Quantum Error Correction

Decoders for the Toric Code

Nando Leijenhorst, 4582640

Bachelor Thesis,
Applied Mathematics and Applied Physics BSc,
Delft University of Technology

Delft, July 3, 2019
Supervisors:
Dr. M.P.T. Caspers
Dr. D. Elkouss Coronas

Abstract

Quantum error correction is needed for future quantum computers. Classical error correcting codes are not suitable for this due to the nature of quantum mechanics. Therefore, new codes need to be developed. A promising candidate is the toric code, a surface code, because of its locality and its high error correcting capability and thresholds (the error probability below which increasing the size of the code decreases the failure rate).

This thesis provides an introduction to quantum error correction and the stabilizer formalism, which is used to introduce the toric code. Several decoders are looked at, including the Minimum Weight Perfect Matching (MWPM) decoder and a recently developed decoder, the Union-Find (UF) decoder [1]. The UF decoder is useful because of its almost-linear time complexity, while only reducing the threshold by a marginal amount compared to the MWPM decoder. In this thesis, the time complexity of the weighted growth version of the UF decoder is analysed.

The toric code and the decoders have been implemented and simulated, and the thresholds have been determined. For the MWPM decoder, the threshold was determined to be $11.5 \pm 0.2\%$, which is not in agreement with the threshold in literature of 10.3% [2], and should not be possible according to the optimal threshold of about 11%. This probably is a result of the low grid sizes (up to a gridsize of 11) of the code used due to the time needed to simulate the MWPM decoder. For the UF decoder, the threshold found was $9.7 \pm 0.9\%$, which is in agreement with the threshold of 9.9% found by N. Delfosse and N. Nickerson [1]. The time complexity of the UF decoder has also been determined, and is indeed almost-linear as expected by the analysis.

Contents

1. Introduction

In the recent years, quantum computing has been developing quickly. Algorithms for quantum computers have been developed, research centres all over the world are developing qubits to make a reliable quantum computer.

A good quantum computer requires speed and accuracy. A problem related to the accuracy is the noise. As the environment can destroy the states in a quantum computer, one would like to have a completely isolated system. However, one would also like to do computations, which need interactions from the environment. It is unavoidable at the moment that this brings errors in the computer [3]. As accuracy is needed for reliable results, these errors need to be corrected.

A good option for codes to correct the errors are surface codes. In this thesis, the toric code of Kitaev [4] (a surface code) will be examined. Some of the properties which make surface codes a good option are the locality (only interactions with neighbouring qubits) and the high thresholds (the error probability below which increasing the size of the code decreases the failure rate).

A decoder is needed to find a correction for an error. Of course, the threshold depends on the decoder, as the decoder determines how to correct an error. As still speed is required, the time complexity of the decoders is an important factor to consider. In this thesis, a recently developed decoder is looked at, with almost-linear time complexity while keeping a high threshold. This decoder is called the Union-Find decoder, a reference to the used Union-Find data-structure algorithm [5]. In the original article, pseudo code for one version of the decoder is given, but it is said that a different version (weighted growth) has a better performance, with the main difference mentioned. In this thesis the weighted growth version is implemented, and the time complexity is analysed and simulated. The peeling decoder, needed in the Union-Find decoder, is also implemented.

In order to introduce the toric code and the Union-Find decoder, first an introduction in the mathematics and quantum mechanics is given. General error correcting codes and the errors they can correct are considered. In Chapter 3, the stabilizer formalism will be introduced, which will be used in Chapter 4 to define the toric code, where we will also see the decoders. After that, in Chapter 5, results of simulations are presented, to examine the error correcting capabilities and the time complexity of the decoders.

This Bachelor thesis has been written as part of the double degree Applied Mathematics and Applied Physics at Delft University of Technology.

2. Theory of Quantum Error Correction

In this chapter, the basic definitions which will be used in other parts of the thesis will be discussed.

In order to be able to talk about quantum error correction, first the needed mathematical tools and the basis of quantum mechanics is discussed. After that, an introduction to quantum computation will be given, as we need to understand what should ideally happen with the states. For quantum error correction, a model for the errors is needed. For this we need to understand quantum channels, introduced in section 2.4. With the channels, we can give specific examples of error channels, like the bit and phase flip channels.

That is the basis on which the quantum mechanics part of quantum error correction is built. We will look at the basics of classical error correction, to see what the difference is with quantum error correction, after which the quantum error correction is fully introduced.

2.1 Linear Algebra

In this thesis, the reader is assumed to know the basics of linear algebra, such as eigenvalues/vectors, projectors, the inner and outer product, and hermitian and unitary operators. In this section, we will first introduce the notation which is used in the thesis. Then the tensor product will be introduced, after which this section will finish with the trace of an operator.

2.1.1 Notation

In this thesis, we will often use the bra-ket notation of Dirac to denote the state vector of a system. In this notation, a vector ψ is written as $|\psi\rangle$ (the 'ket'). The hermitian conjugate of the vector is denoted as $\langle\psi| = |\psi\rangle^\dagger$ (the 'bra'). For example, if we take vectors in $\mathbb{C}^2$,

$$|\psi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} ,$$
$$\langle\psi| = \begin{pmatrix}\alpha^* \beta^*\end{pmatrix} ,$$

where α^* is the complex conjugate of α. The inner product between $|\phi\rangle$ and $|\psi\rangle$ is denoted as $\langle\phi|\psi\rangle$. For vectors in $\mathbb{C}$, this is an obvious notation, as the standard inner product in $\mathbb{C}$ is defined as

$$\langle\phi|\psi\rangle \equiv (|\phi\rangle, |\psi\rangle) = |\phi\rangle^\dagger |\psi\rangle = \langle\phi| |\psi\rangle , \tag{2.1}$$

where $(\,\cdot\,,\cdot)$ denotes the inner product in $\mathbb{C}$. We will write the hermitian conjugate of an operator A as $A^\dagger$.

2.1.2 The tensor product

The tensor product is widely used in quantum mechanics to couple multiple quantum systems, as will be seen later. In this section, we will not use the braket notations. Before we define the tensor product, the required definitions will be given.

Definition 2.1. Let V be a vector space, and let $K \subseteq V$ be a linear subspace of V. Then for $v \in V$, the coset $v + K$ is defined as

$$v + K = \{v + w | w \in K\}. \tag{2.2}$$

Thus a coset is a way to identify multiple vectors as one. For example, let V be $\mathbb{C}^2$, and $K = \{\lambda(1,0)^T | \lambda \in \mathbb{C}\}$, with ^{T} being the transpose. Then $(1,1)^T + K$ is the same set as $(3,1)^T + K$, as are all vectors given by $(\lambda, 1)^T$ with $\lambda \in \mathbb{C}$. Thus we identify each (complex) plane as one element.

Definition 2.2. Let V be a vector space. Let $K \subseteq V$ be a linear subspace of V. The *quotient space* V/K is the vector space given by

$$V/K = \{v + K | v \in V\}, \tag{2.3}$$

with addition and multiplication defined as

$$(v + K) + (w + K) = (v + w) + K,$$
$$\lambda(v + K) = (\lambda v) + K.$$

A quotient space can be used to reduce the dimension of a vector space, by using the cosets defined above. Let us continue with our example. $\mathbb{C}^2/K$ is the vector space in which each plane is one element. As the first element 'does not matter' anymore after taking the quotient space, $\mathbb{C}^2/K$ is isomorphic to $\mathbb{C}$. Thus we reduced the dimension of the vector space by taking a quotient space.

Now we work toward the tensor product by making a large vector space F, of which we will use a specific quotient space to get the properties we need.

Definition 2.3. Let V and W vector spaces. Let F be the vector space spanned by basis vectors (v, w) for $v \in V$ and $w \in W$. Thus

$$F = Span\{(v, w) | v \in V, w \in W\}. \tag{2.4}$$

For example, take $V = \mathbb{C}^2$ and $W = \mathbb{C}^3$. Then every combination of a vector in V and W is a *basis* vector for F. Note that this is a very large vector space, as $(v_1, w) + (v_2, w) \neq (v_1 + v_2, w)$.

Definition 2.4. Let $K \subseteq F$ be the subspace spanned by the following vectors:

$$\forall v_1, v_2 \in V, w \in W : (v_1, w) + (v_2, w) - (v_1 + v_2, w), \tag{2.5}$$
$$\forall v \in V, w_1, w_2 \in W : (v, w_1) + (v, w_2) - (v, w_1 + w_2), \tag{2.6}$$
$$\forall \lambda \in \mathbb{C}, v \in V, w \in W : \lambda(v, w) - (\lambda v, w), \tag{2.7}$$
$$\forall \lambda \in \mathbb{C}, v \in V, w \in W : \lambda(v, w) - (v, \lambda w). \tag{2.8}$$

Now we have all tools to define the tensor product between two vector spaces V and W.

Definition 2.5. Let V and W be vector spaces. When using the definitions of F and K as above, the tensor product between V and W is

$$V \otimes W = F/K\,. \tag{2.9}$$

The elements are denoted as $v \otimes w \equiv (v, w) + K$. $v \otimes w$ is called an elementary tensor.

Let us now try to get some properties of the tensor product. Let $\lambda \in \mathbb{C}, v, v_1, v_2 \in V$, and $w \in W$. Using the vectors in K, defined in equation 2.5 and 2.7, we get

$$\begin{aligned} v_1 \otimes w + v_2 \otimes w &= ((v_1, w) + K) + ((v_2, w) + K) \\ &= (v_1, w) + (v_2, w) + K \\ &= (v_1, w) + (v_2, w) - ((v_1, w) + (v_2, w) - (v_1 + v_2, w)) + K \\ &= (v_1 + v_2, w) + K \\ &= (v_1 + v_2) \otimes w\,, \end{aligned}$$

and

$$\begin{aligned} \lambda(v \otimes w) &= \lambda((v, w) + K) \\ &= \lambda(v, w) + K \\ &= \lambda(v, w) - (\lambda(v, w) - (\lambda v, w)) + K \\ &= (\lambda v, w) + K \\ &= (\lambda v) \otimes w\,. \end{aligned}$$

So the tensor product is linear in the first argument. Repeating this for w gives us linearity in the second argument.

As the superposition principle is often used in quantum physics, the linearity of the tensor product is very useful.

Definition 2.6. Let V and W be vector spaces, with linear operators A and B acting on V and W respectively. Then we define

$$(A \otimes B)(v \otimes w) = Av \otimes Bw\,, \tag{2.10}$$

and, extending the definition to linear combinations of elementary tensors

$$(A \otimes B) \sum_i a_i v_i \otimes w_i = \sum_i a_i A v_i \otimes B w_i\,, \tag{2.11}$$

and to linear combinations of operators

$$\big(\sum_i c_i A_i \otimes B_i\big)(v \otimes w) = \sum_i c_i A_i v \otimes B_i w\,. \tag{2.12}$$

As we often use $\mathbb{C}^n$ as a vector space, we introduce the Kronecker product, which assigns a matrix to a tensor product of matrices.

Definition 2.7. Let A be a n by m matrix, and B a p by q matrix. Then the Kronecker product is

$$A \otimes B = \begin{pmatrix} A_{11}B & A_{12}B & \dots & A_{1m}B \\ A_{21}B & A_{22}B & \dots & A_{2m}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1}B & A_{n2}B & \dots & A_{nm}B \end{pmatrix} . \tag{2.13}$$

The Kronecker product can be used to write out tensor product in a more intuitive way. Equation 2.13 gives a way to represent the tensor product of operators as a matrix.

2.1.3 The trace

In this section, we will look at the trace of an operator, which will be used for the quantum measurements. From here on, the braket notation is be used.

Definition 2.8. Let V be a vector space and $A : V \to V$ a linear operator. Let $\{|i\rangle\}$ be any orthonormal basis for V. The trace of A is

$$\mathrm{Tr}(A) = \sum_i \langle i|A|i\rangle \ . \tag{2.14}$$

Theorem 2.1. $\mathrm{Tr}(A)$ *is basis-independent.*

The proof is added to the appendix, as it has no real significance. Note that this theorem means that we can choose the basis which suits us best. Thus when calculating $\mathrm{Tr}(A\,|v_1\rangle\langle v_2|)$, we can include $|v_2\rangle$ in the orthogonal basis. Then $\mathrm{Tr}(A\,|v_1\rangle\langle v_2|) = \sum_i \langle e_i|A|v_1\rangle\,\langle v_2|e_i\rangle = \langle v_2|A|v_1\rangle\,\langle v_2|v_2\rangle$. If $|v_2\rangle$ is a unit vector, which will often be the case, this reduces to $\langle v_2|A|v_1\rangle$.

Consider two vector spaces V and W, and their tensor product $V \otimes W$. We may want to go back to either of the vector spaces. This can be done with the partial trace.

Definition 2.9. Let V and W be vector spaces, and let $|v_i\rangle \in V, |w_i\rangle \in W$ for $i = 1, 2$. Then the partial trace is defined by

$$\mathrm{Tr}_W(|v_1\rangle\langle v_2| \otimes |w_1\rangle\langle w_2|) = |v_1\rangle\langle v_2|\,\mathrm{Tr}(|w_1\rangle\langle w_2|)\,. \tag{2.15}$$

This is then extended in a linear way as a map $V \otimes W \to V$.

The partial trace is well-defined, which follows from the relations of the tensor product (equation (2.5) - (2.8)).

2.1.4 The spectral decomposition theorem

An important theorem in Linear Algebra is the Spectral Decomposition Theorem. It will be stated here because we will use it often, but as it is considered to be known; we will not prove it.

Theorem 2.2. *Let A be an operator on vector space V. Then A is normal if and only if there exist a basis $|i\rangle$ for V such that A is diagonal with respect to this basis.*

This theorem means that there are λ_i and $|i\rangle$ such that $A = \sum_i \lambda_i |i\rangle\langle i|$ if and only if A is normal.

With this theorem, there is a natural way to define scalar functions on operators. For polynomials we have: $A^n = \sum_i \lambda_i^n |i\rangle\langle i|^n = \sum_i \lambda_i^n |i\rangle\langle i|$, and $\alpha A = \sum_i \alpha\lambda_i |i\rangle\langle i|$, thus for a general polynomial p, we have $p(A) = \sum_i p(\lambda_i) |i\rangle\langle i|$. If we extend this to other functions, this becomes for a general function f

$$f(A) = \sum_i f(\lambda_i) |i\rangle\langle i| \tag{2.16}$$

Of course λ_i should be in the domain of f, thus for example $\sqrt{A}$ is only valid for positive, normal operators A.

2.1.5 Commutators

For scalars α and β, it is true that $\alpha\beta = \beta\alpha$. For operators, this is not always the case. To handle this, we define the commutator and anticommutator of two operators A and B:

Definition 2.10. Let A and B be operators. The commutator of A and B is defined as

$$[A, B] = AB - BA\,. \tag{2.17}$$

and the anticommutator of A and B as

$$\{A, B\} = AB + BA\,. \tag{2.18}$$

If $[A, B] = 0$, A and B are said to commute. Similarly, if $\{A, B\} = 0$, A and B are said to anticommute.

2.2 Postulates of Quantum Mechanics

In this section, the postulates of quantum mechanics will be introduced. We will use a description of the postulates similar to the one in the book of Nielsen and Chuang [6].

Postulate 1. Let A be an isolated system. Then we associate a complex vector space with inner product to A, which is called the *state space*. The system is described by the *state vector*, which is a unit vector in the state space.

As can be seen, this postulate does not state which complex inner product space should be used for an isolated system. Thus we can use any suitable inner product space $\mathcal{H}$, if we find a way to represent all possible states as unit vectors in $\mathcal{H}$. This means that we need an inner product space with the same dimension as our system.

An example of a system is a qubit. A classical bit can be in two states, $|0\rangle$ or $|1\rangle$. A qubit can be in a superposition of two states, i.e. $\alpha|0\rangle + \beta|1\rangle$. However, as we need a unit vector, $|\alpha|^2 + |\beta|^2 = 1$. Note that this can be described as a vector in $\mathbb{C}^2$.

Later in this thesis, we will use a subspace of the state space of a large number of qubits to encode a small number of qubits.

However, if we can only describe a state, we cannot do any computations. For that, the second postulate is needed: what happens over time with the state of a system.

Postulate 2. The *evolution* of an isolated system is described by a *unitary transformation*. That is, the state at a time t_1, $|\psi\rangle$, is related to the state at a time t_2,$\left|\psi'\right\rangle$, by a unitary transformation U, which may only depend on t_1 and t_2:

$$\left|\psi'\right\rangle = U\,|\psi\rangle\ . \tag{2.19}$$

Why is the transformation needed to be unitary? First of all, the resulting state vector needs to be a unit vector as well. So for all states $|\psi\rangle$,

$$\left\langle\psi'\middle|\psi'\right\rangle = \langle\psi|U^\dagger U|\psi\rangle = 1 = \langle\psi|\psi\rangle\ .$$

As this holds for all $|\psi\rangle$, $U^\dagger U = I$, thus U is unitary.

Before we have only seen how a state is described. But in quantum mechanics, we cannot always observe the exact state. This is captured by Postulate 3:

Postulate 3. Let A be an isolated system with state space $\mathcal{H}$. A measurement is described by a set of measurement operators $\{M_m\}$, which act on $\mathcal{H}$. An outcome m is associated with each measurement operator M_m. Given a state $|\psi\rangle$ before the measurement, the probability to get outcome m is

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle\ , \tag{2.20}$$

and given that outcome m occurred, the post-measurement state $\left|\psi'\right\rangle$ is given by

$$\left|\psi'\right\rangle = \frac{M_m\,|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}\,. \tag{2.21}$$

The measurement operators satisfy the completeness relation:

$$\sum_m M_m^\dagger M_m = I\,. \tag{2.22}$$

Note that the probabilities sum to 1, because of the completeness relation:

$$\sum_m p(m) = \sum_m \langle\psi|M_m^\dagger M_m|\psi\rangle = \langle\psi|\sum_m M_m^\dagger M_m|\psi\rangle = \langle\psi|\psi\rangle = 1\,.$$

A special case of measurements which is often used is the projective measurement. Let M be a Hermitian operator (an *observable*). Let $\{P_m\}$ be the projectors on the eigenspaces with eigenvalue m, then $M = \sum_m mP_m$. As P_m are projectors, $P_m^\dagger = P_m$

and $P_m^2 = P_m$. Thus when using these projectors in Postulate 3, the probability to get outcome m is

$$p(m) = \langle\psi|P_m|\psi\rangle \ , \tag{2.23}$$

and the post measurement state, given that outcome m occurred, is given by

$$\left|\psi'\right\rangle = \frac{P_m \left|\psi\right\rangle}{\sqrt{p(m)}}. \tag{2.24}$$

Examples of observables for a qubit are the Pauli matrices. Take for example the Pauli-Z matrix as M. Then in the computational basis,

$$\begin{aligned} Z &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\ &= 1 \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} - 1 \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \\ &= 1 \left|0\right\rangle\!\left\langle 0\right| - 1 \left|1\right\rangle\!\left\langle 1\right| \ . \end{aligned}$$

Let now $|\phi\rangle = \alpha \left|0\right\rangle + \beta \left|1\right\rangle$. Then

$$p(1) = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \left|\alpha\right|^2 .$$

Similarly, $p(-1) = \left|\beta\right|^2$. And when outcome 1 occurs, the post measurement state is

$$\frac{\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}}{\sqrt{p(1)}} = \frac{\alpha}{|\alpha|} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{\alpha}{|\alpha|} \left|0\right\rangle \ .$$

Postulate 3 is vital for quantum error correction, as it provides a way to measure what errors happened.

For quantum error correction, we need to couple multiple qubits to each other. This is captured in Postulate 4, which incorporates the tensor product.

Postulate 4. The state space of a composite system is the tensor product of the state spaces of the component systems. Let systems 1 to n be in states $\left|\psi_1\right\rangle, \ldots, \left|\psi_n\right\rangle$. Then the state of the composite system is $\left|\psi_1\right\rangle \otimes \left|\psi_2\right\rangle \otimes \cdots \otimes \left|\psi_n\right\rangle$

As the notation with $\otimes$ might get tedious sometimes, we often denote $|\psi\rangle \otimes |\phi\rangle$ as $\left|\psi\right\rangle \left|\phi\right\rangle$ or as $|\psi\phi\rangle$.

2.2.1 The density matrix

Instead of using state vectors, we can also describe every Postulate with density operators or density matrices. A density matrix describes a statistical state of a system. That is, there are different states in which the system can be, each with a certain probability. A useful concept is the ensemble of states, which is a way to define in which statistical state a system is.

Definition 2.11. Suppose a system can be in the states $|\psi_i\rangle$, each with probability p_i, such that $\sum_i p_i = 1$. Then the ensemble of states of this system is $\{p_i, |\psi_i\rangle\}_i$.

With the ensemble of states, we can define the corresponding density matrix.

Definition 2.12. Let $\{p_i, |\psi_i\rangle\}_i$ be an ensemble of states of a system. Then the density operator (or density matrix) for this ensemble is given by

$$\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i| \, . \tag{2.25}$$

As an evolution of a state is given by a unitary operator U (Postulate 2), we can write an evolution of the density matrix as

$$\rho' = \sum_i p_i U \, |\psi_i\rangle\langle\psi_i| \, U^\dagger \, . \tag{2.26}$$

Similarly, we can rewrite the probability in the third Postulate:

$$p(m) = \mathrm{Tr}\Big(M_m^\dagger M_m \rho\Big) \, , \tag{2.27}$$

and the post measurement density matrix

$$\rho_m = \frac{M_m \rho M_m^\dagger}{\mathrm{Tr}\Big(M_m^\dagger M_m \rho\Big)} \, . \tag{2.28}$$

The following theorem gives two characteristics of a density matrix.

Theorem 2.3. *An operator ρ is the density matrix of an ensemble $\{p_i, |\psi_i\rangle\}$ if and only if it satisfies the following criteria:*

1. $\mathrm{Tr}(\rho) = 1$
2. *ρ is positive*

Proof. Let ρ be the density matrix of $\{p_i, |\psi_i\rangle\}_i$. Then $\mathrm{Tr}(\rho) = \sum_i p_i \, \mathrm{Tr}(|\psi_i\rangle\langle\psi_i|) = \sum_i p_i = 1$. And for any $|\phi\rangle$, $\langle\phi|\rho|\phi\rangle = \sum_i p_i \, \langle\phi|\psi_i\rangle \, \langle\psi_i|\phi\rangle = \sum_i p_i |\langle\psi_i|\phi\rangle|^2 \geq 0$.
Let ρ now satisfy the two criteria. Then $\rho = \sum_i \lambda_i \, |i\rangle\langle i|$ by the spectral decomposition theorem, and $\sum_i \lambda_i = \mathrm{Tr}(\rho) = 1$. Because ρ is positive, the λ_i are non-negative. So $0 \leq \lambda_i \leq 1$, and thus ρ can be seen as an density matrix for the ensemble $\{\lambda_i, |i\rangle\}$. □

After coupling two systems with ensembles $\{p_i, |\psi_i\rangle\}_i$ and $\{p_j, |\psi_j\rangle\}_j$, the resulting system has the ensemble $\{p_i p_j, |\psi_i\rangle \otimes |\psi_j\rangle\}$). This means that the resulting density matrix ρ can be written as the tensor product of the original density matrices ρ_i and ρ_j: $\rho_i \otimes \rho_j$. Note that we can then use the partial trace to go back to uncoupled systems, as $\mathrm{Tr}_B(\rho_A \otimes \rho_B) = \rho_A \, \mathrm{Tr}(\rho_B) = \rho_A$ for product states. For mixed states, it is less clear that the partial trace is the right operation. Suppose we have coupled systems A and B with ensemble of states $\{p_i, |\psi_i\rangle\}$. The partial trace essentially takes each $|\psi_i\rangle$ of the coupled system, and assigns a probability to the part in system A of this state, which is the trace of the part of system B. This is what is wanted, as there is no extra information about system B, so the probability for system B to be in a certain state is proportional to the trace. Of course we can do this multiple times to couple more than two systems.

2.3 Computation

Now that we have discussed the basics of quantum mechanics, we will start with how we describe computations. Classically, we compute things with *bits*, which can be either 0 or 1. Quantum mechanically, we can have states which represent the classical 0 or 1, but also states which are a superposition of those states, as seen in some examples before. We can define the state vector of a *qubit* as follows

Definition 2.13. The state vector of a qubit is given by a unit vector in $\mathbb{C}^2$.

The two standard basis vectors of $\mathbb{C}^2$ are usually denoted as $|0\rangle$ and $|1\rangle$, thus then the state of a qubit is $\alpha|0\rangle + \beta|1\rangle$. An example of a physical system with two basis vectors is an electron, which can have spin up ($|1\rangle$) or spin down ($|0\rangle$), or a superposition of both. There are many ways to make qubits, but these are not considered in this thesis.

To do quantum computations, we need to manipulate the qubit. As the second Postulate says, this can be done by applying unitary operations. A few important unitary operations on one qubit are the Pauli gates, which will be used often later on. They are defined as follows:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{2.29}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \tag{2.30}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.31}$$

The identity matrix I is sometimes also meant when referring to the Pauli gates. Here the standard basis vectors are the computational basis, $e_1 = |0\rangle$ and $e_2 = |1\rangle$. It can easily be checked that the Pauli gates are unitary. Note that we can rewrite any 2×2 matrix as a linear combination of the Pauli matrices (including I). Specifically, we can write a density matrix ρ as

$$\rho = \frac{1}{2}(I + a_1 X + a_2 Y + a_3 Z), \tag{2.32}$$

with $a_i \in \mathbb{C}$ for $i \in \{1, 2, 3\}$. An important property of the Pauli matrices is that they either commute or anticommute. For example, $[X, Z] = 2iY$, but $\{X, Z\} = 0$.

Another important gate is the controlled NOT gate, or the CNOT gate. This is a two-qubit gate, where the basis vectors of the second qubit (the target qubit) are flipped for the $|1\rangle$ part of the first qubit (the control qubit). With multiple qubits, the basis vectors are the tensor products of the standard basis vectors of one qubit. For example:

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = e_3. \tag{2.33}$$

Similarly, $|00\rangle = e_1$, $|01\rangle = e_2$ and $|11\rangle = e_4$, where e_i is the i-th standard basis vector of $\mathbb{C}^4$. With respect to this basis, the CNOT gate is then:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.34}$$

As can be seen, $|00\rangle$ and $|01\rangle$ are mapped to $|00\rangle$ and $|01\rangle$, whereas $|1i\rangle$ is mapped to $|1\rangle \otimes X|i\rangle$ ($i \in \{0,1\}$). In the same way we can make a controlled U operator, by replacing X by U. The matrix representation becomes then

$$cU = \begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{11} & u_{12} \\ 0 & 0 & u_{21} & u_{22} \end{pmatrix}, \tag{2.35}$$

where u_{ij} are the matrix elements of U.

After doing the computations, we may want to measure the outcome of the computations. This is often done by measuring in the computational basis, with projectors P_0 and P_1 defined below.

$$P_0 = |0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \tag{2.36}$$

$$P_1 = |1\rangle\langle 1| = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.37}$$

Note that $Z = P_0 - P_1$, thus measuring in the computational basis is the same as measuring the Pauli Z operator. As seen in section 2.2, measuring $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ results in $|0\rangle$ or $|1\rangle$, multiplied with a constant, with probability $|\alpha|^2$ and $|\beta|^2$ respectively.

2.4 Quantum Channels

In this section, we will generalise Postulate 2, which introduced the evolution of a quantum state. In order to do this, the density matrices will be used instead of the state vectors. We will denote a quantum channel as Φ, and the result after a quantum channel is applied on a state ρ by $\Phi(\rho)$.

Quantum channels will be used to describe the errors on qubits, and to provide a theorem about which errors can be corrected.

At first, we will show what to expect of an evolution for an open system. This is one of the possible definitions of a quantum channel, and leads easily to the Choi-Krauss representation. However, we will use a different definition, which will be shown to be equivalent.

Let us consider an open system, instead of a closed system like before. Then we can regard this as the system itself, A, together with the environment, E. Thus the total system is $A \otimes E$, which is a closed system. So when using Postulate 2, an evolution of

$\rho \otimes \rho_e$ is given by $U(\rho \otimes \rho_e)U^\dagger$. But if we only want to consider the original system, we need to take the partial trace as stated in section 2.2.1. So the resulting state is $\Phi(\rho) = \mathrm{Tr}_E(U(\rho \otimes \rho_e)U^\dagger)$.

If we now assume that the environment is in a pure state, $|e_0\rangle$, we can extend this to a orthogonal basis for the environment, $\{|e_k\rangle\}$. Using this basis, we can rewrite

$$\begin{aligned}
\Phi(\rho) &= \mathrm{Tr}_E(U(\rho \otimes \rho_e)U^\dagger) && (2.38)\\
&= \sum_k \Big(I \otimes \langle e_k| \cdot |e_k\rangle \Big)(U(\rho \otimes |e_0\rangle\langle e_0|)U^\dagger) && (2.39)\\
&= \sum_k \Big(I \otimes \langle e_k| \cdot |e_0\rangle \Big)(U)\rho\Big(I \otimes \langle e_0| \cdot |e_k\rangle \Big)(U^\dagger) && (2.40)\\
&= \sum_k E_k \rho E_k^\dagger \,. && (2.41)
\end{aligned}$$

Here, $E_k = \Big(I \otimes \langle e_k| \cdot |e_0\rangle \Big)(U)$. With this notation, we mean that I is applied on the first vector space of the tensor product, our first system, and that $\langle e_k| \cdot |e_0\rangle$ is applied on the second vector space of the tensor product, the environment. This is the Choi-Krauss representation, also known as the operator-sum representation.

Let us now define a quantum channel in a different way. What should be expected of a quantum channel? First of all, we would want it to take density operators to density operators. Recalling the criteria for being a density matrix, this translates to preserving the trace (as $\mathrm{Tr}(\Phi(\rho)) = 1 = \mathrm{Tr}(\rho)$) and the positivity of operators (ρ positive implies that $\Phi(\rho)$ is positive). However, if we want to be able to represent measurements as quantum channels, we can let the trace of the resulting state represent the probability that process represented by the quantum channel occurs, thus $0 \leq \mathrm{Tr}(\Phi(\rho)) \leq 1$.

Another thing we would expect of a quantum channel is that it does not matter whether we make an ensemble of states before or after applying the quantum channel. That is, it should be a convex-linear map:

$$\Phi(\sum_i p_i \rho_i) = \sum_i p_i \Phi(\rho_i) \,. \tag{2.42}$$

The last thing we need has to do with coupling systems. Let ρ_1 and ρ_2 be density matrices of two systems, and let Φ be a quantum channel defined on the vector space of ρ_1. Then we would expect that if we apply the quantum channel after coupling the systems would give a valid density matrix. Thus if we denote $\rho = \rho_1 \otimes \rho_2$, then $(\Phi \otimes I)\rho = \Phi(\rho_1) \otimes \rho_2$ should be a valid density matrix. This is the same as requiring that Φ should be a completely positive map.

This is summarized in the following definition.

Definition 2.14. Let Φ be a map from set of density matrices of the input space Q_1 to the set of density matrices of the output space Q_2. Then Φ is a quantum channel if it has the following properties:

1. $\mathrm{Tr}(\Phi(\rho))$ is the probability that the process represented by Φ occurs. Thus $0 \leq \mathrm{Tr}(\Phi(\rho)) \leq 1$ for any density matrix $\rho \in Q_1$.

2. Φ is a convex linear map on the set of density matrices. Thus for probabilities $\{p_i\}$:

$$\Phi(\sum_i p_i \rho_i) = \sum_i p_i \Phi(\rho_i)\,.$$

3. Φ is a completely positive map. That is, for any system R coupled to Q, $\big(I \otimes \Phi\big)A$ is positive for any positive operator A on $R \otimes Q$. Here I is the identity map on R.

The next theorem shows that this definition is equivalent with the operator-sum representation [6, Theorem 8.1] [7, Theorem 4.8].

Theorem 2.4. *Let Φ be a map on Q. Then Φ is a quantum channel if and only if there exist $\{E_k\}_k$ such that*

$$\Phi(\rho) = \sum_k E_k \rho E_k^\dagger\,, \tag{2.43}$$

with $\sum_k E_k^\dagger E_k \leq I$

Proof. Let $\Phi(\rho) = \sum_k E_k \rho E_k^\dagger$, with $\sum_k E_k \dagger E_k \leq I$. It can easily be seen that Φ is linear. We also have $\mathrm{Tr}(\Phi(\rho)) = \mathrm{Tr}\Big(\sum_k E_k^\dagger E_k \rho\Big) \leq \mathrm{Tr}(\rho) = 1$. So to prove that Φ is a quantum channel, we only have to prove that Φ is completely positive.

Let $|\psi\rangle$ be a state of the combined system $R \otimes Q$ and assume A to be a positive operator on $R \otimes Q$. Define $|\phi_k\rangle = (I \otimes E_k^\dagger)\,|\psi\rangle$. This is also a state of $R \otimes Q$. Then we have

$$\begin{aligned}
\langle\psi|(I \otimes \Phi)A|\psi\rangle &= \langle\psi|\sum_k (I \otimes E_k)A(I \otimes E_k^\dagger)|\psi\rangle && (2.44)\\
&= \sum_k \langle\psi|(I \otimes E_k)A(I \otimes E_k^\dagger)|\psi\rangle && (2.45)\\
&= \sum_k \langle\phi_k|A|\phi_k\rangle \geq 0\,. && (2.46)
\end{aligned}$$

So Φ is completely positive. Thus Φ is a quantum channel.

Now let Φ be a quantum channel. We will construct $\{E_k\}_k$. Let $\{|e_i\rangle\}_{i=1}^n$ be a basis for Q. Define $B_{i,j} = \big|e_i\big\rangle\!\big\langle e_j\big|$, and $T = \sum_{i,j} B_{i,j} \otimes B_{i,j}$. Then we have

$$T^2 = \sum_{i,j,k,l} (B_{i,j} \otimes B_{i,j})(B_{k,l} \otimes B_{k,l}) = \sum_{i,j,l} B_{i,j}B_{j,l} \otimes B_{i,j}B_{j,l} = n \sum_{i,l} B_{i,l} \otimes B_{i,l} = nT\,. \tag{2.47}$$

Thus let $|\phi\rangle$ be any eigenstate of T with eigenvalue λ. Then

$$\lambda^2\,|\phi\rangle = T^2\,|\phi\rangle = nT\,|\phi\rangle = n\lambda\,|\phi\rangle\ . \tag{2.48}$$

So $\lambda^2 = n\lambda$, which means that $\lambda \in \{0, n\}$ for all eigenvalues. Thus T is positive. As Φ is completely positive, $\Phi(T) = \sum_{i,j} B_{i,j} \otimes \Phi(B_{i,j})$ is also positive. By the spectral decomposition theorem, we have $\Phi(T) = \sum_k |\psi_k\rangle\!\langle\psi_k|$, where $|\psi_k\rangle$ do not need to be

normalized. Note that because $|\psi_k\rangle \in Q \otimes Q$, we can write $|\psi_k\rangle = \sum_i |e_i\rangle \otimes \left|h_i^k\right\rangle$. Now define $E_k = \sum_i \left|h_i^k\right\rangle\!\left\langle e_i\right|$.

We claim that $\Phi(\rho) = \sum_k E_k \rho E_k^\dagger$. Note that we have $|\psi_k\rangle\langle\psi_k| = \sum_{i,j} \left|e_i\right\rangle\!\left\langle e_j\right| \otimes \left|h_i^k\right\rangle\!\left\langle h_j^k\right| = \sum_{i,j} B_{i,j} \otimes \left|h_i^k\right\rangle\!\left\langle h_j^k\right|$. Thus $\Phi(B_{i,j}) = \sum_k \left|h_i^k\right\rangle\!\left\langle h_j^k\right|$ for all i and j. So now we have for all $B_{i,j}$

$$\sum_k E_k B_{i,j} E_k^\dagger = \sum_{i,j,k} \left|h_i^k\right\rangle\!\left\langle e_i\right| B_{i,j} \left|e_j\right\rangle\!\left\langle h_j^k\right| = \sum_k \left|h_i^k\right\rangle\!\left\langle h_j^k\right| = \Phi(B_{i,j})\,. \tag{2.49}$$

As Φ is linear, we now have for any state $\rho = \sum_{i,j} \rho_{ij} B_{i,j}$ that

$$\Phi(\rho) = \Phi(\sum_{i,j} \rho_{ij} B_{i,j}) = \sum_{i,j} \rho_{ij} \Phi(B_{i,j}) = \sum_{ij} \rho_{i,j} \sum_k E_k B_{i,j} E_k^\dagger = \sum_k E_k \rho E_k^\dagger\,. \tag{2.50}$$

Thus our claim is true.

Lastly, we need to check that $\sum_k E_k^\dagger E_k \leq I$. As $\mathrm{Tr}(\Phi(\rho)) = \mathrm{Tr}\Big(\sum_k E_k \rho E_k^\dagger\Big) = \mathrm{Tr}\Big(\sum_k E_k^\dagger E_k \rho\Big) \leq 1 = \mathrm{Tr}(\rho)$, this condition is also satisfied. □

Note that in the proof, we only found one specific operator-sum representation. However, there are more representations possible, as stated in the next theorem [6, Theorem 8.2] [7, Theorem 4.17 (1), Proposition 4.18].

Theorem 2.5. *Let $\Phi(\rho) = \sum_{k=1}^n E_k \rho E_k^\dagger$ and $\Psi(\rho) = \sum_{l=1}^n F_l \rho F_l^\dagger$ be quantum channels. Then $\Phi(\rho) = \Psi(\rho)$ if and only if there exists a unitary matrix U such that $E_k = \sum_l u_{kl} F_l$ for each k.*

Proof. Suppose $E_k = \sum_l u_{kl} F_l$ for a unitary matrix U. Then we have that

$$\Phi(\rho) = \sum_k E_k \rho E_k^\dagger = \sum_k \sum_l \sum_m u_{kl} F_l \rho u_{km}^* F_m^\dagger = \sum_{k,l,m} u_{kl} u_{km}^* F_l \rho F_m^\dagger \tag{2.51}$$

As U is unitary,

$$\sum_k u_{kl} u_{km}^* = \begin{pmatrix} u_{1m}^* & u_{2m}^* & \dots & u_{nm}^* \end{pmatrix} \begin{pmatrix} u_{1l} \\ u_{2l} \\ \vdots \\ u_{nl} \end{pmatrix} = \langle u_m | u_l \rangle = \delta_{l,m}\,. \tag{2.52}$$

Thus we have

$$\Phi(\rho) = \sum_{l,m} \delta_{l,m} F_l \rho F_m^\dagger = \sum_l F_l \rho F_l^\dagger = \Psi(\rho)\,. \tag{2.53}$$

Let now $\Phi(\rho) = \Psi(\rho)$. Reversing the proof of Theorem 2.4 gives

$$\Phi(T) = \sum_k |\phi_k\rangle\langle\phi_k| = \Psi(T) = \sum_l |\psi_l\rangle\langle\psi_l|\,. \tag{2.54}$$

with $T = \sum B_{i,j} \otimes B_{i,j}$, and $B_{i,j} = \left|e_i\right\rangle\!\left\langle e_j\right|$.

Let $|\sigma\rangle$ be orthonormal to all of the $|\psi_l\rangle$ states, thus $\langle\sigma|\Psi(T)|\sigma\rangle = 0$. Then

$$0 = \langle\sigma|\Psi(T)|\sigma\rangle = \langle\sigma|\Phi(T)|\sigma\rangle = \sum_k \langle\sigma|\phi_k\rangle \langle\phi_k|\sigma\rangle = \sum_k |\langle\sigma|\phi_k\rangle|^2 . \tag{2.55}$$

Thus all $|\sigma\rangle$ which are orthogonal to $|\psi_l\rangle$ are orthogonal to all $|\phi_k\rangle$. Thus we can write $|\phi_k\rangle$ as a linear combination of $|\psi_l\rangle$. Similarly, we can prove that we can write $|\psi_l\rangle$ as a linear combination of $|\phi_k\rangle$. Thus we have

$$|\phi_k\rangle = \sum_l u_{kl} |\psi_l\rangle , \tag{2.56}$$

and

$$|\psi_l\rangle = \sum_k v_{lk} |\phi_k\rangle . \tag{2.57}$$

So

$$|\phi_k\rangle = \sum_l u_{kl} |\psi_l\rangle = \sum_l \sum_m u_{kl} v_{lm} |\phi_m\rangle . \tag{2.58}$$

As $|\phi_k\rangle$ are orthogonal, $\sum_l u_{kl} v_{lm} = \delta_{k,m}$. We also have that $\langle\phi_n|\psi_l\rangle = v_{ln}$. Similarly, as $|\psi_l\rangle$ are orthogonal, $\langle\psi_m|\phi_k\rangle = u_{km}$. Taking this together gives

$$u_{kl} = \langle\psi_l|\phi_k\rangle = \langle\phi_k|\psi_l\rangle^* = v_{lk}^* . \tag{2.59}$$

Thus

$$\delta_{k,m} = \sum_l u_{kl} v_{lm} = \sum_l u_{kl} u_{ml}^* , \tag{2.60}$$

which means that U is unitary. So we have that $|\phi_k\rangle = \sum_l u_{kl} |\psi_l\rangle$. As $|\psi_l\rangle = \sum_i |e_i\rangle \otimes \left|h_i^l\right\rangle$ and $|\phi_k\rangle = \sum_i |e_i\rangle \otimes \left|g_i^k\right\rangle$, we have for each i that $\left|g_i^k\right\rangle = \sum_l u_{kl} \left|h_i^l\right\rangle$. Thus

$$E_k = \sum_i \left|g_i^k\right\rangle = \sum_i \sum_l u_{kl} \left|h_i^l\right\rangle = \sum_l u_{kl} F_l , \tag{2.61}$$

which completes the proof. □

Note that if we have two channels with operator elements $\{E_1, \ldots, E_n\}$ and $\{F_1, \ldots, F_m\}$ (with $n \neq m$), we can use the theorem by appending zero operators (operators which do nothing) to the shorter list to make sure that $n = m$.

2.5 Error Channels

In this subsection, some error channels will be introduced. The errors we will see are the bit flip error, the phase flip error and the erasure error.

2.5.1 The bit flip channel

The intuition one should have for the bit flip channel is that the channel literally flips the $|0\rangle$ state and the $|1\rangle$ state with a certain probability. Suppose we have a state ρ, then the channel can be described as follows. There is a probability p for I to be applied (the state remains the same), and a probability $1-p$ for X to be applied. So the resulting state should be

$$\Phi(\rho) = p\rho + (1-p)(X\rho X) \tag{2.62}$$
$$= p(I\rho I) + (1-p)(X\rho X)\,. \tag{2.63}$$

This can be seen as the operator-sum representation with $E_0 = \sqrt{p}I$ and $E_1 = \sqrt{1-p}X$.

2.5.2 The phase flip channel

The phase flip channel is similar to the bit flip error, except that the phase is flipped with a probability $1-p$ instead of the bits. So the Pauli Z gate is applied instead of the Pauli X gate. Thus

$$\Phi(\rho) = p\rho + (1-p)(Z\rho Z)\,. \tag{2.64}$$

So the operation elements of this channel are $E_0 = \sqrt{p}I$ and $E_1 = \sqrt{1-p}Z$.

2.5.3 The depolarizing channel

The depolarizing channel replaces the state by a completely mixed state with a probability p. This completely mixed state is $\frac{1}{2}I$, so the new state is

$$\Phi(\rho) = (1-p)\rho + p\frac{I}{2} \tag{2.65}$$

However, we need to have the channel as sum of $E_i\rho E_i^\dagger$. It can be checked that

$$\frac{I}{2} = \frac{1}{4}(\rho + X\rho X + Y\rho Y + Z\rho Z)\,. \tag{2.66}$$

See Appendix A for a short proof. Using this, the channel can be rewritten as

$$\Phi(\rho) = (1 - \frac{3}{4}p)\rho + \frac{p}{4}(X\rho X + Y\rho Y + Z\rho Z)\,. \tag{2.67}$$

Or, taking $p' = 3p/4$:

$$\Phi(\rho) = (1-p')\rho + \frac{p'}{3}(X\rho X + Y\rho Y + Z\rho Z)\,. \tag{2.68}$$

This can be interpreted as follows; with a probability $1-p'$, the state is left alone, and each Pauli gate is applied with probability $p'/3$.

2.5.4 The erasure channel

In the erasure channel, a qubit is completely removed from the allowed state space with a probability p. This can be modelled by a channel $\Phi : \mathcal{H}_d \to \mathcal{H}_{d+1}$ as follows:

$$\Phi(\rho) = (1-p)\rho + p\,|e\rangle\langle e|\ , \tag{2.69}$$

with $|e\rangle$ a state orthogonal to all $\rho \in \mathcal{H}_d$. In the qubit case, the dimension d of the input space is 2.

It can be detected when a qubit comes in such a state, after which it can be replaced by a completely mixed state, $I/2$. This can be rewritten using equation 2.66 to obtain a random Pauli error. Note that this channel can be seen as a depolarizing channel with the extra property that the place of the qubits which obtain an error is detected. This is one of the channels which will be used as model for the toric code in chapter 4.

2.6 Error Correction

Before quantum error correction is introduced, a classical example of error correction will be shown, and we will see why this cannot be done in the same way in quantum error correction. After that, we will define what a quantum error correcting code is, and see an example of it. Lastly, a theorem will be given about which errors can be corrected by a code. But before we start with the classical example, some relevant definitions will be given.

First of all, we need to define what a code exactly is.

Definition 2.15. An error correcting code is a linear subspace C of the vector space of all possible bitstrings of length n, the length of the code. The rank k of the code is the dimension of C.

To easily see how much errors a code can correct, we define the weight and the hamming distance of a codeword.

Definition 2.16. The weight of a codeword x is defined as the number of bits that differ from zero, and is denoted as $w(x)$. The Hamming distance between two codewords x and y is the number of elements in which they differ, denoted as $d(x, y)$. The distance of a code C is the minimum Hamming distance between any two codewords in C:

$$d(C) = \min_{x \neq y \in C} d(x, y)\,. \tag{2.70}$$

If a code of length n and rank k has a distance of d, we call it a $[n, k, d]$ code.

For example, the weight of $(0, 1, 0, 1)$ is 2, and the Hamming distance between (0,1,0,1) and (1,1,0,1) is 1. Note that as C is linear, $d(x, y) = d(x + y, 0) = w(x + y)$, where we add modulo 2. And $x + y$ is a codeword if x and y are codewords. Thus the distance of a code C is also the minimum weight of all codewords:

$$d(C) = \min_{x \neq y \in C} d(x, y) = \min_{x \neq y \in C} d(x + y, 0) = \min_{x \in C} w(x)\,. \tag{2.71}$$

The importance of the distance of a code is that it characterises how much errors the code can correct. As we need to map error states e onto the codewords, the best guess would be the codeword with the smallest Hamming distance to the error state. Thus then we would be able to correct all errors on t bits if $d(C) \leq 2t + 1$.

Until now, all definitions have been about classical codes. They can be translated to quantum codes, which will be done later.

2.6.1 The 3-bit repetition code

The classical example we will study is the 3-bit repetition code. Suppose we have a (classical) channel, which can flip a bit with probability p. Let 0 and 1 be encoded by:

$$0 \longmapsto 000 = 0_L \,, \tag{2.72}$$

$$1 \longmapsto 111 = 1_L \,. \tag{2.73}$$

Then if we send one the encoded bits through the channel, say 0_L, we can get one of the following outcomes (allowing for permutations):

state	probability
000	$(1-p)^3\,,$
100	$3p(1-p)^2\,,$
110	$3p^2(1-p)\,,$
111	$p^3\,.$

If we now decode the bits by choosing the bit which occurs most often (majority voting), we get

state	probability
0	$(1-p)^3\,,$
0	$3p(1-p)^2\,,$
1	$3p^2(1-p)\,,$
1	$p^3\,.$

Thus we have the wrong bit with probability $3p^2 - 2p^3$, which is lower than p for $p < \frac{1}{2}$. Thus this is indeed a better way to send the information over the channel, provided that the probability of an error is small enough. As can be seen, the distance of the code is 3, because 0_L differs on 3 places from 1_L. As $d(C) = 3 = 2t + 1$ for $t = 1$, we can indeed only correct an error on 1 bit.

When trying to make a quantum error correcting code out of the 3 bit repetition code, one could naively try the following. Let $|\psi\rangle$ be the state we would like to encode. Classically, we make the encoded state by repeating the bit. So then we have

$$|\psi\rangle \longmapsto |\psi\rangle\,|\psi\rangle\,|\psi\rangle \tag{2.74}$$

We have a problem now, because this is not possible in quantum mechanics. The no-cloning theorem says that we cannot clone a state exactly.

Theorem 2.6. *Then there is no unitary operation U on two qubits such that $U|0\rangle|\psi\rangle = |\psi\rangle|\psi\rangle$ for all one qubit states $|\psi\rangle$.*

Proof. Suppose there exists such a unitary operator U. As we would then have for any $|\psi\rangle$ that $U|0\rangle|\psi\rangle = |\psi\rangle|\psi\rangle$, we have the following for $|\psi\rangle = |0\rangle$ and $|\psi\rangle = |1\rangle$:

$$U|00\rangle = |00\rangle \ , \tag{2.75}$$

$$U|01\rangle = |11\rangle \ . \tag{2.76}$$

But for $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, we have

$$U|0\rangle|\psi\rangle = |\psi\rangle|\psi\rangle \tag{2.77}$$

by definition, and

$$U|0\rangle|\psi\rangle = \alpha|00\rangle + \beta|11\rangle \tag{2.78}$$

by linearity. As $|\psi\rangle|\psi\rangle = \alpha^2|00\rangle + \alpha\beta(|01\rangle + |10\rangle) + \beta^2|11\rangle \neq \alpha|00\rangle + \beta|11\rangle$ for general α and β, we have a contradiction. Thus such a U does not exist. □

Suppose we did not have that problem. Then we would send the encoded bit through a (quantum) channel. But what error could occur? Classically, we can only have 1 error, but in quantum mechanics, the possible errors are in a continuous range. To see what error occurred, we looked (classically) at the outcome, and decoded that with majority voting. This would translate into measuring the states of the three qubits in a certain basis. But as we have seen, this destroys the state itself, and the outcome is probabilistic. Thus even if there was no error, we could have the outcomes $0, 1, 0$ if the original state was a superposition of $|0\rangle$ and $|1\rangle$. We cannot ever find projectors to decode it in the right way for all states, as there always exists a state which encodes to a state with non-zero parts for all basis vectors. So we can safely conclude that this will never work. However, we can do it in a different way, for certain types of errors.

2.6.2 The 3-qubit bit flip code

Suppose the state which should be protected is $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, and it should be protected against bit flip errors. That is, against the error $|\psi\rangle \mapsto X|\psi\rangle$. We can encode the state by $\alpha|000\rangle + \beta|111\rangle$. Note that this can be done by using two $CNOT$ gates (with the qubit we want to protect as the control qubit and two qubits in the state $|0\rangle$ as target bits), and that it is not the same as cloning an arbitrary state. Suppose this is sent through a channel, where each bit can independently be flipped with probability p. To correct a possible error, there is a two stage error correction. First we need to know which error occurred. This is the error detection stage or the syndrome diagnosis stage. In order to know which error occurred, the state is measured with a projective measurement.

Define the following projectors:

$$P_0 = |000\rangle\langle 000| + |111\rangle\langle 111| \ , \tag{2.79}$$

$$P_1 = |100\rangle\langle 100| + |011\rangle\langle 011| \ , \tag{2.80}$$

$$P_2 = |010\rangle\langle 010| + |101\rangle\langle 101| \ , \tag{2.81}$$

$$P_3 = |001\rangle\langle 001| + |110\rangle\langle 110| \ . \tag{2.82}$$

It can be seen that P_0 corresponds to no error, as this is the correct projector for our original state. P_1 to P_3 correspond to a bit flip error on the first to last qubit respectively, as they are the projectors for the state after a bit flip on the first to last qubit.

Suppose a bit flip error on the second qubit occurred. The state is then $\left|\psi'\right\rangle = \alpha\left|010\right\rangle + \beta\left|101\right\rangle$. For measuring the projectors, we assign the outcome i to projector P_i for $i \in \{0, \ldots, 3\}$ (outcome 0 means that no bit has been flipped, the other outcomes give the qubit which has been flipped). Recall that the probability of outcome i is given by $\left\langle\psi'\right|P_i\left|\psi'\right\rangle$. As $\left\langle\psi'\right|P_2\left|\psi'\right\rangle = 1$, the measurement outcome is 2 with certainty. Thus we know that the second qubit has been flipped. Note that the post-measurement state is the same as the state before the measurement, and that we did not get any information about the encoded state itself, only about the error which occurred. Why can we now find projectors which can help us, where we could not at first? Note that we first had $(\alpha\left|0\right\rangle + \beta\left|1\right\rangle)(\alpha\left|0\right\rangle + \beta\left|1\right\rangle)(\alpha\left|0\right\rangle + \beta\left|1\right\rangle)$, which has a non zero part for each of the eight basis vectors. However, now we have $\alpha\left|000\right\rangle + \beta\left|111\right\rangle$, which has only non-zero parts for two basis vectors. By each of the errors, these two basis vectors are mapped onto two different basis vectors, which allows us to define projectors to measure the error syndrome.

After the error detection, the recovery stage takes place. We know which error occurred, thus we can invert the error. In the case of the bit flip code, this means applying the Pauli-X gate on the right qubit.

As before, this works perfectly provided that there occurs only one error, which means that the probability of a wrong correction is again $3p^2 - 2p^3$, like the classical 3-bit repetition code.

Now we have seen an example of a quantum error correcting code, a definition will be given.

Definition 2.17. A quantum error correcting code is a linear subspace C of the state space of n qubits. A code encoding k qubits into n qubits is called a $[n, k]$ code. The linear subspace C is sometimes referred to as the codespace of a code.

All error-free encoded states are in the codespace of a quantum error correcting code. So for the 3-qubit repetition code, $C = Span\{\left|000\right\rangle, \left|111\right\rangle\}$.

Let us denote in the remaining part of the thesis the error channel by Φ, and the recovery operation by $\mathcal{R}$. To succesfully recover a state which lies in C, we need that

$$(\mathcal{R} \circ \Phi)(\rho) \propto \rho\,, \tag{2.83}$$

where $\propto$ means that our corrected state is proportional to the original state. Thus $\rho' = c\rho \propto \rho$, with $0 \leq |c| \leq 1$. As Φ is a quantum channel, which is not necessarily trace preserving, we have $\propto$ instead of $=$. This translates into an equality when Φ is trace preserving. To see which errors a quantum error correcting code can correct, there exist the quantum error correction conditions which are stated in the next theorem [6, Theorem 10.1] [7, Theorem 5.1].

Theorem 2.7 (Knill-LaFlamme)**.** *Let C be a quantum error correcting code, and let P be the projector onto C. Let Φ be an error channel with operator elements $\{E_i\}$. Then there exists an error correcting operation $\mathcal{R}$ which corrects $\{E_i\}$ on C if and only if there*

exists a hermitian matrix α such that

$$PE_i^\dagger E_j P = \alpha_{ij} P \tag{2.84}$$

These equations are called the error correction conditions.

Proof. Suppose there exists a hermitian matrix α such that the error correction conditions are satisfied. We will now construct a recovery operation $\mathcal{R}$.

Note that $\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$, thus if we have for each $|\psi_i\rangle\langle\psi_i|$ that $\mathcal{R}(\Phi(|\psi_i\rangle\langle\psi_i|)) = |\psi_i\rangle\langle\psi_i|$, then $\mathcal{R}(\Phi(\rho)) = \rho$ follows from linearity. Also, as P is the projector onto C, we have $\rho = P\rho P$ for ρ in C.

As α is Hermitian (and thus normal), we know from the spectral decomposition theorem that there exist a basis in which α is diagonal. Thus there exists a unitary matrix $U = (u_{lk})_{lk}$ such that $D = U\alpha U^\dagger$ is diagonal. Define $F_l = \sum_k u_{lk} E_k$. Then we know, as U is unitary, that F_l generate the same channel as the E_k by Theorem 2.5. Note that we now have that:

$$PF_l^\dagger F_k P = \sum_{i,j} PE_i^\dagger u_{li}^* u_{kj} E_j P = \sum_{i,j} u_{li}^* \alpha_{ij} u_{kj} P = d_{lk} P\,, \tag{2.85}$$

where the last equation holds because $U^\dagger \alpha U = D$.

Note that this is a simplification of the quantum error correction conditions, because D is diagonal. So now we have $\Phi(\rho) = \sum_l F_l \rho F_l^\dagger$, with $PF_l^\dagger F_k P = d_{lk} P$.

Note that we have

$$\begin{aligned}
\alpha \otimes P &= \sum_{i,j} |i\rangle\langle j| \otimes \alpha_{ij} P && (2.86)\\
&= \sum_{i,j} |i\rangle\langle j| \otimes PE_i^\dagger E_j P && (2.87)\\
&= \sum_{i,j} (|i\rangle \otimes PE_i^\dagger)(\langle j| \otimes E_j P) && (2.88)\\
&= \begin{pmatrix} PE_1^\dagger \\ \vdots \\ PE_n^\dagger \end{pmatrix} \begin{pmatrix} E_1 P & \dots & E_n P \end{pmatrix} \geq 0\,, && (2.89)
\end{aligned}$$

because this holds for any matrix of the form $A^\dagger A$. As P is a projector, $\alpha \geq 0$. Thus all eigenvalues are non-negative, which means that $d_{ii} \geq 0$. Define $V_i = 0$ if $d_{ii} = 0$, and $V_i = \frac{1}{\sqrt{d_{ii}}} PF_i^\dagger$ if $d_{ii} > 0$. So $PV_i = V_i$. Then we have that

$$V_i V_j^\dagger = \frac{1}{\sqrt{d_{ii} d_{jj}}} PF_i^\dagger F_j P = \frac{1}{\sqrt{d_{ii} d_{jj}}} d_{ij} P\,. \tag{2.90}$$

So if $i \neq j$, $V_i V_j^\dagger = 0$, and otherwise $V_i V_i^\dagger = P$ whenever $V_i \neq 0$. Let $Q = I - \sum_i V_i^\dagger V_i$. Then

$$Q^\dagger Q = (I - \sum_j V_j^\dagger V_j)(I - \sum_i V_i^\dagger V_i) = I - 2\sum_i V_i^\dagger V_i + \sum_{i,j} V_j^\dagger V_j V_i^\dagger V_i = I - \sum_i V_i^\dagger V_i = Q\,, \tag{2.91}$$

because $V_j V_i^\dagger = 0$ if $i \neq j$. Note that we also have

$$V_i Q^\dagger = V_i - \sum_j V_i V_j^\dagger V_j = V_i - V_i P = 0\,. \tag{2.92}$$

Define $\mathcal{R}(\rho) = \sum_i V_i \rho V_i^\dagger + Q\rho Q^\dagger$. Then we have that $\sum_i V_i^\dagger V_i + Q^\dagger Q = I$, thus $\mathcal{R}$ is a trace-preserving quantum channel. We also have that

$$\mathcal{R}(\Phi(|\psi\rangle\langle\psi|)) = \sum_{i,l} V_i F_l \,|\psi\rangle\langle\psi|\, F_l^\dagger V_i^\dagger + \sum_l Q F_l \,|\psi\rangle\langle\psi|\, F_l^\dagger Q^\dagger \tag{2.93}$$

$$= \sum_{i,l} \frac{1}{d_{ii}} P F_i^\dagger F_l P \,|\psi\rangle\langle\psi|\, P F_l^\dagger F_i P + \sum_l Q F_l P \,|\psi\rangle\langle\psi|\, P F_l^\dagger Q^\dagger \tag{2.94}$$

$$= \sum_{i,l} \frac{d_{il}^2}{d_{ii}} \delta_{i,l} P \,|\psi\rangle\langle\psi|\, P + \sum_l Q V_l^\dagger \,|\psi\rangle\langle\psi|\, V_l Q^\dagger \tag{2.95}$$

$$= \sum_i d_{ii} P \,|\psi\rangle\langle\psi|\, P \propto |\psi\rangle\langle\psi|\,, \tag{2.96}$$

as $V_l Q^\dagger = 0$.

So $\mathcal{R}$ is indeed an error correcting operation for $|\psi\rangle\langle\psi|$, and because of linearity also for ρ.

Suppose now that there exists an error correcting operation $\mathcal{R}(\rho) = \sum_k R_k \rho R_k^\dagger$, thus $\mathcal{R}(\Phi(\rho)) = \rho$ for all ρ in C. As $P|\psi\rangle = |\psi\rangle$, we have:

$$\mathcal{R}(\Phi(\rho)) = \mathcal{R}(\Phi(P\rho P)) = \sum_{i,k} R_k E_i P \rho P E_i^\dagger R_k^\dagger = cP\rho P\,, \tag{2.97}$$

where the last equation is valid because $\mathcal{R}$ is a recovery operation. We denoted the proportionality constant as c. As this are two different operator sum representations of $(\mathcal{R} \circ \Phi)$, there exist a unitary matrix U such that $R_k E_i P = u_{ik}\sqrt{c} P$. Thus $P^\dagger E_i^\dagger R_k^\dagger R_k E_j P = u_{jk} u_{ik}^* c P$. As $\sum_k R_k^\dagger R_k = I$, summing over all k gives

$$\sum_k P^\dagger E_j^\dagger R_k^\dagger R_k E_i P = P^\dagger E_j^\dagger E_i P = \sum_k u_{ik}^* u_{jk} c P = \alpha_{ij} P\,, \tag{2.98}$$

where α is hermitian because $\alpha_{ij} = c\sum_k u_{ik}^* u_{jk} = c\sum_k u_{kj} u_{ik}^* = \alpha_{ji}^*$. So we indeed found the error correction conditions. □

However, as errors are continuous, we still cannot correct many errors. The following theorem gives a much bigger class of errors which we can correct.

Theorem 2.8. *Let C be a quantum error correcting code with recovery operation $\mathcal{R}$. Let Φ be an error channel with operator elements $\{E_i\}$, which can be corrected according to the error correction conditions. Then the error channel Ψ with operator elements $F_j = \sum_i f_{ji} E_i$ can also be corrected by C.*

Note that the F_j are linear combinations of E_i; they are not transformed with a unitary matrix, which would give the same error channel.

Proof. Checking the error correction conditions gives

$$P F_i^\dagger F_j P = \sum_{k,l} f_{ik}^* f_{jl} P E_k^\dagger E_l P = \sum_{k,l} f_{ik}^* f_{jl} \alpha_{kl} P = \beta_{ij} P \,. \tag{2.99}$$

As before, β is hermitian because $\beta_{ij} = \sum_{k,l} f_{ik}^* f_{jl} \alpha_{kl} = \sum_{l,k} f_{il}^* f_{jk} \alpha_{lk} = \sum_{l,k} f_{jk} f_{il}^* \alpha_{kl}^* = \beta_{ji}^*$, where we interchanged the summation over k and l and used that α is hermitian. Thus the error correction conditions are satisfied, so we can correct Ψ. □

3. The Stabilizer Formalism

In this chapter, the stabilizer formalism will be introduced. In short, the stabilizer formalism is a way to describe a quantum error correcting code with much less information than we did before. Recall that for the qubit bit flip code, we needed 4 projectors, and the states on which $|0\rangle$ and $|1\rangle$ where mapped. We will see that we can reduce this to only 2 stabilizer operators, and two logical operators which represent the Pauli X and Z gate. The stabilizer formalism will be used to describe the toric code in Chapter 4. The stabilizer formalism can also be used to efficiently simulate quantum error correcting codes on a classical computer; this can be used to simulate errors on the toric code, and see if the error correction works.

3.1 The General Idea

In this section, the general idea of a quantum error correcting code in the stabilizer formalism will be explained.

With an error correcting code, we want to be able to correct errors, but we might also want to do computations. In the stabilizer formalism, we encode $|0\rangle$ and $|1\rangle$ as $+1$ eigenstates of a group of operators (the stabilizers), and $+1$ or -1 eigenstates of another operator, a so-called logical gate. All eigenstates of the stabilizers together form the codespace. When a correctable error happens, this can be measured by measuring the stabilizers, as it will make the state a -1 eigenstate of some stabilizers. After that, the state can be corrected with a suitable operation.

As we also may want to do computations, there are logical gates which act on our encoded states as if they were the original $|0\rangle$ and $|1\rangle$. For example, if we denote the logical states as $|0_L\rangle$ and $|1_L\rangle$, the logical X_L gate should take $|0_L\rangle$ to $|1_L\rangle$ as this is what the normal X gate would do with $|0\rangle$ and $|1\rangle$. The logical gates should take states from the codespace to the codespace, as we still want to be able to correct the state after doing computations.

The main problem for correcting an error is that we should not accidentally apply a logical gate to the state while correcting an error. For this we need a decoder, which determines the correction operation to apply after the measurement.

3.2 Group Theory

Before we start with the stabilizer formalism, the start of group theory will be introduced.

Definition 3.1. Let G be a set. Then $(G, \cdot)$ is a group with binary operation $\cdot$ if the following hold:

G_0: $\forall g_1, g_2 \in G : g_1 \cdot g_2 \in G$

G_1: $\forall g_1, g_2, g_3 \in G : (g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$

G_2: $\exists e \in G : \forall g \in G : g \cdot e = e \cdot g = g$

G_3: $\forall g \in G : \exists g^{-1} \in G : g \cdot g^{-1} = e = g^{-1} \cdot g$

Definition 3.2. Let $(G, \cdot)$ be a group. Then $(F, \cdot)$ is called a subgroup of $(G, \cdot)$ if $F \subseteq G$ and $(F, \cdot)$ is a group.

In this thesis, the groups which will be used consist of operators, and the group operation will be the composition in general. The group operation will not be mentioned, as it is in general either clear from the context which operation it is, or we describe a general group with 'multiplication' as group operation.

We will mostly use a subgroup of the Pauli group on n qubits. The Pauli group on one qubit consists of all Pauli matrices with factors ± 1 and $\pm i$, with matrix multiplication as operation. Thus

$$G_1 = \{\pm I, \pm X, \pm Y, \pm Z, \pm iI, \pm iX, \pm iY, \pm iZ\}. \tag{3.1}$$

The factors are needed to ensure that it is a group. The Pauli group on n qubits is the group with all possible tensor products of elements of G_1. We can describe a group with generators.

Definition 3.3. A set $H = \{g_1, \ldots, g_n\}$ is said to generate a group G if all elements in G can be written as a product of elements in H. The elements of H are called the generators of G, which is denoted as $G = \langle g_1, \ldots, g_n \rangle$.

3.3 Defining a Code

This is the basis needed for the stabilizer formalism. The idea of the stabilizer formalism is that we can define a group of operators, the stabilizers of the code, which leave our codespace invariant. Errors can change our codespace such that part of the stabilizers will give a different outcome when measured, which allows us to correct the errors. First of all, the definition of the stabilizers is needed.

Definition 3.4. Let g be an operator. Then g is called a stabilizer of $|\psi\rangle$ if

$$g |\psi\rangle = |\psi\rangle , \tag{3.2}$$

and $|\psi\rangle$ is said to be stabilized by g.

With this, we can understand why the stabilizers are a group. First of all, if we have two stabilizers, then $g_1 g_2 |\psi\rangle = g_1 |\psi\rangle = |\psi\rangle$, so the product is also a stabilizer (G_0). Secondly, $I |\psi\rangle = |\psi\rangle$, so the identity operator is always part of the stabilizers (G_2). Lastly, if g is a stabilizer, the inverse is also a stabilizer because $g |\psi\rangle = |\psi\rangle \implies |\psi\rangle = g^{-1} |\psi\rangle$. So the inverse is part of the stabilizers (G_3).

With a group of stabilizers, we can define a code.

Definition 3.5. Let S be a group of n-qubit operators. Then the code C stabilized by S consists of all n-qubit states which are stabilized by all elements $g \in S$. S is called the stabilizer of C. Thus $C = \{|\psi\rangle \, | \forall g \in S : g |\psi\rangle = |\psi\rangle\}$.

Let $S \subseteq G_n$ be the stabilizer of a code C. Let us look at the properties the elements of S need to have for C to be non-empty.

First of all, suppose $-I \in S$. Then we need that $-|\psi\rangle = -I|\psi\rangle = |\psi\rangle$. So $|\psi\rangle = 0$ is the only possible state. Thus for non empty codes, $-I \notin S$. Note that this means that the Pauli matrices multiplied by i will not be in a stabilizer, as $(\pm iP)^2 = -I$ for an arbitrary standard Pauli matrix P (X, Y or Z).

Now let $g_1, g_2 \in S$, and suppose they anticommute. Then for all $|\psi\rangle \in C$

$$|\psi\rangle = g_1|\psi\rangle = g_1 g_2|\psi\rangle = -g_2 g_1|\psi\rangle = -g_2|\psi\rangle = -|\psi\rangle \tag{3.3}$$

So if two stabilizers anticommute, we have as before that $|\psi\rangle = 0$ is the only possible state. So no elements should anticommute. As $S \subseteq G_n$, all stabilizers either commute or anticommute, thus all elements of S need to commute. In general, the only elements of S which will be needed are the generators of S.

3.4 Errors

As said earlier, errors can change the codespace such that the stabilizers will give a different outcome when measured. We will only look at Pauli errors, as all other errors can be written as a linear combination as Pauli errors. Thus if Pauli errors on n qubit can be corrected, all errors on n qubits can be corrected. Let us look at how the codespace can change. In short, there are three cases, because the errors either commute or anticommute with the stabilizers.

1. The error E can anticommute with one or more generators. The state will become $E|\psi\rangle$, and as $gE|\psi\rangle = -E|\psi\rangle$ for anticommuting g, it is a -1 eigenstate of g. Then the generators can be measured, and the generators which anticommute with E will give outcome -1.

2. It can also be that $E \in S$. Then E leaves our codespace invariant, thus this 'error' is not an error for the code.

3. It can be that $E \notin S$, but that E commutes with all generators, and thus with all stabilizers. This is a problem, because measuring the generators will not give a different outcome, even though the state has changed.

The set of operators which commute with all elements in S is called the centralizer:

Definition 3.6. Let S be a subgroup of a group G. The centralizer of S is

$$C(S) = \{g \in G | gs = sg \text{ for all } s \in S\} \tag{3.4}$$

In our case, this can be shown to equal the normalizer of S.

Definition 3.7. Let S be a subgroup of a group G. The normalizer of S is

$$N(S) = \{g \in G | gsg^{-1} \in S \text{ for all } s \in S\} \tag{3.5}$$

Lemma 3.1. *Let S be a subgroup of G_n with $-I \notin S$. Then $C(S) = N(S)$.*

Proof. Let $g \in C(S)$. then for all $s \in S$, we have that $gs = sg$. Thus $gsg^{-1} = s \in S$. So $C(S) \subseteq N(S)$.

Let $g \in N(S)$. Let $s \in S$ arbitrarily. Suppose $gs \neq sg$. As our operators are Pauli matrices, they either commute or anticommute. Thus $gs = -sg$. So $gsg^{-1} = -s \in S$. But then we have $-s \in S$ and $s^{-1} \in S$, so $-ss^{-1} = -I \in S$, which is a contradiction. Thus $gs = sg$ which means that $g \in C(S)$. So $N(S) \subseteq C(S)$. □

With this lemma, it can be seen that the third case is when $E \in N(S) \setminus S$

The error cases are stated more clearly in the following theorem, which states which errors can be corrected by a code.

Theorem 3.1. *Let C be a code with stabilizer $S \subset G_n$. Let $\{E_j\} \subseteq G_n$ be a set of Pauli errors. Then $\{E_j\}$ is a correctable set of errors for the code C if $E_j^\dagger E_i \notin N(S) \setminus S$ for all i and j.*

Proof. Let P be the projection onto the codespace of C. We will check the error correction conditions. First of all, suppose $E_j^\dagger E_i \in S$. Then $PE_j^\dagger E_i = P$, so $PE_j^\dagger E_i P = P^2 = P$. Note that $E_i^\dagger E_j = (E_j^\dagger E_i)^\dagger \in S$, so the coefficients for these errors are hermitian indeed.

Now suppose $E_j^\dagger E_i \in G_n \setminus N(S)$, and let $g_1, \ldots, g_{n-k}$ be generators for S. Then there is at least one generator, say g_1, which anticommutes with $E_j^\dagger E_i$. We can assume without loss of generality that g_1 is the only generator which anticommutes with $E_j^\dagger E_i$, because if another generator g_i also anticommutes, we can replace g_i with $g_i g_1$ (which commutes with $E_j^\dagger E_i$) in the generator set. Note that as all our generators stabilize our codespace (thus our codespace is the intersection of the $+1$ eigenspaces of the generators), we can write P as follows:

$$P = \frac{1}{2^{n-k}} \Pi_{l=1}^{n-k} (I + g_l)\,. \tag{3.6}$$

Using the anticommutivity of g_1 with $E_j^\dagger E_i$, we get

$$PE_j^\dagger E_i P = P(I - g_1)E_j^\dagger E_i \frac{1}{2^{n-k}} \Pi_{l=2}^{n-k} (I + g_l) = 0\,, \tag{3.7}$$

because $P(I - g_1) = 0$, as we have $(I + g_1)(I - g_1) = I + g_1 - g_1 - g_1^2 = 0$. For the last step, recall that $g_1 \in G_n$, thus $g_1^2 = I$. So the error correction conditions are satisfied, thus $\{E_j\}$ is a correctable set of errors for the code. □

This theorem states which errors we can correct with a stabilizer code, but it does not say how that can be done. So let us see what happens if an error E occurs which anticommutes with a certain generator g. Note that we only consider Pauli errors and generators. This means that we can write the projectors on the $+1$ and -1 eigenstates of g as follows:

$$P_\pm = \frac{1}{2}(I \pm g)\,. \tag{3.8}$$

Using this, it can be seen that the only possible outcomes of the measurements are $+1$ and -1, and the error state will be projected onto the corresponding eigenstate. Of course, when an error anticommutes with g, we have $gE\,|\psi\rangle = -E\,|\psi\rangle$, so then our error

state is already a -1 eigenstate and we will get -1 as outcome. So the outcomes of the measurements of the generators give information about the error which occured. This is why the outcomes are named the syndrome corresponding to the error:

Definition 3.8. Let $S = \langle g_1, \ldots, g_n \rangle$ be the stabilizers for a code C. The syndrome σ, or the syndrome corresponding to an error E $\sigma(E)$, is the set of all generators which give the measurement outcome -1.

After measuring the generators, the state still needs to be corrected. However, there may be multiple errors which give rise to a certain error syndrome. A decoder is then used to determine which correction will be done.

Let C be a code with stabilizer $S = \langle g_1, \ldots, g_n \rangle$, and let $\{E_j\}$ be a correctable set of Pauli errors for C. Suppose that measuring the generators gives the error syndrome σ. So for $g \in \sigma$, the measurement outcome is -1. If there exists a single error E_j which gives rise to this error syndrome, we can simply apply $E_j^\dagger$ to correct the state, as we assumed the errors to be Pauli errors. However, if E_j and E_i both give the same error syndrome, which error do we need to correct? As for now, only correctable sets of errors are used, it does not matter; both corrections will correct the error syndrome. This can be seen by looking at the error correcting conditions for the stabilizer formalism. We have $E_j^\dagger E_i \notin N(S) \setminus S$, thus the total operation (the error E_i together with the correction $E_j^\dagger$) will be part of the stabilizer group, as it will commute with all stabilizers in S . If the error correcting conditions do not hold, we cannot be sure of the correction needed. For this we need decoders. This will be looked upon in the next chapter, where we examine the toric code and decoders for it.

3.5 Logical Operators

After encoding the original state, there might still be computations which need to be done. If the state needs to be decoded and encoded each time for applying a gate, this can still result in errors which cannot be corrected. Therefore it is useful to be able to apply certain gates on the encoded state, these gates are called logical gates. The logical version of a gate U is denoted as $\hat{U}$. Together with the logical operators, the logical basis states are defined (for one qubit these are $|0_L\rangle$ and $|1_L\rangle$), as the logical basis states follow from the choice of $\hat{Z}$.

Let us now clearly state what a logical operator does. Suppose we have a code with logical basis states $|0_L\rangle$ and $|1_L\rangle$ (the encoded states), and suppose we have an operator U which can act on $|0\rangle$ and $|1\rangle$ (the non-encoded states):

$$U|0\rangle = g(|0\rangle, |1\rangle) \tag{3.9}$$
$$U|1\rangle = h(|0\rangle, |1\rangle). \tag{3.10}$$

The logical operator $\hat{U}$ should apply the same functions on the logical states:

$$\hat{U}|0_L\rangle = g(|0_L\rangle, |1_L\rangle) \tag{3.11}$$
$$\hat{U}|1_L\rangle = h(|0_L\rangle, |1_L\rangle). \tag{3.12}$$

By convention, the logical Pauli gates are constructed from the same physical Pauli gates. This ensures that the logical gates have the same commutation/anticommutation relations. As the states in the codespace of a code C need to map to other states in the codespace, the logical gates need to commute with all generators. This ensures that applying a logical gate is not seen as an error. Note that we actually define logical operators as 'errors' of case 3, which cannot be detected.

Let C be a code with stabilizer $S = \langle g_1, \ldots, g_n \rangle$. Suppose we have defined a logical operator $\hat{U}$. As $\hat{U}|\psi\rangle = \hat{U}g_i|\psi\rangle$, we see that $\hat{U}g_i$ acts identically on our codespace as $\hat{U}$. Thus there are multiple correct choices for logical operators.

3.6 The Logical Basis States

Now that we have defined the logical operators and the stabilizers, we can look at what the logical basis states need to be. In principle, these states do not have to be known to use the code, but it can be good to see that they can be obtained.

To obtain the logical basis states, let us look at what properties they have. First of all, let S be the stabilizer for a code C of which we want to obtain the basis states. Suppose $\hat{Z}$ and $\hat{X}$ are the logical gates defined for this code. Then we know the following for the logical basis states $|0_L\rangle$ and $|1_L\rangle$:

- $\forall g \in S : g|i_L\rangle = |i_L\rangle$ for $i \in \{0, 1\}$
- $\hat{Z}|0_L\rangle = |0_L\rangle$
- $-\hat{Z}|1_L\rangle = |1_L\rangle$

So $|0_L\rangle$ is the $+1$ eigenvector of all stabilizers and of $\hat{Z}$, and $|1_L\rangle$ is the $+1$ eigenvector of the stabilizers and of $-\hat{Z}$. This can be extended to encoding multiple qubits, as each logical basis state will be a $+1$ eigenstate of either $\hat{Z}_i$ or $-\hat{Z}_i$. The density matrix of a basis state is then the projector onto the $+1$ eigenstate of the stabilizers and gates.

3.7 The Distance of a Quantum Error Correcting Code

The distance of a quantum code is related to the logical operators, as these are 'errors' which cannot be corrected. To define the distance, we first need to define the weight of an operators:

Definition 3.9. Let A be an n-qubit operator. The weight of A is the number of qubits on which A acts non-trivially.

For example, the weight of $XIZ (= X \otimes I \otimes Z)$ is two, and the weight of III is zero.

Definition 3.10. The distance d of a quantum code is the minimum weight of any non-identity logical operator.

Similar to the classical code distance, a distance d means that we can correct errors on up to $(d-1)/2$ qubits. This can be understood with Theorem 3.1. Suppose we want

to be able to correct E_1 and E_2, and they both act on t different qubits. Then we need to check that $E_1^\dagger E_2 \notin N(S) \setminus S$. Note that $E_1^\dagger E_2$ can have weight $2t$. As there is a logical operator with weight d, there is a part of $N(S) \setminus S$ with weight d. So we need that $2t < d$, or that $d \geq 2t + 1$, as in the classical code.

3.8 The 3-qubit Bit Flip Code

To make the stabilizer formalism more clear, we will describe the 3 qubit bit flip code with it. Recall that the codewords of the code are:

$$|0_L\rangle = |000\rangle \tag{3.13}$$

and

$$|1_L\rangle = |111\rangle \ . \tag{3.14}$$

As can be seen, the operators that stabilize the code words are III, Z_1Z_2, Z_2Z_3 and Z_1Z_3, which is a group. Only the generators are needed, so we choose for example Z_1Z_2 and Z_2Z_3. The error operators which could be corrected were X_1, X_2 and X_3. It can be checked that these errors can be corrected according to Theorem 3.1. Suppose that X_2 occurs, like in the previous example. Then both Z_1Z_2 and Z_2Z_3 anticommutate with the error, so for both generators we get outcome -1. As X_2 is the only error in our error set which gives this outcome, we can correct it by applying X_2 again.

Let us now look at the logical gates for the 3 qubit bit flip code. The Pauli X gate takes $|0\rangle$ to $|1\rangle$ and vice versa, so $\hat{X}$ should take $|0_L\rangle$ to $|1_L\rangle$ and vice versa. So we can take $\hat{X} = X_1X_2X_3$. Of course, we can make other logical $\hat{X}$ gates by multiplying with one of the stabilizers, but this is the conventional $\hat{X}$ gate (composed of physical X gates).

The Pauli Z gate takes $|1\rangle$ to $-|1\rangle$, and leaves $|0\rangle$ as it is. So $\hat{Z}$ should do the same for the codewords $|0_L\rangle$ and $|1_L\rangle$. As $|1_L\rangle = |111\rangle$, we can either take Z_1, Z_2, Z_3 or $Z_1Z_2Z_3$. Note that these gates can all be transformed into each other by applying stabilizers.

As Z_1 has weight 1, which is the minimal weight for the logical operators, we cannot correct all errors ($d = 1 \leq 2t + 1$, so $t = 0$). This is true, as we cannot distinguish a Z error on one qubit from applying the logical $\hat{Z}$ gate.

As can be seen, all that is needed to describe the 3 qubit bit flip code in the stabilizer formalism are the generators Z_1Z_2 and Z_2Z_3, and the logical gates $X_1X_2X_3$ and Z_1. This is far less than describing the states and the projectors which we did before, where we needed 4 projectors and 2^3 amplitudes for the basis vectors. In general, the number of generators will only scale linearly with the number of qubits as proven in the book of Nielsen and Chuang [6, Proposition 10.5]. Specifically, the number of stabilizers (m) is related to the number of qubits encoded (k) and the number of used qubits (n) in the following way: $m = n - k$. However, the description of the states and the projectors will scale exponentially. Note that even though we may only need 2 logical states for encoding 1 qubit, to describe those states we need the amplitudes of the 2^n basis vectors of the n-qubit vector space. Thus especially for large codes, the stabilizer formalism is a huge improvement.

4. The Toric Code

In this chapter, the toric code of Kitaev [2] [4] will be described using the stabilizer formalism introduced in the previous chapter. First of all, the advantages of the toric code are described. Then we will define the stabilizers, after which the logical gates will be introduced. Then the errors which will be used are described, after which we will look at how we will decode the errors and what can go wrong in the decoding stage. In this thesis, two decoders will be considered. First of all the Minimum Weight Perfect Matching (MWPM) decoder, which is most commonly used. Secondly we will look at an almost-linear time decoder which has recently been introduced as a good alternative [1].

4.1 The Relevance of the Toric Code

The toric code is a surface code, and as said in the introduction, surface codes have certain useful properties. First of all, the locality. As we will see, every measurement to determine the error only involves a small number of qubits (4 qubits). The locality is needed for the first generation of quantum computers, as less distance between interacting qubits means a smaller probability on errors. The second useful property is that increasing the size of the code can reduce the logical error rate (the probability on applying a logical gate instead of correction) when the error probability is below a certain value, the threshold. The threshold of the toric code is high compared to other surface codes. Lastly, it is possible to do universal quantum computation; not only the logical $\hat{X}$ and $\hat{Z}$ gates are possible (which we define in the stabilizer formalism), but an arbitrary gate can be approximated by using magical state destillation [8].

4.2 The Stabilizers

The toric code is defined on a $L \times L$ grid. The qubits are placed on the edges of the grid, such that there are $2L^2$ qubits. This can be seen in Figure 1, where the qubits are denoted by the circles. This grid is made 'toric' by identifying the qubits on the opposed sides of the grid, thus using periodic boundaries.

For the toric code, there are two kinds of stabilizer generators: Plaquette operators and Vertex operators. A plaquette operator consists of Pauli-Z gates applied on the qubits on the edges of a plaquette. A vertex operator or star operator consists of Pauli-X gates applied on the qubits of the edges touching the vertex. See Figure 1 for two examples of the generators.

For the stabilizer formalism, we need to check that the generators commute, and that they are independent. For if they are not independent, we can reduce the set of generators by removing a stabilizer. Of course, when taking two plaquette operators V_1 and V_2 or two vertex operators P_1 and P_2, it is simple to check that they commute, as they consist of commuting Pauli matrices. So let us consider a plaquette operator P and a vertex operator V. There are two cases: either two qubits on which the operators work overlap, or no qubits overlap. In the second case, the operators surely commute, as I

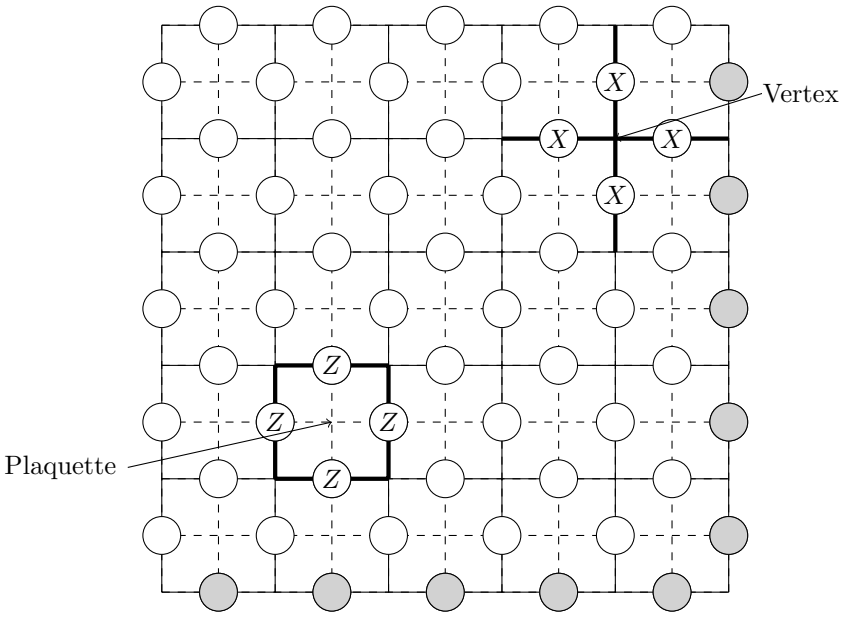


Figure 1: A 5×5 grid for the toric code. The circles represent the qubits. The grey qubits represent the periodic boundary, they are identical with the qubits at the other side of the grid. One vertex operator and one plaquette operator are drawn.

(which is applied on the other qubits) commutes with all operators. In the first case, we need to check that $X \otimes X$ commutes with $Z \otimes Z$. Recall that X and Z anticommute. We have

$$(X \otimes X)(Z \otimes Z) = XZ \otimes XZ = (-ZX) \otimes (-ZX) = (Z \otimes Z)(X \otimes X)\,, \tag{4.1}$$

so $X \otimes X$ indeed commutes with $Z \otimes Z$.

The last thing we need to check is whether the stabilizers are independent. Let us see what happens if two of the same type of stabilizers are multiplied together. Suppose P_1 and P_2 are multiplied together. If $P_1 \neq P_2$, we can have that they overlap on 1 qubit or that they do not overlap. The second case is the easiest, as all Z operators will be applied. In the first case, the operator $Z^2 = I$ will be applied on the overlapping qubit. This means that the Z operators will only be applied on the edge of our stabilizers, as can be seen in Figure 2. One can either think of this as a concatenation of plaquettes, with the Pauli Z gates applied on the edge, or as a closed loop of Pauli Z gates.

So for any amount of vertex operators, the Z operators will be applied on the edges of the total stabilizer. However, there is no edge anymore if we take all vertex generators. This means that no Z operator will be applied, so applying all generators equals the identity operator. This can be seen in a different way too. Each qubit is part of two vertex stabilizers, so if we multiply all vertex generators together, we have $Z^2 = I$ applied on each qubit. Thus to make the generators independent, one generator needs to be left out.

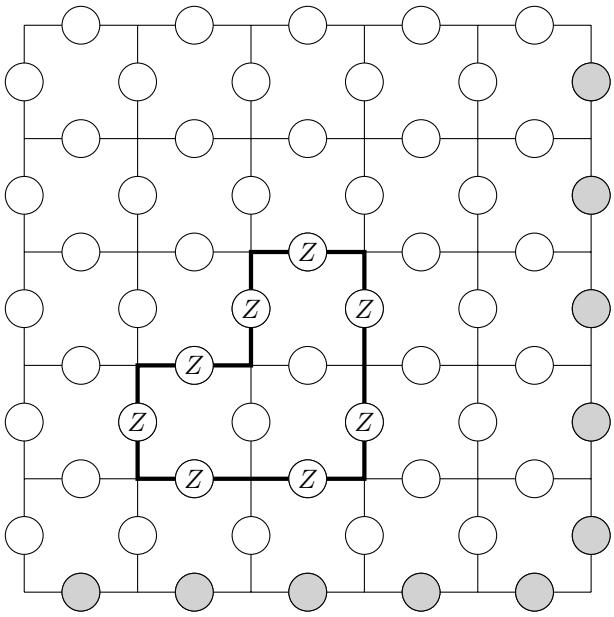


Figure 2: A 5×5 grid for the toric code. The circles represent the qubits. The grey qubits represent the periodic boundary, they are identical with the qubits at the other side of the grid. The result of applying three neighbouring plaquette operators is drawn.

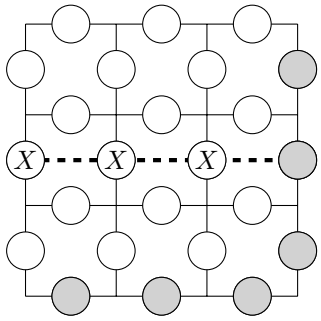
Exactly the same argument holds for the vertex stabilizers. This can be seen by considering a shifted grid (the dashed grid in Figure 1). If the grid is shifted down and right by half a stabilizer, the vertex operators and the plaquette operators are switched. This shifted grid is called the dual grid. For a general surface code, the dual grid is obtained by interchanging the plaquette and the vertex stabilizers (so drawing lines from the plaquettes to eachother). For the toric code, the dual grid is equivalent to the original grid, as the shape of the plaquette operators and the vertex operators does not change. This symmetry in the operators can be used for the correction, as we can correct for errors measured by the plaquette operators and for errors measured by the vertex operators independently and in an identical way. Thus we can focus on one type of error, and the corresponding type of stabilizers. In this chapter, the Z errors and the vertex stabilizers are used, because this makes it easier to visualize the decoding of the code. The results of this can be used to do the same for the X errors and plaquette operators, and applying both an X and a Z error results in a Y error, which is thus corrected in the same way as an X and a Z error.

4.3 The Logical Gates

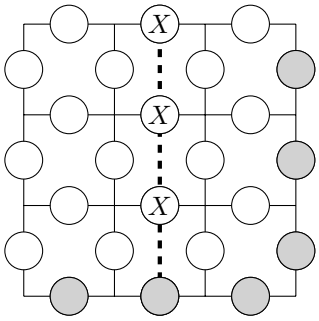
Now that the stabilizers are defined, we need to find a set of logical gates. As can be deduced from what was said before, if an operator touches a stabilizer (they overlap on 1 qubit), they will follow the commutation relations of the operators applied on the

overlapping qubit. This means that, in order to commute, the logical gates can never have a loose end, thus they need to be closed loops, like the stabilizers. However, they cannot be part of the stabilizer. To make such a operator, we make use of the periodic boundary conditions. The only closed loops which are not stabilizers, are the loops which wind around the code. As we have $m = 2L^2 - 2$ independent stabilizer generators, and $n = 2L^2$ qubits, the number of encoded qubits is $k = n - m = 2$. So the logical gates needed correspond to X_1, X_2, Z_1 and Z_2, where the indices denote the qubit on which they work. The logical gates can be seen in Figure 3. As seen in Chapter 3, if a logical gate is multiplied by a stabilizer, it will still act as a logical gate. Thus every loop around the torus (not only the four loops shown in Figure 3) corresponds to a logical gate.

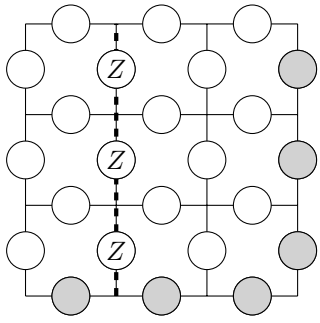
Now that the logical gates are defined, the distance of the code can be found. As a logical operator must wind around the code, it must apply an operation on at least L qubits. This is the minimum weight, and thus the distance of the toric code is $d = L$. Note that this intuitively means that increasing the size of the lattice might increase the robustness of the code (errors on more qubits can be corrected). This is true for certain regimes in certain error models, and will be looked upon later.



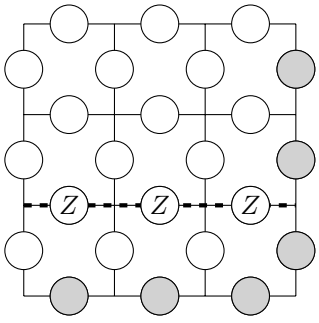
(a) The X_1 logical gate.



(b) The X_2 logical gate.



(c) The Z_1 logical gate



(d) The Z_2 logical gate.

Figure 3: A 3×3 grid for the toric code. The circles represent the qubits. The grey qubits represent the periodic boundary, they are identical with the qubits at the other side of the grid. The drawn loops are examples of the logical gates.

4.4 Errors

As seen in Chapter 2, linear combinations of correctable errors can be corrected. So if an error correcting code can correct Pauli errors of weight n, it can correct all errors of weight n. Let us now see what happens with the measurements of the generators after a Pauli error occurred. Suppose a Pauli Z error occurs on one qubit. Then the plaquette operators (which are made of Pauli Z operators) commute with this error, but the neighbouring vertex operators (made of Pauli X operators) anticommute. So only the two neighbouring vertex operators will give a measurement outcome of -1. Similarly, if a Pauli X error occurs on a qubit, the vertex operators will commute, but the plaquette operators anticommute. Thus X errors are measured on the plaquette operators. Lastly, a Y error can occur on a qubit. As the Pauli Y operator anticommutes with both X and Z operators, this will be measured by both the vertex and the plaquette operators. Thus an Y error can be seen as an X and a Y error on one qubit ($Y = iXZ$).

Of course, multiple errors can happen next to each other. So suppose two Z errors occur on the same vertex V. Then, as seen when checking whether the generators commute, the errors will commute with V. Thus V will give the outcome 1 when measured. However, the other vertices which are next to the errors will still give the outcome -1. This can be seen as a 'path' of errors, where only the outer edges will determine the syndrome. One consequence of this is that each error syndrome has multiple possible errors, an example of this can be seen in Figure 4. As said before, we can do exactly the same on the dual grid for X errors.

As the Pauli operators are self-inverse, an error E is corrected by applying it again. However, as there are multiple possible errors which give the same syndrome, it is not clear which error happened. Suppose E' gives the same syndrome as E. Then the operator $L = EE'$ commutes with all stabilizer generators. So L is either a stabilizer, or L is a logical gate. In the first case, when applying E' after an error E occurred, the correction is successful. In the second case, it is not, as we have accidentally applied a logical gate. To minimize the probability of the second case, a decoding algorithm is used to determine the path to correct along. A few of these decoders will be explained in section 4.7. In section 4.5, two different error models will be explained, which will be used for the decoding.

4.5 Error models

The first type of error model considered here is the uncorrelated noise model, also known as the independent noise model. The idea is that Pauli X errors and Pauli Z errors occur independently of each other. They can also occur both on the same qubit, which can be seen as a Pauli Y error. The errors occur with the following distribution:

$$I: \quad (1-p)^2 \tag{4.2}$$

$$X: \quad p(1-p) \tag{4.3}$$

$$Y: \quad p^2 \tag{4.4}$$

$$Z: \quad p(1-p) \tag{4.5}$$

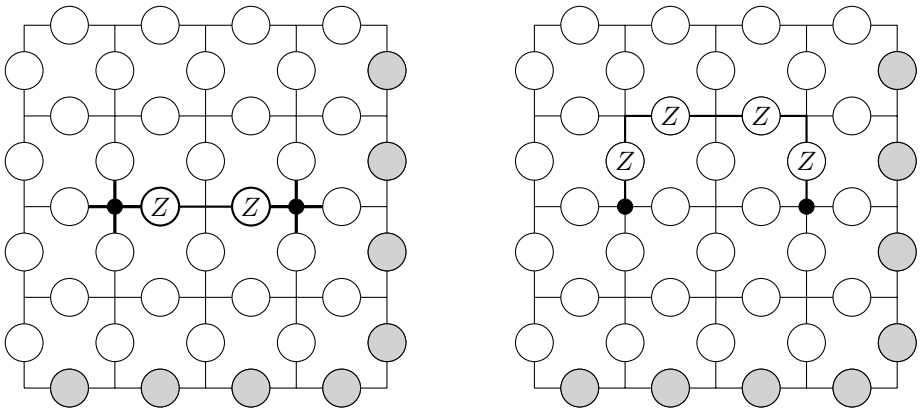


Figure 4: Two different errors with the same syndrome on a 4×4 toric code. The thick vertices represent the syndrome, the Z errors are shown on the qubits on which they occur. The thick edges form the two different paths of the errors.

which is equivalent to two independent models, where for each model an error occurs with probability p. In one model, the error is a Pauli X error, and in the other model a Pauli Z error. All qubits follow this distribution. The symmetry in this model can be used to simplify the decoding process, as we can consider the X and Z errors separately and correct for them accordingly.

In the second model (the Quantum Erasure Channel(QEC)), the errors occur according to the erasure channel, as seen in section 2.5. Recall that this channel erases a qubit with a probability p. This can be detected, and the erased qubit is replaced with a qubit in a completely mixed state, which can be seen as a random Pauli error. As said before, this channel can be seen as the depolarizing channel with the extra property that the place of the errors is known. Thus the place of the errors is known, but it is not known which Pauli error occurs. In practice, the error can for example be induced by the loss of a photon, which can be detected. We call the set of qubits on which an erasure error occurred an erasure.

4.6 The Threshold of the Code

As seen earlier, the distance of the toric code is equal to the lattice size. Thus errors on more qubits can be corrected if the lattice size is increased. However, with the error models seen above, the number of qubits on which errors occur increases too. For low error rates, increasing the lattice size indeed increases the robustness of the code (the errors have a higher chance to be corrected without applying a logical gate), but for high error rates this is not true anymore. The transition point for these behaviours when L goes to infinity is called the code threshold. This can depend on how the syndrome is decoded, thus a threshold always corresponds to a code and a decoder. When a new decoder is defined, the most important properties are the speed of the decoder and the threshold. For if the speed is low, the decoder reduces the speed of the computations

which need to be done. And if the threshold is low, the computations must be done with greater care in order to be correct. These two properties will be analysed in the next chapter, where the result of simulations of two of the decoders described below are shown.

4.7 Decoding the Toric Code

As said earlier, decoding a syndrome is essentially finding paths between the generators with outcome -1. Of course, this can be done in many ways. In this thesis, we will consider three decoding algorithms. First of all, the optimal decoder will be explained, but we will see that this has an exponential time complexity. As the time complexity plays an important role in determining which decoder to use, we will also look at two faster decoders, with a high threshold; an algorithm for the Minimum Weight Perfect Matching (MWPM) problem, and the Union-Find decoder, a recently developed decoder which can run in almost-linear time.

For the decoding, the error models explained in section 4.5 will be used. For most decoders, we will use the uncorrelated noise model; i.i.d. Pauli X and Z errors on each qubit. For the peeling decoder, an intermediate result for the Union-Find decoder, the quantum erasure channel will be used.

4.7.1 The optimal decoder

Let us consider an arbitrary syndrome σ for the toric code. What should an optimal decoder do? It gives a correction such that no logical gate is applied with the highest probability. So it should compute for all errors the probability of occurrence, and whether they are equivalent up to a stabilizer, or that they differ by a logical gate. This results in 4 equivalence classes per type of error, each with a certain probability of occurrence. It then returns the equivalence class with the highest probability (or an error from this class). If the class is chosen with the error in it, the correction is successful, otherwise a logical gate is applied. This decoder is optimal, because the error has the highest probability to be in the class which is returned.

The threshold of the optimal decoder is approximately $p_{th} = 11\%$, obtained by mapping the toric code to the random-bond Ising model [9]. This threshold is optimal, so why is the optimal decoder not widely used? As said earlier, there are two important properties of decoders which should be considered. The threshold, but also the time complexity of the decoder. The threshold may be optimal, but the time complexity is nowhere near optimal. As the number of possible errors corresponding to a syndrome scales exponentially, the time complexity of the optimal decoder is at least exponential. This is why other decoders are considered, with a lower threshold but with a better time complexity.

4.7.2 Minimum Weight Perfect Matching

The second decoder looked at is related to the Mimimum Weight Perfect Matching (MWPM) problem, and can be implemented in polynomial time, considerably better

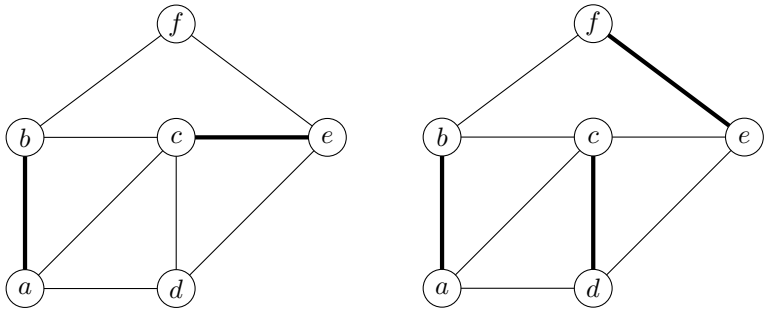
than the exponential time complexity of the optimal decoder. The MWPM problem is a problem in graph theory, which can be used to decode a syndrome by finding the most likely error. To state the problem, first some definitions are needed.

Definition 4.1. A graph G consists of a set of vertices (nodes) V and a set of edges which connect the vertices.

Definition 4.2. Given a graph $G = (V, E)$. A matching $M \subseteq E$ is a set of pairwise non-adjacent edges, thus no two edges have the same vertex as endpoint.

Definition 4.3. Given a graph $G = (V, E)$ and a matching M. M is a perfect matching if every vertex is incident to an edge $e \in M$.

An example of a graph with a matching can be seen in Figure 5a, and a perfect matching in Figure 5b.



(a) A matching M in a graph. The thick lines are the edges in M.

(b) A perfect matching M' in a graph. The thick lines are the edges in M'.

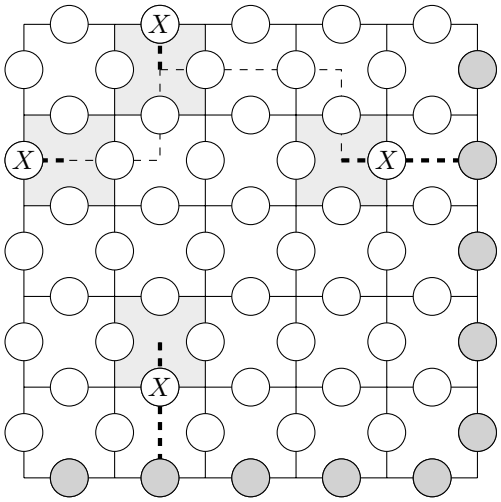
Figure 5: An example of a graph with matchings. The circles are the vertices, the lines are the edges.

With these definitions, the problem statement is as follows:

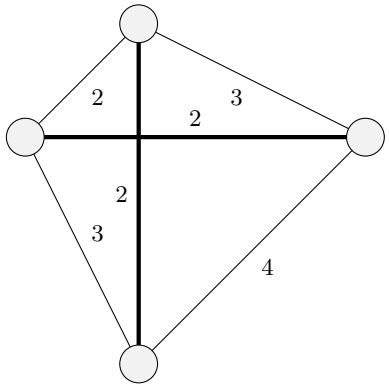
Problem 4.1 (Mimimum Weight Perfect Matching)**.** *Given a graph $G = (V, E)$ with edge weights c_e for $e \in E$, find a perfect matching $M \subseteq E$ of minimum weight $\sum_{e \in M} c_e$.*

One can convert a grid with errors to a complete graph, where the generators with outcome -1 are the nodes. The edges between the vertices have the length of the shortest path (i.e. with the least number of qubits) between those generators. This is the Manhattan distance, which is the sum of the horizontal distance and the vertical distance. For example, if all generators have coordinates in $\mathbb{Z}^2$, the distance between $(4, 2)$ and $(1, 3)$ is $|1 - 4| + |3 - 2| = 3 + 1 = 4$.

An example is shown in Figure 6. As the probability of a certain error with weight m is $p^m(1-p)^{n-m}$ with n the total number of qubits, the most likely error is the error with the least weight (assuming the probability of an error is smaller than $1/2$), thus we need to match every vertex to another vertex, such that the total length of the edges is the smallest, which is exactly the MWPM problem. The MWPM problem can be solved with for example Edmund's Blossom algorithm. The original algorithm has a time complexity



(a) An example of a syndrome, with the shortest paths from one generator to the others. The other shortest paths are omitted for clarity. The solution from the MWPM problem is shown as thick dashed lines, with the operators to correct the syndrome applied on the qubits.



(b) The syndrome converted to a graph, with the shortest paths as edge weights. The vertices represent the generators. The thick lines are the solution of the MWPM problem.

Figure 6: An example of syndrome converted to a graph, with the solution in both the grid and the graph.

of $\mathcal{O}(|E||V|^2)$, which has been improved up to $\mathcal{O}(|E|\sqrt{|V|})$, where $|E|$ is the number of edges and $|V|$ the number of vertices [10]. As we use a complete graph, this is equal to $\mathcal{O}(|V|^2\sqrt{|V|}) = \mathcal{O}(n^2\sqrt{n})$ (the number of vertices is the size of the syndrome, which is linear in the number of qubits). Finding the paths takes $\mathcal{O}(n^2)$ time, as we need calculate the paths between every pair of generators in the syndrome. Thus the total time complexity is still $\mathcal{O}(n^2\sqrt{n})$.

4.7.3 The Union-Find decoder

The fastest known algorithm for the MWPM problem still has a time complexity of $\mathcal{O}(|E|\sqrt{|V|}) = \mathcal{O}(n^2\sqrt{n})$ where n is the number of qubits. To reduce this time complexity, an algorithm has been developed with a time complexity of $\mathcal{O}(n\alpha(n))$ [1], where α is an inverse ackermann function, which is smaller than 3 for any practical input size (see the appendix for the exact definition). The algorithm in this thesis is the weighted growth version of the decoder described in [1], as this has a higher threshold than the original version. It should be noted that it has small differences with the algorithm fully described in [1], as the weighted growth version is not fully described. The name is a reference to the main part of the algorithm, the use of the Union-Find data-structure algorithm developed by Tarjan [5].

To get an almost linear time complexity, the syndrome is first reduced to an erasure syndrome, which can be decoded in linear time. First of all, the peeling decoder for the erasure syndrome will be explained.

The peeling decoder

Let ε be an erasure, and σ the corresponding syndrome. Recall that we defined an erasure to be a set of qubits on which an erasure error occurred. By measuring the stabilizer generators, we obtain the syndrome σ; all generators which anticommute with the erasure errors. Note that every generator in the syndrome has at least one erased qubit next to it. As the only errors occur in the erasure, we must have that for every pair of generators in the syndrome, the path of errors must be in the erasure. This is the main idea of the peeling decoder [11]. The peeling decoder reduces the size of the erasure, while keeping the syndrome adjacent to the erasure by applying corrections (or keeping track of on which qubits corrections should be applied). This is possible, as elements of the syndrome can be moved around by applying corrections on the qubits next to it (the edges in the erasure). At last, the erasure is completely removed, and there is no syndrome anymore. Thus the error is corrected.

To find a correction fast, we construct a spanning forest of the erasure in the first step to avoid cycles. After that, we loop over the leaves (step 3, the leaf is picked in step 4) and either add the incident edge to a set C (the set of qubits on which a correction is needed, initialized in step 2) or not, depending on whether the leaf is in the syndrome or not, as each node in the syndrome should have at least one erased qubit next to it. This is explained in more detail in the boxed text. Every leaf edge encountered in the loop will be removed from the erasure in step 5, which will lead to more leaf edges, and eventually the whole erasure will be removed. In the last step, an error is returned, which consists of Z errors on all qubits on the edges of C.

This will give a set C which would generate the syndrome σ if the errors would be applied on all edges in C, as shown in the next subsection.

Adding an edge to C

Let us now look more closely at when an edge is added to the set C. As the set C contains all edges on which an error will be applied, adding an edge to the set C is the same as (virtually) applying an error to the qubit on that edge. Applying an error on an edge (u, v) will flip the outcome of the generators u and v, thus this is the same as flipping the u and v in the syndrome (if a node is in σ, it will remove the node from σ, and if the node is not in σ, it will add the node). The syndrome need to be kept adjacent to the erasure, as otherwise we cannot apply a correction on an erased qubit to correct the syndrome. So if the pendant vertex of the edge u is not in the syndrome, we must not add the edge to C, whereas if it is, we need to add it to C to remove the u from the syndrome. Of course, we then also need to flip the node at the other side of the edge in σ to keep correctly track of the syndrome. This is done in step 6 to 8.

Note that we either add 1 node to and remove 1 node from σ, or we remove two nodes

from σ. In each case, the number of elements remains the same parity (even or odd). As we need to remove the syndrome, the number of elements of σ in each connected component needs to be even at the start. This is always the case if we consider perfect measurements, as all errors either add or remove an even number of generators to or from σ, or they replace a generator. See Algorithm 1 for the pseudo code of the peeling decoder. In the algorithm, it is said that a syndrome of a Z-error is required. The algorithm for X-errors is completely analogous, where the dual grid is considered. As said in section 4.5, a Y error will be seen and corrected as an X error and a Z error on the same qubit.

See Figure 7 for an example of the peeling decoder.

Algorithm 1 The peeling decoder as described in [11]

Require: A surface $G = (V, E, F)$, an erasure $\varepsilon \subseteq E$ and the syndrome $\sigma \in V$ of a Z-error
Ensure: A Z-error P such that $P \subset \varepsilon$ and $\sigma(P) = \sigma$.
1: Construct a spanning forest F_ε of ε.
2: Initialize C by $C = \varnothing$.
3: **while** $F_\varepsilon \neq \varnothing$ **do**
4: Pick a leaf edge $e = \{u, v\}$ with pendant vertex u
5: Remove e from F_ε
6: **if** $u \in \sigma$ **then**
7: remove u from σ, add e to C and flip v in σ
8: **end if**
9: **end while**
10: **return** $P = \prod_{e \in C} Z_e$.

The correctness of the peeling decoder

Consider now the error constructed by the algorithm. Does this error indeed correct the syndrome, as it should? Suppose the syndrome is adjacent to the erasure in a certain iteration of the loop, where $e = \{u, v\}$ is picked as leaf edge. If $u \in \sigma$, then after the iteration, $u \notin \sigma$. Thus after the iteration, the syndrome is still adjacent to the erasure. The only thing left to prove is that the final iteration completely removes the syndrome. Let us now consider one of the trees which are part of the spanning forest. As noted before, adding an edge to C maintains the parity of the connected component. In the last iteration, the last edge of a tree is picked. As the parity is of the number of elements of σ was even at the start, it is still even in this iteration. There are only two possible members of the syndrome; the two nodes incident to the edge. Suppose they are both in the syndrome. Then the condition of the if statement in step 6 is True, thus both nodes are flipped in σ (removed). Thus the error is corrected. Suppose the nodes are both not in the syndrome. Then the condition is False, thus the nodes are not flipped, and the syndrome remains empty, in which case the error is also corrected. As the parity is even, it cannot be that only one of the two nodes is in the syndrome. As this holds for all trees in the spanning forest, the total error is corrected.

The time complexity of the peeling decoder

Constructing a spanning forest takes linear time, and the loop is over every edge in the forest, which is also linear. The actions inside the loop can be done in constant time if a list of leaves is precomputed or if the forest is stored in a suitable way. Thus the peeling decoder has a linear time-complexity (linear in $|\varepsilon|$, thus linear in the number of qubits n).

Making the erasure

The remaining part of the decoder is making an erasure ε from our syndrome σ. For a good performance, we want the smallest distance between each two generators in the syndrome in the erasure.

The idea is to iteratively grow the erasure, until it is correctable with the peeling decoder. As noted before, the peeling decoder can be used when the syndrome has an even number of elements in each connected component of the erasure. Let us name these connected components of the erasure clusters. To start, each generator is a cluster itself. In every step of the algorithm, we grow the clusters with an odd number of $g \in \sigma$ by half an edge on the boundaries, and merge them when they meet. When two clusters with an odd number of $g \in \sigma$ (two 'odd clusters') meet, the merged cluster will have an even number (an 'even cluster'), which is correctable for the peeling decoder. To grow a cluster, we run over all vertices at the boundary of the cluster, and add half an edge to the edges next to these vertices. For an example, see Figure 8. The algorithm can be easily extended to the combination of erasure errors and Pauli errors by adding the erasure to the initial clusters.

The data structures

Let us first describe the data structures used. The items which need to be stored are the nodes (in clusters) and the edges (to see how far they are grown). In addition, we will store a list of boundary nodes for each cluster, to speed up the growing of the clusters.

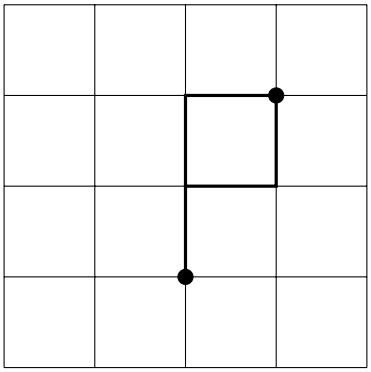
The clusters are stored as trees. At the start, one random node of the cluster is assigned to be the root of the tree, and all other nodes are made children of that node. In the case there is no erasure at the start, all generators are the roots of their own cluster. Note that in the case of only Pauli errors, without erasure, all nodes are clusters on their own, with themself as root. Storing the clusters can easily be done by using a dictionary in Python, by taking a node as key and the parent of that node as value (the root has itself as value).

The edges are stored in a lookup table, in which is stored whether they are not grown (value 0), half grown (1) or fully grown (2).

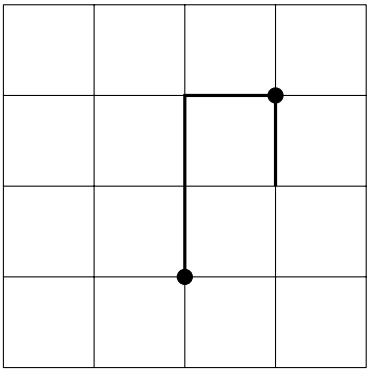
The complete algorithm

The pseudo code of the algorithm can be seen in Algorithm 2, and will now be explained step by step.

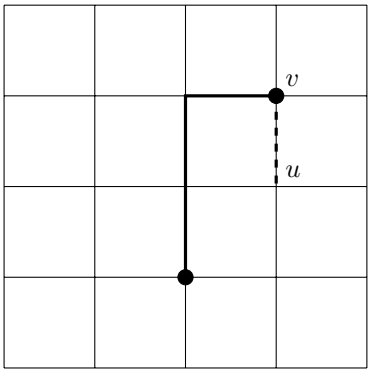
The first step of the algorithm, is to initialize the clusters, the lookup table for the edges and the boundary lists.



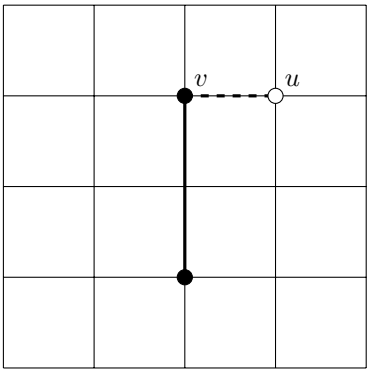
(a) An erasure error. The erased qubits are depicted as thick edges. The black nodes represent the syndrome of vertex generators.



(b) A spanning tree of the erasure

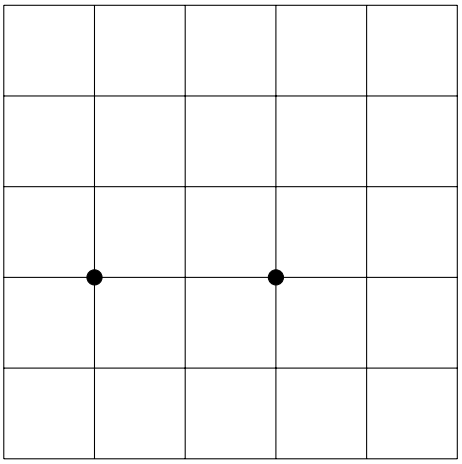


(c) The first step of the peeling decoder. The edge (u,v) will be removed from the erasure. The set C is still empty, because $u \notin \sigma$.

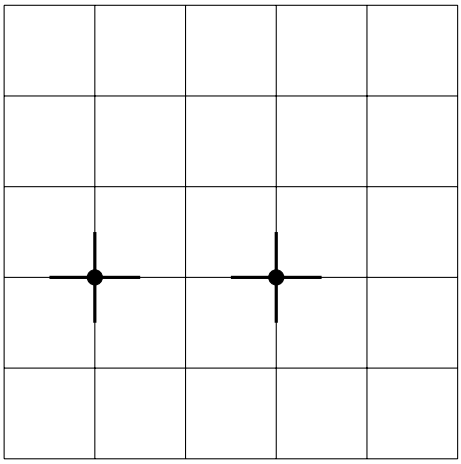


(d) The second step of the peeling decoder. The edge (u, v) will be removed. Because $u \in \sigma$, the edge is added to the set C, v is added to σ and u is removed from σ.

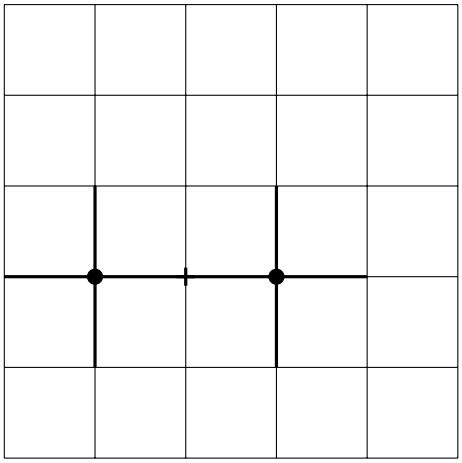
Figure 7: An example of decoding an erasure with the peeling decoder. The edges correspond to the qubits. The thick edges represent the erasure, the black nodes represent the syndrome.



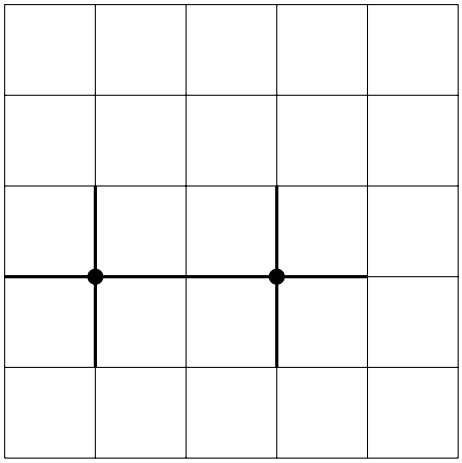
(a) A syndrome. All generators in the syndrome are the initial odd clusters.



(b) The odd clusters grow half an edge.



(c) The odd clusters grow another half edge, and meet.



(d) The clusters merge and form an even cluster. The peeling decoder needs to be applied to find the correction.

Figure 8: A simple example of growing the clusters with the Union-Find decoder. The black nodes represent the syndrome, the thick lines represent the (growing) erasure. (Here, the clusters are both grown in one step; in the actual algorithm, they are grown one by one. It can be checked that it does not matter in this case.)

The growing consists of multiple steps. First of all, always the cluster C with the smallest boundary list is grown. This cluster is picked in step 4 in the algorithm. Which cluster this is can be determined by making use of a priority queue, with the length of the boundary list as primary priority (actually the inverse priority, a smaller boundary lists means a higher priority), and the entry number as secondary priority to let the clusters grow more evenly. Without the secondary priority, a cluster will be grown multiple times in a row while there are multiple clusters of the same size. This can be seen as growing full edges instead of half edges, and will reduce the performance of the decoder.

Why is only the smallest odd cluster is grown?

Taking the length of the boundary list as priority means growing the cluster with the smallest boundary size. To understand this, we can look at the possible error chains. As only odd clusters are grown, we cannot make pairs of the whole syndrome inside the cluster. Thus there is at least one error chain which exits the cluster, as we need to pair at least one syndrome point in the cluster with a syndrome outside the cluster. Growing a cluster means that we add edges to the erasure, of which at least one is correct (at least one error chain). However, for a cluster with a bigger boundary size, there are more incorrect edges which are added to the erasure, which can reduce the performance of the decoder (the threshold will be lower). Thus growing the cluster with the smallest boundary size gives a higher performance than growing all clusters at once.

After picking the smallest cluster C, we run over the boundary list of C in step 5, and grow each incident edge that is not fully grown yet (adding 1 to the value). In this step, we save all edges which are grown to full length. Thus if the value of the edge was 1 before the growing, and is 2 after the growing. For these edges (step 6 in the algorithm), we check whether the incident nodes are part of a different cluster. If so, the clusters are merged.

The merging of the clusters together with checking if nodes are from different clusters is done with a Union-Find algorithm. This has an almost linear time complexity.

The Union-Find algorithm and its time complexity.

R. Tarjan [5] developed a Union-Find algorithm for merging n clusters (Union(u,v), where u and v are the nodes of the clusters which need to be merged) and determining in which cluster a node is m times (the function Find(u) where u is a node). It has a time complexity of $\mathcal{O}(m\alpha(m,n))$ [5], with average time complexity of $\mathcal{O}(\alpha(m,n))$ per operation, where $\alpha(m,n)$ is a version of the inverse Ackermann function defined in the appendix. As the Union-Find decoder has $m \geq 2n$, we use a one-variable version of the inverse Ackermann function with $\alpha(n) = \alpha(2n,n)$. The two important ideas by which the union-find algorithm obtains this time complexity are the following. First of all, the height of the trees is decreased by path compression (every time the function Find(u) is called, all nodes on the path to the root are made children of the root) or similar methods (path splitting or path halving).

Secondly, the smaller cluster is always merged into the bigger cluster (union by size). This ensures that the least amount of nodes have a longer path to the root.

To merge correctly, the boundary lists also need to be fused. This is done in step 7 by appending the boundary list of the smaller cluster to the boundary list of the larger cluster, as they are merged this way.

After the merging, the boundary lists of the clusters which have been changed must be cleaned up by removing all nodes which are not in the boundary anymore. This is done in step 8 of the algorithm.

At last, the priority queue must be updated. This can be done by adding the roots of all clusters which are grown or merged. This may result in duplicates, which is not a problem if we check several things before growing the cluster corresponding to a root. The things to check are the following:

- Is the root actually a root? If this is not the case, the cluster has been merged, and it is probably not the cluster with the smallest boundary list anymore.
- Does the priority correspond to the boundary list of the cluster? If this is not the case, the cluster has been merged, the root is still the same but it is not the smallest odd cluster anymore (it can even be that it is not odd anymore).
- Is the cluster of the given root still odd? It can be the case that the cluster has merged, kept the same root, but became even.

Note that in each case, the root should not have the highest priority; it is either not the root of a cluster, a duplicate, or the root of an even cluster.

Another way the problem of duplicates could be solved is to check for each element which will be added whether it already is in the queue, and replace it if it is. However, this would require looping over the queue after every growth round (which has $\mathcal{O}(n)$ elements), instead of checking a few cases in constant time, and is thus not desirable.

At the end of the algorithm, when all clusters are even, the edges which are fully grown are given as erasure to the peeling decoder, which will give a correction for the original syndrome.

The time complexity

The key point of the algorithm is the growing and merging of the clusters. Before that, the clusters need to be initialized, which can easily be done in linear time. After the growing is finished, the peeling decoder is applied, which also has a linear time complexity. It remains to show that the growing and merging of clusters has a time complexity of $\mathcal{O}(n\alpha(n))$, where n is the number of qubits.

Let us analyse the algorithm step by step. How often do we need to grow a cluster? The shortest Manhattan distance between two clusters is less than or equal to $L = \mathcal{O}(\sqrt{n})$ ($L/2$ vertically and $L/2$ horizontally). In each step, the radius of a cluster grows by 1 edge (half an edge at each side), thus all clusters meet after $\mathcal{O}(\sqrt{n})$ growth steps or less. Thus all steps in the loop should have a time complexity of $\mathcal{O}(\sqrt{n}\alpha(n))$ or less.

In step 4, we can use a priority queue (for example implemented as a heap), which can be updated in constant time.

Algorithm 2 The Union-Find decoder, similar to Algorithm 2 in [1].

Require: The syndrome $\sigma \subset V$ of a Z-error P_Z
Ensure: A Z-error P such that $\sigma(P) = \sigma$.

1: Initialize the cluster trees, Support and the boundary lists for all clusters.
2: Initialize the list of odd cluster roots $\mathcal{L}$
3: **while** $\mathcal{L} \neq \varnothing$ **do**
4: Pick the root of the smallest odd cluster C_s, $s \in \mathcal{L}$.
5: For each v in the boundary of C_s, grow all edges by half an edge. If the edge gets fully grown, add the edge to the fusion list $\mathcal{F}$.
6: For each edge $\{u, v\}$ in $\mathcal{F}$, if Find$(u) \neq$ Find(v), merge the clusters of u and v.
7: For all merged clusters, append the boundary list of the smallest cluster to the boundary list of the largest cluster.
8: For final cluster, update the boundary list by removing all nodes which are not part of the boundary anymore.
9: Append the roots of the odd merged clusters to $\mathcal{L}$.
10: **end while**
11: Apply the peeling decoder with $\varepsilon = \{e \in E | e \text{ is fully grown}\}$ as erasure

In step 5, we loop over the boundary of a cluster. Each node can be at the boundary for at most two rounds of growth. Thus the total time complexity of this step is $\mathcal{O}(n)$. So there are on average $\mathcal{O}(\sqrt{n})$ boundary points per iteration.

In step 6, we loop over a subset of the edges found in step 5, so the loop has time complexity $\mathcal{O}(\sqrt{n})$. For these edges, we need to call the Find function, and possibly the Union function. As we need to call Find at least two times for every Union, $m \geq 2n$. Thus these functions have a cost of $\mathcal{O}(\alpha(n))$, so step 6 has a time complexity of $\mathcal{O}(\sqrt{n}\alpha(n))$ per iteration of the outer loop.

In step 7, we need to append the boundary list for the smaller cluster to the larger cluster for each pair of merged clusters. The time complexity depends on the number of elements in the to be appended list, thus we can calculate the total number instead of calculating the average number per iteration. As always the smallest cluster is merged into the larger one, the final cluster size is at least double the size of the smallest cluster. This means if an element g is followed, the size of the cluster scales faster than 2^k, where k is the number of times the cluster merged to another cluster (so only the merges where the boundary list with g is appended to another boundary list). Note that in most cases, this will be much smaller. As the maximum size of a cluster is n, k scales with $log_2(n)$. Thus the number of times an element is merged is in worst case $\mathcal{O}(\log(n))$. However, only the boundary points are merged. For a cluster of size N (N elements), the number of boundary points is on average $\mathcal{O}(\sqrt{N})$. Let us consider the worst case, where we start with all generators as their own cluster, in such a configuration that they merge each time in pairs, thus the size of the clusters is 2^k with k the number of merges. In each round of merges, there are $n/2^k$ clusters to be merged, each with $\sqrt{2^k}$ boundary points. As the boundary lists of only one of each pair of clusters is merged to the other, we overall obtain a factor $1/2$, which does not matter for the order of the time complexity. As seen before, the clusters can be merged $\log_2(n)$ times, thus the total number of boundary

points which are merged is the result of the following sum:

$$\sum_{k=1}^{\log_2(n)} \frac{n}{2^k} \cdot \sqrt{2^k} = n \sum_{k=1}^{\log_2(n)} \left(\frac{1}{\sqrt{2}}\right)^k \tag{4.6}$$

Using the identity $\sum_{k=1}^{N} r^k = \frac{1-r^{N+1}}{1-r}$, we obtain:

$$n \sum_{k=1}^{\log_2(n)} \left(\frac{1}{\sqrt{2}}\right)^k = n \frac{1 - \frac{1}{\sqrt{2}}^{\log_2(n)+1}}{1 - \frac{1}{\sqrt{2}}} = \frac{n\sqrt{2}}{\sqrt{2}-1} - \frac{n}{\sqrt{2}-1} 2^{-1/2 \log_2(n)} \tag{4.7}$$

$$= \frac{n\sqrt{2}}{\sqrt{2}-1} - \frac{\sqrt{n}}{\sqrt{2}-1} = \mathcal{O}(n) \tag{4.8}$$

Thus step 7 has a total time complexity of $\mathcal{O}(n)$. Note that this is only the case when $\log_2(n) \in \mathbb{Z}$. It is still true when $\log_2(n) \notin \mathbb{Z}$, as shown in the appendix. This can be done far easier when all clusters are grown at the same time. In that case, each point can only be part of a boundary in 2 growth steps, thus the complexity is $\mathcal{O}(n)$.

In step 8, the boundary lists need to be updated. This is done by looping over the used boundary lists, which takes $\mathcal{O}(\sqrt{n})$ time.

Finally, in step 9, the roots of the merged clusters are appended to the list $\mathcal{L}$. As we only grow one cluster each time, all merged clusters merge into one cluster. Thus after one iteration, the final number of odd merged clusters is 0 or 1 (the cluster can be even or odd). Thus appending the root of this cluster to the list is $\mathcal{O}(1)$.

Thus the bottleneck of the algorithm is step 6, which gives a total time complexity of $\mathcal{O}(n\alpha(n))$ ($\sqrt{n}$ from the outer loop, $\sqrt{n}$ from the inner loop and $\alpha(n)$ from the Union and Find operations).

5. Simulations of the Toric Code

In this Chapter, the results of simulating errors and decoding it with the algorithms described in Chapter 4 will be given. With these results, the threshold of the algorithms is obtained.

5.1 The Setup for the Simulations

In order to determine the threshold of the code with the different decoders, it was simulated in Python[1]. The toric code and the decoders were programmed, and then used to simulate errors and corrections for different lattice sizes L and different probabilities on Pauli X errors p_x. Only the Pauli X errors were simulated, as the Pauli Z errors would give the same result. For each combination of L and p_x, an error was simulated a number of times. With these simulations, the threshold was determined by fitting a curve to the data points around $p_x = 0.1$, of the function [12, Equation 2]:

$$p_{succ} = A + Bx + Cx^2 + DL^{-1/\mu} \tag{5.1}$$

where $x = (p_x - p_{th})L^{1/\nu}$. Everything except p_{succ} (the success rate) and p_x is a fitting parameter.[2]

The simulations can be timed to approximate the time complexity. Only the decoding stage has to be timed, as this is the only part that will happen for an implementation in a quantum computer. The simulations were done in Python, on a laptop with a 1.90 GHz AMD Quad Core A10-7300 processor.

5.2 Successful Corrections

As the fitting depend on the success rate, we need to determine when a correction is successful. The errors are known, because they are simulated. Thus to see when a correction is successful, we need to check whether the combination of the error and the correction is equivalent to a logical gate, because applying a logical gate creates an error in the computational state. As the logical gates are loops around the torus, we only need to check if an odd number of qubits has been flipped in the first row and the first column corresponding to an $\hat{X}_1$ and an $\hat{X}_2$ gate. The $\hat{Z}$ gates do not have to be checked, as we only consider X errors in the simulations.

5.3 The Thresholds

As the distance of the Toric Code depends on the size L of the grid, it is natural to think that the percentage of correct corrections per error rate depends on L. A way to

[1] The code is available on Github: `https://github.com/nanleij/The-toric-code.git`

[2] In the original article where the fitting is introducted, p_{succ} is replaced by p_{fail} [13]. This does only matter for the constants, as $p_{fail} = 1 - p_{succ}$.

determine this is to simulate the code with a certain error rate for multiple L. This is done with the MWPM algorithm and with the Union-Find decoder.

5.3.1 The MWPM algorithm

In Figure 9, the resulting graph for $L \in \{3, 5, 7, 9\}$ is plotted for the MWPM algorithm[3]. The horizontal axis represents the probability p_x on a Pauli X error on each qubit. The vertical axis shows the percentage of error-free corrections. The different lines represent the different values of L.

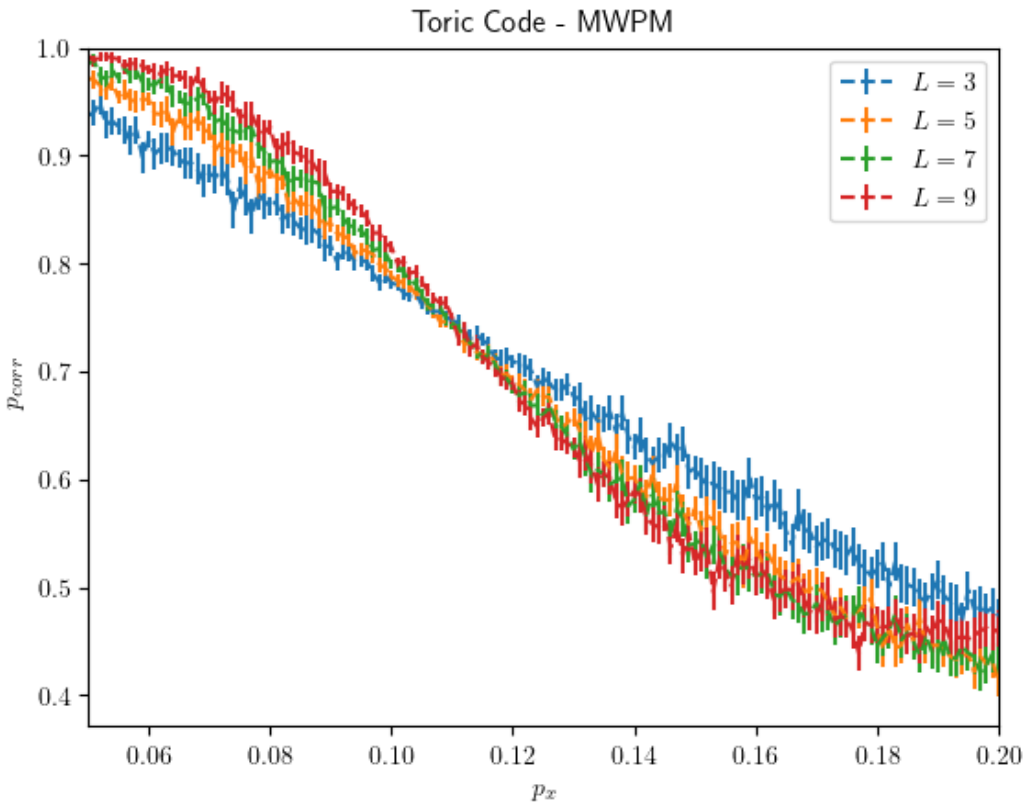


Figure 9: Results of simulation of the Toric Code with the MWPM decoder for multiple odd L from $p_x = 0.05$ to $p_x = 0.20$ with a 95% confidence interval as error bar.

As can be seen, the lines cross each other around $p_x = 11\%$. However, because of finite size effects, the threshold should be a bit lower, which is in agreement with the

[3]The MWPM algorithm was not implemented, a function of the Python package 'NetworkX' was used instead [14]

threshold commonly found in literature of 10.3% [2]. Fitting the curve given in equation 5.1 to the data for $L \in \{5, 7, 9, 11\}$ and for $p_x = 0.095$ to $p_x = 0.112$ gives a threshold of $p_{th} = 0.115 \pm 0.002$, where the error is given by the standard deviation. This is in disagreement with our expectations and the threshold of 0.103% found in literature.

In Figure 10, the fitted curve with the rescaled data points can be seen. As the curve is defined with $x = (p_x - p_{th})L^{1/\nu}$ (the rescaled error rate), this is shown on the horizontal axis. The curve has an offset depending on L, thus to get one curve instead of one for each L, the offset is subtracted from the success probability for each data point, to get the modified success probability. Note that there are many points lying relatively far from the curve, an indication that the fit is not very good. There are multiple possible reasons for the disagreement. First of all, because the simulations were done on a laptop with a 1.9GHz processor, the possibilities for simulating high grid sizes L were very limited. For $L = 11$, the time needed was already about a day for 10.000 simulations per data point, and it would be much higher for large L because of the time complexity.

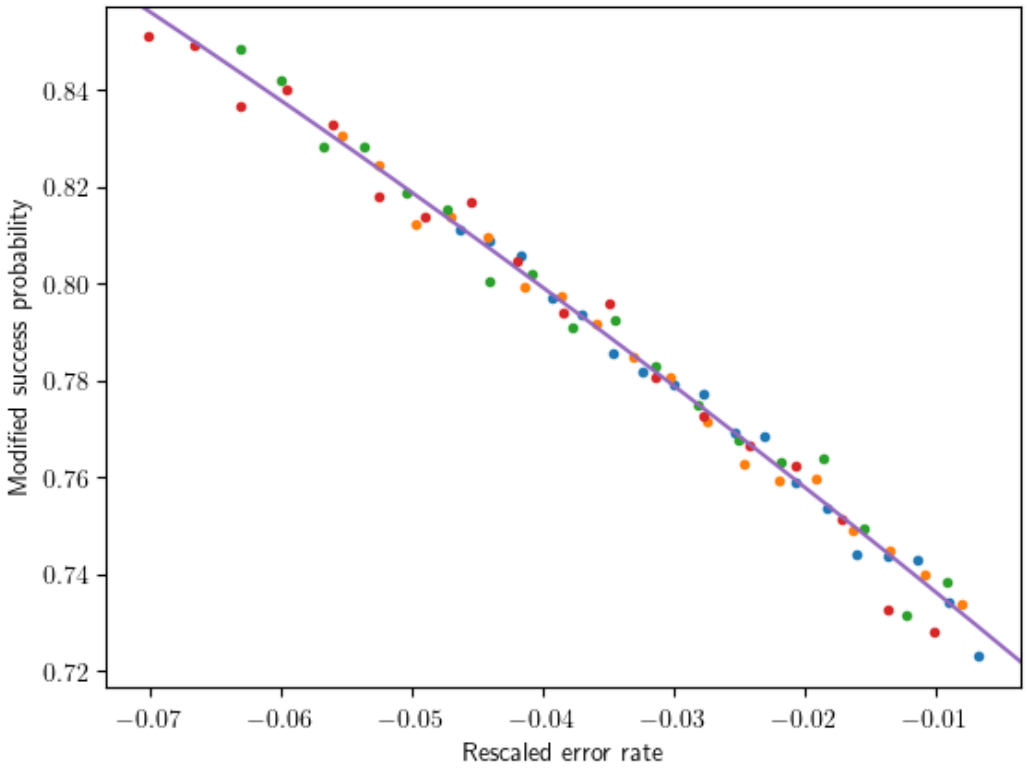


Figure 10: The fitted curve to the data of the MWPM decoder, together with the rescaled data points for $L \in \{5, 7, 9, 11\}$ and $p_x = 0.095$ to $p_x = 0.112$. On the horizontal axis, the rescaled error rate $(p_x - p_{th})L^{1/\nu}$ is shown. The vertical axis represents the modified success probability. The different colors represent different grid sizes L.

5.3.2 The Union-Find decoder

Let us now look at the curves for the Union-Find decoder. As this decoder runs much faster than the MWPM decoder, it was possible to simulate it 10.000 times per data point for $L = 41$. In Figure 11, the graph for the simulations of the Union-Find decoder of $L \in \{9, 17, 25, 33, 41\}$ is plotted, with on the horizontal axis the probability p_x between 0.09 and 0.12. The vertical axis shows as before the percentage of error-free corrections.

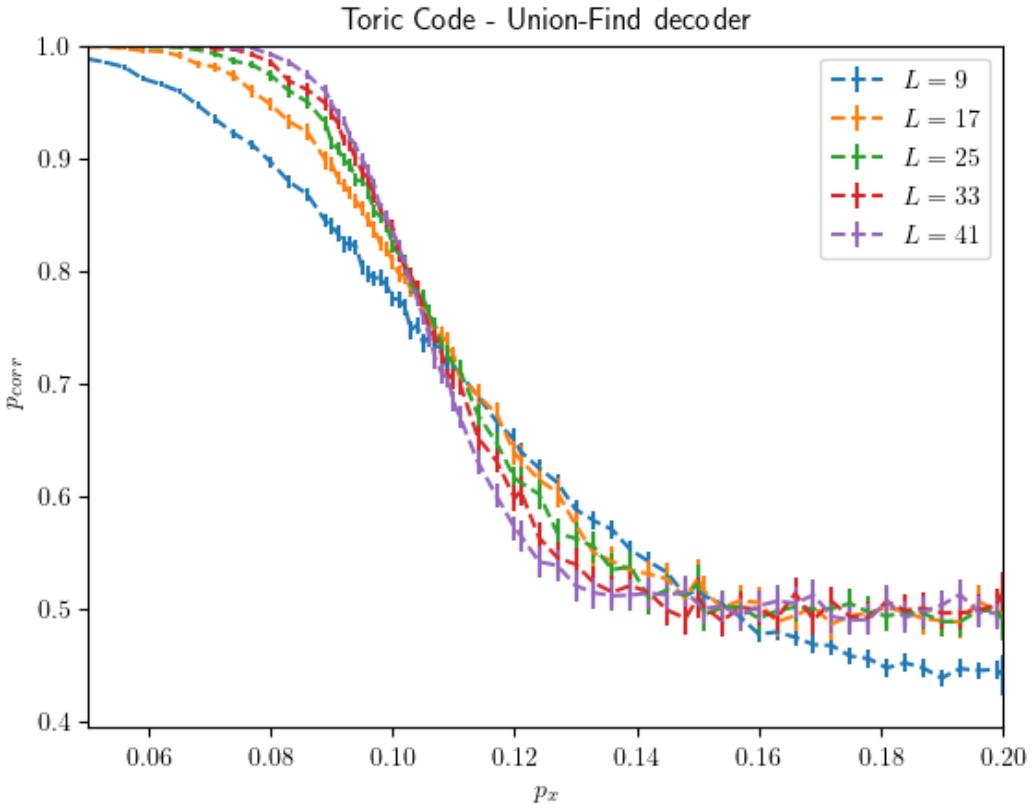


Figure 11: Results of simulation of the Toric Code with the Union-Find decoder for $L \in \{9, 17, 25, 33, 41\}$ from $p_x = 0.05$ to $p_x = 0.20$ with a 95% confidence interval as error bar.

Fitting the curve of equation 5.1 to the data gives a threshold of $p_{th} = 0.097 \pm 0.009$, which is in agreement with the result found by N. Delfosse and N. Nickerson [1] of $p_{th} = 0.099$. In Figure 12, the fitted curve is shown, with the rescaled data points. As the curve depends on L, the term $DL^{-1/\mu}$ was subtracted from the percentage of successful corrections to retrieve one curve, instead of one curve for each L.

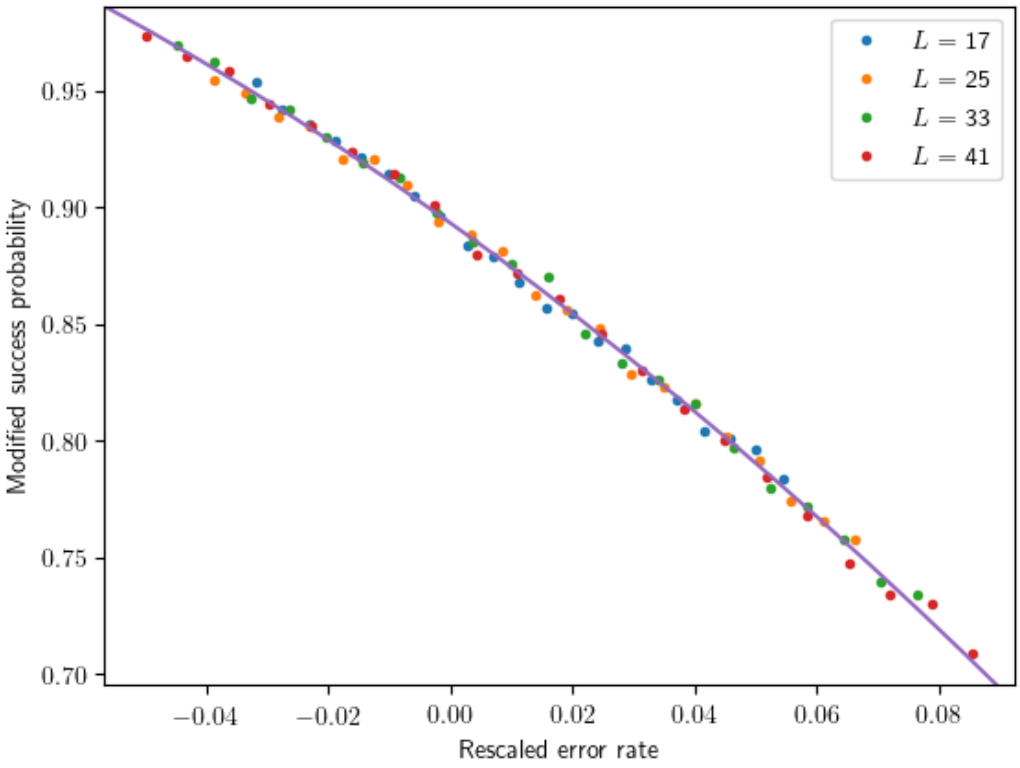


Figure 12: The fitted curve to determine the threshold, together with the rescaled data points for $L \in \{17, 25, 33, 41\}$ and for $p_x = 0.09$ to $p_x = 0.11$. On the horizontal axis, the rescaled error rate $(p_x - p_{th})L^{1/\nu}$ is shown. The vertical axis represents the modified success probability.

5.4 The Difference Between Odd and Even L

Let us look at the same plots, but now for even and odd L. In Figure 13, the plots are shown. As can be seen, for even L the number of error-free corrections goes to 100% when p_x goes to 1, whereas for even L it goes to 0%. This can be explained by how we determined if a correction was correct. If there is an X error on every qubit, there is no error detected, as every generator will commute with the error (an even number of X errors next to a generator always commutes with the generator). The error can be seen as applying the logical gate $X_1^L X_2^L$, which will be equal to the identity if and only if L is even. Thus for odd L, $p_x = 1$ means that $p_{corr} = 0$, and for even L $p_{corr} = 1$. For small deviations, the code corrects toward errors on all qubits, thus the graph should be about (anti) symmetric around $p_x = 0.5$. A consequence of the difference between even and

odd L is that two fitting parameters, D and μ, are different depending on the parity of L [13]. The reason only odd L are considered in the rest of the simulations is that these differences do not matter for the threshold, and considering only one parity decreases the number of fit parameters by 2.

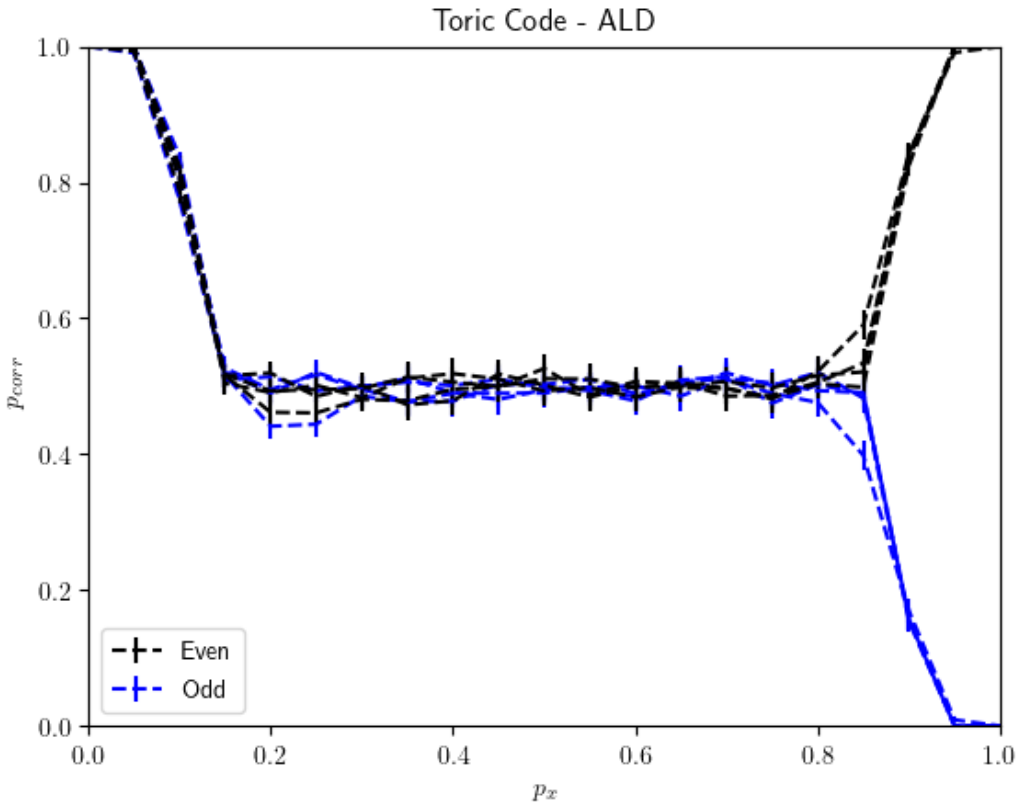


Figure 13: Simulations for even and odd L. As can be seen, for even L the graph is about symmetric, and for odd L it is about antisymmetric. For each data point, 2000 simulations were done.

5.5 The Time Complexity

As one of the biggest advantages of the Union-Find decoder is the time complexity, it is natural to check if it indeed runs in almost linear time. This was done by simulating the decoder for certain probabilities p_x 10.000 times, for different L. Then the total time needed for the decoding per L was plotted versus the number of qubits $n = 2L^2$. This

was done for both the MWPM algorithm and the Union-Find decoder[4].

5.5.1 The MWPM algorithm

The MWPM decoder is fully based on the MWPM algorithm, which has at least a time complexity of $\mathcal{O}(n^2\sqrt{n})$. Knowing this, it is not really necessary to test the implementation used, as it will be worse than the Union-Find decoder. However, it is good to compare the decoders when used for the same error probabilities. In Figure 14, the time needed for simulations with the MWPM algorithm is plotted, with the time on the vertical axis and the number of qubits n on the horizontal axis. Each line represents a different probability on an error p_x, starting at 0.01 for the lowest line, running up to 0.05 for the highest line, with a spacing of 0.01.

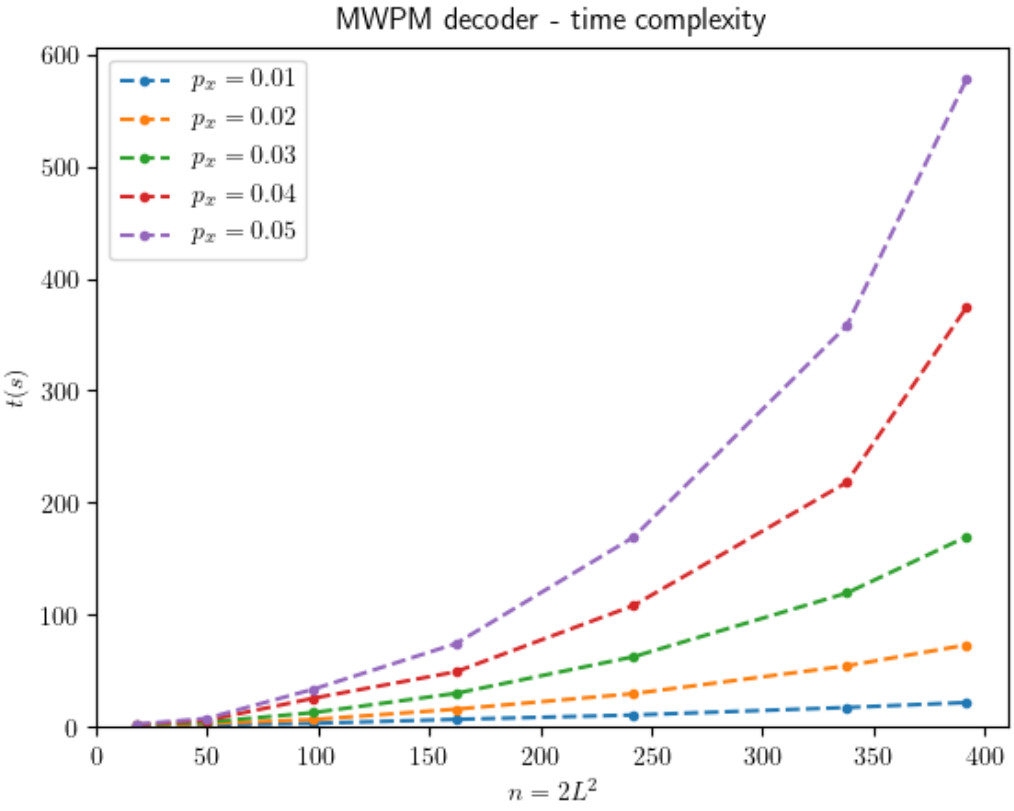


Figure 14: The time needed for 10.000 simulations (per data point) of the MWPM algorithm, the total time plotted versus the number of qubits. The different lines represent different p_x, ranging from 0.1% (the lowest line) to 0.5% (the highest line).

[4]The peeling decoder was also implemented, as it is needed for the Union-Find decoder. However, the time complexity is not determined; if the time complexity is not linear, this will be noted when looking at the time complexity of the Union-Find decoder.

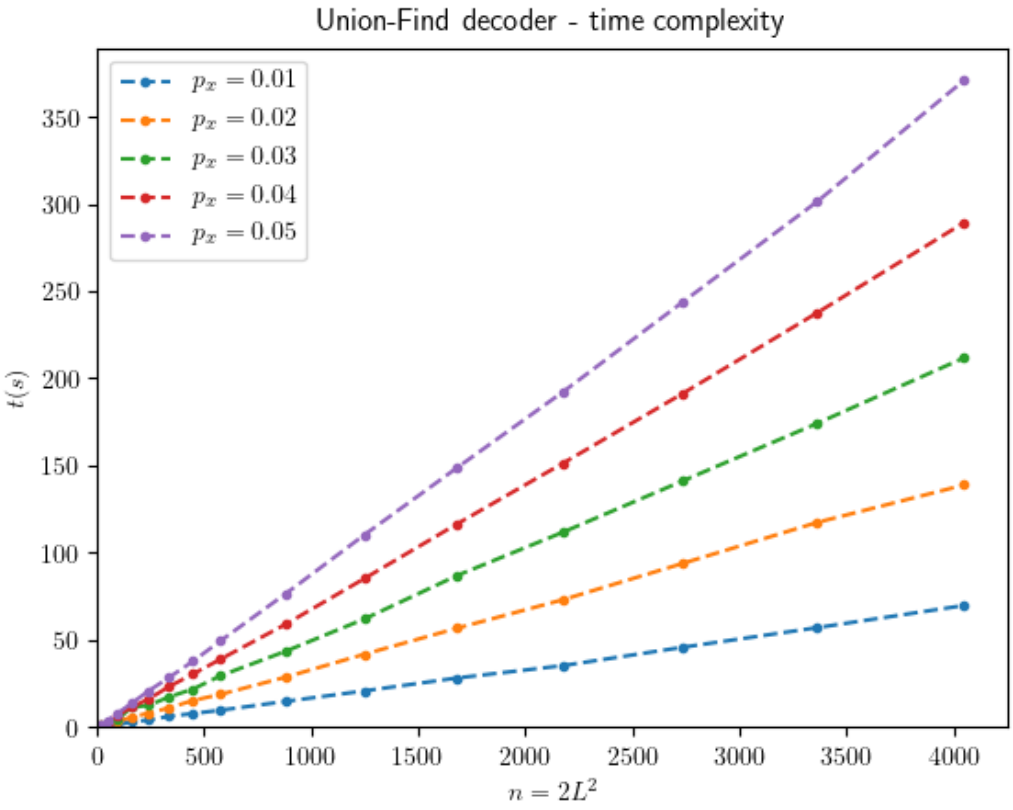


Figure 15: The time needed for 10.000 simulations (per data point) of the Union-Find decoder, the total time plotted versus the number of qubits. The different lines represent different p_x, ranging from 0.1% (the lowest line) to 0.5% (the highest line).

5.5.2 The Union-Find decoder

As the decoder theoretically runs in almost linear time in the number of qubits, the resulting plot should be a straight line for each p_x ($\alpha(2n, n)$ is constant between $n = 16$ and $n = 2^{16}$). This is indeed the case, as can be seen in Figure 15. The same error probabilities as before were used, but it was possible to use higher grid sizes L, up to $L = 45$.

5.5.3 Comparing the decoders

It should be noted that the MWPM algorithm already takes almost 600 seconds to run 10.000 simulations for $p_x = 0.05$ and $L = 14$, whereas the Union-Find decoder uses about half that time for $L = 41$ with the same p_x. As noted before, for higher probabilities

of error p_x, the time needed for the MWPM grows must faster per L, whereas it looks more linearly spaced for the Union-Find decoder. It is interesting to look further at how the average time needed depends on the probability instead of on the number of qubits, as this can also be important for determining how whether decreasing the error rate in a quantum computer is worth it. If the time needed increases fast for increasing p_x (for example quadratically or exponentially), this is sooner the case than when it increases slowly (for example linearly).

For the MWPM algorithm, the time needed to complete the algorithm completely depends on the size of the syndrome. Increasing L for a certain p_x increases the number of syndrome points, but increasing p_x for a certain L increases it also for low p_x, as each qubit with an error will add generators to the syndrome. For higher p_x however, it is more likely that adding an error on a qubit replaces a generator in the syndrome, instead of adding new ones. Thus only for low p_x, the time complexity in p_x will be similar to the time complexity in the number of qubits n.

For the Union-Find decoder, the time needed is less directly related to the number of generators in the syndrome. However, for low p_x the errors will be relatively far apart. This means that most odd clusters will only need to grow half an edge, so the time needed increases linearly with the number of initial clusters (the number of generators in the syndrome). Thus for low p_x, the time complexity will be about linear in p_x.

6. Conclusion

This thesis provides an introduction in quantum error correction, in specific for codes described with the stabilizer formalism. The toric code was examined, and multiple decoders were looked at. The Minimum Weight Perfect Matching decoder and the Union-Find decoder [1] were implemented in Python 3, and simulations were done to determine the threshold and the time complexity of the decoders.

The threshold determined through the simulations were 0.115 ± 0.002 for the MWPM decoder and 0.097 ± 0.009 for the Union-Find decoder. The threshold for the MWPM decoder is not in agreement with earlier results of 0.103 [2], possibly because of the grid sizes used. The threshold for the UF decoder is in agreement with the result of 0.099 found by N.Delfosse and N. Nickerson [1].

The time complexity of the weighted growth function of the Union-Find decoder was analysed, and found to be $\mathcal{O}(n\alpha(n))$, where $\alpha(n)$ is an inverse of the Ackermann function. This is in agreement with the sketch of the analysis of the version without weighted growth [1], even though there are significant differences in the way to prove it. This was tested with simulations, and found to be correct.

Small errors were made during the implementation of the Union-Find decoder, which were not found at first because the errors were made during optimizations after the testing. This can be prevented next time by extensive testing after each optimization or addition to the code.

For further research, it would be interesting to look at the time complexity in the probability on errors p_x. For low p_x, this is expected to be similar to the time complexity in the number of qubits n, but for large p_x this could be different. Another option for further research is implementing the decoder for faulty measurements, and look at the time complexity in the amount of rounds of correction.

References

[1] Delfosse, N., & Nickerson, N. H. (2017). *Almost-linear time decoding algorithm for topological codes*. arXiv:1709.06218v1

[2] Browne, D. (2014). *Topological Codes and Computation*. Lecture Notes, University of Innsbruck, Innsbruck, Germany.

[3] Preskill, J. (2018). *Quantum computing in the NISQ era and beyond*. Quantum, 2, 79

[4] Kitaev, A. Y. (2003). *Fault-tolerant quantum computation by anyons*. Annals of Physics, 303(1), 230

[5] Tarjan, R. E. (1975)*Efficiency of a Good But Not Linear Set Union Algorithm*. Journal of the ACM, 22(2), 215225.

[6] Nielsen, M., Chuang, I. (2010). *Quantum computation and quantum information*. Cambridge: Cambridge University Press.

[7] Paulsen, V. (2016).*Entanglement and Non-Locality*. Lecture Notes, Department of Pure Mathematics and Institute for Quantum Computation, University of Waterloo, Ontario, Canada.

[8] Bravyi, S., & Kitaev, A. (2005). *Universal quantum computation with ideal Clifford gates and noisy ancillas*. Physical Review A, 71(2).

[9] Dennis, E., Kitaev, A., Landahl, A. and Preskill, J. (2001). *Topological quantum memory*. Journal of Mathematical Physics, 43(9), 44524505

[10] Micali, S., & Vazirani, V. V. (1980). *An* $\mathcal{O}(\sqrt{|V|}\cdot|E|)$ *algoithm for finding maximum matching in general graphs*. In 21st Annual Symposium on Foundations of Computer Science (sfcs 1980). IEEE.

[11] Delfosse, N., & Zémor, G. (2017). *Linear-Time Maximum Likelihood Decoding of Surface Codes over the Quantum Erasure Channel*. arXiv:1703.01517

[12] Anwar, H., Brown, B. J., Campbell, E. T., & Browne, D. E. (2014). *Fast decoders for qudit topological codes*. New Journal of Physics, 16(6)

[13] Wang, C., Harrington, J., & Preskill, J. (2003). *Confinement-Higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory*. Annals of Physics, 303(1), pp. 3158

[14] Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). *Exploring network structure, dynamics, and function using NetworkX*. in Proceedings of the 7th Python in Science Conference (SciPy2008), Gel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 1115

[15] Tarjan, R. E., & van Leeuwen, J. (1984). *Worst-case Analysis of Set Union Algorithms*. Journal of the ACM, 31(2)

A. Short proofs and identities

A.1 Proofs related to the trace

In section 2.1.3, the trace was introduced. The following theorem was given, but not proven:

Theorem A.1. $\mathrm{Tr}(A)$ *is basis-independent*

Here we will give the proof. To simplify it, we first prove the following:

Lemma A.1. *The following holds for any orthonormal basis and operators A and B on vector space V:*

$$\mathrm{Tr}(AB) = \mathrm{Tr}(BA)\,. \tag{A.1}$$

Proof. Let $\{|i\rangle\}$ be an orthonormal basis. Then, as $\sum_i |i\rangle\langle i| = I$, we have

$$\begin{aligned}
\mathrm{Tr}(AB) &= \sum_i \langle i|AB|i\rangle \\
&= \sum_i \sum_j \langle i|A|j\rangle\ \langle j|B|i\rangle \\
&= \sum_j \sum_i \langle j|B|i\rangle\ \langle i|A|j\rangle \\
&= \sum_j \langle j|A|j\rangle \\
&= \mathrm{Tr}(BA)\,.
\end{aligned}$$

□

Now we will prove Theorem 2.1.

Proof. Let $\{|v_i\rangle\}$ and $\{|w_i\rangle\}$ be any two orthonormal bases for V. Then there exists a unitary operator U for which

$$|w_i\rangle = U\,|v_i\rangle\ . \tag{A.2}$$

Using the lemma, we can write:

$$\begin{aligned}
\mathrm{Tr}_v(A) &= \mathrm{Tr}_v(AUU^\dagger) \\
&= \mathrm{Tr}_v(U^\dagger AU) \\
&= \sum_i \langle v_i|U^\dagger AU|v_i\rangle \\
&= \sum_i \langle w_i|A|w_i\rangle \\
&= \mathrm{Tr}_w(A)\,.
\end{aligned}$$

□

A.2 Identities

In section 2.5, the depolarizing channel was rewritten using the following equality.

$$\frac{I}{2} = \frac{1}{4}(\rho + X\rho X + Y\rho Y + Z\rho Z)\,. \tag{A.3}$$

Here we will prove that this holds for arbitrary 1-qubit density matrices ρ. First of all, define the following function:

$$\mathcal{E}(A) = \frac{1}{4}(A + XAX + YAY + ZAZ) \tag{A.4}$$

Using $X^2 = Y^2 = Z^2 = I$, we know that $\mathcal{E}(I) = I$ Using this and the anticommutation relations, it can be easily seen that $\mathcal{E}(X) = \mathcal{E}(Y) = \mathcal{E}(Z) = 0$. For example, as X, Y, and Z pairwise anticommute, we have

$$\mathcal{E}(X) = \frac{1}{4}(X + X^3 + YXY + ZXZ) \tag{A.5}$$

$$= \frac{1}{4}(X + X - XYY - XZZ) = 0\,. \tag{A.6}$$

Recalling equation 2.32, an arbitrary density matrix can be written as

$$\rho = \frac{1}{2}(I + a_1 X + a_2 Y + a_3 Z)\,. \tag{A.7}$$

With these results and with the linearity of the function, we can write for arbitrary ρ:

$$\frac{1}{4}(\rho + X\rho X + Y\rho Y + Z\rho Z) = \mathcal{E}(\rho) \tag{A.8}$$

$$= \frac{1}{2}(\mathcal{E}(I) + a_1\mathcal{E}(X) + a_2\mathcal{E}(Y) + a_3\mathcal{E}(Z)) \tag{A.9}$$

$$= \frac{I}{2}\,, \tag{A.10}$$

which is what was needed.

B. The Inverse Ackermann Function

When computing the time complexity of the Union-Find decoder, an inverse of an Ackermann function was used. The inverses used in the article which introduced the Union-Find decoder [1] and in the article which analysed the union-find algorithm [5] are slightly different, as are the used Ackermann functions. The Ackermann function used in the time complexity of the Union-Find decoder is the following [1]. Let the function $A_1 : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ be defined by the following relations:

$$\forall j \in \mathbb{N}_0 : A_1(0, j) = 2j\,, \tag{B.1}$$
$$\forall i \in \mathbb{N}_0 : A_1(i, 0) = 0\,, \tag{B.2}$$
$$\forall i \in \mathbb{N}_0 : A_1(i, 1) = 2\,, \tag{B.3}$$
$$\forall i \geq 1, j \geq 2 : A_1(i, j) = A_1(i - 1, A_1(i, j - 1))\,. \tag{B.4}$$

The used inverse is a one variable function, defined as follows:

$$\alpha_1(n) = \min\{i | A_1(i, 4) \geq \log_2 n\}\,. \tag{B.5}$$

Tarjan used the same Ackermann function in his original paper about the union-find algorithm, but a different inverse. In his second paper, the function was slightly modified. Let $A_2 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be defined by the following relations:

$$\forall j \geq 1 : A_2(1, j) = 2^j\,, \tag{B.6}$$
$$\forall i \geq 2 : A_2(i, 1) = A(i - 1, 2)\,, \tag{B.7}$$
$$\forall i, j \geq 2 : A_2(i, j) = A_2(i - 1, A_2(i, j - 1)\,, \tag{B.8}$$

with the inverse

$$\alpha_2(m, n) = \min\{i | A_2(i, \lfloor m/n \rfloor) > \log n\}\,, \tag{B.9}$$

where $\lfloor \cdot \rfloor$ denotes the floor function.

The union-find algorithm was proven to have a worst case time complexity of $\Theta(m\alpha_2(m, n))$ for $m \geq n$, when using m finds and n unions. As the difference between $\log_2(n)$ and $\log(n)$ is only a small factor compared to the growth of the Ackermann functions, we will look at $\log_2(n)$ for both inverses.

In the Union-Find decoder, one has to call the find function maximal twice as much as the union function, as every two times the find function is called, it is decided on the outcome of that whether the union function is called. The functions are called $\mathcal{O}(n)$ times, thus we are interested in $\alpha_1(n)$ and $\alpha_2(2n, n)$. Thus the inverse ackermann function used in this thesis is $\alpha(n) = \alpha_2(2n, n)$. Note that in the description of Algorithm 2 in [1], the find function is called at least 4 times for every union function, thus $\alpha_2(4n, n)$ is also interesting. So in order to compare the functions, we need the ranges for these functions have certain values.

For the definition of $\alpha_1(n)$, we need $A_1(i, 4)$. Using the equation B.4, we obtain $A_1(1, j) = A_1(0, A_1(1, j - 1)) = 2A_1(1, j - 1)$, thus (using the base case, equation B.3) $A_1(1, j) = 2^j$.

Using equation B.4, we also calculate $A_1(2,j) = A_1(1, A_1(2, j-1)) = A_1(1, A_1(1, \dots 2)) = 2^{2^{2^{\cdot^{\cdot}}}}$ with j two's.

This gives

$$A_1(0,4) = 8\,, \tag{B.10}$$

$$A_1(1,4) = 2^4 = 16\,, \tag{B.11}$$

$$A_1(2,4) = 2^{2^{2^2}} = 2^{16} = 65336\,, \tag{B.12}$$

$$\begin{aligned} A_1(3,4) &= A_1(2, A_1(3,3)) = A_1(2, A_1(2, A_1(3,2))) \\ &= A_1(2, A_1(2, A_1(2, A_1(3,1)))) = A_1(2, A_1(2, A_1(2,2))) \\ &= A_1(2, A_1(2,4)) = a_1 = 2^{2^{\cdot^{\cdot}}} \text{ with } 65336 \text{ two's}\,. \end{aligned} \tag{B.13}$$

This number is bigger than the number of atoms in the universe (about $10^{80} \approx 2^{265} \ll 2^{65336}$), so for any physical grid of qubits, $\alpha_1(n) \leq 2$. (recall that $\log_2(n)$ was used in the definition of α_1.

For $\alpha_2(2n,n)$, we need $A_2(i,2)$. Similar to A_1, we have: $A_2(2,j) = A_2(1, A_2(2, j-1)) = ... = A_2(1, A_2(1, ..., A_2(1,2)) = 2^{2^{\cdot^{\cdot}}}$ with $j+1$ two's. So we have:

$$A_2(2,2) = 2^{2^2} = 16\,, \tag{B.14}$$

$$A_2(3,2) = A_2(2, A_2(3,1)) = A_2(2, A_2(2,2)) = A(2,16) = 2^{65336}\,. \tag{B.15}$$

As before, $2^{A_2(3,2)}$ is much larger than the possible number of qubits, thus $\alpha_2(2n,n) \leq 3$.

For $\alpha(4n,n)$, we need $A_2(i,4)$. Similar to before, we have:

$$A_2(1,4) = 2^4 = 16\,, \tag{B.16}$$

$$A_2(2,4) = 2^{2^{2^2}} = 65336\,. \tag{B.17}$$

Thus we have again that $\alpha_2(4n,n) \leq 2$. Let us summarize the results in the following table.

Table 1: Values for the Ackermann functions.

$i \setminus$ function	$A_1(i,4)$	$A_2(i,2)$	$A_2(i,4)$
1	16	4	16
2	65336	16	65336
3		2^{65336}	

As can be seen, the relevant values of $\alpha_1(n)$ and $\alpha_2(4n,n)$ are about the same (the ranges in n differ by a factor log(2), because of the difference in the definition)[1] , and $\alpha_2(2n,n)$ is only slightly shifted (3 instead of 2 for $n > 2^{16}$, 2 instead of 1 for $2^4 < n < 2^{16}$). In conclusion, the difference in the definition of the inverse Ackermann function does not matter for the results.

[1]This is most likely the reason why the (easier) one-variable definition was used instead of the two variable definition. However, it is the same because the find function is at least 4 times used each time the union function is used. One can argue that the last two times the Find functions is used are

unnecessary. The first two times, there is checked whether nodes are in the same cluster. When they are not, the clusters of the nodes are merged. The third time, the boundary lists of the smaller cluster are appended to the boundary list of the larger cluster. Find is used to determine again which clusters were merged, but this is unnecessary as one can do this immediately after the union, when the roots are still available. The last time, the roots in the initial list are replaced by the new roots, where the new roots are found by using Find for each old root. This is also unnecessary, as one can keep track of the new roots when merging the clusters.

C. The Time Complexity of Appending Boundary Lists

In section 4.7, the time complexity of the Union-Find decoder has been determined. For one step (step 7), it was done in the special case that $n = 2^m$ for some $m \in \mathbb{N}$. The sum which had to be calculated was the following:

$$\sum_{k=1}^{\log_2(n)} \frac{n}{2^k} \sqrt{2^k}, \tag{C.1}$$

because each cluster can at most merge in $\log_2(n)$ rounds, has at least a size of 2^k (thus there are $n/2^k$ clusters), and a boundary of about $\sqrt{2^k}$. For $n = 2^m$ with $m \in \mathbb{N}$, this was computed to be $\mathcal{O}(n)$. When $n \neq 2^m$ for some $m \in \mathbb{N}$, the upper bound of the sum is not an integer anymore, so it will differ slightly. However, it is at least smaller than the same sum with $\lceil \log_2(n) \rceil$ as upper bound, where $\lceil \cdot \rceil$ denotes the ceiling function. Here we will show that this is still $\mathcal{O}(n)$.

$$n \sum_{k=1}^{\lceil \log_2(n) \rceil} \left(\frac{1}{\sqrt{2}} \right)^k = n \frac{1 - \frac{1}{\sqrt{2}}^{\lceil \log_2(n) \rceil + 1}}{1 - \frac{1}{\sqrt{2}}} = n \frac{\sqrt{2}}{\sqrt{2} - 1} - \frac{n}{\sqrt{2} - 1} 2^{-1/2 \lceil \log_2(n) \rceil} \tag{C.2}$$

Take $m = 2^{\lceil \log_2(n) \rceil}$, then $m \geq 2^{\log_2(n)} = n$. Thus

$$n \frac{\sqrt{2}}{\sqrt{2} - 1} - \frac{n}{\sqrt{2} - 1} 2^{-1/2 \lceil \log_2(n) \rceil} = n \frac{\sqrt{2}}{\sqrt{2} - 1} - \frac{n}{\sqrt{2} - 1} 2^{-1/2 \log_2(m)} \tag{C.3}$$

$$= n \frac{\sqrt{2}}{\sqrt{2} - 1} - \frac{n}{\sqrt{m}(\sqrt{2} - 1)} \tag{C.4}$$

Note that $n \leq m \leq 2n$, thus $m = \mathcal{O}(n)$. Thus $n/\sqrt{m} = \mathcal{O}(\sqrt{n})$, and the total sum is $\mathcal{O}(n) - \mathcal{O}(\sqrt{n}) = \mathcal{O}(n)$. So the statement that step 7 is $\mathcal{O}(n)$ is still true when $\log_2(n) \notin \mathbb{N}$.