elaborate on the choice of a good classical code in Sec. 4.3.1.

While we described the connection to BP$_2$ on $\mathcal{T}(\boldsymbol{H}_2)$ based on type-A errors, the same effect occurs on $\mathcal{T}(\boldsymbol{H}_1)$ for type-B errors. In fact, for the special case of symmetric HP codes, we can establish a direct correspondence between the decoding behaviour of parallel BP$_4$ on both error types. Thus, it suffices to perform error analysis for one of the two error types. This is due to the fact that these error types are related via $\boldsymbol{E}_{\text{type-B}} = f(\pi_{\text{c}}(\boldsymbol{E}_{\text{type-A}}))$, as defined in Theorem 1.
This connection is particularly clear for homogeneous type-A and type-B errors (although the errors are not required to be homogeneous to show equivalent decoding behaviour): The homogeneous type-A error $\boldsymbol{E}_{\text{type-A}} = X_o^{i_0} \dots X_o^{i_{w-1}}$ is connected to the homogeneous type-B error $\boldsymbol{E}_{\text{type-B}} = f(\pi_{\text{c}}(\boldsymbol{E}_{\text{type-A}})) = Z_{i_0}^{o} \dots Z_{i_{w-1}}^{o}$, as f interchanges X and Z entries and π_{c} interchanges outer and inner indices. Parallel BP$_4$ thus shows equivalent decoding behaviour on $\boldsymbol{E}_{\text{type-A}}$ and $\boldsymbol{E}_{\text{type-B}}$.

Example 13. We consider the decoding behaviour over the Tanner graph of the symmetric $[[900, 36]]$ HP code, constructed from a classical $[24, 18]$ code. Let $\boldsymbol{E}_a = X_0 X_3 X_8 = X_0^0 X_0^3 X_0^8$ be a weight-3 type-A error. The corresponding type-B error is $\boldsymbol{E}_b = Z_0^0 Z_3^0 Z_8^0 = Z_0 Z_{72} Z_{192}$. From Fig. 3.12 we can see that both errors show equivalent decoding behaviour when decoded with parallel BP$_4$, up to a permutation of VNs and interchanging X and Z beliefs.



(a) Belief history for $\boldsymbol{E}_a = X_0^0 X_0^3 X_0^8$.



(b) Belief history for $\boldsymbol{E}_b = Z_0^0 Z_3^0 Z_8^0$.

Figure 3.12: Belief histories for a pair of type-A and type-B errors, using parallel BP$_4$ decoding.

4. Update Schedule

4.1. Parallel and Serial Update Schedule

The update schedule determines the order, in which messages are passed along the edges of the Tanner graph. In the conventional *parallel* schedule (also referred to as *flooding* schedule) [KFL01], all VNs are updated in parallel. This step is referred to as *vertical step* in Algorithm 1. Subsequent to this, all CNs are updated in parallel in the *horizontal step*. While this schedule is widely used in the literature, Zhang and Fossorier showed that a *serial* update schedule outperforms the parallel schedule in terms of convergence rate [ZF05]. Subsequent theoretical and empirical results indicate that a parallel schedule needs twice the number of iterations compared to a serial schedule [SLG07, CGW10, ZF05].
In contrast to the parallel schedule, "the [vertical step and horizontal step] are interleaved", as stated by [SLG07]. While a serial update along both, CNs and VNs is possible [SLG07], we restrict ourselves to a serial update along the VNs: During the update of VN n, we compute the incoming messages $\Delta_{m\to n}$ for $m \in \mathcal{M}(n)$. We then update all outgoing variable-to-check messages $\lambda_{S_{mn}}(\mathbf{\Gamma}_{n\to m})$ for $m \in \mathcal{M}(n)$ and proceed to the next VN. One iteration is finished once all VNs were updated. The serial BP$_4$ in log-domain is summarized in Algorithm 2.

Algorithm 2 Serial Λ-rule BP$_4$

```
    Input:
1:  $\boldsymbol{S} \in \{I, X, Y, Z\}^{M\times N}$, $\boldsymbol{z} \in \{0,1\}^M$, LLR vectors $\{\mathbf{\Lambda}_n \in \mathbb{R}^3\}_{n=0}^{N-1}$, $\ell_{\max}$
    Initialization: See lines 2-5 in Algorithm 1
    Serial Update:
2:  for $n = 0, \dots, N-1$ do
3:      for $m \in \mathcal{M}(n)$ do
4:          Compute $\Delta_{m\to n} = (-1)^{z_m} \underset{n' \in \mathcal{N}(m)\backslash n}{\boxplus} \lambda_{S_{mn'}}(\mathbf{\Gamma}_{n'\to m})$
5:      for $m \in \mathcal{M}(n)$ do
6:          $\Gamma^W_{n\to m} = \Lambda^W_n + \sum_{m' \in \mathcal{M}(n)\backslash m} \mathbb{1}\{\langle S_{m'n}, W\rangle = 1\}\, \Delta_{m'\to n}$    for $W \in \{X, Y, Z\}$
7:          Compute $\lambda_{S_{mn}}(\mathbf{\Gamma}_{n\to m})$
    Hard Decision: See lines 11-21 in Algorithm 1, except goto Serial Update instead
    of goto Horizontal Step
```

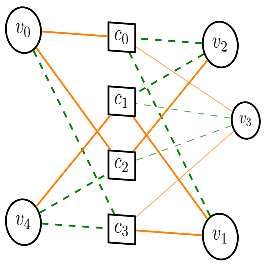
The number of message updates per iteration is identical for parallel BP$_4$ and serial schedule BP$_4$ [KL20]. However, for the parallel schedule it is possible to parallelize the computations for different nodes in both, vertical and horizontal step. If appropriate hardware resources are available, one iteration in a parallel schedule may be performed significantly faster than in a serial schedule. Thus, a distinction between convergence speed in terms of the number of iterations and in terms of computation time has to be drawn. Depending on the available hardware, a parallel schedule might converge faster in terms of computation time, although more iterations have to be performed. However, "the serial schedule can be partially parallelized", as described by Sharon et al. [SLG07]. This can be achieved by partitioning the sequence of traversed VNs into subsets "such that no two variable nodes in a subset are connected to the same check node" [SLG07].

Besides the difference in convergence speed, another aspect is the decoding performance. Kuo and Lai show that a serial schedule outperforms a parallel schedule on the $[[129, 28, 3]]$ HP code [KL20]. We replicate their results and then compare the decoding performance of parallel BP_4 and serial BP_4 on symmetric HP codes. We show that serial BP_4 decoding is less prone to message overestimation and demonstrate how this leads to a performance gap between the two schedules.
To demonstrate how a serial schedule can increase the decoding performance, we recapitulate a small example provided in [KL20]. In addition to [KL20], we elaborate on the symmetric constellation in the stabilizer matrix $\boldsymbol{S}$ that causes parallel BP_4 to fail.

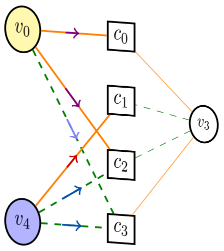
Example 14. Consider the well-known $[[5, 1, 3]]$ code [LMPZ96] with the stabilizer matrix

$$\boldsymbol{S} = \begin{pmatrix} X & Z & Z & X & I \\ I & X & Z & Z & X \\ X & I & X & Z & Z \\ Z & X & I & X & Z \end{pmatrix}.$$

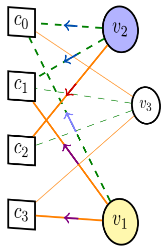
While this code corrects all weight-1 errors, parallel BP_4 fails to converge for the error Y_3 due to a symmetric topology in the Tanner graph $\mathcal{T}(\boldsymbol{S})$, depicted in Fig. 4.1.



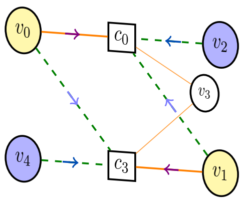
(a) Tanner graph $\mathcal{T}(\boldsymbol{S})$.



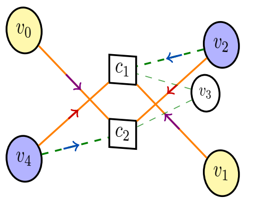
(b) Left subgraph of $\mathcal{T}(\boldsymbol{S})$.



(c) Right subgraph of $\mathcal{T}(\boldsymbol{S})$.



(d) CN c_0 receives the same messages as CN c_3.



(e) CN c_1 receives the same messages as CN c_2.

Figure 4.1: The Tanner graph $\mathcal{T}(\boldsymbol{S})$ contains a symmetric topology, causing parallel BP_4 to fail for the error $\boldsymbol{E} = Y_3$

The Tanner graph $\mathcal{T}(\boldsymbol{S})$ can be partitioned into two isomorph subgraphs, depicted in Fig. 4.1b and Fig. 4.1c. We marked VNs that can be associated with each other with the same color. Assuming a symmetric depolarizing channel, v_0 outputs the same messages to $[c_0, c_2, c_3]$ that v_1 outputs to $[c_3, c_1, c_0]$ after the initialization step. Analogously, v_4 outputs

the same messages to $[c_1, c_2, c_3]$ that v_2 outputs to $[c_2, c_1, c_0]$. Thus, c_0 receives the same messages as c_3 (Fig. 4.1d).
Now assume that all checks are in the same state. This is the case for $\boldsymbol{E} = Y_3$, as this error anti-commutes with every row of $\boldsymbol{S}$. Consequently, c_0 outputs the same messages to $[v_0, v_1, v_2]$ that c_3 outputs to $[v_1, v_0, v_4]$. Analogously, c_1 outputs the same messages to $[v_2, v_4, v_1]$ that c_2 outputs to $[v_4, v_2, v_0]$ (Fig. 4.1e).

The symmetric topology in $\mathcal{T}(\boldsymbol{S})$ in conjunction with the parallel update rule causes the beliefs to oscillate, as shown in the first row in Fig. 4.2. However, a serial BP$_4$ decoder converges after only four iterations, as shown in the second row.

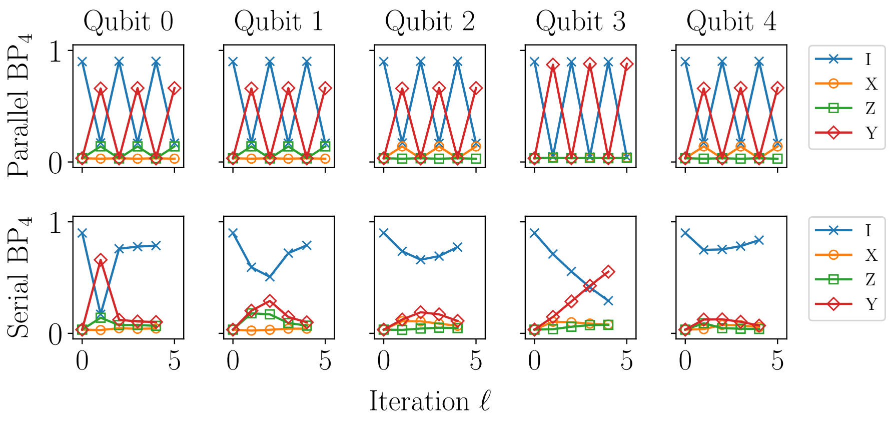


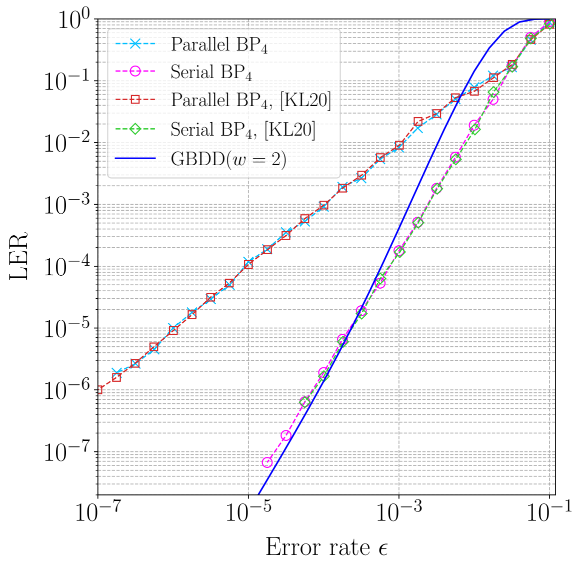
Figure 4.2: Trajectories of the estimated probabilities when decoding the $[[5, 1, 3]]$ code with error Y_3 and an initial $\epsilon_0 = 0.1$.

4.2. [[129, 28, 3]] Hypergraph Product Code

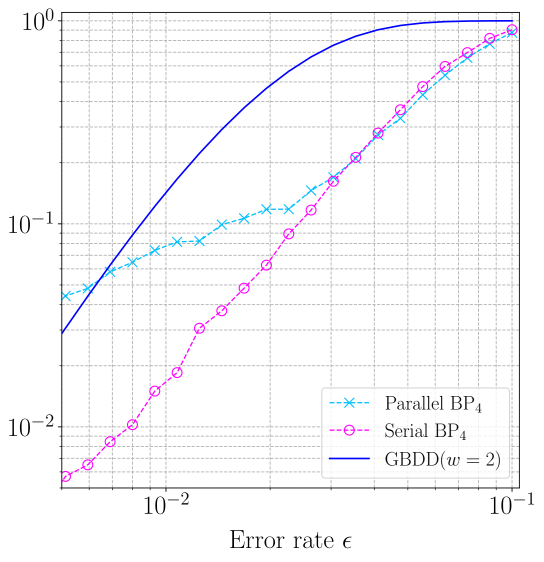
We now show the LER curves for both, parallel and serial BP$_4$ decoder on the $[[129, 28, 3]]$ HP code when initializing ϵ_0 with the true channel parameter ϵ. Like in [KL20], we choose a maximum number $\ell_{\max}$ of twelve iterations. Our analysis confirms that the decoding performance doesn't improve significantly after twelve iterations. As a reference, we redraw the performance curves given in [KL20], as well as the GBDD performance with $\gamma_0 = 1$, $\gamma_1 = 1$ and $\gamma_2 = 98.73\%$, as derived in Sec. 3.1 and described in [LK21].

Figure 4.3a shows that for small ϵ, serial BP$_4$ clearly outperforms parallel BP$_4$. The performance gap between parallel BP$_4$ and serial BP$_4$ grows as ϵ becomes smaller. However, Fig. 4.3b shows that for $\epsilon > 0.03$, both decoders perform comparably. The increasing performance gap indicates that parallel BP$_4$ fails to converge for low weight errors where serial BP$_4$ converges successfully.

This effect can be attributed to message overestimation: If ϵ_0 is initialized with a small value, the message reliability is high, causing parallel BP$_4$ to jump to a wrong higher weight solution for some weight-1 errors. In the following, we demonstrate this effect by means of an example.



(a) LER curves for a wide range of ϵ. We also redraw the curves presented in [KL20].



(b) LER curves for a smaller range of ϵ.

Figure 4.3: LER curves for parallel and serial BP$_4$, initializing ϵ_0 with the channel parameter ϵ for each data point. For comparison, we also plot the performance of a GBDD$(w = 2)$.

Example 15. Consider the weight-1 error $\boldsymbol{E} = Z_{60} = Z_4^0$. According to Fig. 3.6, errors of weight one are likely generated by a channel with $\epsilon = 10^{-2}$. However, when we use $\epsilon_0 = 10^{-2}$, parallel BP$_4$ converges to $\boldsymbol{F} = Z_{30}Z_{45}Z_{60}Z_{75} = Z_2^0Z_3^0Z_4^0Z_5^0$, as shown in the first row of Fig. 4.4. Since $\boldsymbol{FE} = Z_{30}Z_{60}Z_{75} = Z_2^0Z_4^0Z_5^0 \in N(\mathcal{S}) \setminus \mathcal{S}$, the decoder terminates due to a matched syndrome, but $\boldsymbol{F}$ doesn't correct $\boldsymbol{E}$ and decoding results in a failure. Serial BP$_4$ initialized with $\epsilon_0 = 10^{-2}$ converges to the correct solution after two iterations.

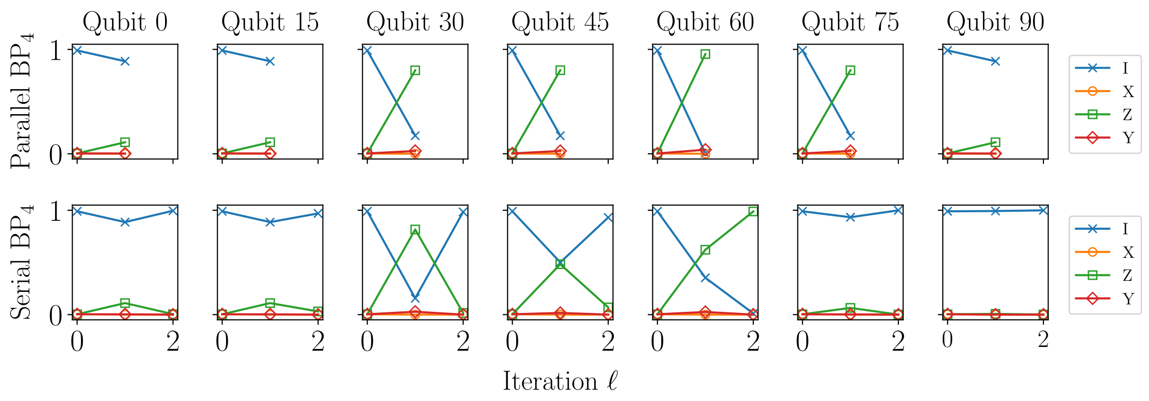


Figure 4.4: Trajectories of the estimated probabilities when decoding the error $\boldsymbol{E} = Z_{60} = Z_4^0$ on the $[[129, 28]]$ code with an initial $\epsilon_0 = 0.01$.

We elaborate on details regarding the quantity of such failures due to message overestimation in the next chapter.

4.3. Symmetric Hypergraph Product Codes

We now compare parallel BP$_4$ and serial BP$_4$ on symmetric HP codes. In this chapter, we focus on the $[[900, 36, 10]]$ HP code, constructed from a regular $[24, 6, 10]$ LDPC for reasons of simplicity. We provide results for the $[[400, 16, 6]]$ HP code and the $[[625, 25, 8]]$ HP code in Appendix C.

4.3.1. Influence of the Classical Code

We observed that the performance gap between parallel BP$_4$ and serial BP$_4$ strongly depends on the choice of the classical PCM used to construct the symmetric HP code. Recall from Section 3.5.3 that BP$_4$ decoding on the HP is related to BP$_2$ on the corresponding classical code. We find a good classical code $\mathcal{C}_\mathrm{C}$ by randomly sampling a sufficiently large number (in our case 10000) of PCMs and selecting the sample which maximizes a measure for suitability w.r.t. both, parallel and serial decoding. To assess the quality of a code, we choose the number of successfully decoded weight-3 errors by BP$_2$ on it's Tanner graph, as we observed that a large fraction of decoding failures of the corresponding symmetric HP can be attributed to type-A or type-B errors of weight three. We remove cycles of length four in a code during the sampling process, since "these cycles appear in the HP codes, multiplying according to the size of the classical parity check matrix", as stated by Raveendran and Vasić [RV21].
For the $[[900, 36]]$ HP code, the classical code has blocklength $N_\mathrm{C} = 24$. Thus, there are $\binom{24}{3} = 2024$ weight-3 errors, which we decode with a BP$_2$ decoder to assess the suitability of the classical code. Figure 4.5 shows a scatter plot of the number of successfully decoded weight-3 errors using parallel BP$_2$ and serial BP$_2$, as well as the marginalized distributions.

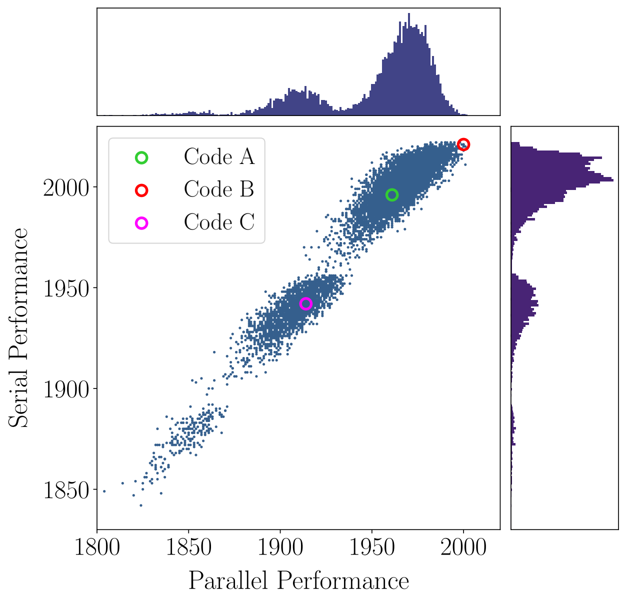
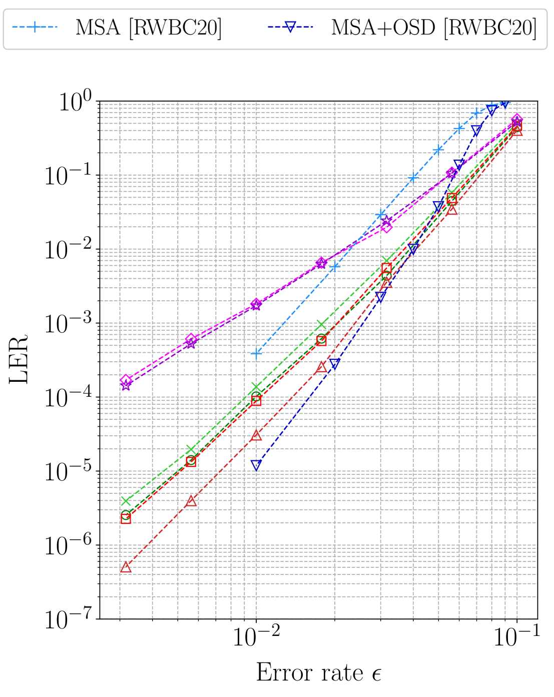


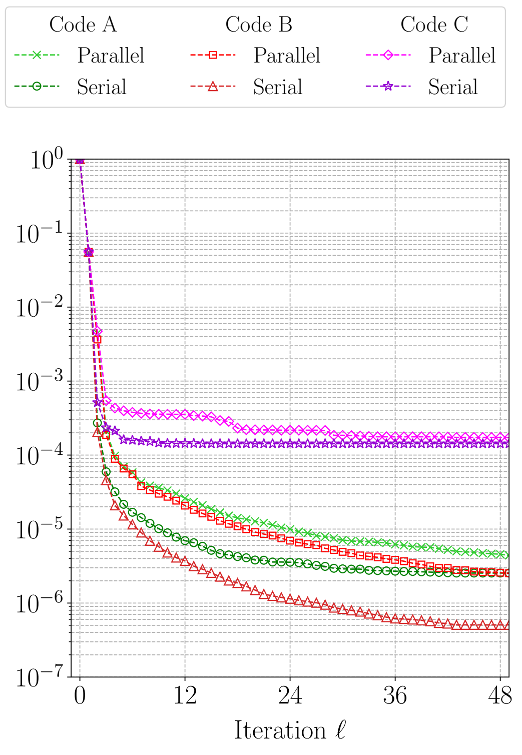
Figure 4.5: Number of decoding successes of BP$_2$ when decoding 2024 weight-3 errors on randomly sampled, regular $[24, 6, 10]$ LDPC codes with $d_\mathrm{v} = 3$ and $d_\mathrm{c} = 4$.

There is a clear correlation between parallel and serial decoding performance, i.e., a code that results in a larger number of decoding successes with a parallel schedule typically performs well under serial decoding as well. We observe similar decoding behaviour for the $[[400, 16]]$ HP code and the $[[625, 25]]$ HP code and provide the corresponding scatter plots in Fig. C.1a and Fig. C.1b, respectively.
From these 10000 PCMs, we choose three classical PCM to construct three HP codes, which we will denote by code A, code B and code C. Code A is constructed from a classical code with average performance, code B is constructed from a classical code with good performance and code C is constructed from a classical code with bad performance w.r.t. both, parallel and serial BP_2. The performances of BP_2 on the corresponding classical codes are marked in Fig. 4.5.

We now compare the performance curves of parallel BP_4 and serial BP_4 for the three codes described above. Again, we initialize ϵ_0 with the true channel parameter ϵ. We choose a maximum number of 100 iterations. Figure 4.6a shows the LER curves after $\ell_{\max} = 100$ iterations for code A, code B and code C.
As a reference, we redraw the results presented by Roffe et al. [RWBC20]. The authors compare a BP decoder without OSD to a decoder with OSD. The BP decoder without post-processing is implemented as two separate parallel BP_2 decoders for X and Z errors, as described in Sec. 2.6. Furthermore, the authors use the min-sum approximation (MSA) [HOP96] as well as message normalization.



(a) LER curves for three codes, constructed from different classical codes.



(b) LER as a function of the number of iterations ℓ for a channel parameter $\epsilon = 10^{-2.5}$.

Figure 4.6: Comparison of the decoding performances of BP_4 on three HP codes, constructed from different classical codes.

The performance gap between parallel BP$_4$ and serial BP$_4$ increases with the performance measure of the classical code: While there is no significant gap between the performances of both schedules for code C (constructed from a bad classical code), serial BP$_4$ outperforms parallel BP$_4$ on code B, especially for low error rates ϵ. Our serial BP$_4$ decoder approaches the performance of the BP decoder using an OSD post-processing step. However, a small gap remains in the low error rate region.
Figure 4.6b depicts the LER as a function of the number of iterations ℓ for the error rate $\epsilon = 10^{-2.5}$. Since we couldn't observe a significant improvement in decoding performance after 48 iterations, we only show the curves for $\ell \leq 48$. Note that the serial BP$_4$ converges faster than parallel BP$_4$ in terms of iterations for all codes, matching the expectation in [SLG07].
The gap between our implementation and the results in [RWBC20] for small ϵ indicate that there is still some room for improvement, especially for the parallel schedule. We investigate the effect of message normalization on the decoding performance in the following chapter.

4.3.2. Errors Causing Decoding Failure

When inspecting the properties of errors which lead to decoding failure, we observed that a majority of them contain more than two entries on the same inner index or the same outer index in the left part of $\boldsymbol{S}$. To demonstrate this, we sample errors of fixed weight and decode them with both, a parallel and a serial schedule. We choose a fixed weight of nine, since this facilitates the categorization of errors and speeds up the simulation time: It is unlikely to randomly sample a harmful subset of, e.g., three qubits, when the error is chosen to be non-identity at exactly three positions. When a larger number of qubits is chosen to be non-identity, a harmful subset is sampled with higher probability. We compute the initial beliefs $\{\boldsymbol{\Lambda}_n\}_{n=0}^{N-1}$ using an initial value $\epsilon_0 = 10^{-2}$, as this error rate maximizes the likelihood to obtain a weight-9 error. We sample and decode errors, until 100 decoding failures are collected.

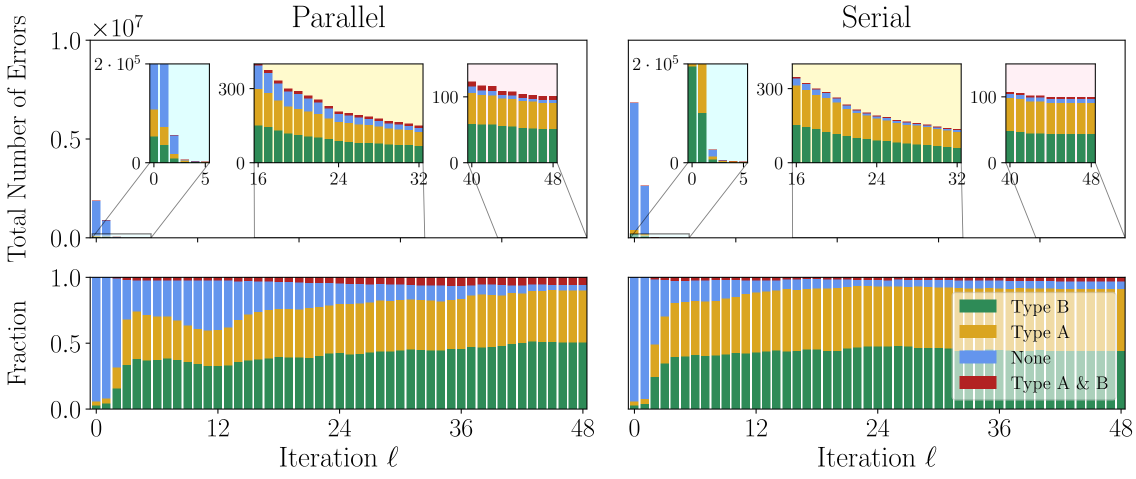


Figure 4.7: Total number (top row) and fraction (bottom row) of decoding failures after ℓ iterations, categorized into four different types. The four types indicate whether the corresponding failures can be attributed to subsets constituting a type-A error, a type-B error, both or none.

The first row in Fig. 4.7 shows the absolute number of decoding failures on code B as a function of iterations for both, parallel BP_4 and serial BP_4. In the second row in Fig. 4.7, we scale the height of each bar by the absolute number of failures in the corresponding iteration. This yields a ratio, where we differentiate between four different error types: If an error contains three or more entries on the same outer index, we assign it to the category "Type A". While such an error is no type-A error in the sense of Definition 5, we observed that for these errors, the decoding failure can be attributed to the subset of entries that constitutes a type-A error. This behaviour is demonstrated in the following example.

Example 16. Consider the weight-9 error $\boldsymbol{E} = X_{50}X_{52}X_{70}Z_{339}Y_{398}Y_{445}Z_{452}X_{552}X_{564}$. Figure 4.8 shows that BP_4 decoding succeeds to correctly identify all entries except for the subset $X_{50}X_{52}X_{70} = X_2^2X_2^4X_2^{22}$, which constitutes a type-A error of weight-3.

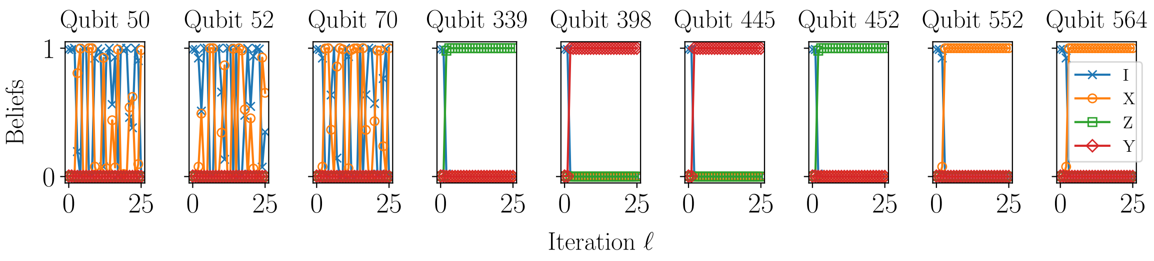


Figure 4.8: The decoding failure of BP_4 can be attributed to a subset of entries that forms a type-A error.

Analogously, an error is categorized as "Type B" if it contains three or more entries on the same inner index. Since the sampled errors are of weight nine, they can contain both, type-A and type-B errors as subsets. If this is the case, the corresponding error is categorized as "Type A & B". If an error contains neither a type-A nor a type-B error, we assign the categorized "None".
Prior to decoding, errors of type "None" constitute over 95% of all sampled errors. After 48 iterations, almost all failure causing errors contain subsets that constitute type-A or type-B errors of weight three or larger.

Further experiments show that nearly all of the failure causing subsets constitute homogeneous type-A and type-B errors. In other words, typically all entries in a type-A subset are X entries. Similarly, typically all entries in a type-B subset are Z errors. To confirm this observation experimentally, we sample and decode weight-3 errors of one type. We categorize each error as "Homogeneous" or "Not Homogeneous" and plot the fraction of decoding failures over the number of iterations in Fig. 4.9.
The a-priori probability for a weight-3 type-A error to consist of only X entries is $(1/3)^3 = 1/27 \approx 3.7\%$, assuming a depolarizing channel. This is coherent with the height of the bar at zero iterations. After 48 iterations, nearly all decoding failures can be attributed homogeneous errors for both, parallel BP_4 and serial BP_4. Type-B errors show identical behaviour, which can be explained by the symmetry of type-A and type-B errors, described in Theorem 1.

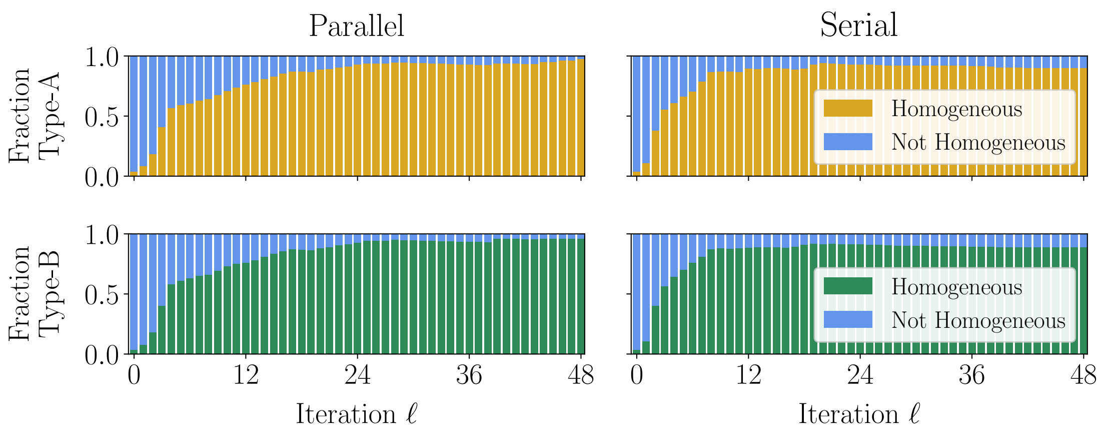


Figure 4.9: Fraction of decoding failures after ℓ iterations, categorized into "Homogeneous" and "Not Homogeneous".

As described in Sec. 3.5.3, failures on homogeneous type-A and type-B errors can be attributed to the decoding behaviour of BP_2 on the corresponding classical code. We conclude this chapter by confirming the above statement experimentally and showing that the choice of the classical code does affect the performance on homogeneous type-A and type-B errors and hence, the general decoding performance.
Table 4.1 shows the number of decoding successes on all homogeneous type-A and type-B errors of weight three in the left part of $\boldsymbol{S}$ for both, parallel BP_4 and serial BP_4 and for code A, code B and code C. Recall from Sec. 4.3.1 that there are 2024 weight-3 errors for a classical code with $N_C = 24$ and thus, the same number of weight-3 type-A/type-B errors for a given column/row in the 24×24 VN grid. Since these errors can occur in each of the N_C columns/rows, the total number of homogeneous weight-3 type-A/type-B errors is $24 \cdot 2024 = 48576$.

Table 4.1: Number of decoding successes for all homogeneous weight-3 type-A and type-B errors when decoding with parallel BP_4 and a serial BP_4 (and $\ell_{\max} = 100$) on code A, code B and code C.

		Code A	Code B	Code C
Type-A	Parallel BP_4	46920	47995	45844
	Serial BP_4	47755	48422	46613
Type-B	Parallel BP_4	46920	47995	45844
	Serial BP_4	47756	48421	46614

The largest number of successes is achieved on code B for both, parallel BP_4 and serial BP_4. This matches our expectation, as code B was constructed from a classical code with good BP_2 performance, whereas BP_2 showed average and bad performance on code A and code C, respectively. Note that the number of successes for type-A and type-B errors are identical when decoding with parallel BP_4, confirming Theorem 1 and it's applicability to type-A and type-B errors.

5. Fixed Initialization and Message Normalization

5.1. Fixed Initialization

Recall from Fig. 4.3b that small values for ϵ_0 cause message overestimation for the parallel BP$_4$ decoder. Kuo and Lai propose to use *fixed initialization* [LK21], i.e., to initialize the prior beliefs with a fixed value ϵ_0, independent of the true channel parameter ϵ. It has been shown for classical BP that fixed initialization can be beneficial in terms of decoding performance and/or convergence speed [HFI10]. Additionally, if ϵ is unknown, channel parameter estimation can be omitted when using fixed initialization. Naturally, this leaves us with the task of choosing a proper ϵ_0, which ensures good decoding performance for a variety of error rates ϵ.
As Fig. 4.3b shows that parallel BP$_4$ and serial BP$_4$ result in nearly identical performance for a $\epsilon > 0.03$, and a performance gap develops for $\epsilon < 0.03$, we choose to compare both schedules with fixed initialization for $\epsilon_0 \in \{10^{-2}, 10^{-1}\}$. The resulting performance curves are shown in Fig. 5.1.

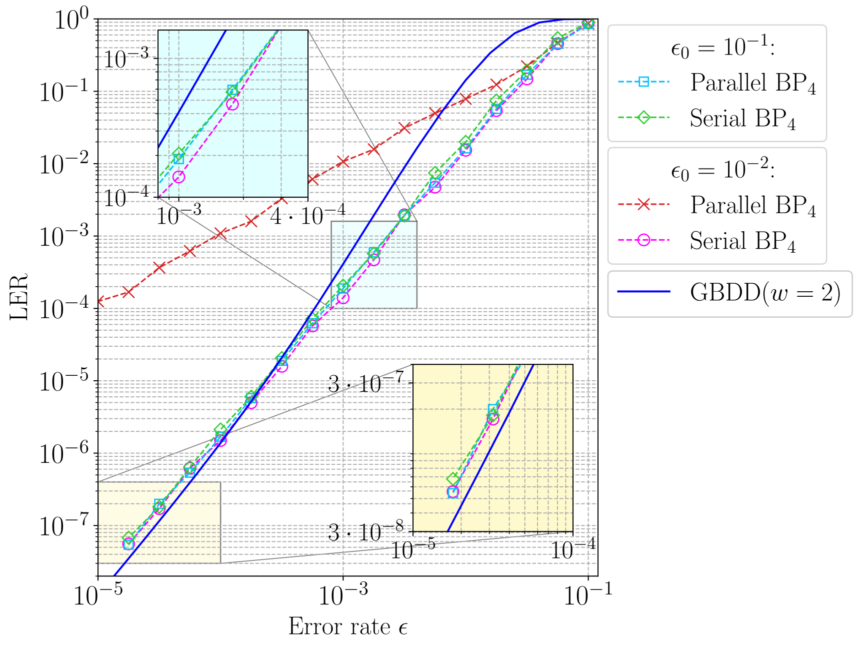


Figure 5.1: LER curves for parallel and serial BP$_4$, initializing ϵ_0 with a fixed value $\epsilon_0 \in \{10^{-2}, 10^{-1}\}$.

While serial BP$_4$ performs close to the GBDD independent of ϵ_0, the performance of parallel BP$_4$ strongly depends on the value of ϵ_0. As mentioned in Chapter 4, the high LER of parallel BP$_4$ can be attributed to the poor performance on low weight errors. Tab. 5.1 lists the number of successfully decoded weight-1 and weight-2 errors for both, parallel and serial BP$_4$ with fixed initialization.

Table 5.1: Number of corrected weight-1 and weight-2 errors for parallel and serial BP_4 decoding with different initial values ϵ_0.

		Weight-1	Weight-2			
			Type-1	Type-2	Type-3	$\sum$
Number of Errors		387	72729	1260	315	74304
Number of Correctable Errors		387	72729	630	0	**73359**
Number of Successes						
$\epsilon_0 = 10^{-2}$	Parallel BP_4	357	65220	224	0	65444
	Serial BP_4	387	72477	198	0	72675
$\epsilon_0 = 10^{-1}$	Parallel BP_4	387	72687	183	0	72870
	Serial BP_4	387	72475	100	0	72575

To demonstrate the effect of message overestimation quantitatively, consider the distribution of iterations until termination when decoding weight-1 errors, depicted in Fig. 5.2. Each bar is divided into two parts where the red part indicates the number of decoding failures after the corresponding number of iterations. For all weight-1 errors, all decoders converge after a maximum number of 3 iterations. It can be seen that parallel BP_4 with $\epsilon_0 = 10^{-2}$ converges faster than the other decoders. However, a significant fraction of decoding attempts result in a decoding failure after one iteration. This happens when the decoder converges to a solution $\boldsymbol{F}$ that is related to the true error $\boldsymbol{E}$ by an element $N(\mathcal{S}) \setminus \mathcal{S}$, as described in Example 15 in the previous chapter.

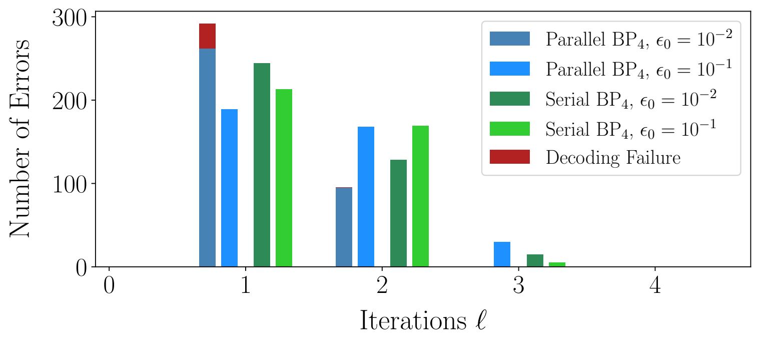


Figure 5.2: Distribution of iterations until termination for weight-1 errors, using parallel and serial BP_4 decoding with different ϵ_0.

Choosing a larger ϵ_0 leads to a reduction of message reliability in the initial step, alleviating the effect of message overestimation. As a consequence, the performance of the parallel BP_4 decoder is significantly improved for $\epsilon_0 = 0.1$. On the other hand, serial BP_4 decoding appears to be agnostic to the choice of ϵ_0 on the $[[129, 28]]$ HP code.

Let us now address the performance difference of BP_4 decoding and GBDD decoding for weight-2 errors, starting with type-1 errors. By definition , a GBDD is able to correct all type-1 errors within it's correction radius. As shown in Tab. 5.1, parallel BP_4 with $\epsilon_0 = 10^{-2}$ results in a decoding failure for a large fraction of type-1 errors, while the other decoders succeed for most of them. Again, this performance difference can be attributed to message overestimation. We provide the corresponding distribution of iterations in Fig. C.3.

Note that all decoders fail for a large fraction of type-2 errors. The bad performance can be attributed to split beliefs, which we described in Sec. 3.5.2. As type-2 errors share a syndrome with another error of equal weight, the decoder attempts to converge to multiple errors simultaneously, resulting in an invalid superposed solution, which doesn't match the syndrome. We provide a more detailed analysis of this effect in the next chapter and propose a simple post-processing procedure to alleviate this decoding issue for the $[[129, 28]]$ HP code.

5.2. Message Normalization

Another approach to alleviate the effect of message overestimation is to normalize the exchanged messages, as proposed by Yazdani et al. [YHB04]. In conventional BP$_4$ decoding, message normalization is not possible, since the messages constitute four-dimensional probability distributions. However, since Kuo and Lai's refined BP$_4$ algorithm only exchanges scalar messages, the application of message normalization is facilitated [KL20]. The authors report a significant performance improvement for bicycle codes, a class of codes proposed by MacKay et al. [MMM04]. Note that message normalization has a similar effect to fixed initialization, as it aims to reduce the message reliability [KL20].
In our work, we follow the approach of Kuo and Lai and apply message normalization in conjunction with a BP$_4$ that exchanges scalar messages. We investigate the effect of message normalization on the decoding performance using symmetric HP codes. As pointed out by Yazdani et al. and demonstrated for quantum codes by Kuo and Lai, normalization can be performed by multiplication of variable-to-check messages by a scalar α_v, as well as a multiplication of check-to-variable messages by a scalar α_c [YHB04, KL20]. As both methods result in similar performance [KL20], we choose to normalize the check-to-variable messages with the scalar α_c.

5.2.1. Variable Message Normalization

Kuo and Lai choose α_c to be constant during decoding and compare different values α_c on bicycle codes [KL20]. Their results indicate that the choice of an optimal α_c (in the sense of minimizing the LER) depends on the error rate ϵ. The authors state that "stronger messages (smaller ϵ) would need larger suppression (larger α_c [...]). It is possible to choose different α_c [...] for different ϵ to achieve a better performance" [KL20]. However, Kuo and Lai don't provide further insights on how to establish the connection between the channel parameter ϵ and α_c.
Our goal is to develop a normalization strategy that optimizes α_c w.r.t. decoding performance without the need for prior channel parameter estimation. The first step towards an optimal choice of α_c without the availability of channel information is to use fixed initialization. Simulation results in the following are conducted with fixed initialization, where we choose $\epsilon_0 = 10^{-2}$. Additionally, we propose to circumvent the obstacle of choosing α_c based on ϵ by using a variable α_c: Literature on message overestimation due to the MSA suggests that an exponential increase of α_c might increase the decoding performance [LS06, EE14]. Indeed, we observed a performance improvement when using a variable scalar instead of a constant one, as demonstrated in the following sections.

To alleviate message overestimation due to application of the MSA, Emran and Elsabrouty propose the function

$$\alpha_c = 1 - 2^{\lceil \ell/S \rceil}$$

and define $\ell = 1$ as the first iteration. By using the term $\lceil \ell/S \rceil$ in the exponent, the function can be implemented efficiently by a shift and a subsequent subtraction [EE14]. The sequence of scaling factors always starts with $\alpha_c = 0.5$. In contrast we introduce two parameters, a and b, which enable us to choose an initial value for α_c and hence control the trajectory more precisely. Our function has the form

$$\alpha_c(a,\, b,\, \ell) = 1 - (1 - a) \cdot 2^{-b \cdot \ell}, \tag{5.1}$$

where we start at $\ell = 0$. The trajectory of $\alpha_c(a,\, b,\, \ell)$ is depicted in Fig. 5.3 for different values (a, b). The parameter a controls the starting value for α_c, i.e., $\alpha_c(a, b, \ell = 0) = a$. The parameter b controls the speed with which α_c converges to the final value of one, as demonstrated in the following sections. Note that, since $\alpha_c(a, b = 0, \ell) = a$, we obtain the normalization with a constant $\alpha_c = a$ when b is set to zero. We can thus take into account message normalization with a constant α_c as a special case of Eq. 5.1.

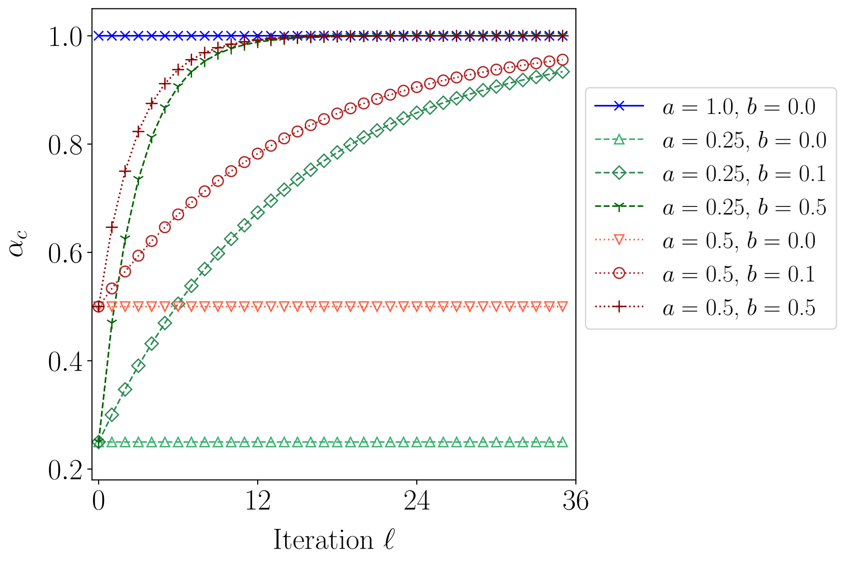
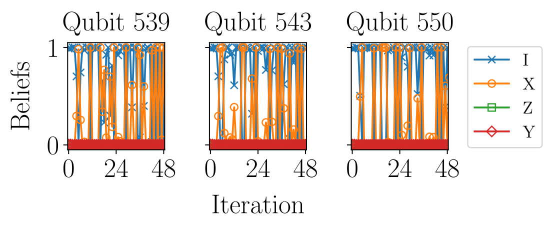


Figure 5.3: Trajectories of $\alpha_c(a, b, \ell)$ for different parameters (a, b).

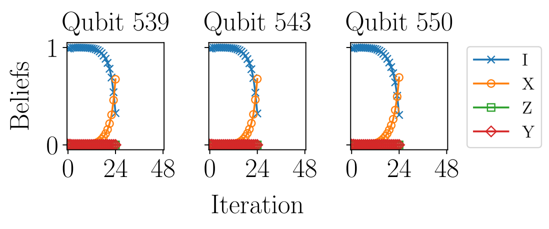
5.2.2. Optimization Normalization Parameters

Applying VMN leaves us with the task of choosing the parameters a and b, such that an objective is optimized. Depending on the choice of this objective, different parameters might be optimal: We observed a trade-off between convergence speed and decoding performance in terms of the LER. To understand this, consider the following example.

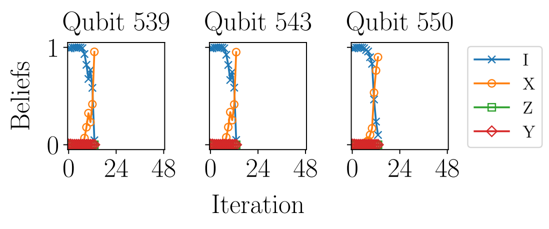
Example 17. Suppose we decode the error $\boldsymbol{E} = X_{539}X_{543}X_{550} = X_{22}^{11}X_{22}^{15}X_{22}^{22}$ on the $[[900, 36]]$ HP code. $\boldsymbol{E}$ constitutes a type-A error of weight three, as all non-identity entries have the same outer index. In the following, we decode with parallel BP$_4$ for a maximum number of 48 iterations and show the corresponding decoding behaviour for different normalization parameters.
Figure 5.4a shows that parallel BP$_4$ without normalization fails on this error due to message overestimation. However, choosing the normalization parameters $(a, b) = (0.25, 0.1)$, the decoder successfully converges after 23 iterations (see Fig. 5.4b). With $(a, b) = (0.25, 0.25)$, the decoder successfully converges after 12 iterations (see Fig. 5.4c), indicating that a larger b increases the convergence speed. However, when b is chosen too large (i.e., $(a, b) = (0.25, 0.5)$), the decoder runs into overestimation again (see Fig. 5.4d).



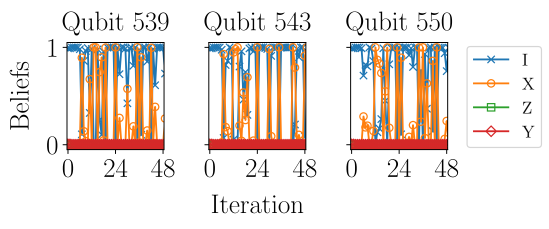
(a) Parallel BP$_4$ with no normalization.



(b) Parallel BP$_4$ with $a = 0.25$ and $b = 0.1$.



(c) Parallel BP$_4$ with $a = 0.25$ and $b = 0.25$.



(d) Parallel BP$_4$ with $a = 0.25$ and $b = 0.5$.

Figure 5.4: Belief histories when decoding the type-A error $\boldsymbol{E} = X_{539}X_{543}X_{550} = X_{22}^{11}X_{22}^{15}X_{22}^{22}$ with a parallel BP$_4$ with $\epsilon_0 = 10^{-2}$ and different normalization parameters.

The above example demonstrates that a compromise has to be struck, which we will refer to as the *overestimation-underestimation-trade-off*: b has to be chosen small enough to prevent overestimation. On the other hand, a small b reduces the convergence speed. Recall that quantum states decohere quickly and fast decoding is necessary, potentially limiting the maximum number of iterations $\ell_{\max}$. A small b lead may to decoding failures if the decoder is not able to converge within $\ell_{\max}$ iterations. Depending on $\ell_{\max}$, different parameters might be optimal. Naturally, the parameters a and b have to be chosen jointly, as they are closely connected. A large initial value for α_c might require a slower increase (i.e., a smaller b) than a smaller initial value.
We search for good normalization parameters by comparing the decoding performances for different a and b at different error rates ϵ. Additionally, we choose different $\ell_{\max}$ to account for the overestimation-underestimation-trade-off. In this thesis, we only perform this analysis for parallel BP$_4$, as simulations show that the performance of serial BP$_4$ on symmetric HP codes is not improved by applying VMN.

Figure 5.5 shows the decoding performance of parallel BP$_4$ on the [[900, 36]] HP code as a function of b, where $b \in \{0, 0.025, 0.05, 0.1, 0.25, 0.5, 1\}$. We draw this function in three separate columns, where each column corresponds to one $a \in \{0.25, 0.5, 0.75\}$. Additionally, each plot contains the curves for $\epsilon \in \{10^{-1.5}, 10^{-2}, 10^{-2.5}\}$, drawn in different colors. We provide five different curves for each set of parameters (a, b, ϵ), where each curve corresponds to one $\ell_{\max} \in \{6, 12, 24, 48, 100\}$. As a benchmark, we also draw the performance of parallel BP$_4$ using message normalization with a constant α_c for each ϵ. Pre-simulations suggest that for constant message normalization, the value $\alpha_c = 0.9375$ results in a good decoding performance on the [[900, 36]] HP code for all error rates ϵ that we considered. The LERs for constant normalization in Fig. 5.5 were obtained for $\ell_{\max} = 100$.

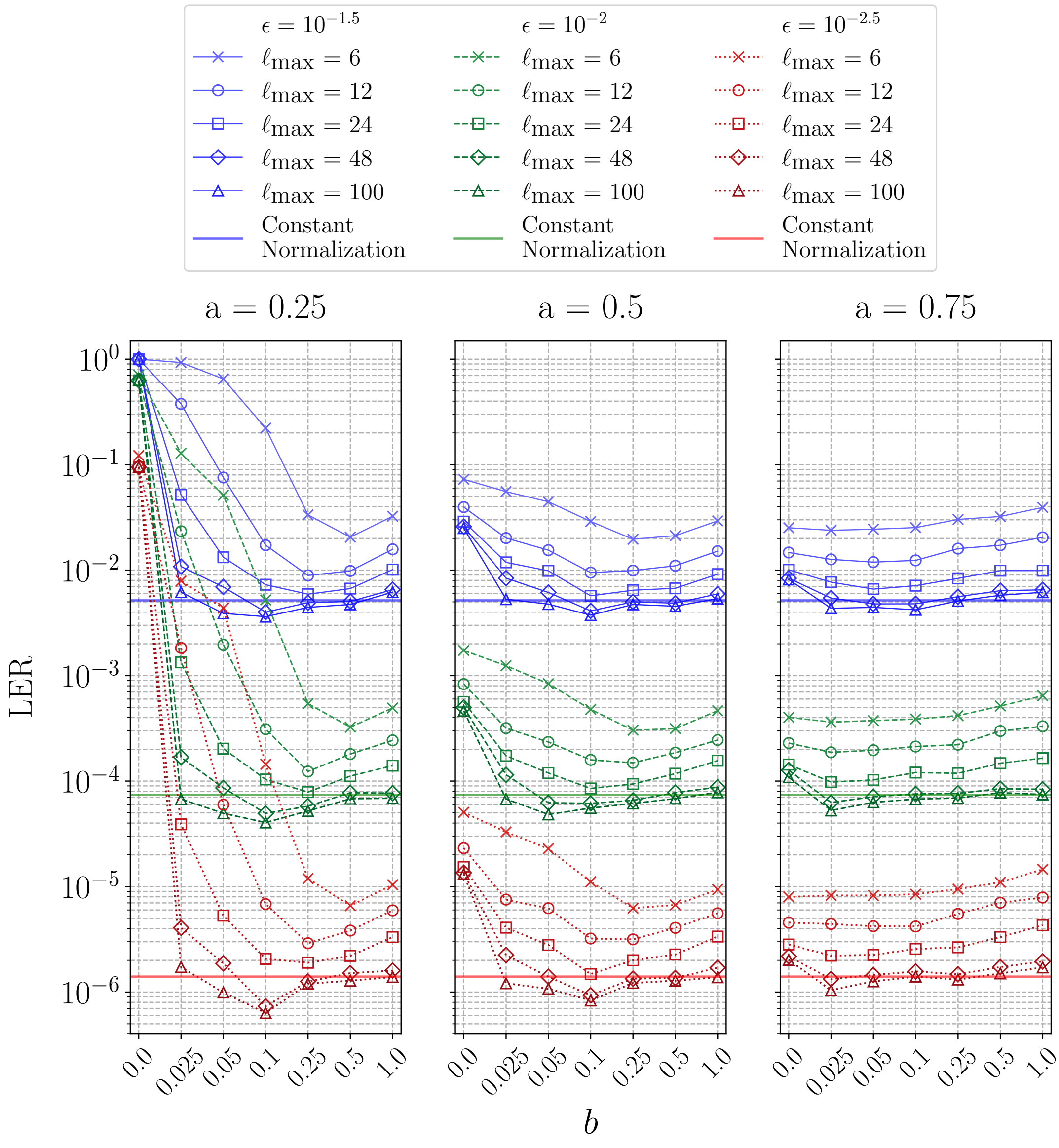


Figure 5.5: LER as a function of the normalization parameter b for different initial values a and error rates ϵ. The optimal value for b depends on both a, as well as maximum number of iterations $\ell_{\max}$.

The overestimation-underestimation-trade-off is reflected by the fact that all curves take on their minimum at a value $0 < b < 1$ (with the exception of the configuration $a = 0.75$, $\epsilon = 10^{-2.5}$, $\ell_{\max} = 6$, where the decoding performance is optimized for $b = 0$). For $a = 0.25$ and $a = 0.5$, the optimal choice for b shifts towards smaller values when more iterations are allowed.
This can be understood by reconsidering Example 17: Suppose that we choose $\ell_{\max} = 20$ iterations. For the corresponding error, decoding with $b = 0.1$ would result in a failure, as the decoder has not converged after 20 iterations due to small message magnitudes. As a consequence, b would have to be chosen larger. On the other hand, a large b can be detrimental to the decoding performance due to message overestimation, reflected by the positive slope in all curves towards larger values for b.
The overestimation-underestimation-trade-off is particularly pronounced for $a = 0.25$ and is less present as a becomes larger. This is due to the fact that the message reliability is strongly reduced for small initial values a, necessitating more iterations before the decoder can converge to the correct solution. However, decoders with a smaller initial value tend to perform better in later iterations. A parallel BP_4 using our VMN approach can outperform a parallel BP_4 using constant message normalization, given that a sufficient number of iterations are performed. Figure 5.5 shows that the largest performance gap to constant message normalization is achieved for $a = 0.25$. In this case, $b = 0.1$ yields the best decoding performance after 100 iterations. Hence, we choose $(a, b) = (0.25, 0.1)$ as a representative for VMN in what follows.

Figure 5.6 shows the LER of parallel BP_4 and serial BP_4 with VMN (i.e., $(a, b) = (0.25, 0.1)$) and without message normalization (i.e., $(a, b) = (1, 0)$) as a function of the iterations ℓ.

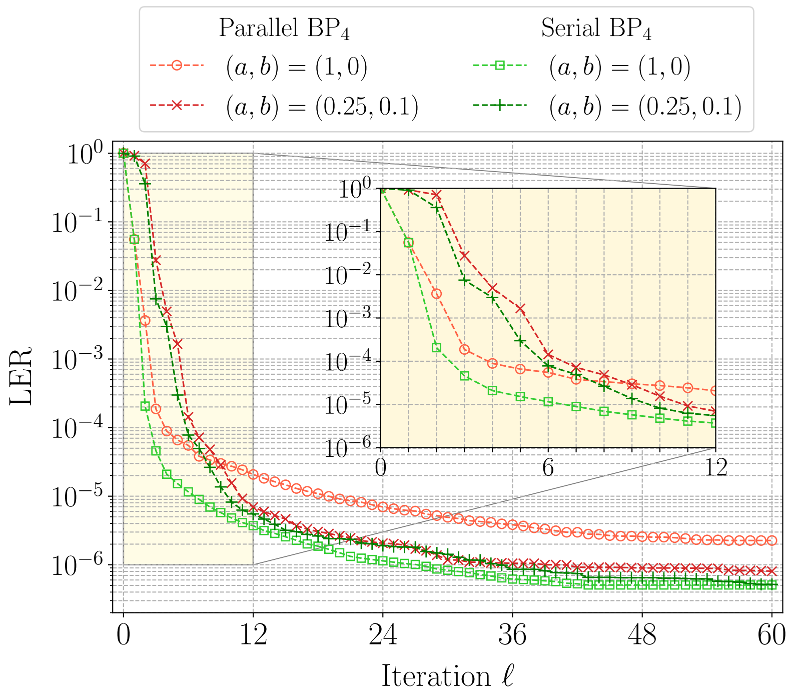


Figure 5.6: LER as a function of the iteration ℓ for parallel BP_4 and serial BP_4 with VMN and without message normalization.

While parallel BP_4 with $(a, b) = (1, 0)$ outperforms parallel BP_4 with $(a, b) = (1, 0)$ in the first eight iterations, the normalization leads to an increase in decoding performance for $\ell > 8$ and reduces the gap between parallel and serial performance. Again, we can observe that VMN can be beneficial, provided that a sufficient number of iterations is performed. On the other hand, applying VMN to serial BP_4 slows down the convergence speed but doesn't result in an increased performance. This is consistent with the results presented for the $[[129, 28]]$ HP code, where fixed initialization improved the parallel performance but leaves the serial performance mostly unchanged. We conclude that serial BP_4 doesn't benefit from VMN.

We end this chapter with the comparison of the performance curves, obtained for no normalization, constant normalization and variable normalization. Figure 5.7 shows the performance curves of parallel BP_4 using VMN with $(a, b) = (0.25, 0.1)$, constant message normalization with $(a, b) = (0.9375, 0)$ as well as no message normalization, i.e., $(a, b) = (1, 0)$. As a reference, we draw the LER curve of serial BP_4 with no normalization.

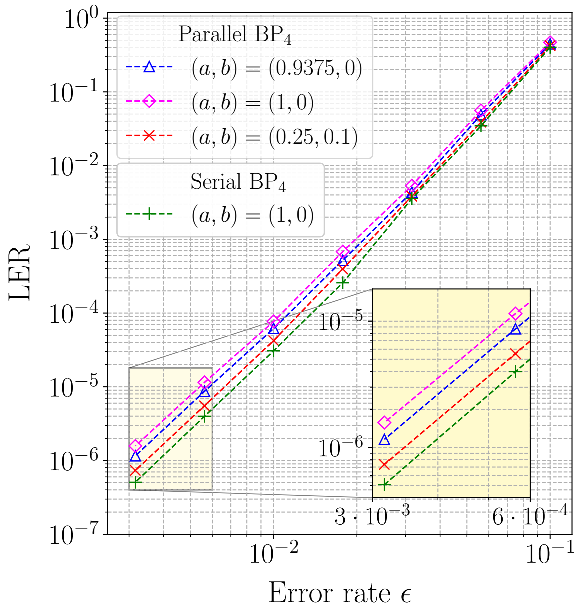


Figure 5.7: LER curves for parallel BP_4 with variable, constant and no normalization as well as serial BP_4 with no normalization.

Parallel BP_4 with VMN slightly outperforms parallel BP_4 with constant message normalization. Thus, the gap in decoding performance between parallel BP_4 and serial BP_4 is reduced. However, a small gap remains.
The corresponding curves for the $[[400, 16]]$ HP code and the $[[625, 25]]$ HP code are depicted in Fig. C.4. Note that for the $[[400, 16]]$ HP code and the $[[625, 25]]$ HP code, the scalars $\alpha_c = 0.875$ and $\alpha_c = 0.9375$ yielded the best performance for constant message normalization, respectively. While we can see a slight improvement in the performance using VMN on the $[[625, 25]]$ HP code, VMN doesn't improve the performance on the $[[400, 16]]$ HP code compared to constant message normalization.

6. Post-Processing Step

A common approach to alleviate decoding issues of BP on quantum codes is to apply post-processing methods posterior to BP decoding. While these methods can improve the decoding performance significantly [PK21, RWBC20], they are often computationally expensive. As an example, the OSD post-processing step proposed by Panteleev and Kalachev [PK21] scales cubically with the blocklength of the code [RV21]. Unfortunately, this may be prohibitive in a QEC system, where quantum states decohere quickly and fast decoding algorithms of low complexity are needed.
Based on our notation and the observation that type-2 errors cause split beliefs on the $[[129, 28]]$ HP code, we propose a post-processing procedure of low complexity. Our approach incorporates knowledge on the structure of type-2 error (described in Sec. 3.3) as well as the structure of the HP code and the classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$ to resolve these split beliefs.

6.1. Split Beliefs in the [[129, 28]] Hypergraph Product Code

Recall from Sec. 3.3 that an optimal decoder can correctly identify 630 type-2 errors. However, Tab. 5.1 shows that both, parallel and serial BP$_4$ only correct a small fraction of these errors. The bad performance for type-2 errors can be attributed to split beliefs, described in Sec. 3.5.2: As type-2 errors share their syndrome with another error of equal weight, BP$_4$ attempts to converge to both errors simultaneously. This is demonstrated in the following example.

Example 18. Consider the pair of type-2 errors $\boldsymbol{E}_1 = Z_0Y_{15} = Z_0^0Y_1^0$ and $\boldsymbol{E}_2 = X_{15}Z_{45} = X_1^0Z_3^0$. Since $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ are related via $Z_0^0Z_1^0Z_3^0 \in N(\mathcal{S})\setminus\mathcal{S}$, they share the same syndrome but don't correct each other. Figure 6.1 shows the belief history for a parallel BP$_4$ decoder, initialized with the syndrome $\boldsymbol{z}(\boldsymbol{E}_1) = \boldsymbol{z}(\boldsymbol{E}_2)$. The decoder attempts to converge to both errors simultaneously. However, the resulting superposition doesn't satisfy the syndrome equation. Since $\boldsymbol{z}(\boldsymbol{E}_1) = \boldsymbol{z}(\boldsymbol{E}_2)$, the belief histories are identical for both errors and decoding results in a failure if either of these errors arises. On the other hand, the optimal decoder chooses either $\boldsymbol{E}_1$ or $\boldsymbol{E}_2$ and decodes correctly in half of the cases.

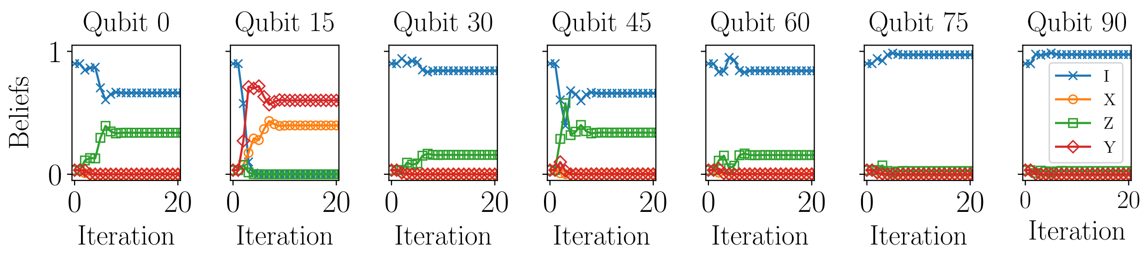


Figure 6.1: For pairs of type-2 errors $(\boldsymbol{E}_1, \boldsymbol{E}_2)$, a BP$_4$ decoder attempts to converge to both error simultaneously. This results in an invalid superposition.

Figure 6.2 shows that a large fraction of type-2 errors are not successfully decoded after twelve iterations. This indicates that a majority of type-2 errors show similar decoding behaviour as described in Example 18. Note that not all type-2 errors result in a decoding failure: Figure 6.2 shows that small fraction is decoded successfully. Interestingly, the decoder with the largest success rate on type-2 errors is the parallel BP$_4$ with $\epsilon_0 = 10^{-2}$, which performs significantly worse on weight-1 errors and weight-2 type-1 errors (as described in Sec. 5.1). This indicates that a large message reliability due to message overestimation helps to escape some split beliefs.

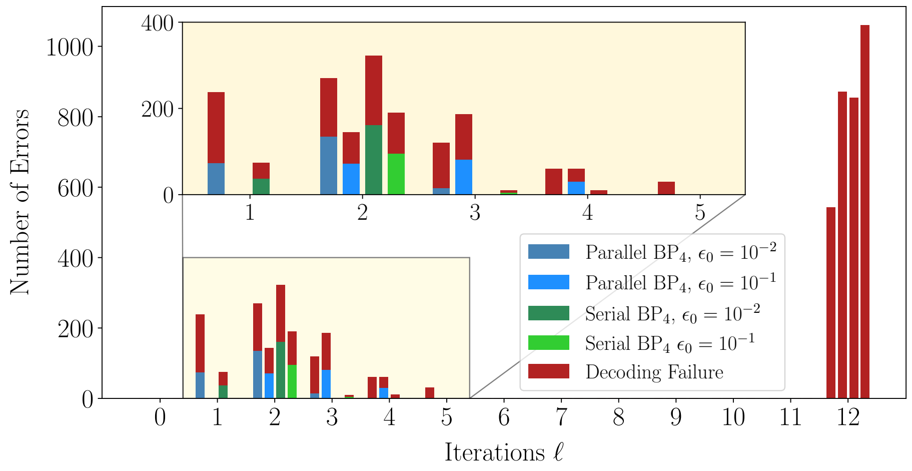


Figure 6.2: Distribution of iterations until termination for type-2 errors, using parallel and serial BP$_4$ decoding and fixed initialization with $\epsilon_0 \in \{10^{-1}, 10^{-2}\}$.

Note that we can detect decoding failures due to split beliefs, since the estimated error doesn't match the syndrome. We propose a simple yet effective post-processing step that allows the decoder to choose one out of two possible type-2 errors, given that decoding failed due to type-2 ambiguity. We describe our approach in the following section.

6.2. Including Knowledge on the Error Structure

Our approach uses the fact that we have knowledge on the structure of type-2 errors. In the following we will assume that all decoding failures after $\ell_{\max}$ iterations are due to split beliefs. This assumption is reasonable in the low error rate region, where a majority of decoding failures is caused by weight-2 errors due to the low a-priori probability for errors of larger weight: For the parallel BP$_4$ with $\epsilon_0 = 10^{-1}$, decoding fails for 489 correctable weight-2 errors. 447 of these failures occur due to type-2 errors, whereas only 42 failures occur due to a type-1 error. Assuming that a split belief occurred, we can infer the corresponding error pair of type-2 errors from the syndrome and choose to decide for one of them.
Recall from Sec. 3.3 that for type-2 errors, one error is of the form $\boldsymbol{E}_1 = Z^i_{o_0} Y^i_{o_1}$, while it's twin is of the form $\boldsymbol{E}_2 = X^i_{o_1} Z^i_{o_2}$. Since both errors occur with identical probability, we choose to decide for $\boldsymbol{E}_1$.

Let us now define the syndrome corresponding to the upper half of $\boldsymbol{S}$ as $\boldsymbol{z}_X$ and the syndrome corresponding to the lower half of $\boldsymbol{S}$ as $\boldsymbol{z}_Z$, i.e.

$$\begin{aligned}\boldsymbol{z}_X &= (z_0, \cdots, z_{M_1N_2-1}),\\ \boldsymbol{z}_Z &= (z_{M_1N_2}, \cdots, z_{M_1N_2+N_1M_2-1}).\end{aligned}$$

Due to the structure of HP codes, only X checks participate in the computation of $\boldsymbol{z}_X$. Analogously, only Z checks participate in the computation of $\boldsymbol{z}_Z$. Since we assume that a type-2 error occurred, we don't take into account non-identity entries on qubits corresponding to the right part of $\boldsymbol{S}$ in the syndrome computation: In other words, we assume that $E_n = I\ \forall n \geq N_1N_2$. In this case, the right part of the error doesn't affect the syndrome since I entries commute with every edge type.
Figure 6.3a and Fig. 6.3b illustrate the post-processing procedure. While we chose to use the HP code from Sec. 3.2.1 (i.e. a HP code constructed from two repetition codes) for reasons of simplicity, the same considerations apply to all HP codes. To summarize our description from Sec. 3.2.1, VNs in the left part of $\boldsymbol{S}$ with identical inner index i are connected to the checks $cx_0^i, ..., cx_{M_1-1}^i$ via X checks according to the Tanner graph $\mathcal{T}(\boldsymbol{H}_1)$. Analogously, VNs in the left part of $\boldsymbol{S}$ with identical outer index o are connected to the checks $cz_o^0, ..., cx_o^{M_2-1}$ via Z checks according to the Tanner graph $\mathcal{T}(\boldsymbol{H}_2)$.
Due to linearity, the syndrome of a weight-2 error can be expressed as the sum of the syndromes for the corresponding weight-1 errors:

$$\begin{aligned}\boldsymbol{z}_X &= \boldsymbol{z}_X(Z_{o_0}^i Y_{o_1}^i) = \boldsymbol{z}_X(Z_{o_0}^i) + \boldsymbol{z}_X(Y_{o_1}^i) \qquad \text{mod } 2,\\ \boldsymbol{z}_Z &= \boldsymbol{z}_Z(Z_{o_0}^i Y_{o_1}^i) = \boldsymbol{z}_Z(Z_{o_0}^i) + \boldsymbol{z}_Z(Y_{o_1}^i) \qquad \text{mod } 2.\end{aligned} \tag{6.1}$$

The post-processing procedure consists of three steps, using the above equations as well as knowledge on the code structure to deduce the indices o_0, o_1 and i from the syndrome $\boldsymbol{z}$.

Step One

In the first step, we use the fact that $Z_{o_0}^i$ commutes with every Z check in the lower half. Thus, we have $\boldsymbol{z}_Z(Z_{o_0}^i) = \boldsymbol{0}$ and consequently, $\boldsymbol{z}_Z = \boldsymbol{z}_Z(Y_{o_1}^i)$. This means that $\boldsymbol{z}_Z$ is generated by an error with exactly one Y entry with outer index o_1 and inner index i.
We can now use our knowledge on the grid-structure of the code: The error $Y_{o_1}^i$ only induces a non-trivial syndrome for checks in the column with outer index o_1. We can thus easily determine o_1 and thereafter, define the auxiliary syndrome $\boldsymbol{z}_1$, consisting of all CNs in column o_1, i.e. $\boldsymbol{z}_1 = ([z_Z]_{o_1}^0, ..., [z_Z]_{o_1}^{M_2-1})$. Since $Y_{o_1}^i$ is a weight-1 error, $\boldsymbol{z}_1$ will be equal to the column i of $\boldsymbol{H}_2$. The inner index i is thus easily found by a column search through $\boldsymbol{H}_2$, which we will denote by

$$i = \arg\underset{j}{\text{where}}\ \left([\boldsymbol{H}_2]_{:,j} = \boldsymbol{z}_1\right).$$

Step Two

In the second step, we rearrange Eq. 6.1 to obtain $\boldsymbol{z}_X(Z_{o_0}^i) = \boldsymbol{z}_X + \boldsymbol{z}_X(Y_{o_1}^i) \mod 2$, as our goal is to determine the missing outer index o_0. It suffices to consider this equation for the i-th row in the VN grid, since the errors $Z^i_{o_0}$ and $Y^i_{o_1}$ only induces a non-trivial syndrome for checks in row i. We denote the syndrome corresponding to row i by $\boldsymbol{z}_2$. Note that we can easily generate the syndrome of the weight-1 error $\boldsymbol{z}_X(Y^i_{o_1})$ by picking column o_1 of $\boldsymbol{H}_1$. Thus, we obtain $\boldsymbol{z}_2 = ([z_X]^i_0, ..., [z_X]^i_{M_1-1}) + [\boldsymbol{H}_1]_{:,o_1}$.

Step Three

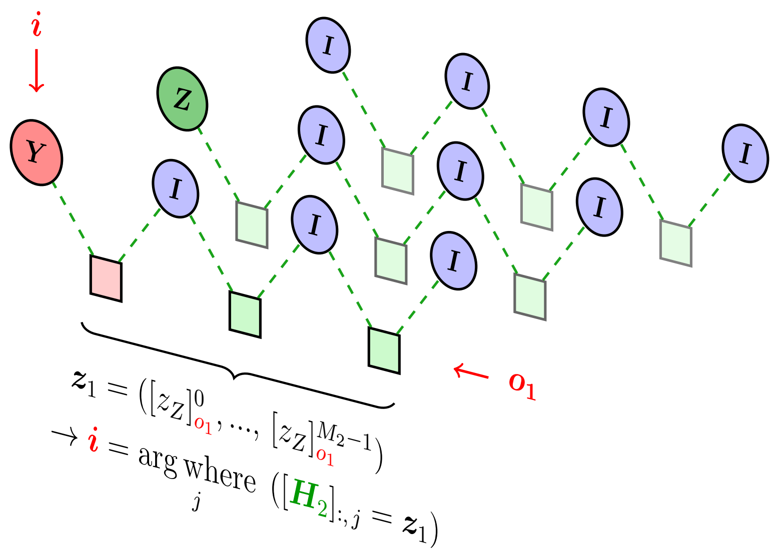
In step three, we identify the remaining outer index o_0 of the Z entry: As $\boldsymbol{z}_2$ can be thought of as the syndrome of a weight-1 error (using the PCM $\boldsymbol{H}_1$), we identify the outer index o_0 by searching through the columns of $\boldsymbol{H}_1$, i.e.

$$o_0 = \arg \underset{j}{\text{where}} \ ([\boldsymbol{H}_1]_{:,j} = \boldsymbol{z}_2) .$$

Once step three is completed, we recovered the indices i, o_0 and o_1 from the syndrome. We can then output the error $\boldsymbol{E} = Z^i_{o_0} Y^i_{o_1}$. The post-processing procedure is summarized in Algorithm 3.

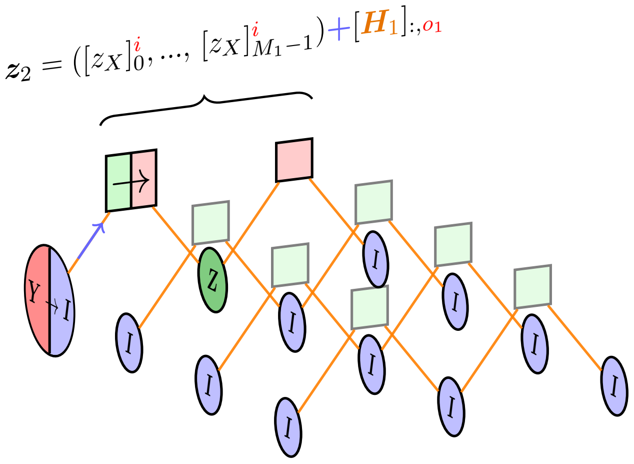
Algorithm 3 Post-Processing for $[[129, 28]]$ HP code

```
    Input:
1:  $\boldsymbol{z} \in \{0,1\}^M$
    Split syndrome according to upper and lower part of $\boldsymbol{S}$:
2:  $\boldsymbol{z}_X = (z_0, \dots, z_{M_1 N_2 - 1})$
3:  $\boldsymbol{z}_Z = (z_{M_1 N_2}, \dots, z_{M_1 N_2 + N_1 M_2 - 1})$
    Step 1: Identify outer and inner index of $\boldsymbol{Y}$ entry:
4:  $O = \{o : [z_Z]^i_o = 1\}$
5:  $o_1 \leftarrow$ only element in $O$
6:  $\boldsymbol{z}_1 = ([z_Z]^0_{o_1}, \dots, [z_Z]^{M_2-1}_{o_1})$
7:  $i = \arg \underset{j}{\text{where}} \ ([\boldsymbol{H}_2]_{:,j} = \boldsymbol{z}_1)$
    Step 2: Compute residual syndrome:
8:  $\boldsymbol{z}_2 \leftarrow ([z_X]^i_0, \dots, [z_X]^i_{M_1-1}) + [\boldsymbol{H}_1]_{:,o_1}$
    Step 3: Identify outer index of remaining $\boldsymbol{Z}$ entry:
9:  $o_0 = \arg \underset{j}{\text{where}} \ ([\boldsymbol{H}_1]_{:,j} = \boldsymbol{z}_2)$
10: return $\boldsymbol{E} = Z^i_{o_0} Y^i_{o_1}$
```



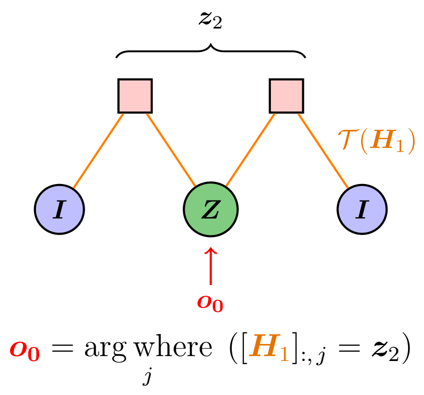
i
Y
Z
I
o_1
$z_1 = ([z_Z]^0_{o_1}, \ldots, [z_Z]^{M_2-1}_{o_1})$
$\rightarrow i = \arg\,\underset{j}{\text{where}}\, ([H_2]_{:,j} = z_1)$

(a) Step 1: Only the Y entry induces non-trivial entries in $\boldsymbol{z}_Z$. Thus, $\boldsymbol{z}_Z$ can be used to determine o_1 and i.



$z_2 = ([z_X]^i_0, \ldots, [z_X]^i_{M_1-1}) + [H_1]_{:,o_1}$
$Y \rightarrow I$
Z
I

(b) Step 2: Using o_1 and i from the previous step, the residual syndrome is obtained by flipping all checks in the Tanner graph $\mathcal{T}(\boldsymbol{H}_1)$ corresponding to the inner index i.



z_2
$\mathcal{T}(H_1)$
I
Z
o_0
$o_0 = \arg\,\underset{j}{\text{where}}\, ([H_1]_{:,j} = z_2)$

(c) Step 3: The outer index o_0 is obtained as the index of the column of $\boldsymbol{H}_1$ identical to the residual syndrome $\boldsymbol{z}_2$.

Figure 6.3: Visual representation of the proposed post-processing procedure. For illustration purposes, we depict the HP constructed from the $[3, 1, 3]$ repetition code and the $[4, 1, 4]$ repetition code.

6.3. Decoding Performance with Post-Processing

We now compare the performances of BP_4 decoding on the $[[129, 28]]$ HP code with and without the post-processing procedure described above. Since we observed the best results for parallel BP_4 with fixed initialization and $\epsilon_0 = 10^{-1}$ (see Tab. 5.1), we only apply post-processing to this configuration. Figure 6.4 shows the corresponding performance curves.

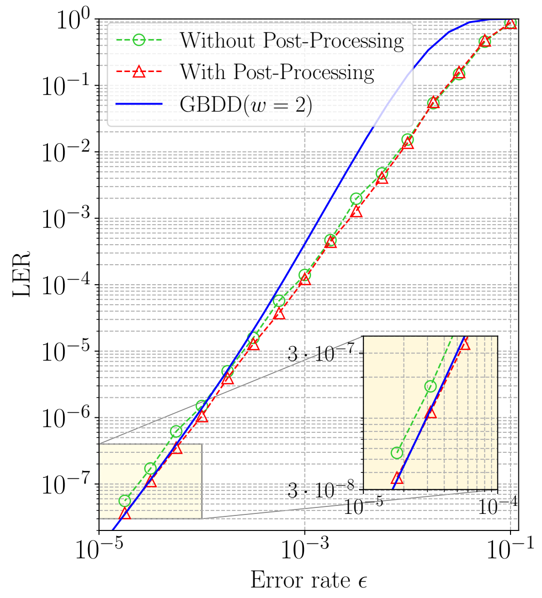


Figure 6.4: LER curves for a parallel BP_4 with fixed initialization $\epsilon_0 = 10^{-1}$, without and without post-processing step.

While both decoders perform close to GBDD($w = 2$), a small gap between the GBDD performance and the performance of the unmodified decoder remains in the low error rate region. This gap can mostly be attributed to split beliefs, as the majority of decoding failures are caused by type-2 errors. Our modification closes this gap.
Numerical data for the performance of the modified parallel BP_4 decoder for weight-1 and weight2 errors is provided in Tab. 6.1. As a reference, we show the number of errors corrected by a GBDD($w = 2$) as well as the number of errors corrected by the unmodified decoder. While the unmodified decoder fails for a large fraction of type-2 errors, the modified decoder corrects 618 out of 630 correctable type-2 errors. In total, only 54 errors theoretically correctable errors are not corrected after applying our post-processing procedure, demonstrating the effectiveness of our approach.

Table 6.1: Number of corrected weight-1 and weight-2 errors for parallel BP_4 decoding with fixed initial values $\epsilon_0 = 0.1$.

		Weight-1	Weight-2			
			Type-1	Type-2	Type-3	$\sum$
Number of Correctable Errors		387	72729	630	0	**73359**
Number of Successes						
$\epsilon_0 = 10^{-1}$	Parallel BP_4	387	72687	183	0	72870
	Modified Parallel BP_4	387	72687	618	0	**73305**

7. Conclusion and Outlook

7.1. Conclusion

In this thesis, the effect of several modifications to BP on the decoding performance was investigated. To gain a better understanding of decoding failure causing errors, we introduced a new notation, which emphasizes the relation between properties of classical codes and properties of the HP code constructed from these codes. With the help of this notation, we identified all correctable weight-1 and weight-2 errors of the $[[129, 28]]$ HP code, which is constructed from two classical BCH codes. We used this information to derive the performance of the GBDD with a correction radius of $w = 2$, which is also provided in [LK21]. The GBDD serves as a good approximation of an optimal decoder in the low error rate region.
We confirmed the results of Kuo and Lai that serial BP_4 approaches the GBDD performance in the low error rate region and demonstrated that a parallel BP_4 yields a comparable performance when fixed initialization is applied. Our simulation results showed that both, parallel and serial BP run into split beliefs for pairs of errors closely linked to low-weight codewords of one of the classical BCH codes, used to construct the HP code. We demonstrated that the remaining gap between BP decoding and an optimal decoder in the low error rate region can mostly be attributed to split beliefs. Based on the knowledge on the structure of the split belief causing errors, we proposed a low-complexity post-processing procedure which specifically aims at resolving these split beliefs, making use of our notation. Using this post-processing procedure, the gap to the GBDD can nearly be closed in the low error rate region.

We introduced type-A and type-B errors, which are closely related to the structure of HP codes and have a convenient representation in our notation. We established a connection between the decoding behaviour of parallel BP_4 on type-A and type-B errors for symmetric HP. Experimental results indicate that the majority of decoding failures can be attributed to errors of these types. We provide experimental evidence that decoding failures on these errors are related to the decoding behaviour of BP_2 on the classical code and propose a method to choose a good classical code from which a good HP code can be constructed. Using such an optimized code, the performance of serial BP_4 without post-processing approaches the performance of BP followed by an OSD post-processing step as presented by Roffe et al. in [RWBC20].

We proposed to choose the normalization factor according to a function depending on the number of iterations and two parameters and perform an optimization of these parameters. We show that message normalization with a constant scalar can be slightly outperformed on the $[[625, 25]]$ HP code and the $[[900, 36]]$ HP code and the gap between parallel BP_4 and serial BP_4 can be nearly closed.

7.2. Outlook

While our post-processing procedure closes the remaining performance gap in the low error rate region, it is specifically designed for the $[[129, 28]]$ HP code. It would be interesting to see if our approach can be generalized to a broader class of codes. The preceding error analysis could get more sophisticated, especially when degenerate codes are considered. If sufficient algebraic structure is still available in the classical codes used to construct the HP code, a post-processing procedure could benefit from the availability of efficient classical decoders.
For the $[[900, 36]]$ HP code, a performance gap to BP with subsequent OSD post-processing remains. Further analysis could shed light on the structure of errors where BP fails and OSD succeeds. Another interesting question is, whether the application of OSD in combination with our modifications could further enhance the decoding performance. Potentially, the performance of normalized BP decoding could benefit from a more sophisticated strategy to choose the normalization scalar. Such a strategy may incorporate information derived from the running beliefs during decoding [CGW10]. Furthermore, we only considered variable message normalization on symmetric HP codes. It remains an open question, how this technique affects the decoding behaviour for other quantum codes.

A. Abbreviations

BCH	Bose–Chaudhuri–Hocquenghem
BDD	bounded distance decoder
BP	belief propagation
BSC	binary symmetric channel
CN	check node
CSS	Calderbank-Shor-Steane
GBDD	generalized bounded distance decoder
HP	hypergraph product
LDPC	low-density parity-check
LER	logical error rate
LLR	log-likelihood ratio
MSA	min-sum approximation
OSD	ordered statistics decoding
PCM	parity-check matrix
QEC	quantum error correction
QLDPC	quantum low-density parity-check
QSC	quantum stabilizer code
RV	random variable
VN	variable node
VMN	variable message normalization

B. Proofs

The permutation matrix $\boldsymbol{P}_\mathrm{c}$ interchanges outer and inner indices in the left part $\boldsymbol{S}_\mathrm{l}$ and the right part $\boldsymbol{S}_\mathrm{r}$, respectively. The permutation matrix $\boldsymbol{P}_\mathrm{r}$ interchanges outer and inner indices in the upper part $\boldsymbol{S}^X$ and the lower part $\boldsymbol{S}^Z$ and subsequently interchanges the permuted upper and lower part. A visual representation of this transformation is provided in Fig. 3.8. The following equation demonstrates that indeed, $\boldsymbol{P}_\mathrm{r} \cdot \boldsymbol{S} \cdot \boldsymbol{P}_\mathrm{c} = \boldsymbol{S}' = f(\boldsymbol{S})$:

$$\boldsymbol{P}_\mathrm{r} \cdot \boldsymbol{S} \cdot \boldsymbol{P}_\mathrm{c} = \boldsymbol{P}_\mathrm{r} \cdot \begin{pmatrix} X \cdot (\boldsymbol{H} \otimes \boldsymbol{I}_n) & X \cdot (\boldsymbol{I}_m \otimes \boldsymbol{H}^T) \\ Z \cdot (\boldsymbol{I}_n \otimes \boldsymbol{H}) & Z \cdot (\boldsymbol{H}^T \otimes \boldsymbol{I}_m) \end{pmatrix} \cdot \boldsymbol{P}_\mathrm{c}$$

$$= \boldsymbol{P}_\mathrm{r} \cdot \left(\begin{array}{ccc||ccc}
X \cdot H_{11} \cdot \boldsymbol{I}_n & \cdots & X \cdot H_{11} \cdot \boldsymbol{I}_n & X \cdot \boldsymbol{H}^T & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
X \cdot H_{m1} \cdot \boldsymbol{I}_n & \cdots & X \cdot H_{m1} \cdot \boldsymbol{I}_n & 0 & \cdots & X \cdot \boldsymbol{H}^T \\
\hline\hline
Z \cdot \boldsymbol{H} & \cdots & 0 & Z \cdot H_{11} \cdot \boldsymbol{I}_m & \cdots & Z \cdot H_{m1} \boldsymbol{I}_m \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & Z \cdot \boldsymbol{H} & Z \cdot H_{1n} \cdot \boldsymbol{I}_m & \cdots & Z \cdot H_{mn} \boldsymbol{I}_m
\end{array}\right) \cdot \boldsymbol{P}_\mathrm{c}$$

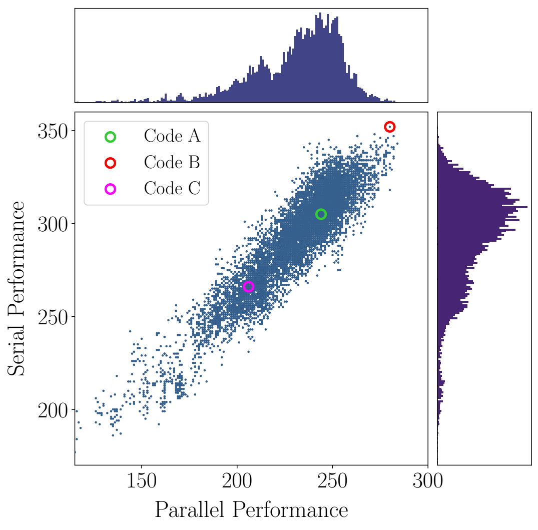
$$= \boldsymbol{P}_\mathrm{r} \cdot \left(\begin{array}{ccc|c|ccc||ccc|c|ccc}
X \cdot H_{11} & \cdots & 0 & & X \cdot H_{1n} & \cdots & 0 & X \cdot H_{11} & \cdots & X \cdot H_{m1} & & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & X \cdot H_{11} & & 0 & \cdots & X \cdot H_{1n} & X \cdot H_{1n} & \cdots & X \cdot H_{mn} & & 0 & \cdots & 0 \\
\hline
\multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c||}{\vdots} & \multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c}{\vdots} \\
\hline
X \cdot H_{m1} & \cdots & 0 & & X \cdot H_{mn} & \cdots & 0 & 0 & \cdots & 0 & & X \cdot H_{11} & \cdots & X \cdot H_{m1} \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & X \cdot H_{m1} & & 0 & \cdots & X \cdot H_{mn} & 0 & \cdots & 0 & & X \cdot H_{1n} & \cdots & X \cdot H_{mn} \\
\hline\hline
Z \cdot H_{11} & \cdots & Z \cdot H_{1n} & & 0 & \cdots & 0 & Z \cdot H_{11} & \cdots & 0 & & Z \cdot H_{m1} & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
Z \cdot H_{m1} & \cdots & Z \cdot H_{mn} & & 0 & \cdots & 0 & 0 & \cdots & Z \cdot H_{11} & & 0 & \cdots & Z \cdot H_{m1} \\
\hline
\multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c||}{\vdots} & \multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c}{\vdots} \\
\hline
0 & \cdots & 0 & & Z \cdot H_{11} & \cdots & Z \cdot H_{1n} & Z \cdot H_{1n} & \cdots & 0 & & Z \cdot H_{mn} & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & & Z \cdot H_{m1} & \cdots & Z \cdot H_{mn} & 0 & \cdots & Z \cdot H_{1n} & & 0 & \cdots & Z \cdot H_{mn}
\end{array}\right) \cdot \boldsymbol{P}_\mathrm{c}$$

$$= \boldsymbol{P}_\mathrm{r} \cdot \left(\begin{array}{ccc|c|ccc||ccc|c|ccc}
X \cdot H_{11} & \cdots & X \cdot H_{1n} & & 0 & \cdots & 0 & X \cdot H_{11} & \cdots & 0 & & X \cdot H_{m1} & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & & X \cdot H_{11} & \cdots & X \cdot H_{1n} & X \cdot H_{1n} & \cdots & 0 & & X \cdot H_{mn} & \cdots & 0 \\
\hline
\multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c||}{\vdots} & \multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c}{\vdots} \\
\hline
X \cdot H_{m1} & \cdots & X \cdot H_{mn} & & 0 & \cdots & 0 & 0 & \cdots & X \cdot H_{11} & & 0 & \cdots & X \cdot H_{m1} \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & & X \cdot H_{m1} & \cdots & X \cdot H_{mn} & 0 & \cdots & X \cdot H_{1n} & & 0 & \cdots & X \cdot H_{mn} \\
\hline\hline
Z \cdot H_{11} & \cdots & 0 & & Z \cdot H_{1n} & \cdots & 0 & Z \cdot H_{11} & \cdots & Z \cdot H_{m1} & & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
Z \cdot H_{m1} & \cdots & 0 & & Z \cdot H_{mn} & \cdots & 0 & 0 & \cdots & 0 & & Z \cdot H_{11} & \cdots & Z \cdot H_{m1} \\
\hline
\multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c||}{\vdots} & \multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c}{\vdots} \\
\hline
0 & \cdots & Z \cdot H_{11} & & 0 & \cdots & Z \cdot H_{1n} & Z \cdot H_{1n} & \cdots & Z \cdot H_{mn} & & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & Z \cdot H_{m1} & & 0 & \cdots & Z \cdot H_{mn} & 0 & \cdots & 0 & & Z \cdot H_{1n} & \cdots & Z \cdot H_{mn}
\end{array}\right)$$

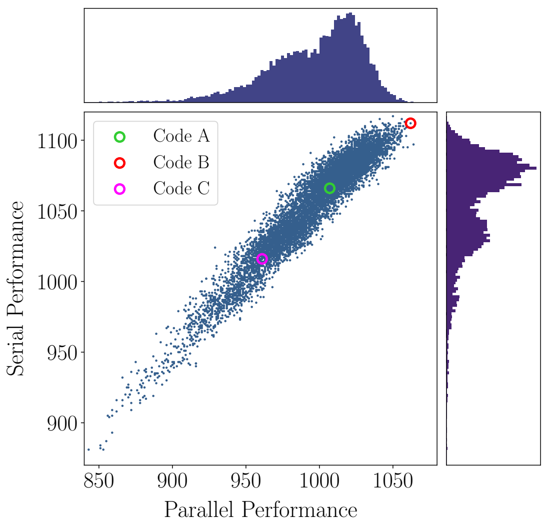
$$
= \left(\begin{array}{ccc|c|ccc||ccc|c|ccc}
Z\cdot H_{11} & \cdots & 0 & & Z\cdot H_{1n} & \cdots & 0 & Z\cdot H_{11} & \cdots & Z\cdot H_{m1} & & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & Z\cdot H_{11} & & 0 & \cdots & Z\cdot H_{1n} & Z\cdot H_{1n} & \cdots & Z\cdot H_{mn} & & 0 & \cdots & 0 \\
\hline
\multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c||}{\vdots} & \multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c}{\vdots} \\
\hline
Z\cdot H_{m1} & \cdots & 0 & & Z\cdot H_{mn} & \cdots & 0 & 0 & \cdots & 0 & & Z\cdot H_{11} & \cdots & Z\cdot H_{m1} \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & Z\cdot H_{m1} & & 0 & \cdots & Z\cdot H_{mn} & 0 & \cdots & 0 & & Z\cdot H_{1n} & \cdots & Z\cdot H_{mn} \\
\hline\hline
X\cdot H_{11} & \cdots & X\cdot H_{1n} & & 0 & \cdots & 0 & X\cdot H_{11} & \cdots & 0 & & X\cdot H_{m1} & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
X\cdot H_{m1} & \cdots & X\cdot H_{mn} & & 0 & \cdots & 0 & 0 & \cdots & X\cdot H_{11} & & 0 & \cdots & X\cdot H_{m1} \\
\hline
\multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c||}{\vdots} & \multicolumn{3}{c|}{\vdots} & \ddots & \multicolumn{3}{c}{\vdots} \\
\hline
0 & \cdots & 0 & & X\cdot H_{11} & \cdots & X\cdot H_{1n} & X\cdot H_{1n} & \cdots & 0 & & X\cdot H_{mn} & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & & X\cdot H_{m1} & \cdots & X\cdot H_{mn} & 0 & \cdots & X\cdot H_{1n} & & 0 & \cdots & X\cdot H_{mn}
\end{array}\right)
$$

$$
= \boldsymbol{P}_{\mathrm{r}} \cdot \begin{pmatrix} Z\cdot(\boldsymbol{H}\otimes\boldsymbol{I}_n) & Z\cdot(\boldsymbol{I}_m\otimes\boldsymbol{H}^T) \\ X\cdot(\boldsymbol{I}_n\otimes\boldsymbol{H}) & X\cdot(\boldsymbol{H}^T\otimes\boldsymbol{I}_m) \end{pmatrix} \cdot \boldsymbol{P}_{\mathrm{c}} = \boldsymbol{S}' = f(\boldsymbol{S})
$$

C. Supplementary Figures

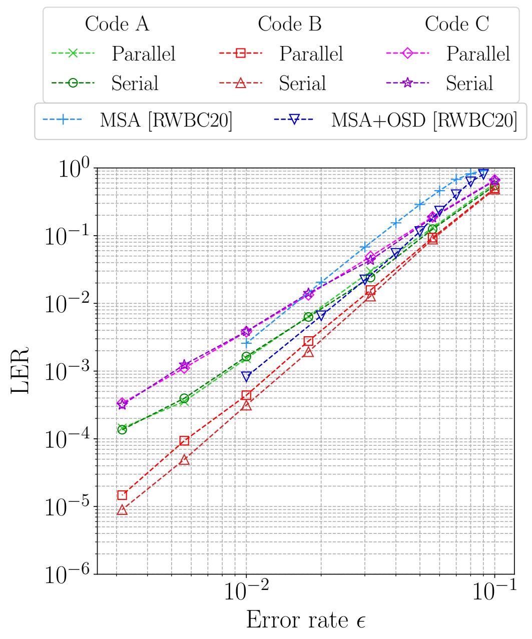


(a) For the $[[400, 16]]$ HP, there are 560 weight-3 errors, since $N_C = 16$.

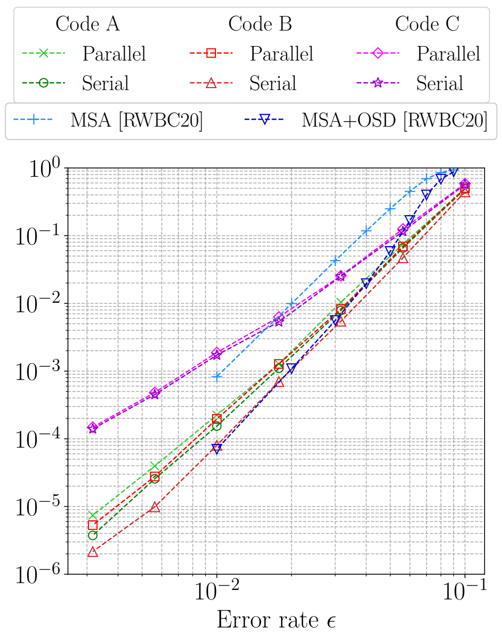


(b) For the $[[625, 25]]$ HP, there are 1140 weight-3 errors, since $N_C = 20$.

Figure C.1: Number of decoding successes of BP_2 when decoding weight-3 errors on randomly sampled, regular LDPC codes with $d_v = 3$ and $d_c = 4$.



(a) $[[400, 16]]$ HP code.



(b) $[[625, 25]]$ HP code

Figure C.2: LER curves for parallel BP_4 and serial BP_4 on three HP codes, constructed from classical codes with different BP_2 performances (see Fig. C.1). As a reference, we redraw the curves presented in [RWBC20].

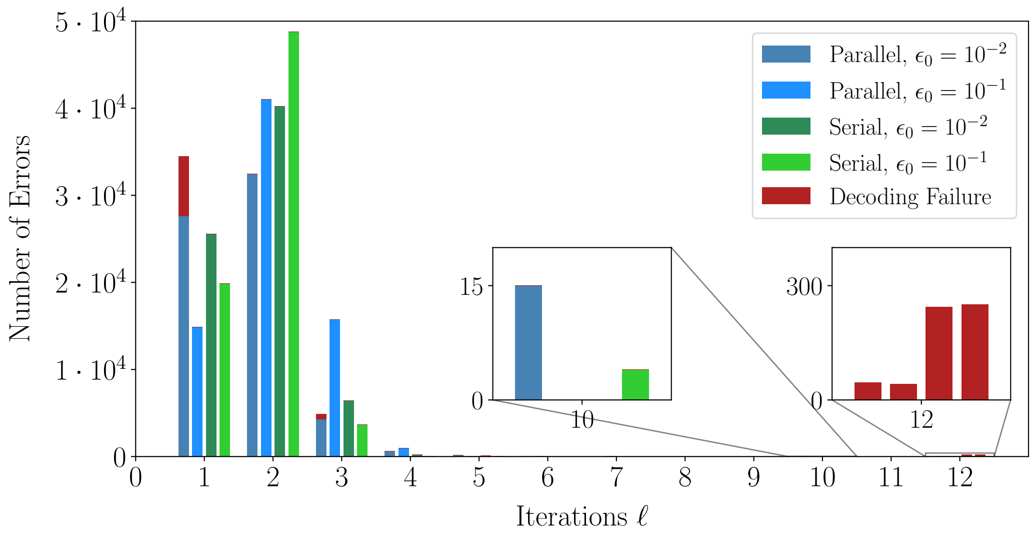
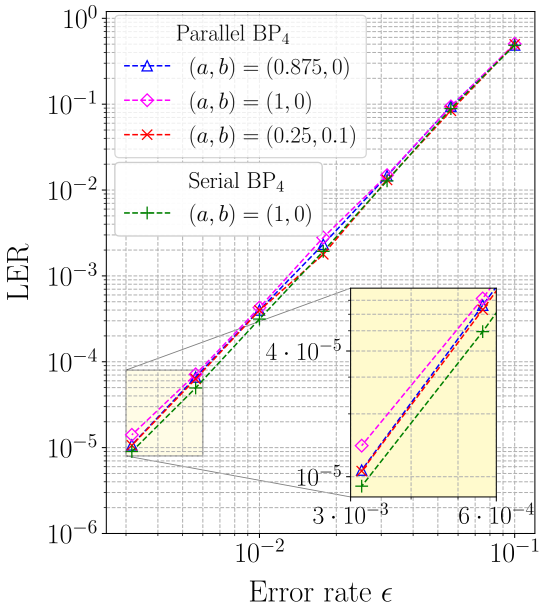
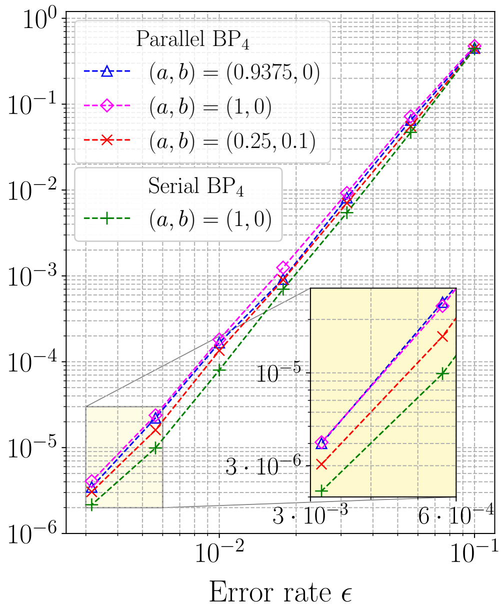


Figure C.3: Distribution of iterations until termination for type-1 errors of weight two, using parallel and serial BP_4 decoding and fixed initialization with $\epsilon_0 \in \{10^{-1}, 10^{-2}\}$.



(a) [[400, 16]] HP code.



(b) [[625, 25]] HP code.

Figure C.4: LER curves for parallel BP_4 with variable, constant and no normalization as well as serial BP_4 with no normalization.

Bibliography

[BBA$^+$15] Z. Babar, P. Botsinis, D. Alanis, S. X. Ng, and L. Hanzo, "Fifteen years of quantum LDPC coding and improved decoding strategies," *IEEE Access*, vol. 3, pp. 2492–2519, 2015.

[CF02] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 406–414, 2002.

[CGW10] A. I. V. Casado, M. Griot, and R. D. Wesel, "LDPC decoders with informed dynamic scheduling," *IEEE Transactions on Communications*, vol. 58, no. 12, pp. 3470–3479, 2010.

[CRSS97] A. Calderbank, E. Rains, P. Shor, and N. Sloane, "Quantum error correction via codes over GF(4)," in *Proceedings of IEEE International Symposium on Information Theory*, 1997, pp. 292–.

[DM98] M. Davey and D. MacKay, "Low-density parity check codes over GF(q)," *IEEE Communications Letters*, vol. 2, no. 6, pp. 165–167, 1998.

[DSS98] D. P. DiVincenzo, P. W. Shor, and J. A. Smolin, "Quantum-channel capacity of very noisy channels," *Phys. Rev. A*, vol. 57, pp. 830–839, 1998.

[EE14] A. A. Emran and M. Elsabrouty, "Simplified variable-scaled min sum LDPC decoder for irregular LDPC codes," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, 2014, pp. 518–523.

[FL95] M. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, 1995.

[Gal62] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, 1962.

[Got97] D. Gottesman, "Stabilizer codes and quantum error correction," Ph.D. dissertation, California Institute of Technology, Jan. 1997.

[Gro96] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 1996, p. 212–219.

[HEAD01] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, vol. 2, 2001, pp. 1036–1036E.

[HFI10] M. Hagiwara, M. Fossorier, and H. Imai, "Fixed initialization decoding of LDPC codes over a binary symmetric channel," *IEEE Transactions on Information Theory*, vol. 58, no. 6, pp. 784–788, 2010.

[HMvdW$^+$20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[HOP96] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, 1996.

[Hos20] M. Hostetter, "Galois: A performant NumPy extension for Galois fields," 11 2020. [Online]. Available: https://github.com/mhostetter/galois

[KCL21] K.-Y. Kuo, I.-C. Chern, and C.-Y. Lai, "Decoding of quantum data-syndrome codes via belief propagation," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 1552–1557.

[KFL01] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, 2001.

[KL20] K.-Y. Kuo and C.-Y. Lai, "Refined belief propagation decoding of sparse-graph quantum codes," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 2, pp. 487–498, 2020.

[LK21] C.-Y. Lai and K.-Y. Kuo, "Log-domain decoding of quantum LDPC codes over binary finite fields," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–15, 2021.

[LMPZ96] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek, "Perfect quantum error correcting code," *Phys. Rev. Lett.*, vol. 77, pp. 198–201, Jul 1996.

[LP19] Y.-H. Liu and D. Poulin, "Neural belief-propagation decoders for quantum error-correcting codes," *Phys. Rev. Lett.*, vol. 122, p. 200501, 2019.

[LS06] G. Lechner and J. Sayir, "Improved sum-min decoding for irregular LDPC codes," in *4th International Symposium on Turbo Codes*, 2006, pp. 1–6.

[MMM04] D. MacKay, G. Mitchison, and P. McFadden, "Sparse-graph codes for quantum error correction," *IEEE Transactions on Information Theory*, vol. 50, no. 10, pp. 2315–2330, 2004.

[MN02] D. Mackay and R. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 8, pp. 1645–1646, 2002.

[NC10] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010.

[PC08] D. Poulin and Y. Chung, "On the iterative decoding of sparse quantum codes," *Quantum Information and Computation*, vol. 8, p. 987–1000, 2008.

[Pea82] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Proceedings of the Second AAAI Conference on Artificial Intelligence.* AAAI Press, 1982, p. 133–136.

[PK21] P. Panteleev and G. Kalachev, “Degenerate quantum ldpc codes with good finite length performance,” *Quantum*, vol. 5, p. 585, 2021.

[ROJ19] A. Rigby, J. C. Olivier, and P. Jarvis, “Modified belief propagation decoders for quantum low-density parity-check codes,” *Phys. Rev. A*, vol. 100, p. 012330, Jul 2019.

[RV21] N. Raveendran and B. Vasić, “Trapping sts of quantum LDPC codes,” *Quantum*, vol. 5, p. 562, 2021.

[RWBC20] J. Roffe, D. R. White, S. Burton, and E. Campbell, “Decoding across the quantum low-density parity-check code landscape,” *Phys. Rev. Research*, vol. 2, p. 043423, 2020.

[Sha48] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[Sho94] P. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[Sho95] P. W. Shor, “Scheme for reducing decoherence in quantum computer memory,” *Phys. Rev. A*, vol. 52, pp. R2493–R2496, 1995.

[SLG07] E. Sharon, S. Litsyn, and J. Goldberger, “Efficient serial message-passing schedules for LDPC decoding,” *IEEE Transactions on Information Theory*, vol. 53, no. 11, pp. 4076–4091, 2007.

[TZ14] J.-P. Tillich and G. Zémor, “Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength,” *IEEE Transactions on Information Theory*, vol. 60, pp. 1193–1202, 2014.

[YHB04] M. Yazdani, S. Hemati, and A. Banihashemi, “Improving belief propagation on graphs with cycles,” *IEEE Communications Letters*, vol. 8, no. 1, pp. 57–59, 2004.

[ZF05] J. Zhang and M. Fossorier, “Shuffled iterative decoding,” *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, 2005.