Fast Post Processing Algorithms For Fault Tolerant Quantum Computation using Surface Codes

by

Arman Zaribafiyan

BSc, Isfahan University of Technology, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Electrical and Computer Engineering)

The University Of British Columbia
(Vancouver)

September 2014

Abstract

Local architecture of qubits in two dimensional lattices is known as one of the candidates to run fault tolerant quantum computation with high threshold. Topological quantum error correcting codes, such as surface codes, are used to make this architecture robust against various quantum error models. In this research we present a fast, concise, operationally inexpensive and highly parallelizable decoding algorithm for surface codes without using concatenation which has a threshold range of 8.6% to 10.5% varying based on a parameter called *OMSS*. Thanks to parallelization of the proposed algorithm, the time complexity scales logarithmically in the lattice size for small *OMSS*. This can be compared to the work of [9, 10] which has a threshold of 8% for the bit-flip error channel within the same complexity order.

Preface

This dissertation is original and unpublished work by the author, A. Zaribafiyan.

The ”Adjusment of Weights” idea in chapter 3 is suggested by my supervisor Prof. Robert Raussendorf, and the rest of the work is my original work and implementation.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Glossary

MWPM Minimum Weight Perfect Matching

OMSS Optimal Matching Subgraph relative Size

FTQEC Fault Tolerant Quantum Error Correction

Acknowledgments

First and foremost, I would like to thank Prof. Robert Raussendorf for his priceless guidance, for being my teacher and fully supportive supervisor. This thesis would not have been possible without his advice, suggestions and constant encouragement.

Furthermore I would like to extend my gratitude to Prof. Konrad Walus for being my supportive co-supervisor, my colleagues in the Quantum Information Group, especially Dr. Leon Loveridge for proof reading the thesis and providing valuable comments. I would also want to thank all of my friends at UBC for their discussions and for creating memorable and pleasant moments during the course of this research.

Finally, a heartfelt and deep thanks go to my family, who have always encouraged me and given their unconditional support through my studies at the University of British Columbia. I could not have completed this thesis without their support.

Dedication

To my dearest wonderful Love,

Negar,

who made me able to conduct this research with her true love and patience, despite we were geometrically far distant from each other,
and to my family who always supported me by all means.

Chapter 1

Introduction

If I have seen farther it is by standing on the shoulders of Giants.
— Sir Isaac Newton (1642 - 1727)

1.1 Plan of the Thesis

Quantum computers have shown promising results solving problems which are currently computationally hard for classical computers [19][36][25]. Although there has been groundbreaking progress in realizing such devices [2][20][26][21][27], scaling them to a size that can handle real world problems is still a high-tech challenge. This is because quantum computers use the quantum mechanical properties of certain physical systems which are very fragile and can be distorted by the environmental noise. Accumulation of this noise makes it hard to scale these systems up. However, *threshold theorem* proved that fault tolerant quantum computation with noisy devices is possible with arbitrary accuracy provided that the noise level of the subsystems are below a threshold [1]. So quantum computation in its large size meaning is impossible without considering the effect of noisy systems.

Fault tolerance is the idea of using error correction to protect a quantum state from quantum noise. There are various proposed methods for fault tolerant quantum computation using different error correcting approaches. Different error correcting schemes are mostly compared by the *error threshold* they can achieve and the complexity of their encoding/decoding algorithms.We will discuss error thresh-

old in more detail in section 1.4.2. Among all these error correction schemes, *surface codes* with their simple architecture and high error threshold value [1] are so far one of the best candidates to be the chosen protocol for fault tolerant quantum computation.

As it will be more elaborated in the remainder of this thesis, certain classical processing is involved in the quantum error correction with surface codes. The error threshold of a decoding scheme is closely related to this classical processing step. In the case of surface codes this classical post processing boils down to finding the minimum weight perfect matching on a complete graph. This post processing is essential to fault tolerant quantum computation and the quantum speed up advantage can be lost if we need to spend a lot of time on the classical post processing. Therefore, we need to reduce the time complexity of these classical algorithms as much as we can in order not to delay the quantum computation with slow classical processing. There is also a trade of between the complexity and the threshold gained using these algorithms. So an ultimate goal would be to devise a low complexity post processing algorithm which maintains a high threshold. There has been different approaches in the literature to either reduce the complexity of the classical processing or gain a higher error threshold [11, 12, 24]. We used a greedy algorithm with a weight adjustment step to both reduce the complexity to that of [9]. Then we have also shown that a hybrid approach using our greedy approximation and Edmonds matching algorithm [12] can achieve an error threshold higher than [9] while keeping the complexity at the same level. Our proposed algorithm and it's numerical results and comparisons are explained in detail in chapter 3.

In the remainder of the thesis, first we discuss the importance and significance of using quantum computers along with a short summery of quantum mechanics background. Then we introduce the idea of fault tolerant quantum computation and its use of error correction. A whole section is dedicated to fundamentals of error correcting codes. Then with a definition of quantum error models in section 1.3.3, we initiate the discussion on quantum error correcting codes. A series of examples on quantum error correcting codes will help us establish a formal analogy between linear classical codes and quantum codes. Surface codes are a subclass

[1] Due to their robustness against local errors

of a more general family of quantum error correcting codes, which are called stabilizer codes. So in section 1.3.10, we introduce the stabilizer formalism and then we conclude the first chapter by an introduction to Surface codes in section 1.4, which are the main target of this research. This introduction includes the formal definition of surface codes, the geometric picture and how the topology plays a role in the robustness of these codes against errors, and then we explain how the actual error correction is done using surface codes in section 1.4.2. A formal definition of error threshold as a measure of the quality of an error correcting code is given in the same section.

In chapter 2 we review the steps of error correction with surface codes in more detail and in an algorithmic fashion. Then we introduce our numerical analysis method and how the simulator is developed. We conclude this chapter by a numerical test example to show that our developed numerical simulation engine is consistent with the literature. Then in chapter 3 we propose our algorithm along with different ways to improve it and its numerical results along with the bench markings with the current methods in the literature. Finally we conclude the thesis in chapter 4 along with a number of proposed challenges for future works.

1.2 Quantum Computers

1.2.1 Motivation

It's easy to imagine how the birth of computers changed our lives. Smaller yet faster processors were rapidly manufactured one after another enabling us to do more sophisticated calculations in shorter time and using less resources. It opened a wider area of problems that we can tackle efficiently. Despite all this rapid progresses, there are classes of problems which are generically very hard to solve [17]. What do we mean by hardness of a problem? Each paradigm of calculation brings a complexity theory with itself. If you have a universal paradigm of computation, then there has to be a mapping between any arbitrary operation on an input and the set of instructions in your computer. Complexity theory looks at the number of instructions you need to employ in order to run a certain operation. The more instructions you need to run a certain operation, the harder it will be for the com-

puter to handle that operation. Looking at the classical gate model paradigm of computation, the complexity theory classifies problems into certain groups based on their hardness:

P problems The set of problems which can be solved with a Turing machine in polynomial time.

NP problems Class of computational problems for which solutions can be computed by a non-deterministic Turing machine in polynomial time.

NP-hard Class of problems at least as hard as hardest problems in NP.

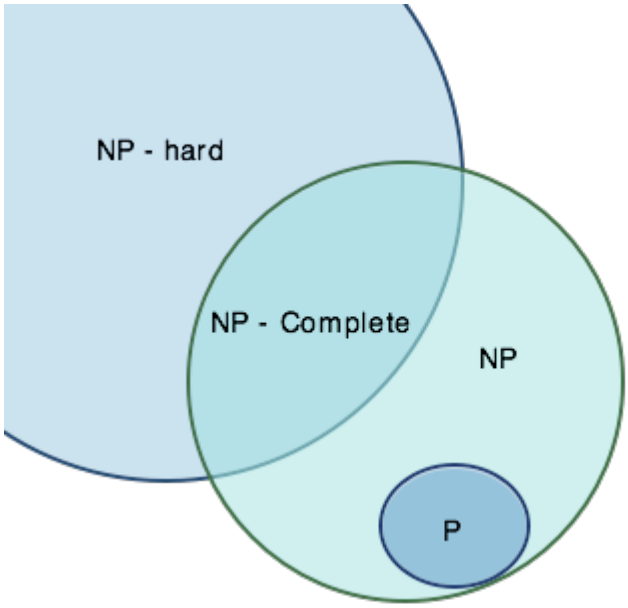
NP-complete class of problems which are NP and any NP problem can be reduced to them in polynomial time.

Figure 1.1 illustrates these problem classes and their relation in a Venn diagram. Some problems are computationally intensive to be solved on a classical computer. Although you might be able to check whether a potential answer is correct or not, finding the solution may not happen in a polynomial time. [17] has a long list of famous Np-complete problems. The most important fact is that all this complexity discussion is coming from our notion of computation, the classical Turing machine. If a problem is hard to solve in a Turing machine model, it does not mean that it is also hard to solve in another computation paradigm. And this is exactly where an alternative notion of computation becomes important, especially to tackle these hard problems.

Quantum computers are proven to have computational resources for an alternative paradigm of computation in which classically challenging problems can be solved faster (in some cases the speed-up is even exponential). A list of many NP-Complete problems can be found in [25] which can be tackled by current realizations of quantum computers [2]. As another example, we can point to Shor's factoring algorithm [36] and Grover's search algorithm [19] as the most famous quantum algorithms in the literature.

Quantum computation as a general notion is the idea of using systems which are governed with quantum physic rules and has special properties like tunneling,

Figure 1.1: Classes of problems based on their complexity.



entanglement, superposition, and to harvest these properties as computation resources. There are several different ways to realize these quantum systems in a small scale. Photons, ion traps [21], cold atoms in optical lattices [26], superconducting charge (or flux) qubits [20, 38], are all examples of potential experimental realization of quantum systems. Although the theory of quantum computation for any of these approaches is rapidly progressing, scientists still have technological limitations to scale these systems up to a size able to handle real world challenging problems. In spite of these engineering challenges, some approaches to quantum computation have shown promising results in terms of scalability.

1.2.2 A Glimpse of Quantum Mechanics

In physics, a system is explained by the Hamiltonian. Hamiltonian is an operator that relates the total energy of the system to the energies and forces applied to the particles in the system either as a result of an external source (exp. a potential due to a magnetic field) or due the existence of the particles themselves. A Quantum system is a physical system in which its Hamiltonian is an operator with quantized

eigenvalues (energies) acting on a very special vector space called a Hilbert space. By diagonalizing the Hamiltonian operator we can get all information about the energy levels of a physical system. Equation 1.1 shows the Hamiltonian operator for a one-particle quantum system where $V(r,t)$ is the potential applied to the particle and ∇^2 is just $\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}$ operator.

$$H = -\frac{\hbar^2}{2m}\nabla^2 + V(r,t) \tag{1.1}$$

The quantized eigenvalues are the possible energies that a quantum system can acquire and eigenvectors are the state of the system corresponding to a specific energy (eigenvalue). Since states are vectors on the Hilbert space we can use dirac bra-ket notation to show a state with a ket $|\psi\rangle$ or its complex conjugate with a bra $\langle\psi|$.

$$H|\psi\rangle = E_\psi|\psi\rangle \tag{1.2}$$

Equation 1.2 is showing that Hamiltonian as an operator, has eigenstate $|\psi\rangle$ with corresponding eigenvalue E_ψ which is the energy of that state and can be measured. The lowest possible energy eigenvalue and eigenstate is of significant importance and it's called the ground state of the system. All other energy states whit larger energies are called excited states.[2] Although solving a Hamiltonian and finding the ground state of a quantum system can be a hard problem itself and needs a more detailed discussion, but it's outside the scope of this introduction so for now we assume we are working with systems in which we are aware of their energy eigenstates.

As an example we can go through details of a quantum system consisting of one particle with two energy levels. These two level systems are called qubits. They are either systems with only two energy levels or systems with many energy levels

[2]This particular naming convention is coming from the fact that naturally a system tends to get to the configuration which has the lowest possible energy. In order to move a system out of its lowest energy state we have to do work and literally excite the system in order to put it in a configuration with higher energy. A very simple example is the potential energy stored in a ball hanging from a certain distance of the ground. When we drop the ball it moves toward the ground to lose potential energy. if we want the ball to move in opposite direction we have to excite it and apply force to it.

but only the two smallest energies are taken in account. Let's call the lowest energy state (often called ground state), state 0 or $|0\rangle$ and the second energy state (often called as first excited state) state 1, $|1\rangle$. Right here an intuitive analogy between bits (binary number either 0 and 1) and qubits (Quantum system in either 0 or 1 state) can be established. The first and hence the most fundamental difference is that a qubit (two level quantum system) can also exist in a superposition of its eigenstates. If $|\psi\rangle$ is denoting the state of a qubit, superposition means the following is a valid state of this system:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.3}$$

where α and β are complex numbers in general. Although this quantum system exists in a superposition of states, when it's measured in σ_z basis, it collapses to one of the eigenstates of that basis ($|0\rangle$ or $|1\rangle$) with a certain probability proportional to the square of the constants in the superposition relation 1.3 . Unlike this situation, an energy eigenvalue can be degenerate. Degeneracy happens when an eigenvalue corresponds to more than one eigenstate of the observable (measurement operator). In general an eigenvalue corresponds to a subspace of the Hilbert space, called the eigenspace of that eigenvalue. The number of eigenstates in the eigenspace defines the degree of the degeneracy. When each eigenvalue of the system corresponds to only one eigenstate, the system is non-degenerate under that observable and the eigenstates of the observable form a basis for the Hilbert space, i.e. each state of the Hilbert space can be represented as a superposition of the eigen states of that specific observable. Later in this section, we will talk more about measurements and the probabilistic nature of observing a certain energy eigenvalue and eigenstate in a quantum system.

All quantum properties of two level quantum systems (qubits) can be understood using the Algebra of Pauli Operators σ_x, σ_y, σ_z and the Identity operator I [16]. The Hilbert space of a qubit is two dimensional and all operators defined on this space have a 2 by 2 matrix representation. The Pauli operators can be represented as follows:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{1.4}$$

In fact the equation 1.3 we saw earlier, is a general expression for a two level quantum system diagonalized in σ_z basis. σ_z is often called the computational basis, and its eigen vectors are written as $|0\rangle$ and $|1\rangle$. Each of the Pauli operators has two eigenvalues +1 and -1 and the following eigen-vectors corresponding to + and - eigenvalues:

$$\psi_{x^+} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}, \psi_{x^-} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}, \tag{1.5}$$

$$\psi_{y^+} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\i\end{pmatrix}, \psi_{y^-} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-i\end{pmatrix}, \tag{1.6}$$

$$\psi_{z^+} = \begin{pmatrix}1\\0\end{pmatrix}, \psi_{z^-} = \begin{pmatrix}0\\1\end{pmatrix}. \tag{1.7}$$

The commutation relations between these operators is really important when studying their application to a quantum state:

$$[\sigma_x, \sigma_y] = 2i\sigma_z, \tag{1.8}$$

$$[\sigma_y, \sigma_z] = 2i\sigma_x, \tag{1.9}$$

$$[\sigma_y, \sigma_x] = -2i\sigma_z \tag{1.10}$$

Each operator commutes with itself, so $[\sigma_x, \sigma_x] = 0$ and two different operators anticommute $\{\sigma_x, \sigma_y\} = 0$. In general:

$$\{\sigma_i, \sigma_j\} = 2\delta_{ij} \quad i, j \in \{x, y, z\} \tag{1.11}$$

If we have a qubit prepared in state $|\psi\rangle$ of equation 1.3, then

$$\sigma_z|\psi\rangle = \sigma_z(\alpha|0\rangle + \beta|1\rangle) \tag{1.12}$$
$$= \alpha(\sigma_z|0\rangle) + \beta(\sigma_z|1\rangle) \tag{1.13}$$
$$= \alpha(+1|0\rangle) + \beta(-1|1\rangle) \tag{1.14}$$
$$= \alpha|0\rangle - \beta|1\rangle \tag{1.15}$$

The third line holds since $|0\rangle$ and $|1\rangle$ are eigenvalues of σ_z Pauli operator with eigenvalues +1 and -1 respectively. In fact, applying the σ_z operator can be a very simple, yet important example of a quantum error occurring on a two level quantum system, changing our qubit state.

So far we have been talking about a closed quantum system. However, we know that in order to use a quantum system, manipulate it, or get information about it we need interact with that system. One way that a closed quantum system can interact with its environment is through measurement.

Quantum measurements are also operators applied to a quantum state. It's a postulate in quantum mechanics that an observable M (measurement operator) is an operator that maps the Hilbert space of a quantum system to a subspace of that Hilbert space. As we discussed earlier, if the system is not degenerate under an observable, the eigen vectors of the observable form a basis for the Hilbert space of the quantum system under measurement. In fact, the result of a measurement (i.e. what we can expect to see in a lab) can be any of the eigenvalues of the observable. Since they are measurement values the eigenvalues have to be real and thus the observable must be Hermitian (self-adjoint) operators. A quantum state living in a Hilbert space can be represented as a superposition of the eigen-vectors of that observable defined on the same Hilbert space. The Born rule named after Max Born states that if an observable M is measured in a quantum system in a normalized state $|\psi\rangle$ then:

- The measurement results is one of the eigenvalues m of the hermitian operator M with a certain probability.
- The probability of observing an specific eigenvalue m is $Pr(m) = \langle\psi|P_m^\dagger P_m|\psi\rangle$ where $P_m = |V_m\rangle\langle V_m|$ is the projection operator for the eigen vector V_m cor-

responding to the eigenvalue m if the eigenspace of m is 1 dimensional (i.e. non-degenerate eigenvalue), and $P_m^\dagger$ is its complex conjugate.

If eigenvalue m is degenerate of degree g_m, then its eigenspace forms a g_m-dimensional subspace of the Hilbert space. All possible degenerate eigen states of this eigenvalue are vectors in this space and can be written as the linear combination of its orthonormal basis $|Em_i\rangle$. In this case we can also define a similar projection operator $P_m = \sum_{i=1}^{g_m} |Em_i\rangle\langle Em_i|$.

- After the measurement the state of the system is projected to $\frac{P_m|\psi\rangle}{\sqrt{\langle\psi|P_m^\dagger P_m|\psi\rangle}}$.

- The V_m eigen vectors satisfy the completeness equation, $\sum_m P_m^\dagger P_m = 1$. The completeness equation shows the fact that the probabilities of observing the eigenvalues add up to 1:

$$1 = \sum_m Pr(m) = \sum_m \langle\psi|P_m^\dagger P_m|\psi\rangle \tag{1.16}$$

To summarize these points, when we measure an observable of a quantum system, the state of the system is projected to a subspace of the system's Hilbert space. In fact, if the eigenvalue is non-degenerate, the system is being projected to an eigen vector of that observable and the real value we measure and see in the lab is the eigenvalue corresponding to that eigen-vector. Observing each eigenvalue has its own probability and all these probabilities add up to 1 [28]. As an example we suppose a quantum system initially in state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. If we measure this system in the σ_z basis, the system will be projected to either state $|0\rangle$ or state $|1\rangle$ which are the eigen-vectors of observable σ_z. Please note that σ_z is indeed a hermitian operator and the state of the system ψ is already written as a superposition of eigen-vectors of σ_z operator. Now according to the Born Rule:

- The measurement outcome can be either +1 or -1 which are the eigen-values of operator σ_z and:

$$V_{+1} = |0\rangle \text{ and } V_{-1} = |1\rangle \tag{1.17}$$

with projection operators:

$$P_{+1} = |0\rangle\langle 0| \text{ and } P_{-1} = |1\rangle\langle 1| \tag{1.18}$$

- The probability of each measurement outcome and finding the system in the corresponding eigen-vector is:

$$\begin{aligned}
Pr(m=+1) &= \langle\psi|P_{+1}^{\dagger}P_{+1}|\psi\rangle && (1.19)\\
&= \langle\psi|P_{+1}|\psi\rangle && (1.20)\\
&= \langle\psi|0\rangle\langle 0|\psi\rangle && (1.21)\\
&= (\alpha^*\langle 0| + \beta^*\langle 1|)|0\rangle\langle 0|(\alpha|0\rangle + \beta|1\rangle) && (1.22)\\
&= (\alpha^*\underbrace{\langle 0|0\rangle}_{=1} + \beta^*\underbrace{\langle 1|0\rangle}_{=0})(\alpha^*\underbrace{\langle 0|0\rangle}_{=1} + \beta^*\underbrace{\langle 0|1\rangle}_{=0}) && (1.23)\\
&= \alpha^*\alpha = |\alpha|^2 && (1.24)
\end{aligned}$$

or (using the same steps):

$$Pr(m=-1) = \langle\psi|1\rangle\langle 1|\psi\rangle = |\beta|^2 \tag{1.25}$$

- The system is projected in to either of the following states:

$$\frac{P_{+1}|\psi\rangle}{\sqrt{\langle\psi|P_{+1}^{\dagger}P_{+1}|\psi\rangle}} \text{ ,or } \frac{P_{-1}|\psi\rangle}{\sqrt{\langle\psi|P_{-1}^{\dagger}P_{-1}|\psi\rangle}} \tag{1.26}$$

in 1.24 and 1.25 we calculated the part under the square root in the denominator. So the state of the system after this measurement is either of the following states:

$$\frac{P_{+1}|\psi\rangle}{\sqrt{Pr(m=+1)}} = \frac{P_{+1}|\psi\rangle}{\sqrt{|\alpha|^2}} \tag{1.27}$$

$$= \frac{|0\rangle\langle 0|\psi\rangle}{|\alpha|} \tag{1.28}$$

$$= \frac{|0\rangle\langle 0|(\alpha|0\rangle + \beta|1\rangle)}{|\alpha|} \tag{1.29}$$

$$= \frac{|0\rangle(\alpha\langle 0|0\rangle + \beta\langle 0|1\rangle)}{|\alpha|} \tag{1.30}$$

$$= \frac{\alpha}{|\alpha|}|0\rangle \tag{1.31}$$

or :

$$\frac{P_{-1}|\psi\rangle}{\sqrt{Pr(m=-1)}} = \frac{P_{-1}|\psi\rangle}{\sqrt{|\beta|^2}} \tag{1.32}$$

$$= \frac{\beta}{|\beta|}|1\rangle \tag{1.33}$$

It can be shown that the phase factors like $\frac{\beta}{|\beta|}$ can be ignored and the states in 1.31 and 1.33 are in fact $|0\rangle$ and $|1\rangle$ states.

These calculations are showing that measuring a quantum system in state $|\psi\rangle$ in the σ_z basis has two possible outcomes, +1 with probability $|\alpha|^2$ and -1 with probability $|\beta|^2$. The state of the system after the measurement will be projected to either $|0\rangle$ if +1 was measured and $|1\rangle$ if -1 was measured.

Measurement is only one example of interaction between a closed quantum system and its environment. However, it has been shown that if a system of many qubits is prepared in a highly entangled state, we can use this system to do universal quantum computation, by only performing measurements on these qubits [5]. This paradigm of quantum computation is called measurement-based quantum computation (MBQC).

This was a very short high level introduction to a few postulates of quantum mechanics. For the remainder of this thesis we suppose that the reader is comfortable with postulates of quantum mechanics, especially properties and definitions

of two level quantum systems (qubits).

1.2.3 Quantum Computation

First stated by Feynman in 1982 [15], the peculiar properties of quantum physics can be exploited as a universal resource for computation. As we discussed briefly earlier in this chapter, there are many different ways to exploit this potential power. This diversity brings different paradigms of computation with it, some are close to the classical notion of computation and the idea of universal Turing machine [39] and some of them work in a totally different manner. For example the gate-model quantum computation is a quantum mechanical generalization of the classical computers we are working with [3].

The idea behind the gate-model computation is to have a universal set of operations (gates). A set of gates operating on boolean data [4] is called functionally complete (universal) if any possible logical truth table (logical operation) can be expressed as a combination of these gates in a boolean expression. For example, $\{AND, NOT\}$ boolean operations form a universal set for classical computation, meaning any other operation like *OR*, XOR, *NAND*, etc. can be built combining only *AND* and NOT gates. Thus the notion of universality is essential for a computer in order to run any possible operation on the provided data. The same logic applies to the gate model approach to quantum computation. A universal set of quantum gates is needed in order to have a functionally complete quantum computer.

As we mentioned earlier and similar to classical computers, the qubit is the

[3] As an example of a completely different approach to quantum computation we can name the Adiabatic Quantum Computation or AQC in short [14]. This approach works on the basis that a quantum system will stay in its lowest energy state if its Hamiltonian operator evolves very slowly. It's proved in [13] that any operation can be decomposed to the problem of finding the ground state of a specific Hamiltonian starting from another Hamiltonian and evolving it slow enough in to be in an adiabatic fashion. It's easy to see that this approach is different from the gate model approach. Instead of encoding an operation in a combination of a limited number of gate types, AQC encodes a problem into the desired Hamiltonian of a quantum system. The system starts from a known and prepared Hamiltonian and evolves slowly in time. The adiabatic theorem proves that if the system was originally in its ground state then if the evolution was slow enough with a high probability it will remain in it's ground state all the way through the process and what we can measure at the end of the process is nothing but the ground state of the desired Hamiltonian.

[4] or bits in the notation of classical computers.

Table 1.1: Different single and two qubit quantum gates (unitary operators) along with their matrix representation and their application on computational basis states $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$.

Gate	Matrix rep.	Example of Application
Pauli X gate	$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$X\|\psi\rangle = \alpha\|1\rangle + \beta\|0\rangle$
Pauli Z gate	$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$X\|\psi\rangle = \alpha\|0\rangle - \beta\|1\rangle$
Phase shift gate	$R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$	$R_\theta\|\psi\rangle = \alpha\|0\rangle + \beta e^{i\theta}\|1\rangle$
Controlled-NOT gate	$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	If the first qubit is in $\|1\rangle$ it applies a pauli X to the second qubit, otherwise leaves it as it is. For example: $\|01\rangle \rightarrow \|01\rangle$, but $\|11\rangle \rightarrow \|10\rangle$.

unit of quantum data, which stores the information about the state of the two-level quantum system (the qubit). Similar to the classical gate model approach, we manipulate these qubits by successive application of different gates from our universal set in order to conduct a certain complex operation. These operations are called quantum algorithms. A quantum operation in general can be any unitary operator acting on the Hilbert state of the quantum computer. Unlike classical boolean operations, being unitary makes quantum operations reversible. Table 1.1 includes a few examples of quantum gates along with their matrix representations.

As an example of a quantum algorithm running on a universal gate-model quantum computer we can point to Shor's famous factoring algorithm [36]. Having an ideal quantum computer with its universal set of gates working on sufficiently many qubits without any noise, Shor's factoring algorithm makes it possible to factor a large integer N in polynomial time $\mathcal{O}(log^3(N))$. One should note that this is super-polynomially faster than the best classical factoring algorithm *general number field sieve* [30] [5]. This quantum algorithm is efficient enough to break RSA

[5]The complexity of this algorithm is of order $\mathcal{O}\{exp(c(log^{\frac{1}{3}}(n))(loglog(n))^{\frac{2}{3}})\}$.

cryptography systems. The security of RSA systems is defined based on infeasibility of finding prime factors of large integer numbers using classical computers. The record for the largest number factored using Shor's algorithm on a quantum device was set in 2012 for factoring 21 [27] [6].

1.2.4 Fault Tolerant Quantum Computation and Threshold Theorem

Although experimental realization of ideal quantum computers are possible in prototype sizes, large scale quantum computation cannot happen free of noise. This is exactly why fault tolerance is an important factor when it comes to scalable quantum computers. The chance of destructive correlation with environment and quantum decoherence[7] increases with the size of the quantum system. So it's almost inevitable to implement a large scale quantum computer with ideal gates, measurements and interactions in general. Now the question is whether quantum computers can still work with faulty operations?

Due to the threshold theorem, large scale quantum computation is possible even with noisy gates and systems provided that the noise is below a certain threshold [1].

Threshold theorem for quantum computation: A quantum circuit containing $p(n)$ gates may be simulated with probability of error at most ε using

$$O(poly(\frac{logp(n)}{\varepsilon})p(n)) \tag{1.34}$$

gates on hardware whose components fail with probability at most p, provided p is below some constant threshold, $p < p_{th}$, and given reasonable assumptions about the noise in the underlying hardware.

Fault tolerant quantum computation is only possible if there is a quantum error correction step in the computation process which repeatedly brings the faulty system to the valid code space and does not allow the error to propagate throughout

[6] The largest number factored so far with a quantum computer is 143. But this was done using the adiabatic quantum computation paradigm, not the Shor's algorithm on a gate-model quantum computer.

[7] for a detailed formulation of decoherence refer to section 1.3.3.

the computation process. If the error is below threshold and is controlled in this manner then the faulty quantum system can simulate an ideal quantum system even when the number of qubits scale up. This is the significance of quantum error correction and the reason we are interested to study error correcting codes. Different fault tolerant quantum computation schemes together with various error correcting codes can demonstrate different fault tolerance thresholds. High threshold fault tolerant quantum computation is possible using the proper error correcting codes [33].

1.3 Error Correction

Quantum Error correction plays an important role in quantum information processing and quantum computation. Exactly like its classical analogue, quantum error correction is the concept of restoring the state of a quantum system after it's subjected to noise. A quantum system can easily be disturbed by its environment if not protected. This will cause quantum decoherence which can be modeled as an error channel affecting quantum bits (qubits) in the system. Quantum error correction can be understood as a generalization of error correction for classical communication through noisy channels.

1.3.1 Classical Error Correction

The best way to approach quantum error correction is to start with the basics of classical error correction. In *information theory*, a *word* or *code-word* is a representation of a particular message which is supposed be processed, communicated, stored or etc. The unit of information is often taken to be a bit and can be represented by a vector in Galois field $\mathbb{F}_2^1$ which is basically the field consisting of vectors [0] and [1]. A unit of information is defined to be the ability of distinguishing two opposing binary situations, 0 or 1, Yes or No, True or False. A word or message then can be represented as a series of zeros and ones carrying bits of information of that particular message. So from now on, a word is going to be a vector of length n in $\mathbb{F}_2^n$, as an example $m_1 = 1101$ is a message with 4 bits of information.

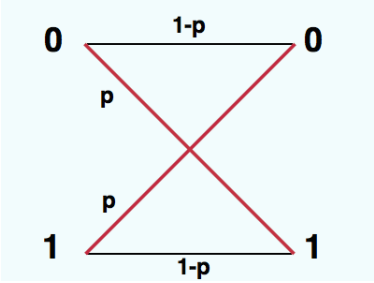
Since in a real world realization of computers and communications systems,

we always have noisy channels and faulty gates, any processing or communication or etc happening to message m_1 can change its content with a certain probabilistic model.

$$m_1 = 1101 \xrightarrow{Erroneous\ Procedure} \mathscr{E}(m_1) = 1\underline{0}01 \tag{1.35}$$

As you can see in 1.35, the second bit in message m_1 is flipped. There are various error models and this was only a simple example, for now we can suppose that the erroneous procedure is a noisy communication channel over which we want to transmit message m_1. A noisy communication can be modeled as an error channel. An error channel affects each individual bit of information independently. Bits can change, remain the same, or be erased with different probabilities. Figure 1.2 depicts a simple symmetric bit-flip channel.

Figure 1.2: Symmetric bit-flip error channel. A bit is independently changed with probability p, or it keeps its original value with probability 1-p.



Now we use a simple error correction to protect our message against this or any other one bit flip error like that through the noisy channel. The very summarized version of the classical error correction idea is to add redundancy to the representation of a given word. As the simplest example, we use the 3-bit majority code. The encoding E_n is relatively simple. Make 3 copies of each logical information bit (adding redundancy), so instead of representing each logical bit of information with 1 physical bit, we represent each logical bit of information with 3 physical bits:

Table 1.2: Encoding table for the 3-bit majority code.

Logical information	Physical (redundant) representation
0	000
1	111

$$E_n(m_1 = 1101) = 111\,111\,000\,111 = M_1 \tag{1.36}$$

$$M_1 = 111\,111\,000\,111 \xrightarrow{\mathscr{E}(.)} \mathscr{E}(M_1) = 11\underline{0}\,111\,0\underline{1}0\,\underline{0}11 = \mathscr{E}(M_1) \tag{1.37}$$

$$D_e(\mathscr{E}(M_1)) \xrightarrow{\text{Error Correction}} \underbrace{110}_{1}\,\underbrace{111}_{1}\,\underbrace{010}_{0}\,\underbrace{011}_{1} = 1101 = m_1 \tag{1.38}$$

The decoding is as simple as a majority vote. Even if an error happens to one of the physical bits, the remaining two are still representing what the initial logical information was. So when we receive each 3 physical bits, count 0's and 1's and who ever has the majority has a higher probability to be the initial bit of information. We should note error correction relies on the probabilistic nature of error and hence the result of an error correction has a probability to be correct. Our goal is to maximize this success probability. The following example can be really helpful to understand the probabilistic behavior of error correction:

$$m_0 = 1 \rightarrow M_0 = E_n(m_0) = 000 \tag{1.39}$$

$$m_1 = 1 \rightarrow M_1 = E_n(m_1) = 111 \tag{1.40}$$

$$\mathscr{E}_1 \rightarrow \text{Two bit flip on first and second bits} \tag{1.41}$$

$$\mathscr{E}_2 \rightarrow \text{One bit flip error on third bit.} \tag{1.42}$$

$$\left.\begin{array}{l} \mathscr{E}_1(M_0 = 000) = \underline{11}0 \\ \mathscr{E}_2(M_1 = 111) = 11\underline{0} \end{array}\right\} \quad \text{Same Syndrome.} \tag{1.43}$$

These two messages $\mathscr{E}_1(M_0)$ and $\mathscr{E}_2(M_1)$ are decoded to 1 because of the majority vote decoding, which is only correct for the second case. If we assume that each bit-flip error is an independent random event with probability p_e, then the

probability of error $\mathscr{E}_1$ is p_e^2 whereas it's only p_e for $\mathscr{E}_2$. So the error correction tries to guess the most probable error and correct for it, this minimizes the error correction failure probability. Thus this code is only able to correct up to one bit flip error and it fails in the event of more than one bit error. Consequently, the majority vote error correction will succeed with probability $1-3p_e^2+2p_e^3$:

$$\begin{aligned} Pr\{\text{Success}\} &= 1 - Pr\{\text{Failure}\} && (1.44)\\ &= 1 - (Pr\{\text{Only 2 erros}\} + Pr\{\text{Only 3 errors}\}) && (1.45)\\ &= 1 - (\binom{3}{2} p_e^2(1-p_e) + \binom{3}{3} p_e^3) && (1.46)\\ &= 1 - 3p_e^2 + 2p_e^3 && (1.47) \end{aligned}$$

More sophisticated codes with more complex encoding and decoding schemes have been developed for various error models, however, all classical error correcting codes share the fundamental ideas of the 3-bit majority-vote code.

1.3.2 Linear Codes

Linear codes are the most famous and commonly used class of classical error correcting codes because the mathematical formalism makes it easier to both construct and analyze these codes. Formal definition of a linear error correcting code is as follows:

Definition 1. *A (n,k) linear code is a k-dimensional linear subspace $\mathscr{C}$ of n-dimensional vector space $\mathbb{F}_q^n$, where $\mathbb{F}_q$ is the finite filed with q elements. For special case $q=2$ this structure will be a **binary** linear code.*

A vector in $\mathscr{C}$ is called a code-word. It's easy to show that there are in general q^k code words in a linear code. Looking back at our notation in the introductory 3-bit repetition code example, a (n,k) binary linear code is able to encode 2^k different messages by adding $n-k$ redundant bits to the representation of each message (code-word).

Since code $\mathscr{C}$ is a linear subspace of $\mathbb{F}_2^n$ of rank k, it has a basis of rank k, which means the whole code space $\mathscr{C}$ can be represented as span of k code-words. The

code generator matrix $G_{k\times n}$ is the row concatenation of these basis code-words. Encoding a k bit message is now as easy as matrix multiplication on the specified finite field. [8]

$$M_1 = E_n(m_1) = m_1.G \tag{1.48}$$

$$H.M_1^T = H.(m_1.G)^T = 0 \tag{1.49}$$

For each linear code there exists an $H_{(n-k)\times n}$ matrix whose kernel is $\mathscr{C}$ and it's used for error correction and decoding. This matrix is called the parity check matrix. Since the code basis is the kernel of H, any code-word in $\mathscr{C}$ will be zero. On the other hand, if H is multiplied by an erroneous message (a vector outside the code space) the result will be non-zero which will be used as a syndrome to guess which family of errors might have happened and to correct them. The following are generator and parity check matrices for our previous 3-bit repetition code also known as the (3,1) Hamming code:

$$G = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, H = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{1.50}$$

This code has two code-words $c_0 = [0]$ and $c_1 = [1]$. It's easy to verify the properties we mentioned on this code:

$$c_0.G = [0]\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = [000] = C_0 \tag{1.51}$$

$$c_1.G = [1]\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = [111] = C_1 \tag{1.52}$$

$$H.C_1^T = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0 \tag{1.53}$$

[8] The reader should note that since the codes are in general vector spaces defined on finite field $\mathbb{F}_q^n$, all encoding, decoding, syndrome detection and etc. algebraic operations are carried out on the same finite field. So in the very special case of binary codes defined on $\mathbb{F}_2^n$ all arithmetic operations are modulo 2.

we can also apply the parity check matrix to erroneous messages in order to observe the syndromes:

$$M_1 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \tag{1.54}$$

$$H.M_1^T = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} . \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = 1 \neq 0 \tag{1.55}$$

We now detected a possible one bit-flip error on message M_1 and try to correct it by mapping it to the closest valid code-word in the code-space. This kind of decoding is called the nearest-neighbor decoding which in this case is as simple as the majority vote among the 3 bits.

1.3.3 Quantum Error Model and Decoherence

Before we move on to quantum error correcting codes, first we have to devise a model for quantum errors. Any interaction between a quantum system and its environment can cause the state of the quantum system to change. The interaction can change a quantum system to a classical state (Like a projective measurement in computational basis [9]), or it can just end up flipping a quabit or a relative phase in the superposition. In order to model the quantum errors which are coming from the interaction between the system and its environment, we look at these errors as operators acting on the state of the system. So far we have been looking at quantum mechanical systems from a vector space point of view. However, there are other ways to represent a quantum mechanical system. The *density operator* or often called the *density matrix* is another way of representing a quantum system which is mathematically equivalent to the state vector in Hilbert space approach, and it's mostly used when studying systems interacting with their environment.

The density operator provides a convenient way for representing quantum systems whose states are not fully known. As an example, let's assume that the only information we have about a quantum system is that it is in one of the states $|\psi_i\rangle$

[9] which ruins the superposition property of the quantum state, we discussed this earlier in 1.2.2

with a corresponding probability p_i. This quantum system is a statistical mixture of subsystems in different states $|\psi_i\rangle$. Each subsystem is in its own state but when we study the union of these subsystems as a whole, the only thing we can say about it, is the statistical distribution of the states among these different subsystems. So when this system is observed the statistical nature of the mixture makes the outcome probabilistic. This is why we assign a probability p_i to each possible state $|\psi_i\rangle$ in a mixture. The $\{p_i, |\psi_i\rangle\}$ is called an *ensemble of pure states*. *Pure state*, in contrast to a *mixed state*, is a quantum state that can not be represented as a convex combination (mixture) of other states. The density operator for the mentioned system can be written as:

$$\rho \equiv \sum_i p_i |\psi_i\rangle\langle\psi_i| \tag{1.56}$$

Now let's study the evolution of this ensemble due to a unitary operator U.[10] If the system is in state ψ_j with probability p_j then after the unitary evolution it is going to be in state $U\psi_j$ with probability p_j. According to this and equation 1.56 we have:

$$\rho \xrightarrow{U} \sum_i p_i U|\psi_i\rangle\langle\psi_i|U^\dagger = U\rho U^\dagger \tag{1.57}$$

The notion of measurement can also be translated to the density operator language. If we perform a measurement with projection operators P_m and the ensemble is in state $|\psi_i\rangle$ then the probability of observing m is:

$$Pr(m|\text{system in i'th state}) = \langle\psi_i|P_m^\dagger P_m|\psi_i\rangle = tr(P_m^\dagger P_m|\psi_i\rangle\langle\psi_i|) \tag{1.58}$$

Now that we calculated the measurement probability for one possible state $|\psi_i\rangle$, we can generalize it to the whole ensemble:

[10] Operator U is unitary if and only if $U^\dagger = U^{-1}$.

$$Pr(m) = \sum_i p_i Pr(m|\text{system in i'th state}) \tag{1.59}$$

$$= \sum_i p_i tr(P_m^\dagger P_m |\psi_i\rangle\langle\psi_i|) \tag{1.60}$$

$$= tr(P_m^\dagger P_m \rho) \tag{1.61}$$

Using what we had in section 1.2.2 on Born's rule and the same logic we folllowed here we can show that the density operator for the ensemble after this measurement is:

$$\rho_m = \sum_i p_i \frac{P_m|\psi_i\rangle\langle\psi_i|P_m^\dagger}{tr(P_m^\dagger P_m \rho)} \tag{1.62}$$

$$= \frac{P_m \rho P_m^\dagger}{tr(P_m^\dagger P_m \rho)} \tag{1.63}$$

All postulates of quantum mechanics can be reformulated in density operator language [28]. The density operator also provides an easy way of looking at subsystems in a composite quantum system. This is exactly what we are looking for because quantum errors are result of the quantum system having correlation with it's environment and make a composite system with it. If we have two physical systems A and B interacting with each other, then the density operator of the whole composite system of the two ρ^{AB} can be reduced to the density operator of each system just by partial trace over the other system, for example:

$$\rho^B = tr_A(\rho^{AB}) \tag{1.64}$$

Now suppose we have a system of interest (for example our desired qubits) which we call the quantum computer or QC in short and we represent it with density operator ρ^{QC}. This system is interacting with its environment with density operator ρ^{env} resulting in a total density operator like $\rho = \rho^{QC} \otimes \rho^{env}$[11] . From our

[11]This is an important assumption that the composite system of the quantum computer and its

reduced density matrix method we saw earlier, the final result $\mathscr{E}(\rho)$ of a unitary evolution of the system interacting with the environment can be written as follows:

$$\mathscr{E}(\rho) = tr_{env}[U(\rho^{QC} \otimes \rho^{env})U^{\dagger}] \tag{1.65}$$

Using the orthonormal basis $|e_i\rangle$ for the state space of the environment and supposing that environment has the density operator $\rho^{env} = |e_0\rangle\langle e_0|$ initially, we can turn the above equation in to a new representation called the *operator-sum representation*:

$$\mathscr{E}(\rho) = \sum_i \langle e_i|U[\rho \otimes |e_0\rangle\langle e_0|]U^{\dagger}|e_i\rangle \tag{1.66}$$

$$= \sum_i E_i \rho E_i^{\dagger} \tag{1.67}$$

where $E_i = \langle e_i|U|e_0\rangle$ is called the *Kraus* operator acting on the quantum computer state space. When the sum for $\mathscr{E}(\rho)$ contains more than one element it shows that quantum computer system is subject to quantum decoherence because of its correlation with its environment. Coherence in a quantum state is the ability to make interference which is a direct result of the superposition property. The environment noise can cause a quantum system to lose coherence and become a classical state losing all its useful quantum properties for quantum computation. Decoherence can be modeled as lose of information from the quantum computer to its environment. Now that we developed the mathematical machinery, we can start a counter attack to preserve the coherence of a quantum state through the quantum computation process. This is exactly what we call the quantum error correction.

A comparison between classical error models and quantum error model can now be helpful. Like classical communication channels, the effect of environmental noise on a quantum system can be modeled as an error channel with the initial

environment are in a product state as stated above. The correlation between system and environment can be much more general and complex but it's reasonable to assume to have the product state since it shows up in many practical cases. The same logic and line of thought can be applied to more general correlations but it's out of the scope of this thesis. We encourage the reader to look at chapter 8 of [28] for a detailed discussion of these situations.

state (which carries information) as the input and the noisy transmitted version of that as the output. In classical communication, while the information is transmitted through the channel, each possible error operation can act on it with a certain probability. Errors act on individual bits and the probability distribution of error on different bits are independent yet identical. So each bit is going through the same error channel but it ends up with a probabilistic outcome. Using this property we can model a communication channel as a one-bit error channel. Depending on the channel under study, the bit might remain correct with a probability and change due to different errors (like a bit flip) with another probability. These probabilities should add up to 1 since eventually one of these events should occur to each bit of information. Figure 1.2 shows an example of a classical error channel called binary symmetric channel. It's called symmetric because the probability of a 0 changing to 1 or a 1 changing to 0 is the same. We can use our operator-sum representation to build a quantum bit flip error channel. According to equation 1.67 if:

$$E_0 = \sqrt{p}I = \sqrt{p}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad E_1 = \sqrt{1-p}X = \sqrt{1-p}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.68}$$

then

$$\rho \to \mathscr{E}(\rho) = p\rho + (1-p)X\rho X \tag{1.69}$$

will be the operator sum representation of a quantum bit flip channel. A qubit is preserved in its state with probability p or its $|0\rangle$ and $|1\rangle$ states are flipped with probability $1-p$. Unlike classical error models that only includes bit flips, as seen in equation 1.67 quantum noise operations can be more general than just a bit flip. However, it's shown that an arbitrary independent error on a single qubit can be written as a linear combination of pauli operators σ_x (Bit flip), σ_z (Phase flip), σ_y (both, $\sigma_y = \sigma_x\sigma_z$) and the identity operator I [28]. This reduction makes error analysis and study of quantum error correction codes easier [8, 28]. Using example 1.15, if $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is a general quantum state, the following is an example of different quantum errors acting on it:

$$\sigma_z|\psi\rangle = \alpha|0\rangle - \beta|1\rangle \qquad \text{(Phase Flip)} \tag{1.70}$$

$$\sigma_x|\psi\rangle = \alpha|1\rangle + \beta|0\rangle \qquad \text{(Bit Flip)} \tag{1.71}$$

$$\sigma_y|\psi\rangle = i\sigma_x\sigma_z|\psi\rangle = \alpha|1\rangle - \beta|0\rangle \qquad \text{(Both errors)} \tag{1.72}$$

As a general purpose example, we use the density operator formalism to model *depolarizing quantum error channel*. If each of the bit flip, phase flip, σ_y has the occurrence probabilities $p/3$ and with probability $1-p$ the state remains with no error (identity operator). If a quantum system is initially in a mixed state with density matrix ρ then the following shows its evolution due to this example of a quantum error channel:

$$\rho \to \mathscr{E}(\rho) = \underbrace{(1-p)\rho}_{\text{No error}} + \frac{p}{3}\underbrace{\sigma_x\rho\sigma_x}_{\text{Bit Flip}} + \frac{p}{3}\underbrace{\sigma_z\rho\sigma_z}_{\text{Phase Flip}} + \frac{p}{3}\underbrace{\sigma_y\rho\sigma_y}_{\text{Both Errors}} \tag{1.73}$$

To conclude this part of the introduction on quantum error model, we want to discuss the relation between quantum noise and time evolution of a quantum system and Schrodinger equation. In order to do so, we use a pedagogical example of a quantum system which its time evolution will give us the error model for a depolarizing channel.

Suppose we have a 1 qubit quantum computer system ρ^{QC} which is interacting with its environment ρ^{env} forming the total density operator ρ. Using the following identity:

$$\frac{I}{2} = \frac{\rho + X\rho X + Y\rho Y + Z\rho Z}{4} \tag{1.74}$$

we can think of the depolarizing channel acting on the quantum computer single qubit density matrix ρ^{QC} in a way that it is either untouched with probability $1-p$ or it's replaced with a completely mixed state $I/2$ with probability p:

$$\mathscr{E}(\rho^{QC}) = \frac{pI}{2} + (1-p)\rho^{QC} \tag{1.75}$$

in order to realize this as the time evolution of the quantum computer and its environment, we take the environment to be a 2 qubit system and the following Hamiltonian between the total 3 qubit, computer and environment systems. The first and second qubits are the environment and the third qubit is the quantum computer system:

$$H = J(t)|1\rangle\langle 1|(\overrightarrow{S}_2.\overrightarrow{S}_3) \tag{1.76}$$

where $J(t)(\overrightarrow{S}_2.\overrightarrow{S}_3) = \frac{J(t)}{4}(X_1X_2 + Y_2Y_3 + Z_2Z_3)$ is the Heisenberg Hamiltonian between the second qubit in the environment and the third qubit which is the quantum computer system. It's shown that the time evolution of Heisenberg Hamiltonian for a time of π will give us the unitary evolution of a swap $e^{\frac{-i\pi\overrightarrow{S}_2.\overrightarrow{S}_3}{4}}$ between the two qubits 2 and 3. Thus the time evolution of the whole system described in Hamiltonian 1.76 will give us a controlled-swap operation between 2nd and 3rd qubits, controlled by the state of the first qubit.

Now let's suppose the qubit of the quantum computer system, qubit number 3, is initially in an arbitrary mixture ρ^{QC} and we initialize the two qubits of the environment, qubit 1 and qubit 2, in the following mixture $(1-p)|0\rangle\langle 0| + p|1\rangle\langle 1|$ and $I/2$ respectively. Now if this system evolves according to the Hamiltonian explained above, the first qubit in the environment will be in state $|0\rangle$ with probability $(1-p)$ which leaves the ρ^{QC} untouched, or it is in state $|1\rangle$ with probability p which swaps the state of the quantum computer system with mixture $I/2$. Using the identity we saw earlier for $I/2$ we see that this is exactly like a depolarizing channel acting on our 1 qubit quantum computer system:

$$\mathscr{E}(\rho^{QC}) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z) \tag{1.77}$$

This shows how the time evolution of a system and its environment can cause an

error model similar to depolarizing channel on the system in reality as we mathematically modeled earlier. Similar examples can be used to show how a physical quantum system evolving in time is exposed to errors which are modeled using the mathematical machinery of error channels we discussed in this part. Now that we have a good understanding of these error models, we can focus on preserving our system from these errors using the error correction schemes.

1.3.4 Quantum Error Correction

It's intuitive to consider the same ideas we used for classical error correction to protect quantum states against quantum noise, but as we discussed in 1.2.2, quantum mechanics brings odd properties which makes information processing different. For example, if we want to use the 3-bit repetition code idea to build a quantum code we have to note the following issues:

No Cloning We can not copy Quantum states, i.e. starting from general state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ we can not make $|\psi\rangle^{\otimes 3} = |\psi\rangle|\psi\rangle|\psi\rangle$.

Measurement destroys quantum information As we discussed in 1.2.2, measuring the quantum state $|\psi\rangle$ to get information about the error, will kill the quantum state itself.

Different errors unlike classical information processing where all errors can be expressed as bit-flips, A continuum of quantum errors can happen to a quantum state.

We studied the last item earlier in the quantum error model section. Now that we pointed out the differences between classical and quantum error correction and defined quantum error channel we are ready to build our first simple quantum error correcting code, the 3-qubit bit flip code, which is a generalization of the 3-bit repetition code considering the differences mentioned here.

1.3.5 3-qubit Bit-flip Code

This code only concentrates on bit-flip (σ_x) errors, so the error channel is:

$$\rho \to \mathscr{E}(\rho) = p_0\rho + (1-p_0)\underbrace{\sigma_x\rho\sigma_x}_{\text{Bit Flip}} \tag{1.78}$$

We use the same idea of the 3-bit repetition code we studied earlier. The logical $|0\rangle$ and $|1\rangle$ states will be $|000\rangle$ and $|111\rangle$ respectively. For a general state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, we have:

$$|0\rangle \to |0\rangle_L = |000\rangle \tag{1.79}$$
$$|1\rangle \to |1\rangle_L = |111\rangle \tag{1.80}$$
$$|\psi\rangle \to |\psi\rangle_L = \alpha|0\rangle_L + \beta|1\rangle_L = \alpha|000\rangle + \beta|111\rangle \tag{1.81}$$

Now according to our bit flip channel definition in 1.78 a bit-flip error can happen on any of the three qubits involved in the code, so one of the following four possible scenarios might happen:

$$|\psi\rangle_L \xrightarrow{I} \alpha|000\rangle + \beta|111\rangle \tag{1.82}$$
$$|\psi\rangle_L \xrightarrow{\sigma_{x_1}} \alpha|100\rangle + \beta|011\rangle \tag{1.83}$$
$$|\psi\rangle_L \xrightarrow{\sigma_{x_2}} \alpha|010\rangle + \beta|101\rangle \tag{1.84}$$
$$|\psi\rangle_L \xrightarrow{\sigma_{x_3}} \alpha|001\rangle + \beta|110\rangle \tag{1.85}$$
$$\tag{1.86}$$

In this context, the underscore indexes of each Pauli operator points to the qubit that it's acting on. So far we have managed to decompose all different quantum errors to a limited set of Pauli operators, and we also managed to make a 3-qubit repetition encoding. But what happens when we want to decode and error correct? we know that we can not measure the quantum state. If we measure the state it will kill the quantum superposition and won't even give us much information about the original state since it only collapses to one of the eigenstates. So instead of measuring the state we do what we did for linear classical codes. We use the idea of parity check matrix H and apply another operator which acts trivially on the

code space.

Since the $|\psi\rangle_L$ state is written in the computation basis, we try the σ_z operator. Knowing the fact that:

$$\sigma_z|0\rangle = +1|0\rangle \tag{1.87}$$

$$\sigma_z|1\rangle = -1|1\rangle \tag{1.88}$$

We put an even number of σ_{z_i} operators acting on different qubits and apply them to the general state, for example:

$$\begin{aligned}
\sigma_{z_1}\sigma_{z_2}|\psi\rangle_L &= \sigma_{z_1}\sigma_{z_2}(\alpha|000\rangle + \beta|111\rangle) && (1.89)\\
&= \sigma_{z_1}\sigma_{z_2}\alpha|000\rangle + \sigma_{z_1}\sigma_{z_2}\beta|111\rangle && (1.90)\\
&= \alpha|000\rangle + (-1)^2\beta|111\rangle && (1.91)\\
&= \alpha|000\rangle + \beta|111\rangle && (1.92)\\
&= |\psi\rangle_L && (1.93)
\end{aligned}$$

As far as the 3-qubit code is concerned we have found an operator which acts almost like the parity check matrix in the classical case. So now let's see the application of this operator on an erroneous state $\mathscr{E}_1|\psi\rangle = \sigma_{x_1}|\psi\rangle = \alpha|100\rangle + \beta|011\rangle$:

$$\begin{aligned}
(\sigma_{z_1}\sigma_{z_2})\mathscr{E}_1|\psi\rangle &= \sigma_{z_1}\sigma_{z_2}\sigma_{x_1}|\psi\rangle && (1.94)\\
&= \sigma_{z_1}\sigma_{x_1}\sigma_{z_2}|\psi\rangle && (1.95)\\
&= (-1)\sigma_{x_1}\sigma_{z_1}\sigma_{z_2}|\psi\rangle && (1.96)\\
&= -\sigma_{x_1}|\psi\rangle && (1.97)\\
&= -\mathscr{E}_1|\psi\rangle && (1.98)
\end{aligned}$$

The second line holds since two operators on two different qubits are two operators on two disjoint Hilbert spaces and thus they commute. The third line holds since different Pauli operators (σ_x and σ_z) acting on one qubit anticommute. This relation suggests that the erroneous state $\mathscr{E}_1|\psi\rangle_L$ is an eigenstate of the $(\sigma_{z_1}\sigma_{z_2})$ operator with eigenvalue -1. Before we also saw that this operator acts trivially on the code space i.e., any logical state $|\psi\rangle_L$ is a +1 eigenstate of this operator. Not all possible

one bit flip error anticommute with this operator, though. In order to cover all possible bit-flip errors we need to have a set of such operators. The set

$$\{\sigma_{z_1}\sigma_{z_2}I_3,\ I_1\sigma_{z_2}\sigma_{z_3}\} \tag{1.99}$$

is the set of operators we are looking for. It's easy to see that any one bit flip error will anticommute with at least one of the operators in this set. The operators in the set commute with each other and act trivially on the code space. The decoding is as simple as applying these two operators to a possibly faulty state and measuring the results. These two operators are basically acting as parity check row vectors in the parity check matrix (look at example 1.50).

Now we have made our first simple quantum code to correct for one bit flip error. The problem with this simple code is that it only corrects one bit-flip error. However, as we discussed earlier, quantum errors are much more general than this. We reduced the individual qubit errors to bit-flip, phase-flip and both. The nice thing about these set of errors is that although they are fundamentally different but they can be corrected with the same logic. This actually motivates how we want to protect our state against phase flips using our 3 bit repetition idea again.

1.3.6 3-qubit Phase-flip Code

The operator for a single phase flip is σ_z Pauli operator so the error channel model looks like the following:

$$\rho \to \mathscr{E}(\rho) = p_0\rho + (1-p_0)\underbrace{\sigma_z\rho\sigma_z}_{\text{Phase Flip}} \tag{1.100}$$

Now we want to use the 3-qubit repetition idea to encode a general qubit state $|\psi\rangle = \alpha|0\rangle + |1\rangle$ in 3 physical qubits so that it's protected against a single phase flip operation. The analogy between the classical 3 bit repetition and quantum 3 qubit repetition code was simple because of the shared idea of a bit flip. but we know that a σ_z acting on computational bases changes only the phase which in fact what motivates the mind. So in order to make the analogy it's good to find a new basis in which the σ_z operator can flip states (like what σ_x does to the computational basis). To acheive this let's look at the X basis and the eigen basis of σ_x operator. From

equation 1.5 we can write this basis as follows. Without loss of generality we call these states $|+\rangle$ and $|-\rangle$:

$$|+\rangle = \psi_{x^+} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{1.101}$$

$$|-\rangle = \psi_{x^-} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{1.102}$$

Now let's see what the phase flip operator does to these eigen basis:

$$\begin{aligned} \sigma_z|+\rangle &= \frac{1}{\sqrt{2}}(\sigma_z|0\rangle + \sigma_z|1\rangle) \\ &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle \end{aligned} \tag{1.103}$$

$$\sigma_z|-\rangle = |+\rangle \tag{1.104}$$

The second line holds because $|0\rangle$ and $|1\rangle$ states are +1 and -1 eigenstates of the σ_z operator respectively. Now we see that if we prepare our qubits in the eigenbasis of the σ_x operator. the phase flip operator will act like a bit flip operator changing $|+\rangle$ to $|-\rangle$ and $|-\rangle$ to $|+\rangle$. The rest of the encoding decoding procedure is now exactly identical to that of the 3-qubit bit flip code. We start with the logical information $|\psi\rangle_L = \alpha|+\rangle + betta|-\rangle$ and encode it in 3 copies of the logical eigenbasis:

$$|+\rangle \rightarrow |+\rangle_L = |+++\rangle \tag{1.105}$$

$$|-\rangle \rightarrow |-\rangle_L = |---\rangle \tag{1.106}$$

$$|\psi\rangle \rightarrow |\psi\rangle_L = \alpha|+\rangle_L + \beta|-\rangle_L = \alpha|+++\rangle + \beta|---\rangle \tag{1.107}$$

Since the rest of the procedure is identical we only show one example of how the phase flip error act on this code:

$$|\psi\rangle_L \xrightarrow{I} \alpha|+++\rangle + \beta|---\rangle \tag{1.108}$$

$$|\psi\rangle_L \xrightarrow{\sigma_{x_2}} \alpha|+-+\rangle + \beta|-+-\rangle \tag{1.109}$$

A majority vote decoding can then be used to decode the faulty state back into one of the two logical states $|-\rangle_L$ or $|+\rangle_L$. Exactly similar to the 3-qubit bit flip code we can now have a series of check operators to give us error syndromes. error syndrome give us information about the possible error occurred on the state without actually disturbing the quantum state itself. Since we are working in the σ_x basis, use this pauli operator as a parity check syndrome. Similar to 3-qubit bit flip code let's look at the implementation of even number of X pauli operators like $\sigma_{x1}\sigma_{x2}$ on the logical states:

$$\sigma_{x_1}\sigma_{x_2}|\psi\rangle_L = \sigma_{x_1}\sigma_{x_2}(\alpha|+++\rangle + \beta|---\rangle) \tag{1.110}$$

$$= \sigma_{x_1}\sigma_{x_2}\alpha|+++\rangle + \sigma_{x_1}\sigma_{x_2}\beta|---\rangle \tag{1.111}$$

$$= \alpha|+++\rangle + (-1)^2\beta|---\rangle \tag{1.112}$$

$$= \alpha|+++\rangle + \beta|---\rangle \tag{1.113}$$

$$= |\psi\rangle_L \tag{1.114}$$

Now let's see the application of this operator on an erroneous state $\mathscr{E}_1|\psi\rangle = \sigma_{z_1}|\psi\rangle = \alpha|-++\rangle + \beta|+--\rangle$:

$$(\sigma_{x_1}\sigma_{x_2})\mathscr{E}_1|\psi\rangle = \sigma_{x_1}\sigma_{x_2}\sigma_{z_1}|\psi\rangle \tag{1.115}$$

$$= \sigma_{x_1}\sigma_{z_1}\sigma_{x_2}|\psi\rangle \tag{1.116}$$

$$= (-1)\sigma_{z_1}\sigma_{x_1}\sigma_{x_2}|\psi\rangle \tag{1.117}$$

$$= -\sigma_{z_1}|\psi\rangle \tag{1.118}$$

$$= -\mathscr{E}_1|\psi\rangle \tag{1.119}$$

The second line holds since two operators on two different qubits are two operators on two disjoint Hilbert spaces and thus they commute. The third line holds since

different Pauli operators (σ_x and σ_z) acting on one qubit anticommute. This relation suggests that the erroneous state $\mathscr{E}_1|\psi\rangle_L$ is an eigenstate of the $(\sigma_{x_1}\sigma_{x_2})$ operator with eigenvalue -1. Before we also saw that this operator acts trivially on the code space i.e., any logical state $|\psi\rangle_L$ is a +1 eigenstate of this operator. This is exactly what we had earlier for the bit flip channel. So we can construct a set of these check operators to cover all possible phase flip positions on 3 qubits. The resulting set is similar to that of 1.99 but with the exception that here we used σ_x operator in our check syndromes.

$$\{\sigma_{x_1}\sigma_{x_2}I_3,\ I_1\sigma_{x_2}\sigma_{x_3}\} \tag{1.120}$$

So far we managed to design two separate yet similar codes to correct either for one bit flip or one phase flip error on a qubit state. However, as mentioned earlier these errors can happen at the same time on a qubit state. So a good code should be able to correct for all these types of errors at simultaneously. Although we need a more complex code to achieve this, it hav been shown that the same logic for phase flip and bit flip code can be concatenated to make the 9-qubit Shor code [8, 28], which can correct up to one bit flip and one phase flip error simultaneously.

1.3.7 9-qubit Shor Code

Let's take a look at the depolarizing channel as a reasonable error model where all different single errors are possible:

$$\rho \to \mathscr{E}(\rho) = \underbrace{(1-p)\rho}_{\text{No error}} + \frac{p}{3}\underbrace{\sigma_x\rho\sigma_x}_{\text{Bit Flip}} + \frac{p}{3}\underbrace{\sigma_z\rho\sigma_z}_{\text{Phase Flip}} + \frac{p}{3}\underbrace{\sigma_y\rho\sigma_y}_{\text{Both Errors}} \tag{1.121}$$

The concatenation of idea of the 9-qubit shor code to protect against both phase flip and bit flip errors is simple. Let's start with the 3-qubit phase flip code and it's encoding to logical states:

$$|+\rangle \rightarrow |+\rangle_L = |+++\rangle \tag{1.122}$$

$$|-\rangle \rightarrow |-\rangle_L = |---\rangle \tag{1.123}$$

$$|\psi\rangle \rightarrow |\psi\rangle_L = \alpha|+\rangle_L + \beta|-\rangle_L = \alpha|+++\rangle + \beta|---\rangle \tag{1.124}$$

This encoding protects against one phase flip on any of the 3 qubits involved but no protection for bit flip errors. Now let's write this logical basis in the computational basis:

$$|+\rangle_L = |+++\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{1.125}$$

$$|-\rangle_L = |---\rangle = \underbrace{\frac{|0\rangle - |1\rangle}{\sqrt{2}}}_{\text{Phase qubit 1}} \otimes \underbrace{\frac{|0\rangle - |1\rangle}{\sqrt{2}}}_{\text{Phase qubit 2}} \otimes \underbrace{\frac{|0\rangle - |1\rangle}{\sqrt{2}}}_{\text{Phase qubit 3}} \tag{1.126}$$

Now that we have the logical states in computational basis, let's protect each of the 3 qubits involved with a 3-qubit bit flip repetition code:

$$\underbrace{\frac{|0\rangle + |1\rangle}{\sqrt{2}}}_{\text{Phase qubit i}} = \frac{|000\rangle + |111\rangle}{\sqrt{2}} \tag{1.127}$$

If we repeat this for all 3 phase protected qubits then we will end up with the following 9-qubit logical state:

$$|0\rangle_{Shor} = \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \tag{1.128}$$

$$|1\rangle_{Shor} = \frac{1}{2\sqrt{2}} \underbrace{(|000\rangle - |111\rangle)}_{\text{qubits 1 to 3}} \otimes \underbrace{(|000\rangle - |111\rangle)}_{\text{qubits 4 to 6}} \otimes \underbrace{(|000\rangle - |111\rangle)}_{\text{qubits 7 to 9}} \tag{1.129}$$

Now that we have the logical states the encoding is simple. A general 1 qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ can be encoded to these logical states $|\psi\rangle_L = \alpha|0\rangle_{Shor} +$

$\beta|1\rangle_{Shor}$. The error correction and decoding is also similar to the 3-qubit phase flip and bit flip codes. We have to follow the same idea of check operators to detect and locate errors (either phase or bit flip errors) on these 9 physical qubits. It's important to note that these 9 physical qubits are used to protect only 1 logical qubit state against all single quantum errors. Equation 1.129 shows that consecutive 3 qubit groups (qubit 1 to 3 , 4 to 6, and 7 to 9) each represent one bit flip repetition code. So the check operators for bit flip error syndromes should be extended on the physical qubits of these groups. In a simple analogy with the 3-qubit bit flip check operators we had earlier in equation 1.99 we can devise the bit flip check operators for these 9 qubit code:

$$\begin{aligned}
\text{phase qubit 1}: &\; Z_1\ Z_2\ I_3\ I_4\ I_5\ I_6\ I_7\ I_8\ I_9 \\
&\; I_1\ Z_2\ Z_3\ I_4\ I_5\ I_6\ I_7\ I_8\ I_9 \\
\text{phase qubit 2}: &\; I_1\ I_2\ I_3\ Z_4\ Z_5\ I_6\ I_7\ I_8\ I_9 \\
&\; I_1\ I_2\ I_3\ I_4\ Z_5\ Z_6\ I_7\ I_8\ I_9 \\
\text{phase qubit 3}: &\; I_1\ I_2\ I_3\ I_4\ I_5\ I_6\ Z_7\ Z_8\ I_9 \\
&\; I_1\ I_2\ I_3\ I_4\ I_5\ I_6\ I_7\ Z_8\ Z_9
\end{aligned} \tag{1.130}$$

The same logic can be used to derive the phase flip check operators (error syndrome operators) but with a difference. Since the phase flip code was the first layer of oue two layer concatanated encdoing, as stated earlier now groups of three qubits (1,2,3), (4,5,6) and (7,8,9) represents 3 phase flip repetition qubits. So using equation 1.131 we can now derive the 9 qubit phase flip check operators:

$$\{\underbrace{\sigma_{x_1}\sigma_{x_2}\sigma_{x_3}}_{\text{Phase qubit 1}}\ \underbrace{\sigma_{x_4}\sigma_{x_5}\sigma_{x_6}}_{\text{Phase qubit 2}}\ \underbrace{I_7\ I_8\ I_9}_{\text{Phase qubit 3}}\ ,\ \underbrace{I_1\ I_2\ I_3}_{\text{phase qubit 1}}\ \underbrace{\sigma_{x_4}\sigma_{x_5}\sigma_{x_6}}_{\text{phase qubit 2}}\ \underbrace{\sigma_{x_7}\sigma_{x_8}\sigma_{x_9}}_{\text{phase qubit 3}}\} \tag{1.131}$$

We can summarize all these check operators in a big matrix of pauli operators:

$$
\left[\begin{array}{ccc|ccc|ccc}
Z_1 & Z_2 & I_3 & \multicolumn{3}{c|}{\multirow{2}{*}{I}} & \multicolumn{3}{c}{\multirow{2}{*}{I}} \\
I_1 & Z_2 & Z_3 & & & & & & \\
\hline
\multicolumn{3}{c|}{\multirow{2}{*}{I}} & Z_4 & Z_5 & I_3 & \multicolumn{3}{c}{\multirow{2}{*}{I}} \\
\multicolumn{3}{c|}{} & I_1 & Z_5 & Z_6 & \multicolumn{3}{c}{} \\
\hline
\multicolumn{3}{c|}{\multirow{2}{*}{I}} & \multicolumn{3}{c|}{\multirow{2}{*}{I}} & Z_7 & Z_8 & I_3 \\
\multicolumn{3}{c|}{} & \multicolumn{3}{c|}{} & I_1 & Z_8 & Z_9 \\
\hline
X_1 & X_2 & X_3 & X_4 & X_5 & X_6 & I_7 & I_8 & I_9 \\
I_1 & I_2 & I_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9
\end{array}\right] \tag{1.132}
$$

Now the error correction and decoding is simple. we repeat what we did for the 3-qubit bit flip and phase flip codes. First we apply all these check operators to the physical qubits. if the logical state of these 9 physical qubits is a +1 eigenvalue of all these 8 check operators, it means that there is no error that we can correct for and the state is one of the logical possible $|0\rangle_{Shor}$ or $|1\rangle_{Shor}$ states. Otherwise, if any of these operators syndrome is -1 instead of a +1 then we detect and locate the error exactly like how we did this for the repetition codes.

There a couple of important final remarks on what we saw about the 9-qubit Shor code. First of all, this code can only encode 1 qubit state into a group of physical qubits. So one actually needs 9 qubits in order to protect the state of 1 qubit carrying quantum information. Second, the 9-quibt Shor code is only able to correct for single qubit errors and it fails on the event of two simultaneous errors or more. So we need more sophisticated codes in order to correct for more number of errors and also maintain the redundancy in a reasonable level.

Third point is that this code is able to correct for both bit flip and phase flip errors by using two different codes concatenated together. As you can see from the table of check operators, we can perfectly divide these operators to two sets of X-check operators and Z-check operators for phase flip and bit flip codes respectively. The idea of using two codes in parallel one for phase flip and one for bit flip errors is an important and practical trick which made us able to build quantum codes which are actually able to correct for all possible quantum error types in our model.

The final remark is the relation between these quantum codes and classical

codes. The idea of check operators and error syndromes is very similar and analogue to parity check matrices and syndromes in linear classical codes we saw earlier. Although these quantum codes we studied so far were simple generalization for pedagogical purposes actual complex and applied quantum codes are also made of classical codes. As an example Calderbank, Shor and Stean showed a systematic way of building a family of quantum error correcting codes directly from classical error correcting codes [28] called CSS codes. Similar to what we pointed here CSS codes also consist of two classical codes in parallel to make a quantum error correction code robust against both bit flip and phase flip errors.

1.3.8 CSS Codes

The analogy we build up between quantum and classical error correction by various simple examples gives rise to an important family of quantum codes generated out of classical linear codes we studied earlier. CSS codes named by initials of its inventors are the structured realization of quantum codes based on classical codes. These codes are a subclass of quantum stabilizer codes which we'll study later in this chapter. Their structured construction makes it easier to look for a quantum code with desired properties.

The CSS code is built of two $[n,k_1]$ and $[n,k_2]$ linear classical codes $\mathscr{C}_1$ and $\mathscr{C}_2$ respectively with the condition that $\mathscr{C}_2 \subset \mathscr{C}_1$ and $\mathscr{C}_1$ and $\mathscr{C}_1^{\perp}$ can both correct up to t errors. The resulting code will be a $[n,k_2-k_1]$ quantum code which can correct up to t qubit errors. Now we have to focus on the logical states and their encoding for a $CSS(\mathscr{C}_1,\mathscr{C}_2)$ quantum code. If x is a codeword in $\mathscr{C}_1$ the following will define our logical states[12]:

$$|x+\mathscr{C}_2\rangle = \frac{1}{\sqrt{|\mathscr{C}_2|}} \sum_{y\in\mathscr{C}_2} |x+y\rangle \tag{1.133}$$

This code will encode k_1-k_2 qubits in n qubits, so it has $2^{k_1-k_2}$ such logical encoded states. As we mentioned earlier this code is able to correct for both bit flip and phase flip errors. We choose E_1 and E_2 to be binary vectors of size n and

[12] All the arithmetic operations between classical codewords x and y are binary and done in $\mathbb{Z}_2$.

their i'th element is 1 if there is a bit flip happening on i'th qubit for E_1 and a phase flip on i'th qubit for E_2 and 0 otherwise. In this way we have two vectors that can model the random errors happening on the n physical qubits, and then the following will be an erroneous state.

$$\mathscr{E}_1\mathscr{E}_2|CSS\rangle = \frac{1}{\sqrt{|\mathscr{C}_2|}}\sum_{y\in\mathscr{C}_2}(-1)^{(x+y).E_2}|x+y+E1\rangle \tag{1.134}$$

where $\mathscr{E}_1$ and $\mathscr{E}_2$ are the corresponding quantum error operators acting on a logical CSS code state to cause bit flip and phase flip errors located at the support of E_1 and E_2. Exactly similar to classical error correction we use the parity check matrix to detect bit flip or phase flip errors. Let's try the $\mathscr{C}_1$ code parity check matrix H_1 to detect bit flip errors. Since we can not apply that directly to the physical qubits we suppose we have enough ancilla qubits prepared in $|0\rangle$ and in a product state with our actual data storing qubits. Since the parity check matrix acts trivially on the code space we'll have the following result using it:

$$|x+y+E_1\rangle|0\rangle \xrightarrow{H_1} |x+y+E_1\rangle|H_1(x+y+E_1)\rangle = |x+y+E_1\rangle|H_1E_1\rangle \tag{1.135}$$

$$H_1(\mathscr{E}_1\mathscr{E}_2|CSS\rangle)\underbrace{|0\rangle}_{Ancilla} = \frac{1}{\sqrt{|\mathscr{C}_2|}}\sum_{y\in\mathscr{C}_2}(-1)^{(x+y).E_2}|x+y+E1\rangle|H1E_1\rangle \tag{1.136}$$

$$= (\mathscr{E}_1\mathscr{E}_2|CSS\rangle)|H_1E_1\rangle \tag{1.137}$$

The last line shows that application of the parity check matrix will keep the data qubits in the state they were before and gives us a syndrome of bit flip errors happened on the ancilla qubit. We can now measure the ancilla qubit without disturbing the quantum system and get information about the bit flip errors. After we learned which qubits had been exposed to bit flip errors we simply apply a NOT gate to correct for the bit flip error happened on them. This will bring the state to a CSS code state only with phase flip errors:

$$\mathscr{E}_2|CSS\rangle = \frac{1}{\sqrt{|\mathscr{C}_2|}} \sum_{y\in\mathscr{C}_2} (-1)^{(x+y).E_2} |x+y\rangle \tag{1.138}$$

The same logic with a slightly different implementation uses the $\mathscr{C}_2$ code parity check matrix H_2 to get information about phase flip errors and then we correct for those errors in the same fashion to bring the erroneous codeword back to its code space.

A simple example of a CSS code is the $[7,1]$ *Steane* code whcih uses the linear classical $[7,4,3]$ *Hamming code* to generate a $[7,4]$ code as $\mathscr{C}_1$ and a $[7,3]$ code as $\mathscr{C}_2$. This CSS code encodes 1 logical qubit in 7 physical qubits and it can correct 1 phase flip and bit flip error. One can compare this new quantum code with the 9-qubit Shor code which used 9 qubits to achieve the same goal, and understand how this new construction can help design better codes. The $[7,1]$ Steane code can protect 1 logical qubit state so it only has two logical codewords $|0\rangle_L$ and $|1\rangle_L$:

$$\begin{aligned} |0\rangle_L = \frac{1}{\sqrt{8}}(&|0000000\rangle + |1010101\rangle + |0110011\rangle + |1100110\rangle \\ &+ |0001111\rangle + |1011010\rangle + |0111100\rangle + |1101001\rangle) \end{aligned} \tag{1.139}$$

$$\begin{aligned} |1\rangle_L = \frac{1}{\sqrt{8}}(&|1111111\rangle + |0101010\rangle + |1001100\rangle + |0011001\rangle \\ &+ |1110000\rangle + |0100101\rangle + |1000011\rangle + |0010110\rangle) \end{aligned} \tag{1.140}$$

This gave us an understanding of how our knowledge of classical error correcting codes can be used to build good quantum error correcting codes. We also learn why we always need two codes running in parallel to be able to correct for both quantum bit flip and phase flip errors.

The heart of error detection procedure in a classical linear code is its parity check matrix H. During the last few sections on error correction we have seen how intuitively this useful notion is generalized to quantum error correcting codes as check operators. These check operators work very similar to parity check matrices to give us syndromes about errors without disturbing the state carrying information.

In the case of CSS codes is easier to establish this relation since these codes are directly made of classical linear codes with their parity check matrices serving as check operators. Now let's have a more close look at check operators for $[7,1]$ *Steane* code:

$$\left[\begin{array}{ccccccc|ccccccc} I_1 & I_2 & I_3 & X_4 & X_5 & X_6 & X_7 & & & & & & & \\ I_1 & X_2 & X_3 & I_4 & I_5 & X_6 & X_7 & & & \mathrm{I} & & & & \\ X_1 & I_2 & X_3 & I_4 & X_5 & I_6 & X_7 & & & & & & & \\ \hline & & & & & & & I_1 & I_2 & I_3 & Z_4 & Z_5 & Z_6 & Z_7 \\ & & & \mathrm{I} & & & & I_1 & Z_2 & Z_3 & I_4 & I_5 & Z_6 & Z_7 \\ & & & & & & & Z_1 & I_2 & Z_3 & I_4 & Z_5 & I_6 & Z_7 \end{array}\right] \tag{1.141}$$

As we discussed earlier, the two linear codes work at the same time to provide resilience against both bit flip and phase flip errors. The check operators due to each linear code looks similar except the that one is a product of Pauli X operators to detect phase flips and the other is a product of Z operators to detect bit flips. This similarity between the two codes suggests a new regrouping and representation of these check operators as it can be seen in 1.142. In general, for a CSS code $CSS(\mathscr{C}_1,\mathscr{C}_2)$ with H_1 and H_2 as the parity check matrices for linear codes $\mathscr{C}_1$ and $\mathscr{C}_2$ respectively, we can write the quantum error correcting check operator matrix in the following format by replacing 1's in the first parity check matrix by X operators and 1's in the second parity check matrix by Z operators.

$$\left[\begin{array}{c|c} H_1^X & \mathrm{I} \\ \hline \mathrm{I} & H_2^Z \end{array}\right] \tag{1.142}$$

We can test the application of these check operators on the logical states of the Steane code, let's pick the first check operator and apply it to the $|0\rangle_L$. First we investigate the application of this operator on each term in the superposition:

$$\text{term 1: } X_4X_5X_6X_7|0000000\rangle = |0001111\rangle \rightarrow \text{term 5}$$
$$\text{term 2: } X_4X_5X_6X_7|1010101\rangle = |1011010\rangle \rightarrow \text{term 6}$$
$$\text{term 3: } X_4X_5X_6X_7|0110011\rangle = |0111100\rangle \rightarrow \text{term 7}$$
$$\text{term 4: } X_4X_5X_6X_7|1100110\rangle = |1101001\rangle \rightarrow \text{term 8}$$
$$\text{term 5: } X_4X_5X_6X_7|0001111\rangle = |0000000\rangle \rightarrow \text{term 1}$$
$$\text{term 6: } X_4X_5X_6X_7|1011010\rangle = |1010101\rangle \rightarrow \text{term 2}$$
$$\text{term 7: } X_4X_5X_6X_7|0111100\rangle = |0110011\rangle \rightarrow \text{term 3}$$
$$\text{term 8: } X_4X_5X_6X_7|1101001\rangle = |1100110\rangle \rightarrow \text{term 4} \tag{1.143}$$
$$\rightarrow X_4X_5X_6X_7|0\rangle_L = |0\rangle_L \tag{1.144}$$

This shows how the check operators act like identity operator on logical encoded states. In other words, the encoded states are the +1 eigenstates of check operators. These check operators form a group called *stabilizer group*. Each of these operators are called one stabilizer of the code because they stabilize the encoded states to itself, while any other state out of the encoded space is a -1 eigenstate of at least one of these stabilizers. This group theoretic understanding of check operators gave birth to an important family of error correcting codes called stabilizers codes which we will discuss in more detail in the remainder of this chapter. Before that, in the next short subsection we review a list of common steps shared between all different quantum error correcting codes, specially to address what the similarities and the differences are between classical and quantum error correction.

1.3.9 General QEC Picture and Concatenated Codes

Although various families of quantum error correcting codes have been developed for various error models and properties, There are certain fundamental items which are shared between all quantum error correcting schemes. The following list can be both a take away summery of what we mentioned about quantum error correction and it also serves as a list of common facts shared among all quantum error correcting codes:

1. The unit of quantum information is the state of a qubit, which is a vector in a two dimensional vector space (Hilbert space) in $\mathbb{C}^2$.

2. Quantum noise acts like error operators on a qubit states. These errors can be discretized into bit flip and phase flip errors acting on qubits.

3. In order to protect the qubits against these quantum errors we need quantum error correction. We can not make copies of quantum information but we can make copies of the known eigenbasis in which these states are represented. to encode a quantum state one only needs to represent the state in the logical basis instead of the actual physical basis.

4. Using this we encode the state of a set of qubits in the states of a bigger set of physical qubits. the information we want to protect is called the logical state or encoded state and the set of redundant qubits we use to encode this information to is called the physical qubits.

5. Classical error correcting codes with special conditions can be used to build quantum error correcting codes. CSS codes are an example of an easy structure that can be used to make quantum codes out of existing classical codes. These codes consists of two separate error correcting codes, one for bit flips and one for phase flip errors.

6. Exactly like parity check matrix in classical linear codes, quantum codes also have a set of check operators. these operators act trivially on the states in the code space but they leave error syndrome in ancilla qubits when they hit the error operators.

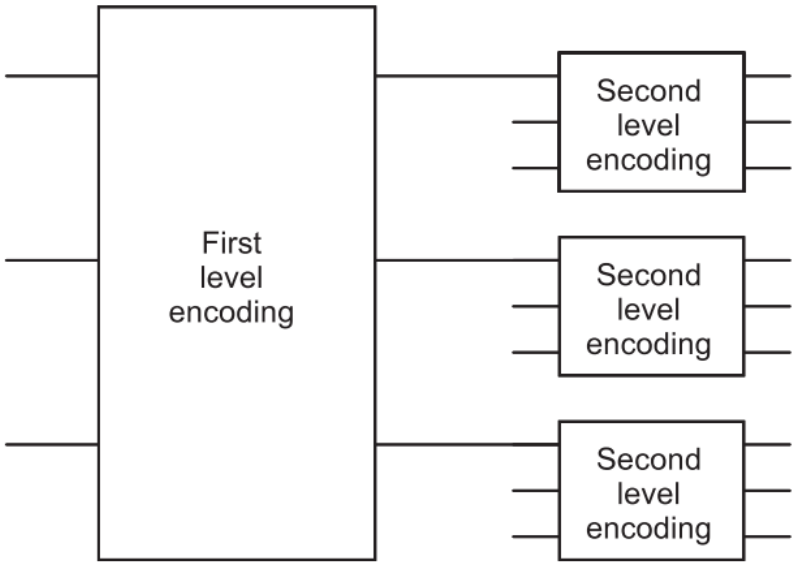
7. The error syndromes on ancilla qubits can be measured to detect and locate the errors without disturbing the encoded quantum information.

8. After detecting the erroneous physical qubits we apply the proper operation to the erroneous state to recover it and undo the effect of errors bringing it back to the code space.

9. Exactly similar to classical codes, each quantum code has its own properties, the number of logical qubit states that it can encode, the number of physical

qubits that it needs, the number of phase flip and bit flip errors that it can correct and ...

Figure 1.3: A schamatic representation of a two level concatanated codes. The logical qubits are encoded into a seris of qubits in the first layer. Then a second code ise used to encode each individual qubit in the first layer to even more physical qubits in the second layer. [28]



Now that we have a general understanding of quantum error correction and its relation with classical error correcting codes, we can discuss the idea of concatenated coding and explain the threshold theorem in more depth. As the name suggests the concatanated coding is the idea of using error correcting codes repeatedly. As we mentioned in previous sections we can encode quantum information in the state of physical qubits by using a quantum code. However, each of those physical qubits used in the first encoding carries a certain quantum state and nothing stops us to use another quantum code to encode those states in a bigger group of physical qubits. Figure 1.3 illustrates a two level concatenation of quantum codes in general. This repetition can go on and we can always add another layer of encoding. any quantum code can correct up to a certain number of quantum errors.

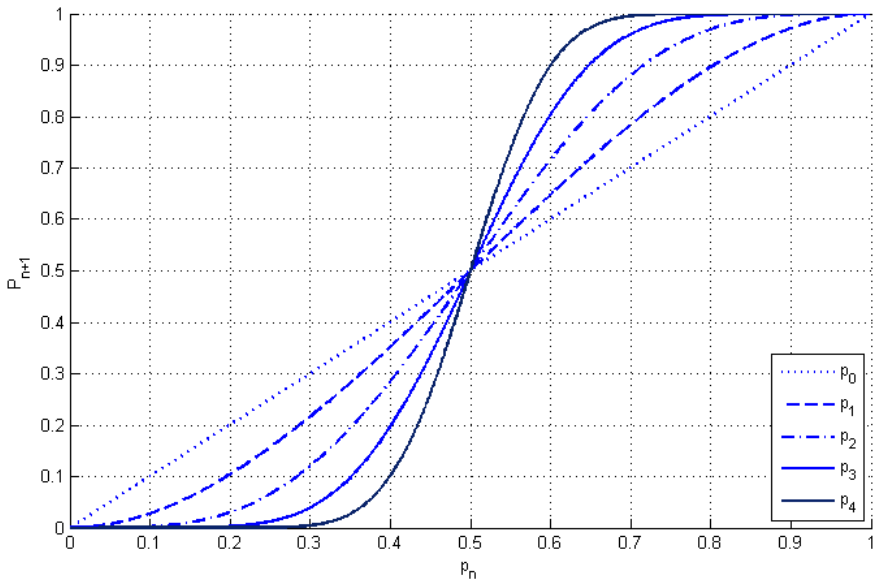
The 9 qubit and the 7 qubit codes are both able to only correct up to a single arbitrary error, which means they fail to protect the state if more than two or more errors occur on the physical qubits. We actually calculated the success probability of the 3 qubit code which was $1-3p_e^2+2p_e^3$ where p_e is the probability of a single error on the physical qubit. Thus by one encoding layer we were able to reduce the error probability from p_e to less than $3p_e^2$. With the same logic we can show that each quantum code based on its detection/correction capacity can reduce the initial individual error probability p_e to at most cp_e^t where c is a constant and t depends on the code distance. Without loss of generality let's work with a code which can correct up to one error and add another layer of error correction. This time the failure probability is reduced to at most $c(cp_e^2)^2$. This concatenated repetition can go on and on up to an arbitrary k number of layers. In that case, the final failure probability is reduced to at most $\frac{(cp_e)^{2^k}}{c}$. Now if we want this concatenated code to have a certain accuracy ε, the failure probability should be less than ε:

$$\frac{(cp_e)^{2^k}}{c} < \varepsilon \tag{1.145}$$

If we choose k large enough this inequality can hold provided that $cp_e < 1$ or $p_e < \frac{1}{c}$. This argument shows that arbitrary accuracy on a quantum memory is possible by using enough layers of concatenation provided that the individual error probability is lower than a certain threshold $p_{th} = \dfrac{1}{c}$. A similar argument can be used to generalize this result to the case when we want to have faulty gates and do quantum computation. Figure 1.4 illustrates how adding more layers of concatenation can help decreasing the failure error provided that the initial error value is below a threshold. This is a brief explanation of how threshold theorem is proved and helps us to have fault tolerant quantum computation using concatenation of error correcting schemes.

This summery is a good motivation to start our next section on Stabilizer codes. One of the significance of stabilizer codes is the very well structured formalism to represent a code. All the vague analogies we made between classical and quantum codes are rigorously formalized in the stabilizer formalism.

Figure 1.4: The residual error after n'th encoding layer versus the initial error for a concatenated 3qubit bit flip code. Below a certain threshold as we increase the encoding layers the residual error drops down to the desired value.



1.3.10 Stabilizer Codes

Stabilizer codes introduced by Daniel Gottesman [18], are a family of quantum codes with special group theoretical structures. This structure makes them easier to find and to analyze. Stabilizer formalism is the powerful group theoretical framework in which these codes are discussed.

Let's assume a subspace $\mathscr{C}$ and error operators E_i in a way that an erroneous state $E_1|\psi\rangle$ is orthogonal to $|\psi\rangle$ for any state $|\psi\rangle$ in $\mathscr{C}$. A measurement can be conducted to show whether the system is in state $|\psi\rangle$ or $E_1|\psi\rangle$ to detect if error E_1 has happened. In order to distinguish errors we can generalize this condition to any two errors on an arbitrary state on the code space:

$$\langle\phi|E_1E_2|\psi\rangle = 0 \tag{1.146}$$

If this relation holds for any two states in subspace $\mathscr{C}$ and any two possible single qubit errors, then $\mathscr{C}$ is a valid quantum error correcting code. Now one way to build such a subspace $\mathscr{C}$ with these properties is to fill C with +1 eigenstates of an operator G such that any error pair will anticommute with G:

$$\langle\phi|E_1E_2|\psi\rangle = \langle\phi|E_1E_2G|\psi\rangle \tag{1.147}$$

$$= -\langle\phi|GE_1E_2|\psi\rangle = -\langle\phi|E_1E_2|\psi\rangle = 0 \tag{1.148}$$

The second line holds because G and E_1E_2 anticommute. Note that since $\mathscr{C}$ is a +1 eigen space of operator G then:

$$\forall\psi \in \mathscr{C}, G|\psi\rangle = |\psi\rangle \tag{1.149}$$

G fixes (acts as an identity operator on) all code-words in $\mathscr{C}$. The set of all operators like G which fix the code-words form a group S, called the stabilizer of the code [18]. The stabilizer group S of a $[n,k]$ stabilizer code has a minimal representation in terms of $n-k$ independent generators. In order to describe a code in stabilizer formalism one only needs to find the generators of group S, all other operators that fix the code can be written as products of these generators. In fact what we had in equation 1.99 was the generator for the stabilizer group of the 3-qubit bit-flip code.

The generators of the stabilizer group S are quantum operators on the physical qubits of the code. These operators consist of combinations of the Pauli operators σ_x to anticommute with phase flip errors, σ_Z operators to anticommute with bit flip errors, and Identity operators. These operators act trivially on the code space while they leave error syndromes when acting on an erroneous state. This is exactly the formal definition of what we called a check operator so far. Like all other quantum codes, each stabilizer code can encode a specific number of logical qubits in a specific umber of physical qubits and it's able to correct for a specific number of single qubit errors happening on the physical qubits. Now what are the error operators that a stabilizer ocde can correct. Before mentioning the error correction condition we need a couple of well-known definitions and the following theorem[13]:

[13]For the proof of the following two theorems we encourage the reader to refer to [18]

Theorem 1. *The set of tensor products of Pauli operators σ_x, σ_y, σ_z and I with a possible overall factor of -1 or $\pm i$ forms a group $\mathscr{G}$ under multiplication.*

We use the notation $\mathscr{G}_n$ when we want to refer to the group of tensor products of the Pauli operators on n qubits. For example $XIZII$ is an element of G_5 which applies a σ_x to the first qubit and a σ_z to the third one, and act trivially on the rest of the qubits. As we used earlier, we put indecies for each Pauli operator to show which qubit it is acting on, so for the same example we write: $X_1I_2Z_3I_4I_5$. For every group like $\mathscr{G}_n$ we have a notion of group normalizer which is defined as follows:

Definition 2. *The **normalizer** of a subgroup S in the group $\mathscr{G}$ is defined to be* $\mathrm{N}_{\mathscr{G}}(S) = \{g \in \mathscr{G} \mid gS = Sg\}$.

The stabilizer group S is a subgroup of the bigger Pauli group $\mathscr{G}_n$. With this definition the normalizer of this subgroup consists of all elemtns in $\mathscr{G}_n$ which commute with all elements in stabilizer group S. In other words, The normalizer of the stabilizer group consists of all combinations of Pauli operators in $\mathscr{G}_n$ on the physical qubits which commute with all stabilizers in S. The following theorem now tells us which errors we can correct using a stabilizer code:

Theorem 2. ***Error-correction conditions for stabilizer codes*** *Let S be the stabilizer group for a stabilizer code $C(S)$. Suppose $\{E_j\}$ is a set of operators in $\mathscr{G}_n$ such that $E_j^\dagger E_k \notin N(S) - S$ for all j and k. Then $\{E_j\}$ is a correctable set of errors for a stabilizer code $C(S)$.*

When using a stabilizer code, we encode the quantum state we want using the logical eigenbasis defined by the codewords of the code. In order to correct for errors, we apply all elements of the stabilizer group to the system and measure the error syndromes on ancilla qubits which are interacting with the physical qubits affected by the stabilizer. The collected error syndromes gives us information to detect and locate the errors. Then we apply proper operations to undo those errors and preserve the logical quantum state encoded in the physical qubits.

The following is the stabilizer group of a $[5,1]$ stabilizer code which encodes one logical qubit state in 5 physical qubits and can correct up to 1 arbitrary single qubit error. This code has 5-1=4 stabilizer generators:

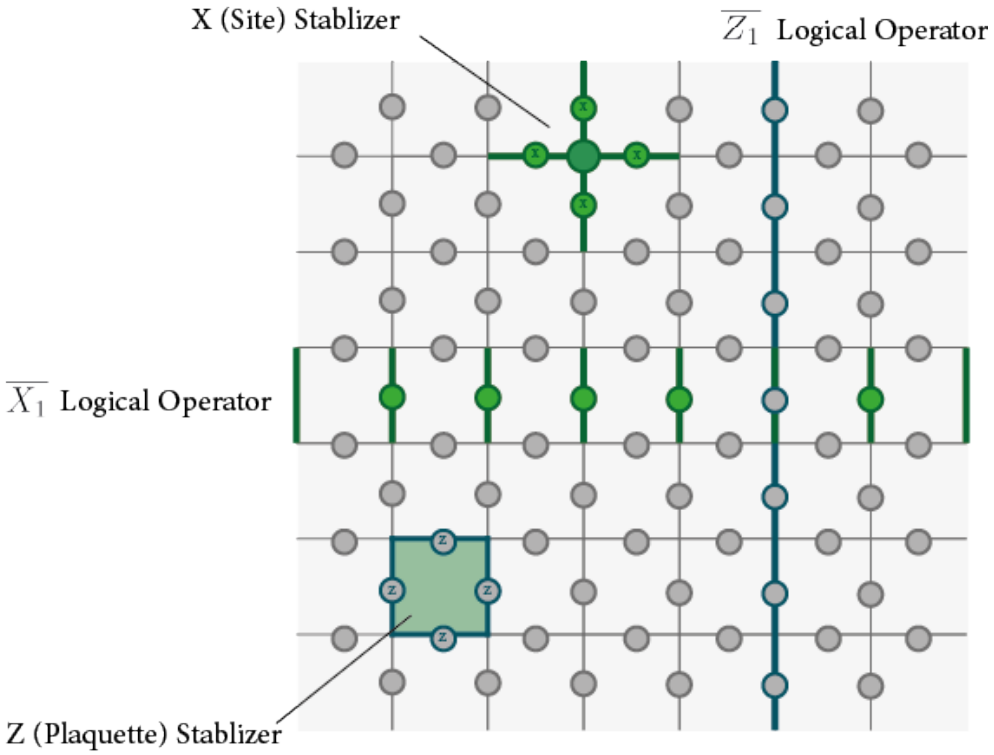
$$
\begin{aligned}
G_1 &= X_1 Z_2 Z_3 X_4 I_5 \\
G_2 &= I_1 X_2 Z_3 Z_4 X_5 \\
G_3 &= X_1 I_2 X_3 Z_4 Z_5 \\
G_4 &= Z_1 X_2 I_3 X_4 Z_5
\end{aligned} \tag{1.150}
$$

1.4 Surface Codes

Quantum surface codes are a family of stabilizer codes ([18]) and were introduced by Bravyi and Kitaev [4] as an improved planar version of Alexi Kitaev's toric codes [7]. Fault tolerant quantum computation is possible with high threshold using the surface code as the error correcting code [33]. The main purpose of this research is to study the classical post processing in the error correction step of fault tolerant quantum computation using surface codes [35] [7]. A thorough and nicely written review of surface codes can be found in Fowler et al. [16].

The underlying principles of quantum error correction is exactly the same for surface codes. However, surface codes have additional advantages because they also make use of the especial geometric positioning of the physical qubits. So far what we have learned about quantum coding was purely theoretical. We never mentioned how the physical qubits will put together interact and etc. Surface code is an example that not only demonstrates the principles of quantum error correction for pedagogical purposes but it also illustrates how a physical system can exploit these theoretical principles to make a robust quantum memory. Another advantage of surface codes is that not only they are good codes to prevent quantum information from quantum noise, but they also make quantum operations possible. The especial physical geometry and interaction of qubits with each other makes us able to realize quantum gates. Raussendorf and colleagues [33] show how we can use surface codes not only for protecting quantum states but also doing fault tolerant quantum computation with them.

Figure 1.5: Lattice definitions. X and Z stabilizers and Logical X and Z operators.



1.4.1 Geometric Picture

Surface codes are formed from a two dimensional lattice architecture of qubits on a surface of nontrivial topology. The data qubits (physical qubits which store information about the encoded state) are placed on the edges of the 2D lattice of linear size L, having $2L^2$ qubits in total. Ancilla qubits are placed on the plaquettes and vertices of the lattice and there is a nearest neighbor Ising interaction between each ancilla qubit and its neighboring data qubits. The ancila qubits can be measured to get error syndromes about the data qubits adjacent to them.

The two dimensional lattice architecture, the local nearest neighbor interaction, and high tolerance of surface codes to errors (as high as 1% per operation error)

[14], makes it one of the most realistic methods of building a solid-state quantum computer (Fowler et al. [16]). We use the stabilizer formalism to discuss the error correction on surface codes.

The stabilizer formalism suggests us to introduce the stabilizers of this code first. In order to have an easier notation, we use X and Z instead of σ_x and σ_z operators, respectively. The following defines the stabilizers on the surface code and their commutation relations:

$$Z_P = \bigotimes_{l \in P} Z_l \tag{1.151}$$

$$X_s = \bigotimes_{l \in s} X_l \tag{1.152}$$

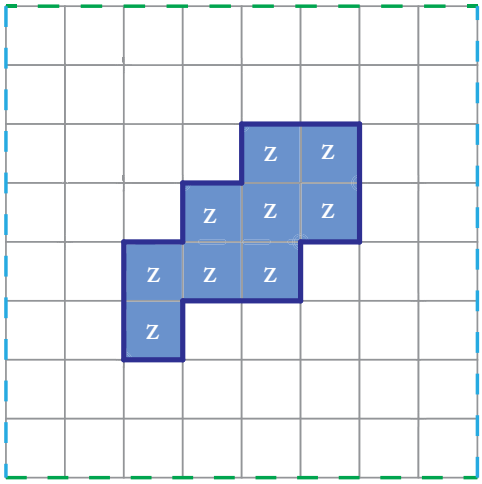
$$[Z_{P_i}, X_{P_j}] = 0, \forall i, j \leq L \tag{1.153}$$

The Z_P stabilizers are called the *plaquette* stabilizer and X_s stabilizer are called *site* stabilizers. As we can see in the equations 1.151-1.153 these operators are defined on all sites and plaquette of the lattice. The commutation condition between stabilizer operators is satisfied because as it can be seen in figure 1.5 they are either applied to 4 different qubits or if they share qubits they always share an even number of them. This is the first time we see how the geometry of defining the the lattice and the square or star like shape of stabilizer operators is playing a role in the code definition.

Now let's study this geometry more. Applying stabilizers to the data qubits will not change the encoded information, because the quantum state of the system is a +1 eigenstate of the stabilizer operators. How can we interpret this in a geometric picture. Without loss of generality let's focus on plaquette stabilizers. From 1.2.2 we know that Pauli operators are both unitary and hermitian because of that two consecutive application of a Pauli operators is the same as the Identity operator or $\sigma_i^2 = I$ for $i \in x, y, z$. Combining this fact with the geometric picture of a group of plaquette stabilizers in figure 1.6 shows that the net effect of the stabilizer operators

[14] Just as a simple comparison, a two dimensional lattice implementation of Steane and Bacon-shor codes with nearest neighbor interactions has an error threshold of 2×10^{-5} compared to 0.01 of surface codes. [37]

Figure 1.6: A trivial cycle of Pauli Z operators shown in solid dark blue lines as a multiplication of the Z stabilizer operators on a group of plaquettes on the lattice. Each plaquette applies a pauli Z operator on each of the 4 data qubits adjacent to it. Now, two adjacent plaquettes share exactly one data qubit because of the square arcitecture of the lattice. Two adjacent plaquette operators apply the Z Pauli operator twice on the shared qubit. Since $\sigma_i^2 = I$ a group of adjacent plaquettes act trivially on the data qubits inside and only apply Pauli operator on a closed loop of data qubits forming the boundary of the plaquettes. This cycle is a trivial cycle on this lattice. The lattice boundaries with the same dashed line color are identified which forms a troric topology in this case.



on the lattice of qubits is equal to a closed loop of Pauli Z operators. Their effect on qubit inside the loop is canceled because each qubit inside the closed loop is hit twice and $\sigma_z^2 = I$. On the other hand we know that the encoded quantum state in this surface code is the +1 eigenstate of all stabilizer operators. So this closed loop of Pauli operators is also an stabilizer of the code since it can be written as multiplication of the stabilizer group generators of the surface code. So the result of the special geometric picture of surface codes is that any closed loop of Pauli operators [15] which can be written as multiplication of stabilizer generators[16], acts

[15] Either they are random noise or applied by us as the controllers of the quantum computer.

[16] Or in this geometric picture, any closed loop of Pauli operators which can be tiled with stabilizer generators.

trivially on the code space. In order to formally define what we just discussed we need a couple of definitions:

Definition 3. *A 1-chain is a mapping from $\mathbb{Z}_2 = \{0,1\}$ to each edge on the lattice.*

The set of all 1-chains form a vector space over $\mathbb{Z}_2$. The sum of two 1-chains is disjoint union of edges with a non-zero value in each 1-chain. a 0-chain is then a similar assignment of $\mathbb{Z}_2$ values to the sites and a 2-chain is the assignment of such values to the plaquettes of the lattice. A boundary operator σ is then defined to take 2-chains to 1-chains and 1-chains to 0 chins. The boundary of a 2-chain (plaquette) is the sum of 4 edges (1-chians) surrounding it and the boundary of a 1-chain is the two sites (0-chians) at its ends. This brings us to the notion of *trivial cycle* on a surface code.

Definition 4. *A chain with trivial boundary is called a cycle.*

Pauli operators can be a expressed as tensor products of X and I operators times a tensor product of Z and I operators. The tensor product of Z and I operators can define a 1-chain where edges acted on by Z operators are mapped to 1 and the rest of the edges are mapped to 0. As we discussed earlier, such a 1-chain commutes with all plaquette operators. However, it only commutes with site operators where an even number of Z operators act on the edges adjacent to that specific site. Thus the 1-chain should have a trivial boundary and consequently it is a cycle.[17]

Definition 5. *A cycle is homologically trivial if it is the boundary of a 2-chain.*

So a tensor product of Z or X Pauli operators on the lattice which forms a closed loop an can be tiled by stabilizer generators is a trivial cycle as it's shown in figure 1.6. There also exists homologically nontrivial cycles on the lattice topology.

Definition 6. *A cycle is homologically nontrivial if it can not be expressed as the boundary of any 2-chain.*

Whether a combination of Pauli operators make a trivial cycle on a surface or not depends on the boundaries of the surface. Figure 1.5 illustrates a group of Pauli

[17] A similar argument can be made on the dual lattice to show that tensor products of X and I operators are also cycles on the dual lattice. The dual lattice is the lattice obtained by changing all plaquette to sites, site to plaquettes, vertical edges to horizontal edges and vice versa.

Z operators named as *logical Z* operator. If the two horizontal boundaries of the surface are identified together and the two vertical boundaries are also identified together, then this operator can also make a closed loop (a cycle). However, this closed loop can not be tiled with Plaquette operators. So a series of Pauli operators on data qubits which connects two identical boundaries of the surface code will also make a loop in definition but this loop of operators can not be written as multiplication of stabilizer operators. This operator commutes trivially with the stabilizers so it does not generate an error syndrome. However, it acts non-trivially on the code space which affects the logical quantum information stored in the lattice.

Let's rephrase what we just discussed a take a closer look at it. A non-trivial cycle of operators will not act trivially on the quantum state of the lattice, or in other words the encoded logical state of the system is not a +1 eigenstate of this operator. So applying this loop of Pauli operators not only will change the state of the individual physical qubits on the lattice but it will also change the logical encoded state of the system. [18] This is in fact the reason we call these operators "Logical" operators, because they act on the system and change the logical state of the system which we wanted to protect from changing. Similar to the classical case of a bit flip, if we know which logical operator has happened on the lattice we can simple apply the same operator again and cancel its effect. However, the problem with the non-trivial cycles is that they commute with stabilizer operators. As we discussed earlier in 1.3.10, stabilizers act like parity check matrices and give us syndromes about the errors. They key idea behind the stabilizers is in their commutation relation. Stabilizers commute with each other and they can detect error operators which do not commute with them. This is how a stabilizer give us an error syndrome. Now if an operators occurs on the qubits which changes the state of physical qubits and it can not be detected by the stabilizers it will act as an un-correctable error on the logical information. So if we apply a series of Pauli operators which make a non-trivial cycle on the code surface,it means we have applied a logical operator on the encoded state. However, If these operators happen due to a random event (like a random noise) they change the logical information and can not be detected. Thus they act as un-correctable errors.

[18] Remember that the logical encoded state can remain the same if the state of physical qubits change up to trivial cycle of Pauli operators.

This discussion of trivial and non-trivial cycles shows how the geometry of physical qubits, the shape [19] of stabilizer operators and also the topology of the surface of the lattice form the unique proprieties of the surface code.

1.4.2 Surface Codes, Error Correction and Threshold

When we realize the surface code, the physical data qubits on the lattice are subject to quantum noise. According to our earlier discussion in section 1.3.3, we can discretize quantum noise as combinations of bit flip and phase flip errors on physical qubits. Errors can happen on a single qubit or they can happen on adjacent qubits and form a longer chain of error operators[20]. Figure 1.7 illustrates how single random errors on adjacanet qubits on the lattice can form long chains of Pauli phase or Bit flip operators.

The key to detect an error is to use error syndromes provided by the stabilizer generators. As we studied so far, stabilizer generators are like check operators. On repeated cycles, we measure the error syndrome of all plaquette and site stabilizers. As it was mentioned in section 1.3.10 if an error operator anticommutes with a stabilizer operator we see a -1 eigenvalue of that stabilizer which is an error syndrome. However, as it's illustrated in figure 1.8 chains of error operators commute with both plaquette and site stabilizers except at their boundaries. So the only information we detect about errors is the location of the boundaries of those errors on the lattice 1.9.

The next is to correct for detected errors. If we knew which qubits were subject to random error operators we could apply the same operator to those qubits again and eliminate the error bringing back the system into its encoded state. However, we showed previously that the only information we can get from error syndrome is the location of the boundaries of chains of error operators. Although we do not know the exact position of each chain of errors occurred, but we also showed that

[19] While we are interested to keep the geometric intuition, the reader should note that what we explain is exactly in line with the abstract definition in stabilizer formalism. For example, by talking about the shape of the stabilizer we focus on the data qubits where a certain stabilizer acts non-trivially. As it was mentioned in section 1.3.10 each stabilizer generator acts on a subset of physical qubits by Pauli operators σ_z or σ_x and it leaves the rest of the qubits untouched (Acts like an Identity operator on them).

[20] The reader should remember that in this context by error chain we mean a chain of Pauli operators σ_x, σ_y and σ_z on the data qubits.

Figure 1.7: The gray edges are the physical data qubits on the lattice. Each black solid line on an edge shows one Pauli Z operator (Phase flip) on that specific data qubit. When these errors occur on adjacent qubits they make longer chains of errors.

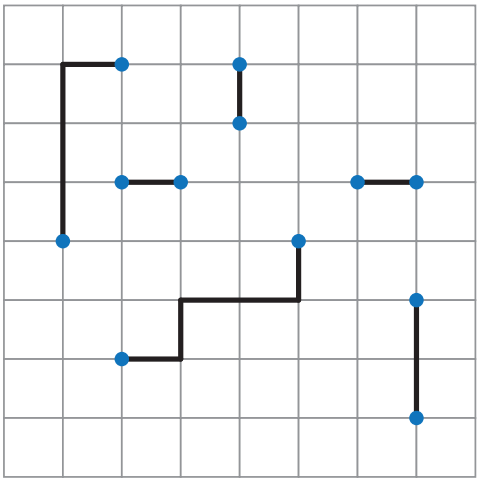


Figure 1.8: How error chains commute with all stabilizers except at their boundaries. The black lines show Pauli Z errors, the blue crosses show X stabilizers at various sites on the lattice and the green and red lines show where the stabilizer commutes and anticommutes with errors respectively. The commutation relations depends on the parity of qubits shared between error chains and stabilizers.

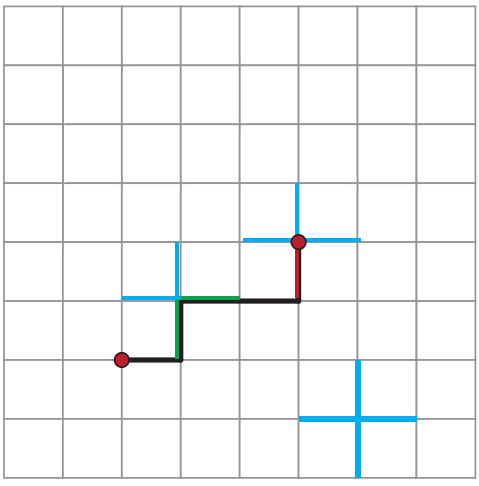
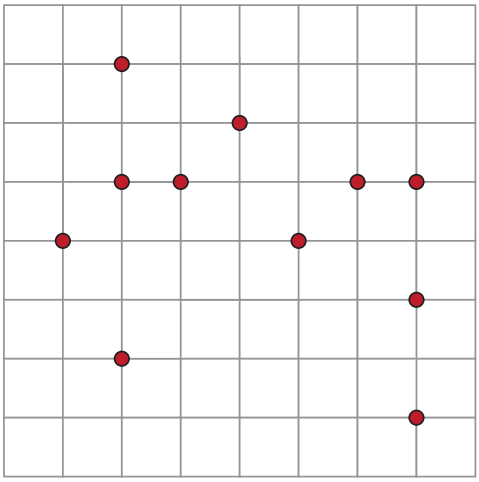


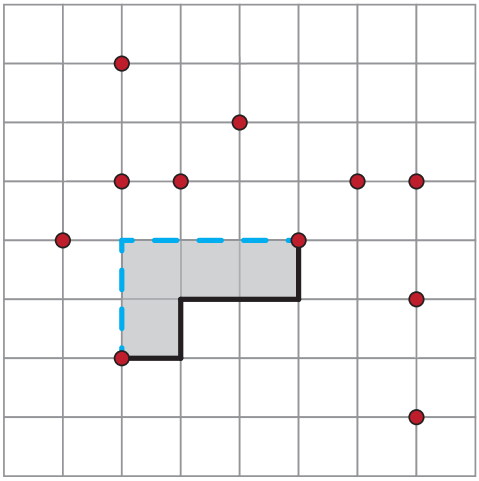
Figure 1.9: Stabilizer syndromes only reveal information about the boundaries of the chains of errors.



the trivial cycles of Pauli operators act trivially on the code space. In other words a trivial cycle of Pauli operators is a multiplication of stabilizer generators which fixes the encoded state. Thus it does not matter where exactly the error chain is. We can guess what the error chain was and connect the boundaries by a new chain called the *recovery chain* which is just our guess of what the original chain was. Since the boundaries of the recovery chain and the original error chain are identical they will form a cycle of Pauli operators on the lattice. If this cycle is a trivial cycle then we have successfully eliminated that error chain. Figure 1.10 illustrates how the original unknown error chain and our guess for the recovery chain can make a trivial cycle eliminating the effect of the error chain.

Since the single qubit errors are random variables which are independent and has identical error probability distributions. The probability of an error chain decreases when it lengths become longer. Using a taylor expansion of the probability of error chains of a certain length and looking at the dominating factor in the large size limit we can see that this probability is in fact exponential in the length of the error and it's proportional to p_{error}^{l}. So the shortest possible recovery chain between two error boundaries has the most probability to be the actual error. On the other hand since we do not want to generate non-trivial cycles, we want to avoid

Figure 1.10: An example of an error chain with our guess for a recovery chain only having information about the boundaries of that error chain. The black line is the error chain and the blue dashed line is the recovery chain. The Pauli Z plaquette operators inside the loop are emphasized to show how a combination of stabilizers on the lattice can generate a trivial cycle.



long chains as much as we can. Therefore the criteria for the recovery path is to find the shortest path we want to connect. Then we apply the same Pauli operator to the data qubits on this path and by making the trivial cycle we get rid of that error.[21]

Now if we have a lot of these error chains on the surface, after measuring all stabilizers, we will see an even number of these defective boundaries which we call defects. We do not know which two defects are coming from the same chain because we don't know the chains. Now by globalizing the argument we had earlier, the best guess of the error correction process is to pair up these defects and put recovery chains between each pair in such a way that we globally minimize the length of these recovery paths. The task of finding a disjoint pairing of vertices in a graph is called perfect matching and if we want to do this task in a way

[21] A more sophisticated interpretation of error correction on surface codes is that boundaries of error chains can be thought as electric and magnetic quasi-particles which are created at a point and then drift apart by an error chain. The error correcting procedure must find a recovery chain between each pair of quasi-particles of the same type, to bring them back together and annihilate them [23].

that we minimize the length of the paths we chose to pair the vertices, it becomes an optimization problem called minimum weight perfect matching problem. The classical post processing part of the error correction is exactly where we solve this optimization problem to make the best guess pairing up the defects. This is the exact statement of the problem we want to address in this research. Chapter 3 of the thesis is a detailed discussion of an idea of how to improve the complexity of this post processing step in the error correction on surface codes.

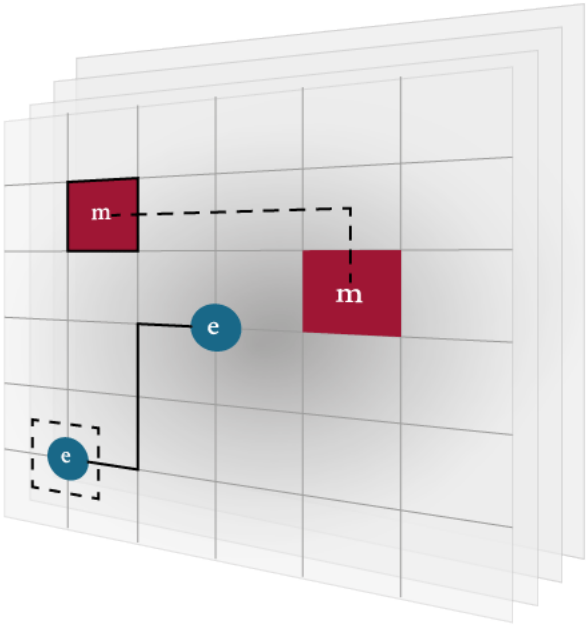
We will have a detailed algorithmic study of the error correction on surface codes in section 2.1 , and we will discuss how the probability of failure exponentially decays as we make the lattice larger if the error probability is below a certain critical point called the error threshold. The existence of this critical point relies on the probabilistic nature of the error correction. We expect that when the error probability becomes higher, longer chains become more probable and thus we will have more chance of making the wrong guess and generating a non-trivial cycle on code surface. This will result in the failure of the error correction. So intuitively we expect that when we increase the error probability above a certain value the error correction is more probable to fail. This threshold shows the robustness of an error correction algorithm and this is one factor to compare different approaches.

Theorem 3. *Success probability of error correction with surface codes can be arbitrarily close to 1 by making the lattice size large enough provided that the error probability is below a certain threshold.[7]*

This threshold is calculated numerically using Monte Carlo simulation. for a fixed error rate as we enlarge the lattice size the success probability increases and it approaches 1 in the large size limit.

The advantage of this syndrome measurement and classical post processing is that the post processing and recovery chain generation can be postponed as far as we collect syndrome results with a proper pace. The idea is that for example if a physical bit is flipped we can continue keeping the quantum memory and then when we need the quantum information reverse the bit flip and process the data. Likewise, we can keep the error syndromes at discrete times and then when we need to use the quantum information stored in our lattice we look at the error syndromes we have accumulated and do the post processing accordingly. Errors

Figure 1.11: Error chains and their boundaries as quasi particles. These chains can be recognized as logical operators on the surface code.



can accumulate through time, but because of this discussion we just had that is fine. However, we still need to collect syndrome measurements with a frequency that does not allow errors to accumulate and pass the threshold. We know that in the large size limit the error correction will most likely fail if the error probability is bigger than the threshold. This argument only applies when we want to have a quantum memory. [22] What if we want to do a certain computation on this quantum data by applying quantum gates? for a group of gates called the *Clifford gates* [23] it is still possible to trace the error syndromes and do the post processing later. However, This family of gates is not enough to have a universal gate set. In order to make universal quantum computation we need to add *non-Clifford* gates. The post processing can not be postponed to a point after a *non-Clifford* gate. So when we are doing actual quantum gates with our surface codes. we can only post pone the

[22] A quantum memory is a devise that can store and preserve some quantum information for a desired amount of time and then one can read off that information from it.

[23] The clifford group consists of the Hadamard, Controlled-z and the $\frac{\pi}{2}$ rotation gates.

postprocessing untill we hit a *non-clifford* quantum gate in the circuit of quantum gates we want to apply. Section 1.4.4 briefly explains how we construct a universal set of quantum gates for fault tolerant quantum computation with surface codes. For a more detailed study of how quantum gates can be done with surface codes we encourage the reader to look at [33].

1.4.3 Defects, Holes and Scaling

As we discussed earlier the topology and geometry of the lattice defines the shape of stabilizers and the error correction properties of this code. in the simple case of a torus topology where the horizontal boundaries of the surface are identified and the vertical boundaries are also identified we will have the toric code with two different types of non-trivial cycles which means we can encode two logical encoded quantum states in this architecture of $2L^2$ physical qubits. This simple version is called the toric code. The number of logical qubit states that we can encode is dependent on the topology of the surface. By changing this topology we can change the encoding properties of the code. In case of the toric topology the number of non-trivial cycles and hence the number of logical encoded states is two times the number of genus in the torus.

Defects are used to add non-trivial boundary conditions to the topology of the cluster. Defects can be carved out of the cluster by local Z-measurements of all qubits in the region of a defect.

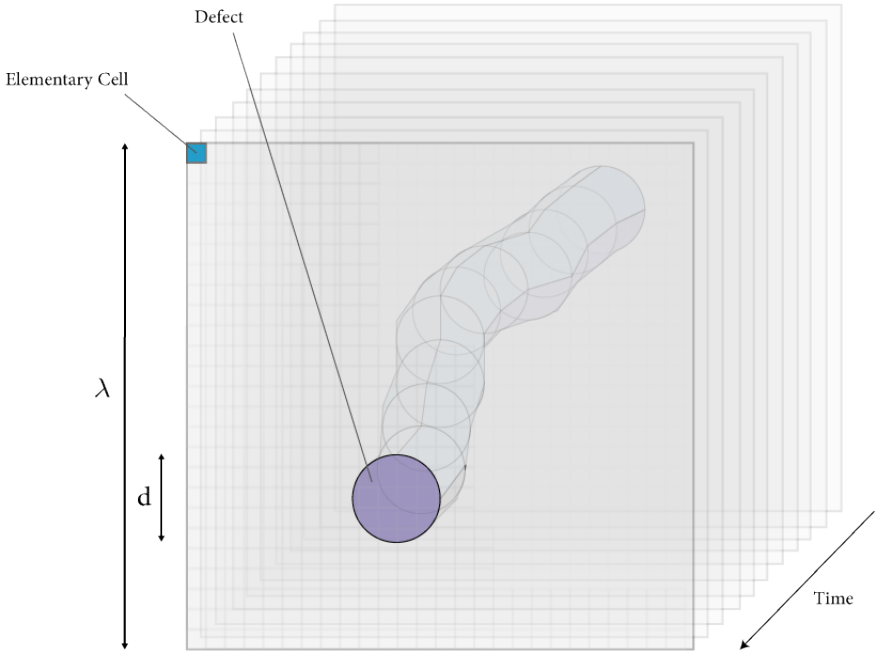
We can add encoded information by adding more defects but the problem is that short cycles wrapping around a defect are homologically nontrivial cycles which cannot be detected nor corrected, hence error correction fails in those cases. So we have to re-scale the surface to have thicker defects and hence bigger logical cells.

The logical cell is re-scaled from elementary cell of the lattice by a factor of λ and defects are thickened by a factor of d, figure 1.12.

1.4.4 Fault Tolerant Quantum Computation with Surface Codes

The surface code so far we were studying can act as a memory that can preserve the logical quantum state from certain quantum errors. However, it's proven that these codes can be used for quantum computation too Raussendorf and Harrington

Figure 1.12: Scaling factors on a lattice and time evolution of a moving defect. An elementary cell is shown for reference on the top left of the front lattice.



[33]. Initially the CNOT gate could be conducted with surface code using a 3D architecture of stacked surface layers. Later, Raussendorf showed that the CNOT gate can be carried out on the surface code by braid transformations on a single surface [33–35]. This is a great improvement. However, in order to have universal computation resource we need to be able to carry out a universal set of gates. All other complicated quantum computations can be carried out by composing them out of the universal gate set. If we add two single qubit rotations to our two qubit gate CNOT, this will make a universal gate:

CNOT Controlled NOT gate.

S Gate Rotation by $\frac{\pi}{4}$ along the Z axis

T Gate Rotation by $\frac{\pi}{8}$ along the Z axis

The two rotations are not easy to accomplish. They need special states prepared on ancilla qubits and injected to the surface in order to happen. These special states are called magic states and since they're hard to prepare it's easier to start with a noisy version and then distill a state with higher fidelity through an iterative process called magic state distillation [3]. The following ancilla states must be prepared with high fidelity in order to realize the rotation gates:

$$|Y\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{1.154}$$

$$|A\rangle = \frac{|0\rangle + e^{\frac{i\pi}{4}} |1\rangle}{\sqrt{2}} \tag{1.155}$$

Applying the magic state distillation each time will increase the fidelity in the output states and in this fashion it is expected that it will lower the overall operational cost of fault tolerance but that is generally not the case, since it also increases the number of distillation operations and noisy input states exponentially. Each level of magic state distillation has its own scaling factors.

The operational overhead for each gate type is a function of

1. Circuit size
2. Scaling factors
3. Circuit layout

As is shown in the figure 1.14 as the circuit becomes bigger, after a certain point it iss beneficial to add another level of magic state distillation rather than continuing with the same old noisy states. The bumps on certain curves in this figure is exactly when the overhead with an extra distillation level is less than the overhead of continuing with the same level.

Figure 1.13: Implementing a Controlled-NOT gate by twisting a pair of primal and dual holes representing the control and target qubits.

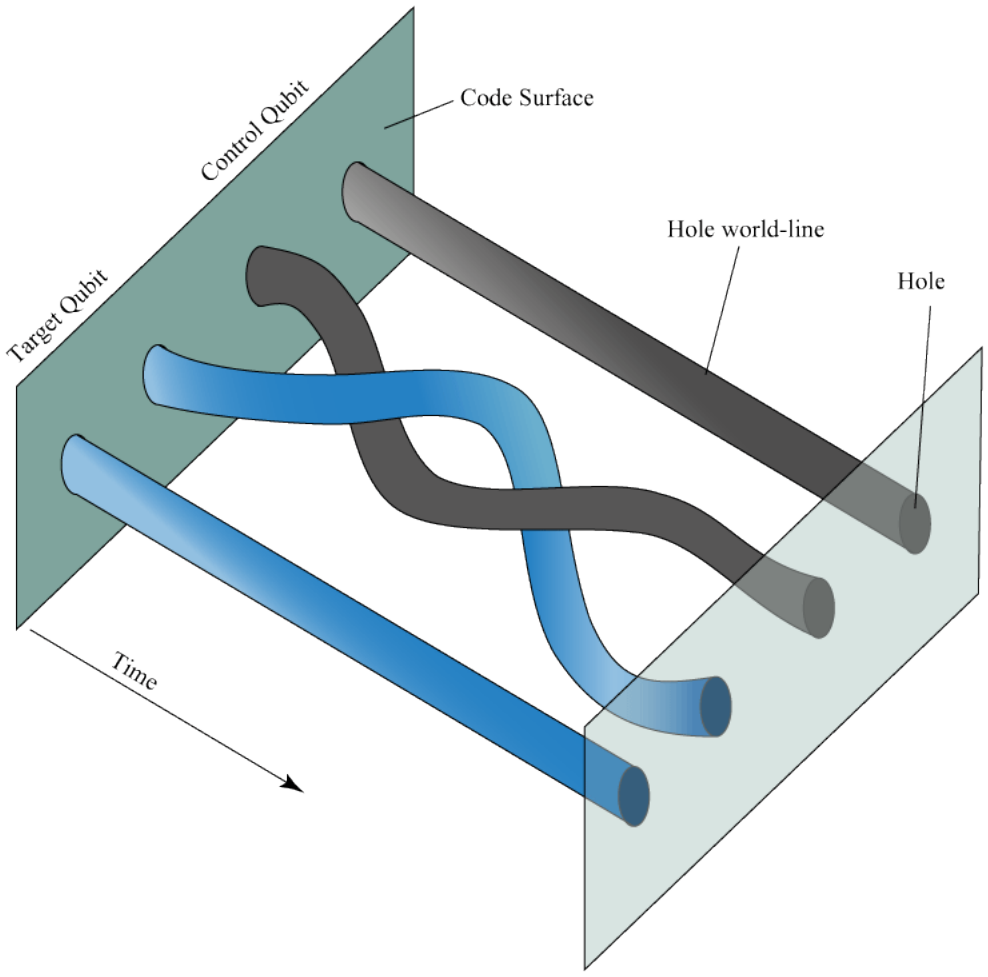
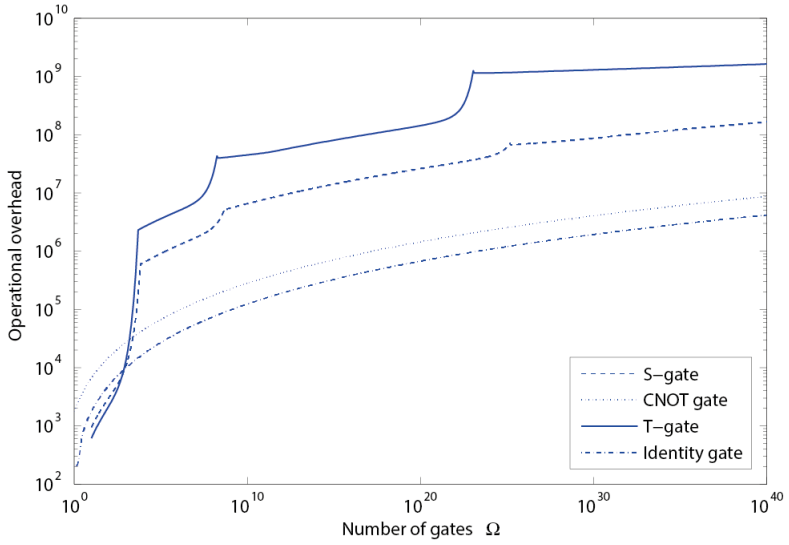


Figure 1.14: The operational overhead of building fault tolerant gates as a function of circuit size for various gates, at error probability $p_{error} = 10^{-4}$.



Chapter 2

Surface Code Simulation

It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.
— Richard P. Feynman (1918 - 1988)

In this chapter we try to develop a simulator engine for surface codes in order to carry out numerical analysis on the error correcting procedure. This chapter starts with a brief explanation of different modules in the error correcting procedure which both helps to understand the classical algorithm and also the topological robustness of surface codes against quantum errors.

2.1 Overview of the Error Correcting Algorithm

The toric code and the planar surface code have the same accuracy threshold [16], so I started my simulations with the toric code model which was simpler to work with. Since we are only considering the memory error I assumed that our measurements are perfect and thus used the Random-Bond Ising Model (RBIM) to compute the accuracy threshold. In order to build the algorithm, first I start with a simpler error channel which has only Z errors.

$$\rho \longrightarrow (1-p_z)\rho + p_z\sigma_z\rho\sigma_z^\dagger \tag{2.1}$$

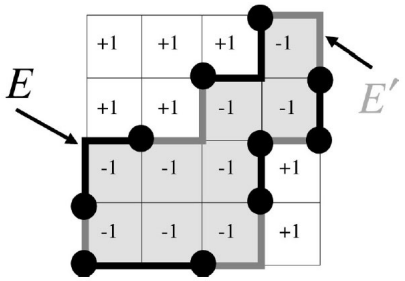
Generalizing the result to the real depolarizing channel in equation (2) is straight forward since the X errors are assumed to be independent from Z errors and they're

treated in the exactly same manner but on the dual lattice. The only important thing here is to note that Y errors introduce error correlations on the primal and dual lattices.

$$\rho \longrightarrow (1-p)\rho + \frac{p}{3}(\sigma_z \rho \sigma_z^\dagger + \sigma_x \rho \sigma_x^\dagger + \sigma_y \rho \sigma_y^\dagger) \tag{2.2}$$

A random error generator is used to generate an error sample on an $L \times L$ lattice. The Chain E is the set of links (qubits) on the lattice which suffered the Z error. As we know, the only important point is to know the boundary ∂E of the one-chain to locate the Ising vortices. We can imagine Ising vortices as defects which nucleates in pairs and then drift apart on the links of the one-chain. To recover from the errors, we have to generate another one-chain as a recovery chain bounded by the boundary of a one chain (the position of the defects). It's like bringing the two defects together again in order to re-annihilate.

Figure 2.1: The error chain E and the recovery chain E'. This image is taken from Dennis et al. [7]



So if we consider a complete graph with the Ising vortices as its vertices, then the problem of finding a chain to re-annihilate the defects is translated to finding the set of edges in the mentioned graph, in which every vertex has one and only one adjacent. Considering a complete graph there are lots of such sets called matchings of a graph. If we compute the physical distance between two defects in the lattice and assume it as the weight of the edge linking these two together, then our problem is to find the minimum weight matching of the mentioned graph. I used Edmonds perfect matching algorithm to find the minimum weight matching, which is our minimum cost recovery chain to overcome the errors.

The recovery is successful only if the chain $D = E + E'$ has a trivial homology

class. Otherwise, A homologically non-trivial cycle has happened on the lattice and hence the recovery fails. I repeat this procedure for many randomly generated error samples to estimate the failure probability with respect to the error rate.
For $p < p_{th}$ the failure probability approaches 0 exponentially as $L \to \infty$, a more detailed discussion in section **??** justifies the following relation between the residual topological error $e_{failure}$ and the linear size of the lattice size L:

$$e_{failure} \sim e^{-\kappa(p)L} \tag{2.3}$$

If we use Monte Carlo simulation to estimate the failure probability for different L, we expect to observe that for $p < p_{th}$ the failure probability decreases if L increases while for $p > p_{th}$ the failure probability grows with lattice size L. So if we have failure probability versus p for different L, There would be a unique point where all lines cross each other, which is the estimated value for threshold p_{th}.

2.1.1 Error Channel and Random Generator

In order to generate random errors on the lattice we used *boost* library which includes MT19937, a variant of the Mersene Twister, a famous random number generator with a period of $2^{19937} - 1 \approx 10^{6000}$.

2.1.2 Finding Ising Vorticies

If we turn the lattice into a giant graph with lattice sites as the vertices and the error chain as the edges of the graph then Ising vortices or defects will be the nodes with odd degree.

2.1.3 Edmonds Perfect Matching Algorithm

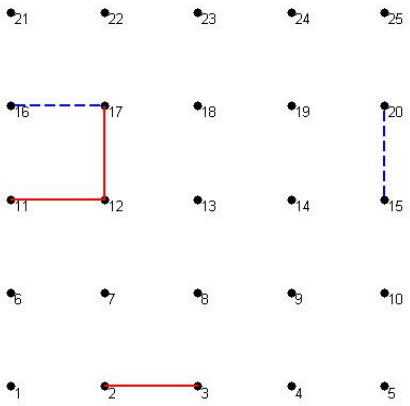
As I mentioned before, the problem of finding the recovery chain can be translated to the problem of finding the minimum weight matching on a complete graph showing defects as its vertices and E chain links as its edges. Here the weight of a chain is the sum of the vertical and horizontal distance of its boundary (defect pair) in the lattice.
The most complex part of our simulation is the perfect matching algorithm with a

complexity of $\mathcal{O}(l^6)$ [12]. As was mentioned before, this complexity makes it hard for us to do numerical analysis for larger lattice sizes and smaller error probabilities. So we want to address this problem and by adding heuristics to the matching algorithm make the post processing error correction (decoding step) faster. The entire chapter 3 of this research will be dedicated to this new idea along with its numerical results. In the simulator we used an implementation of the Edmonds matching algorithm called Blossom [24].

2.1.4 Path Generator

After we found which sites are going to be linked with an E' chain, we have to generate paths on the lattice from each defect to its pair.

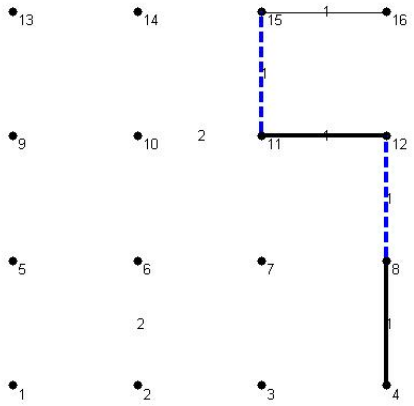
Figure 2.2: L=5 Lattice. Error chain is shown with red solid lines and the recovery chain is shown in blue stripped line.



2.1.5 Homologically Non-trivial Cycles

In order to see whether we can successfully preserve the lattice from memory errors we have to check if $D = E + E'$ chain builds a homologically non-trivial cycle. The existence of a non-trivial cycle is checked by a simple algorithm which looks for rows or columns of qubits in the lattice which includes an odd number of error chains.

Figure 2.3: $D = E + E'$ builds a homologically non trivial cycle in this lattice. The solid black lines are the Error chains E's and dotted blue lines are recovery chains E'. The vertices are the ancilla qubits on the sites of the lattice and edges (lines) are the data qubits affected by either the error or the recovery chains.



2.2 Monte Carlo Simulation

Putting all these blocks together we'll have a toric code failure probability simulator:

1. Generate an error sample on the lattice (E chain)
2. Locate defects (Ising Vortices)
3. Find the minimum-weight matching for defects (E' recovery chain)
4. Examine the homology class of $D = E + E'$ cycle
5. Declare a success or a failure

The following is a sample result of running the simulator for $L = 3, p = 0.1$:
Error sample matrix:

$$\tau_{m,n} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Location of defects (Ising Vortices):

$$\xi_{m,n} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

weighted adjacency matrix for a graph with defects as vertices:

$$G_{i,j} = \begin{pmatrix} 0 & 4 & 2 & 2 \\ 4 & 0 & 2 & 4 \\ 2 & 2 & 0 & 2 \\ 2 & 4 & 2 & 0 \end{pmatrix}$$

This matrix is fed into the Edmonds perfect matching algorithm to find the minimum cost matching between defects. The result is a matrix which clarifies which pair of defects should be matched:

$$M_{i,j} = \begin{pmatrix} 5 & 5 \\ 3 & 5 \\ 3 & 3 \\ 1 & 5 \end{pmatrix}$$

which says that for example the defect on $\xi_{5,5}$ must be matched to the defect on $\xi_{3,5}$ and so on ... The result is a lattice including both E chains and E' chains:

$$\chi_{m,n} = \begin{pmatrix} 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

then the whole lattice is translated to adjacency matrix of a giant graph with sites of the lattice as its vertices and the E or E' chains as the edges:

$$\chi_{m,n} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

This matrix is used to check the homology class of the cycle. There is a non-trivial cycle in this example and the simulator returns +1 as a failure:

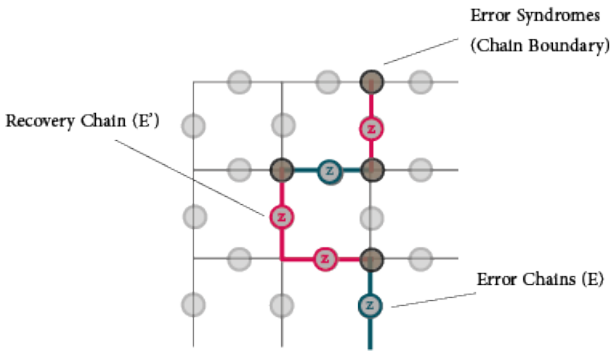
$$\text{Total Failure} = 1$$

A graph of the code is also provided in simulation; for clarification, a schematic sample of this output is shown in figure 2.4.

2.2.1 Numerical Results

The key point to find the p_{th} is to repeat the simulation for a lot of error samples and for different lattice sizes L. After error analysis we figured out that 10^5 error

Figure 2.4: Simulator illustrates the lattice graph including E and E' chains.



samples gives us reliable results. The detailed error analysis can be found in section 2.5.

The architecture in this numerical simulation is exactly that of [7] for the bit-flip error channel. Section 2.5 discuss the slight difference between this study and the literature. This can be immediately compared with that of [9] which is also introducing a new decoding algorithm with threshold 8.2% for bit-flip error channel below the 10.5% but with an advantage of operational complexity $L^2 log(L)$ compared to L^6 of Edmonds perfect matching algorithm.

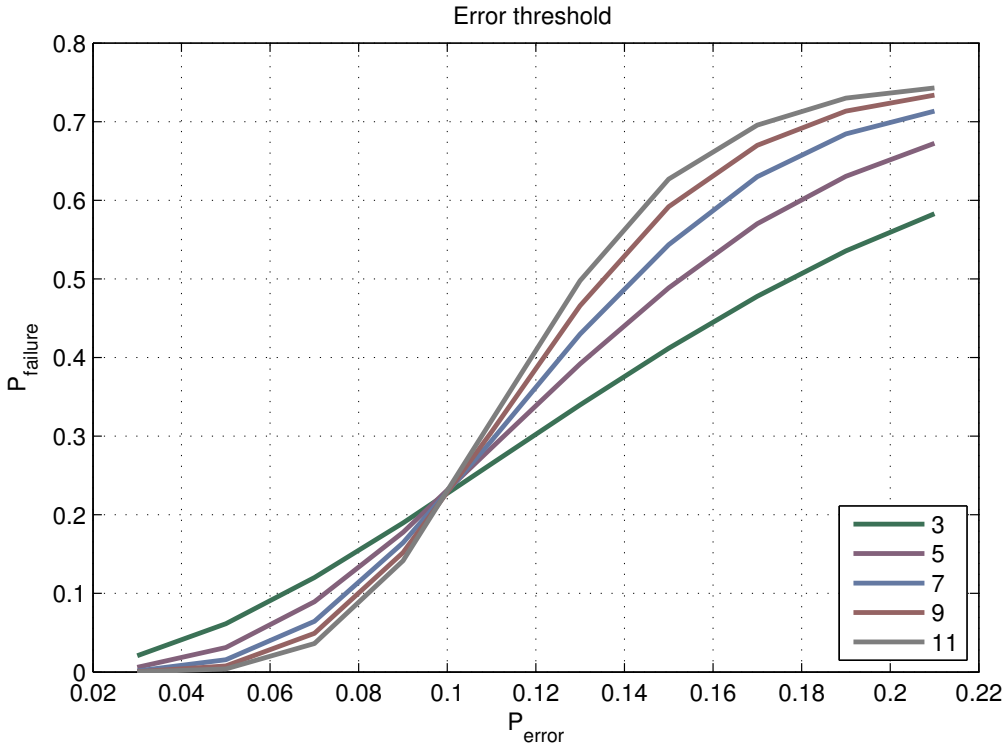
$$\boxed{\textit{From figure 2.5 :}\ p_{th} \approx 0.1 = \%10}$$

2.3 Larger Lattice Sizes

Although higher threshold is an advantage for a decoding algorithm, it's important to note that none of the quantum computations are going to happen with error probabilities close to threshold. On the other hand, in order to conduct a real world fault tolerant computation we will be working with lattices of size 100 to 1000 and probably bigger than that based on different parameters in the computation such as tolerable error, type of gates, number of gates , error level, etc. So what we actually need to do is to study the behavior of these codes for small error probabilities (where they really would operate) and for large lattice size limit.

Both of these studies are computationally unfeasible. The number of qubits grows quadraticaly with the lattice size and the perfect matching algorithm is

Figure 2.5: Error threshold averaging 10^5 error samples

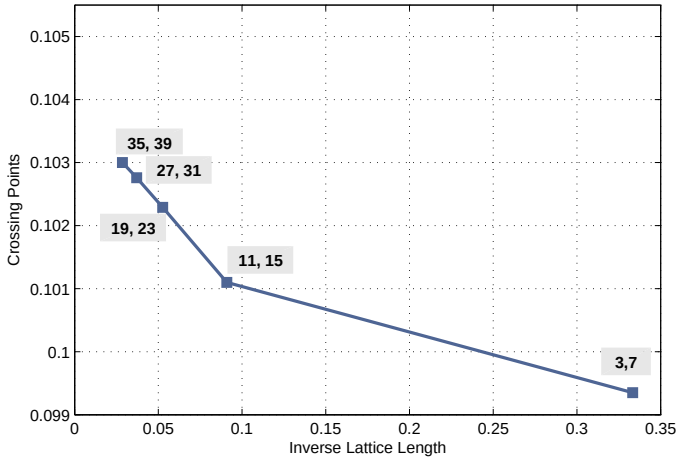


$\mathcal{O}(L^6)$ which really slows the process down as we go to larger lattice sizes. In the smaller errors case, we need more random samples in order to observe less probable error chains, a couple of orders of magnitude more random samples will also dramatically slow down the Monte Carlo simulation process. These two reasons make it challenging to gather high quality numerical results. We use several different methods and approaches to make the simulation faster and the general purpose of this research is also to design a matching algorithm which is less complex than the Edmonds matching algorithm.

2.4 Crossing Points

The simulation results for $L = 3,5,\ldots,11$ gives us a threshold of 10% while we found a 10.5% threshold in the literature [7] for the same error channel. The first step is to investigate the crossing points of curves for different lattice sizes. As you can see in figure 2.6, there is an asymptotic approach toward the 0.105 value for threshold mentioned in previous works, as we go to larger lattice sizes.

Figure 2.6: Investigating the crossing point with larger lattice sizes



The next important thing is to compare the results with the numerical error in the Monte Carlo simulation.

2.5 Error bars

After a random noise occurred on our lattice there might be only two different outcomes: either the topological memory can survive the errors and preserve the quantum information or it may fail to correct the errors and the information is lost. When there's only the possibility of a failure or a success we can model our system by a Bernoulli trial. If we repeat a Bernoulli trial for N_{max} times, then one can easily deduce that the Bernoulli distribution approaches a Gaussian distribution as N_{max} goes to infinity. Like any other random variable with a specific distribution we expect a mean and variance for this experiment. Our Monte Carlo simulations

in this research can be modeled as a chain of Bernoulli trials for N_{max} times, and the numerical error $E_{Numerical}$in the resulting data is the standard deviation σ of this Bernoulli distribution.

$$E_{Numerical} = \frac{\sigma}{\mu} \tag{2.4}$$

For a Bernoulli trial with failure probability $P_{failure}$ and N_{max} repetitions:

$$\mu = N_{max} \times P_{failure} \tag{2.5}$$

$$\sigma^2 = N_{max} \times P_{failure} \times (1 - P_{failure}) \tag{2.6}$$

Using these relations together:

$$E_{Numerical} = \frac{\sqrt{N_{max} \times P_{failure} \times (1 - P_{failure})}}{N_{max} \times P_{failure}} = \sqrt{\frac{(1 - P_{failure})}{N_{max} \times P_{failure}}} \tag{2.7}$$

Now we apply the numerical results at the $P_{error} = P_{threshold}$ in the relation above, the number of repetitions is 10^5 and the failure probability $P_{failure} = 0.22$ around the threshold point:

$$E_{Numerical} = \sqrt{\frac{(1 - 0.22)}{0.22 \times 10^5}} = 0.5954\% \tag{2.8}$$

As we can see the numerical error is the main reason behind the 0.5% between our results and the previous ones obtained from the literature [7, 35].

$$\boxed{\text{Numerical error for } 10^5 \text{ samples} \approx 0.6\%}$$

Chapter 3

Fast and Simple Error Correction Post Processing

Be Fearful When Others Are Greedy and Greedy When Others Are Fearful. — Warren Buffett

Although the complexity of the post processing algorithm for Fault Tolerant Quantum Error Correction (FTQEC) is polynomial in the linear size of the lattice, since this is just a step in a quantum computation process, it has to be comparably fast otherwise the speed up using a quantum processor won't be observed by the long delay due to the classical post processing algorithms. The other reason that we need a fairly fast and simple post processing step is that a lot of theoretical studies on these quantum architectures involves a lot of Monte Carlo numerical calculations which needs millions of runs of these algorithms with large lattice sizes. So the computational cost of these algorithms are very important and even a small speed up accumulates and shows up a great advantage in terms of time.

This is basically the motivation behind the study of post processing algorithms. In this following chapter we propose a very simple greedy heuristic algorithm which serves as the post processing engine. Using parallelising techniques it has a time complexity logarithmic in the lattice size with high threshold. The results can be compared to the re-normalization group approach of Duclos-Cianci and Poulin [9, 10]. Using the idea of concatenating codes they are able to get to a threshold

of almost 8% with a logarithmic time complexity in terms of the lattice size. Our work, however, results in a family of hybrid algorithms with an adjustable threshold varying from 8.6 to 10.5 % with a time complexity that can be asymptotically as low as that of Duclos-Cianci and Poulin [9]. The implementation of our algorithm is much easier due to the simple heuristic we used and it's expected to work more efficiently on low error regimes compared to [10]. Another advantage of our work is that it can make use of approximation algorithms to lower the operational complexity when the threshold is adjusted to be higher than 8.6%.

3.1 Minimum Weight Perfect Matching (MWPM) Algorithm

As we discussed earlier the error correction step on the lattice is reduced to pairing defects and then having proper operations on a path between them to annihilate those defects. Pairing defects has to happen in such a way that the paths between each pair is the best guess for the error chain that actually happened between them. Since the error probability distribution is i.i.d (independent and identically distributed) the probability of a path is inversely proportional to it's length. So our guess will have the highest probability to be the actually error chain occurred if we choose the shortest path possible between the two defective qubits. This means that defects should be paired in such a way that the distance between each two paired defects is a minimum.

This pairing problem is reduced exactly to a weighted perfect matching problem on a graph where it's vertices are the defects. There is an edge between two vertices (defects) if there is a path between them on the lattice. There might be several paths between two defects on the lattice but we are interested in the shortest path between nodes. Hence, The weight of each edge will be the lengths of the shortest path between those two defects.

Although a WPM always has an optimal answer on a complete graph, due to topological properties of the surface code, the decoding process is really robust according to this pairing process. As we mentioned earlier only nontrivial cycles will make the code fail so even if a pairing is not the optimal possible matching of defects it will still correctly compensate for errors if there is no nontrivial cycle,

and the error correction will succeed. This is exactly the robustness that we employ to build a sub-optimal matching algorithm without any dramatic reduction in the threshold.

3.2 A Simple Heuristic

Thus we can easily see that the whole post processing step is translated to a finding a minimum cost perfect matching on a graph where vertices are the defects and the weight of each edge between these vertices is the length of the shortest distance between those two defects. Such a minimum cost perfect matching will cover all vertices , so all defects will be annihilated and since it has a generic optimization of the cost (length of the paths) it always matches vertices in such a way that the global length of all paths is the minimum it can be. Literally it tries to match defects which are closer to each other to reduce the length of the paths.

But the resulting defect graph on which we want to apply the perfect matching algorithm has a very nice property that can be used as a heuristic added to our post processing algorithm to dramatically lower the complexity of matching step. Since the defects are on a lattice (a connected graph) there is at least one path between each random defect. So the graph that we build based on defects and paths between them will be a complete graph since as I mentioned there is at least one path between each two defects on a lattice.

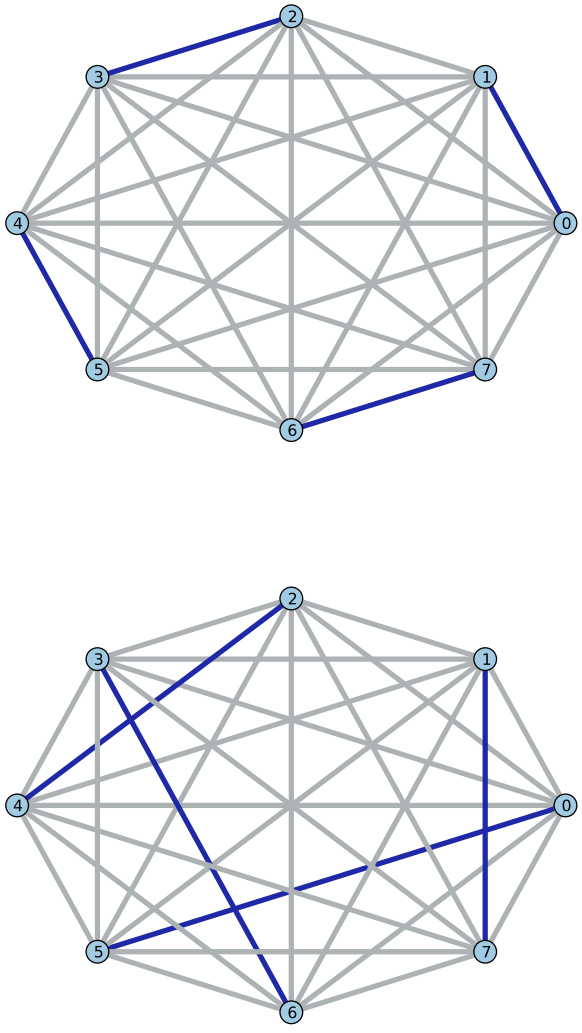
Finding a perfect matching on a complete graph with even number of vertices is simpler than finding a perfect matching on a random graph. In fact since there exists an edge between any two vertices on a complete graph , any random pairing of the vertices will be a perfect matching. So we almost have no difficulty finding a perfect matching on a complete graph.

Simple perfect matching algorithm on a complete graph is written in pseudo-code in 3.1.

However as we discussed earlier we are not just interested in a perfect matching-more precisely we are interested in finding a minimum cost perfect matching where the cost is the length of the paths between matched vertices.

This assumption can make the problem a lot more complex because now it's mixed with an optimization problem of minimizing the cost. However, we will use

Figure 3.1: Random matchings on a complete graph with 8 vertices



Algorithm 3.1 Simple random perfect matching algorithm on a complete graph.

Initialize G a complete graph of size n
$G_{vertices} \leftarrow$ vector of vertices in G
$G_{edges} \leftarrow$ vector of edges (vector of pairs of vertices)
while number of vertices in G is not zero **do**
    $x_1, x_2 \leftarrow$ random pair of vertices from $G_{vertices}$
    append $[x_1, x_2]$ to the matching list
    delete x_1 and x_2 from the $G_{vertices}$
    delete all edges adjacent to x_1 and x_2 except the one between them
end while

a simple heuristic to apply a very small modification to the simple algorithm we discussed above to change it to a minimum cost perfect matching algorithm.

By adding a greedy heuristic to the simple matching algorithm on a complete graph, the program tries to minimize the cost of matching. The heuristic is as follows: since we want to minimize the cost of the perfect matching, instead of choosing a random pair of vertices each time, we pick the pair with the lowest cost for the edge between them.

In terms of the error correction on the lattice of qubits, this heuristic means that when starting the error correction, look at all defects and calculates their mutual distances, then match the two defects which are closer to each other than any other pair. Apply the most trivial chain of Pauli operators between them and annihilate them. Then we can assume that these two defects have disappeared and go to the next pair with the shortest distance and repeat this procedure until all defects are paired up.

So the only thing we need to change in the previous trivial algorithm is to sort the list of edges according to their weights, this will help us find the lowest cost pair in each iteration.

3.3 Sub-optimality and Adjustment of Weights

As we discussed earlier, like other approaches to finding the simplest post processing algorithm, our proposed method is also a sub-optimal approach. All these sub-optimal approaches are then compared based on the highest threshold that they

Algorithm 3.2 Perfect matching algorithm on a complete graph with sorted weights.

Initialize G a complete graph of size n
$G_{vertices} \leftarrow$ vector of vertices in G
$G_{edges} \leftarrow$ vector of edges (vector of pairs of vertices)
$G_{edges} \leftarrow$ **Sort** G_{edges} in ascending order
while number of vertices in G is not zero **do**
    $[x_1, x_2] \leftarrow$ The left-most edge in G_{edges}
    append $[x_1, x_2]$ to the matching list
    delete x_1 and x_2 from the $G_{vertices}$
    delete all edges adjacent to x_1 and x_2 except the one between them
end while

can get and the distance of their threshold to the optimal matching threshold. So as it's expected there is a trade of between the complexity of the algorithm and how close it is to the the optimal threshold.

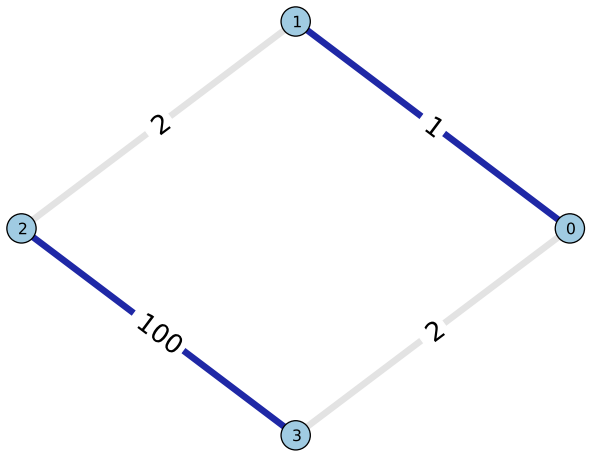
Although the sorting algorithm with all its simplicity gives us a relatively good threshold which is comparable to other approaches, but we can study why this simple algorithm is sub optimal and maybe manage to apply minor modifications to get a slightly more complex algorithm with a significant gain in terms of threshold.

Figure 3.2 shows a simple random example where sorting the list of weights of edges fails to give us the optimal matching. As we can see the sorted list of weights is [1,2,2,100]. Applying our simple greedy matching algorithm, the first edge to be picked up will be (0,1) with weight 1. After that we have to erase all other adjacent edges to vertices 0 and 1. So the only edge remained in the graph will be (2,3) with cost 100. The total cost of this matching is 101, but we can simply observe that if we pick edges (0,3) and (1,2) we'll cover all vertices of the graph in a matching with cost 4.

This example shows that optimizing the cost of the perfect matching is a global optimization problem. If we use divide and conquer methods to break it into smaller sub-problems over subsets of vertices, optimizing those sub-problems doesn't guarantee that the global problem will also be optimized.

We add a simple step to redefine the weights of each edge in the defect graph in order to partially tackle this problem. The idea behind this weight modification

Figure 3.2: A simple not realistic and extreme example of the situation when sorting fails in minimizing the matching cost.



is to consider the edges that we are left with after choosing a specific edge in the matching and removing its ends from the defect graph.

We iterate over all edges of the graph and we check what remains if this edge is chosen in the matching. Then we look at the remaining edges and add a penalty inversely proportional to the sum of their weights to the weight of that specific chosen edge. In this way if choosing an edge removes a lot of small weight edges it gets more penalty than an edge which removes a lot of higher weight edges when it's chosen in the matching.

3.4 Complexity of the Algorithm

The motivation behind this study was to design and develop a simpler and faster algorithm for the post processing step. Thus, it is really important to evaluate the complexity of the proposed algorithm. Going through the operations, one can see that the complexity is dominated by the sorting algorithm which is the heart of this heuristic.

Table 3.1: List of selected comparison-based sorting algorithms

Name	Best	Average	Worst	Memory
Quick sort	$n\log(n)$	$n\log(n)$	n^2	average: $\log(n)$, worst case: n
Merge sort	$n\log(n)$	$n\log(n)$	$n\log(n)$	worst case n
In-place Merge sort	-	-	$n(\log(n))^2$	1
Insertion sort	n	n^2	n^2	1
Bubble sort	n	n^2	n^2	1
Selection sort	n^2	n^2	n^2	1

Sorting is perhaps the most well-known and oldest computer algorithm. In fact almost all computer algorithm text books start with a chapter on sorting. These algorithms are often classified based on their computational complexity.

Different sorting algorithms are designed for different sorting scenarios. Operation complexity, use of memory, stability and etc., are the parameters which become important when choosing a proper sorting algorithm for a certain situation. Our goal is to reduce the complexity of the matching algorithm so the memory is not a problem here and we want to run the algorithm with the least complexity possible.

As we can observe having sorting as the bottleneck of our algorithm we introduce a lot of freedom in choosing our sorting scheme that best fits the needs and requirements of a specific application. But the advantage is not only in a wide variety of choices when it comes to sorting algorithms. In fact sorting algorithms are very well studied for paralleling and multi-threading, and there were great achievements in reducing their time complexity using paralleling ideas. In the next section we'll explain more how we will benefit from this advantage.

3.4.1 Parallel Processing

Parallel processing is the idea of dividing the tasks in an algorithm and running them at the same time step on many different processing devices. In this way the operational complexity of an algorithm is divided by the number of parallel devices which are running different parts of the code and then the overall time complexity of the parallel algorithm will be reduced proportional to the number of parallel

devices.

Among all different algorithms those which are designed based on a divide and conquer method have a great potential to be parallelized since they are already designed for divided work load. For example *merge sort* is a great example where different levels of multi-threading and paralleling can be applied to reduce its time complexity from $n\log(n)$ to $\log^3(n)$. There is a detailed study of the operational complexity of merge sort in section 2.3.1 of Cormen et al. [6]. on page 803 of the same reference, you can find a detailed implementation of a multi-threaded parallel merge sort and its complexity analysis.

There are other sophisticated parallel and multi-threaded implementations of sorting algorithms, even combining a couple of them as a hybrid sorting engine that show much better time complexity performance. David Powers used CRCW PRAM with n processors to develop an implementation of quick sort which works with a time complexity of $\log(n)$ [31] [32]. There are lots of other similar studies published in the last 20 years.

As a conclusion of this section, sorting algorithm are a well studied subject in computer science and there are lots of resources on how to optimize them in terms of their time complexity. Using multi-threading and paralleling techniques it's possible to get a random list of n elements sorted in time $\log(n)$.

3.4.2 Complexity Analysis

As discussed earlier the complexity of matching algorithm is reduced to the complexity of the sorting steps which is dependent on the size of the list we want to sort. This list consists of all possible edges in a complete graph where it's vertices are the defects on the two dimensional qubit lattice. If N is the number of edges:

$$N = \binom{v}{2} = \frac{v(v-1)}{2} \tag{3.1}$$

where v is the number of defects on the lattice. Now that we have the relation between elements in the sorting list and defects on the lattice, we have to count the number of defects. But as we discussed in chapter 1, defects are drawn from a set of independent and identically distributed (i.i.d) random variables. So we have to find the expected value (average) of the number of defects on a lattice.

Each qubit can have an error with probability p_{error} or stay as it is with probability $(1-p_{error})$. A lattice of linear size L has exactly $2L^2$ qubits on it, so if X is a random variable with its value equal to the total number of defects on an instance of a lattice with size L:

$$v = E[X] = p_{error} \times (\text{total number of qubits}) = 2p_{error}L^2 \tag{3.2}$$

Combining equation 3.1 and 3.2, we will get the final relation between the size of the sorted list and the inputs of the matching algorithm. We can easily write N as a function of input parameters L and p_{error}:

$$N(L, p_{error}) = \binom{2p_{error}L^2}{2} = \frac{2p_{error}L^2(2p_{error}L^2-1)}{2} \approx L^4 \tag{3.3}$$

Equation 3.3 suggests that the matching algorithm has to sort a list of size proportional to L^4. Using techniques that we discussed in previous sections the sorting can be then operated in $log(L)$ time. So the overall time complexity of this algorithm is logarithmic in L.

3.5 Numerical Results: Threshold

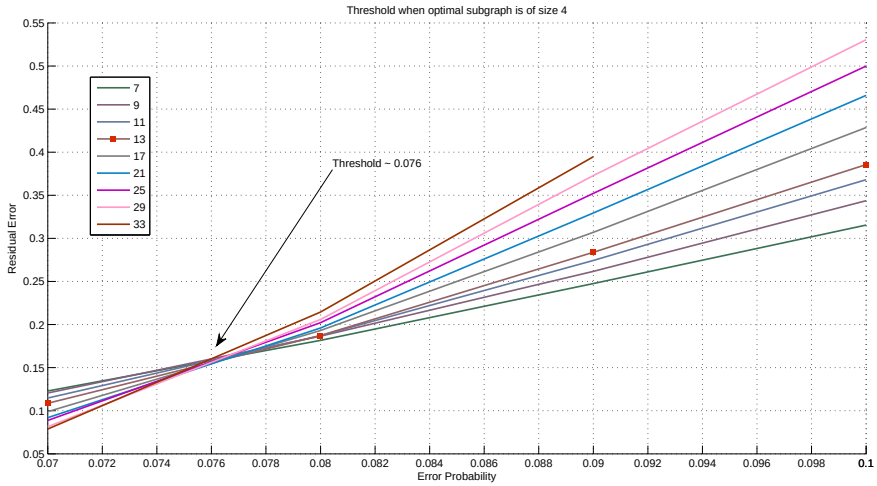
As we discussed in the introduction section, since the analytic study of the threshold is unfeasible it's common in the literature to use Monte Carlo methods to get statistical information about the the error profile and the threshold.

As we discussed in error bars analysis in chapter 1, the uncertainty in random sample results in Monte Carlo simulation dictates the minimum number of samples we need in order to have a meaningful distance between our data points. For the following study of threshold each data point is the average of 10^5 random samples having an uncertainty close to 0.05 for the $e_{failure}$.

The numerical results state a threshold of 7.6% for our pure sorting-based perfect matching algorithm. So far we have designed a very simple greedy heuristic which improves the time complexity of post processing from almost L^6 to $\log(L)$ with only decreasing the threshold from 10.5% to 7.6%. But this is not the end of story.

Although the performance of the proposed algorithm is close to the optimal

Figure 3.3: The threshold calculation for the refined sorting based matching algorithm.



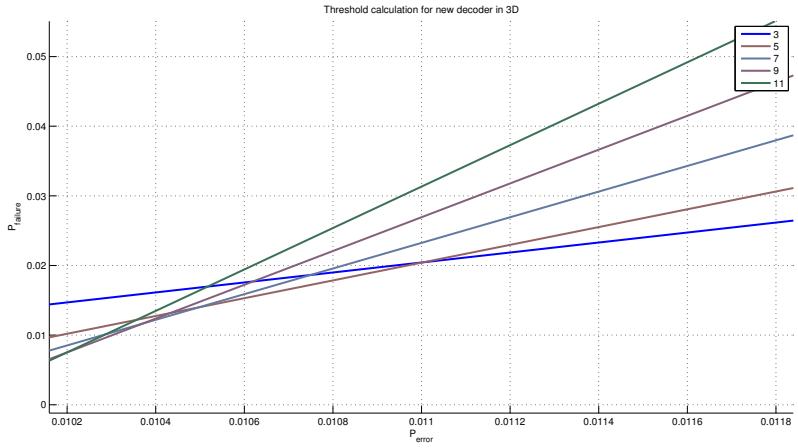
case with a lot of improvement in terms of time complexity and simplicity, we want to push this forward and implement a hybrid matching engine to reach for even closer to optimal thresholds while maintaining the simplicity of the algorithm. This idea is explained in detail in the next section.

3.5.1 3 Dimensional Topological Memory

So far we have assumed that our measurements are ideal, but in a noisy environment measurement has an error probability q_e. It's shown that the 2 dimensional surface code and faulty measurements can be modeled as a three dimensional architecture of qubits using surface codes plus the third dimension where measurements happen through time. The error correction can be mapped to a random plaquette $\mathbb{Z}_2$-gauge model in 3 dimensions, Dennis et al. [7].

As you can see in figure 3.4 our method has a threshold of 1% for the 3 dimensional surface code and the bit flip channel. The RPGM fault tolerance error threshold for a depolarizing channel is shown to be %3.2 [29], The minimum weight perfect matching decoding algorithm has a lower threshold of %2.9 [35]. Unfortunately there is no study of 3D surface code for re-normalization group decoder in Duclos-

Figure 3.4: Threshold calculations for the 3 dimensional topological quantum memory architecture using surface codes.



Cianci and Poulin [11] to compare with.

3.6 A Family of Hybrid Algorithms and Improved Threshold

Now that we have a very fast and straight forward greedy algorithm to do the matching let's take a step back and look at what's exactly happening in the real world when we are matching defects.

As we discussed 2.1, chains of errors will introduce defects at their boundaries. When we want to eliminate these defects, we need to guess what the error chain was and undo the Pauli operations due to error. But thanks to the stabilizers we can only see the boundaries of the chains (defects) not the exact error chain itself. Again the stabilizer code and the topology of the surface will come to our help. Since a trivial cycle of Pauli operators on the surface will not act as a logical operator and keeps the system in its encoded state, we do not need to guess the exact path of the error chains. Having the boundaries the only thing we need to do is to try making trivial cycles with the defects. The defects are already chained together by a chain of error, and so the only thing we need to do is to find a path between two defects and

apply the specific Pauli operator to it. This path in addition to the chain of errors will make a homologically trivial cycle on the code surface and eliminate those defects.

While doing this defect pairing and getting rid of them we have to be careful about one thing. This process will succeed if we manage to build trivial cycles with the error chains. A non-trivial cycle will apply a logical operation to the surface and change the logical data that we're storing on the surface.

Another way of looking at this is to study the length of the error chains. A non-trivial cycle on a torus for example has to wrap around the surface of the code, which means the non-trivial cycle has to be of length L. This is a very long chain of errors and its probability of occurrence will decrease when we go to larger lattice sizes.

So the idea is to avoid generating large chains because they have higher probability of becoming a non-trivial cycle. Since we do not know what exactly the error chains are, we try to choose the shortest path between the two for error correction, so that the error correcting path added to the real error chain will make the smallest possible cycle and in this way try to minimize the probability of generating a non-trivial cycle.

This argument has two important meanings for us:

1. First, we always have to pick the shortest path between two paired qubits to make sure we minimize the probability of building a non-trivial cycle along with the error chain between those two. This leads to the idea of minimum cost perfect matching. Minimizing the cost means always trying to pair the qubits in sush a way so as to have the minimum overall length of path between all qubits with each other and avoiding building long chains on the lattice. A more local, hence sub-optimal version of the same idea will be trying to do it in a greedy way and always tries to pair the two nearest defects, which we used to build our simple algorithm with it.

2. The second point is the building block of the idea behind this section. In addition to the first point what we can get from this is that the physical system doesn't care how you pair up the defects as far as you don't make a non-trivial cycle. On the other hand non-trivial cycles are chains of length L

which has less probability to occur and their probability is reduced as we go to larger lattice sizes. These two points together suggest that we only need to be careful when we are dealing with a long chain of errors. In other words, when defects are closer than $\frac{L}{2}$ to each other we do not need to be pairing them in the most optimal way. But when we are dealing with defects far from each other, if we pick a wrong pair then there is a high probability that we end up generating non-trivial cycle on the lattice.

These two points will lead us to the general idea behind a hybrid decoding engine for surface codes which has the highest threshold among all sub-optimal approaches, while maintaining the time complexity of operations logarithmic in size of the lattice.

3.6.1 The General Idea

Combining the two facts discussed in the previous section we use branch and cut techniques to build a hybrid matching algorithm of the defect graph. As we discussed before, when we face long error chains we really need to be careful in matching but when it comes to defects which are closer to each other even a simple sorting and matching the defects greedily won't harm the overall performance of the surface code. So we try a branch and cut approach to find a perfect matching on the complete defects graph.

The general idea is to apply the perfect matching in it's simplest way until we remain with defects which are all distanced from each other. At that stage we need to be careful because picking a sub-optimal matching might cause a long chain and consequently a non-trivial cycle on the surface. This is where we stop the greedy algorithm and we apply the optimal Edmonds matching algorithm to pair up the rest of the defects.

Please note that shorter chains are more probable to happen on a lattice and since they are short in comparison to the length of the lattice they have less probability to make a non-trivial cycle, even if they are matched sub-optimally. On the other hand longer chains are less probable and since they are long and close to the size of the lattice they have higher probability of making a non-trivial cycle. So the idea is to apply the sub-optimal and fast algorithm to the bigger portion of the

defects (which have short distance from each other) and then apply the optimal and slow algorithm to the remaining small number of defects which are far from each other. In this way we act fast where we can and act optimally and slow where we should be careful and in this way both maintain a high threshold (probability of successful decoding) and a low time complexity.

When we start our greedy sort based matching algorithm, we choose the shortest edges and we pair the defects at their ends. We have to note that we do the matching process step by step so each time one edge is picked, it's added to the matching list, it's vertices and all edges adjacent to those vertices are removed from the original graph and then the same procedure is repeated for the remaining graph. But there is an important fact we have to remember: each time we are done with one step, the remaining graph is still a complete graph. In fact it is a complete graph with 2 fewer vertices compared to the complete graph before that step.

Now we define a parameter called the Optimal Matching Subgraph relative Size (OMSS) as a barrier. We continue repeating the greedy algorithm until we match OMSS percent of the total number of defects on the lattice. Then we turn off the greedy algorithm and apply the optimal Edmonds perfect matching algorithm to the rest of the defects in order to minimize the risk of building non-trivial cycles between vertices which have a long distance.

The OMSS parameter defines what portion of the matching problem is solved with the simple greedy algorithm and what portion is solved with the slower yet optimal Edmonds matching. So this parameter controls a trade off between time complexity and threshold. A bigger OMSS means optimal Edmonds is applied to a bigger subgraph so the threshold will be closer to the optimal with the cost that the overall algorithm becomes slower. The two extreme cases are when $OMSS = 0\%$ and $OMSS = 100\%$. The first one means all defects are matched using the simple fast greedy algorithm. It gives us the sub-optimal 7.6% threshold and time complexity of order $log(L)$. The second case $OMSS = 100\%$ where all defects are matched using Edmonds algorithm gives us the optimal 10.5% threshold and well the high complexity of the Edmonds matching of order at least L^6. Varying the *OMSS* parameter between 0 and 100 % will result in a gradual increase of threshold from 7.6% to 10.5% with the cost of adding complexity.

The next step is to see the behavior of this trade off. We need to study the

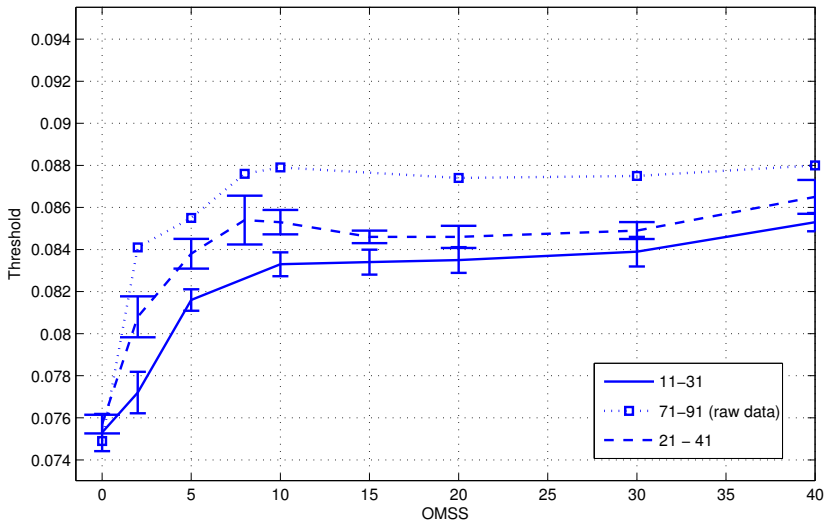
change in threshold with respect to increasing OMSS.

3.6.2 Numerical Results: Spectrum of Thresholds

We calculated the threshold for various OMSS around a fixed lattice size to see how the threshold is improved by increasing the parameter.

As we can see in figure 3.5, the threshold strictly increases non-linearly with *OMSS* increased. The other behavior that we observe is the overall shift in the threshold graphs when we move toward larger lattice size. It seems from figure 3.5 that the threshold is approaching a large-size limit for a fixed OMSS as we increase the size of the lattice. This observation is well studied in the next section.

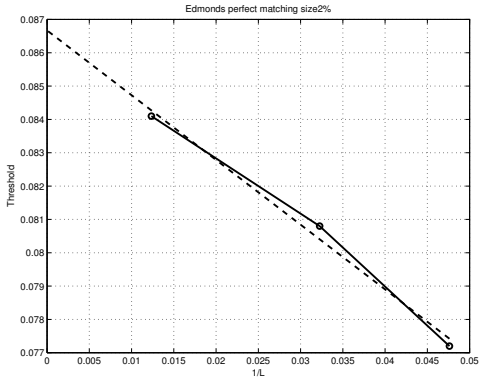
Figure 3.5: Threshold values vs. OMSS, for different lattice sizes.



3.6.3 Large Lattice Size Limit

In order to investigate the large lattice size limit behavior of the threshold with varying *OMSS* we try to plot fixed *OMSS* graphs for threshold versus $\frac{1}{L}$. This helps us to extrapolate the infinite size limit of the threshold for a fixed *OMSS*. Figure 3.6 shows a sample of the extrapolation method to obtain the numerical

Figure 3.6: Extrapolating the large size limit thresholds for a 2% portion of the Edmonds matching algorithm.



results for a fixed $OMSS = 2\%$. We get an extreme extrapolated threshold for each fixed *OMSS*.

Putting all large size limit thresholds we got via extrapolation and drawing them versus their *OMSS* we will get the large size limit behavior of the threshold by varying the size of the optimal matching sub-graph. Figure 3.7 shows this large size limit behavior. The large size limit graph has two important parts. The first rapid step-like jump from 7.6% to 8.6% threshold at the beginning and then the parabolic behavior from 8.6% to 10.5% threshold.

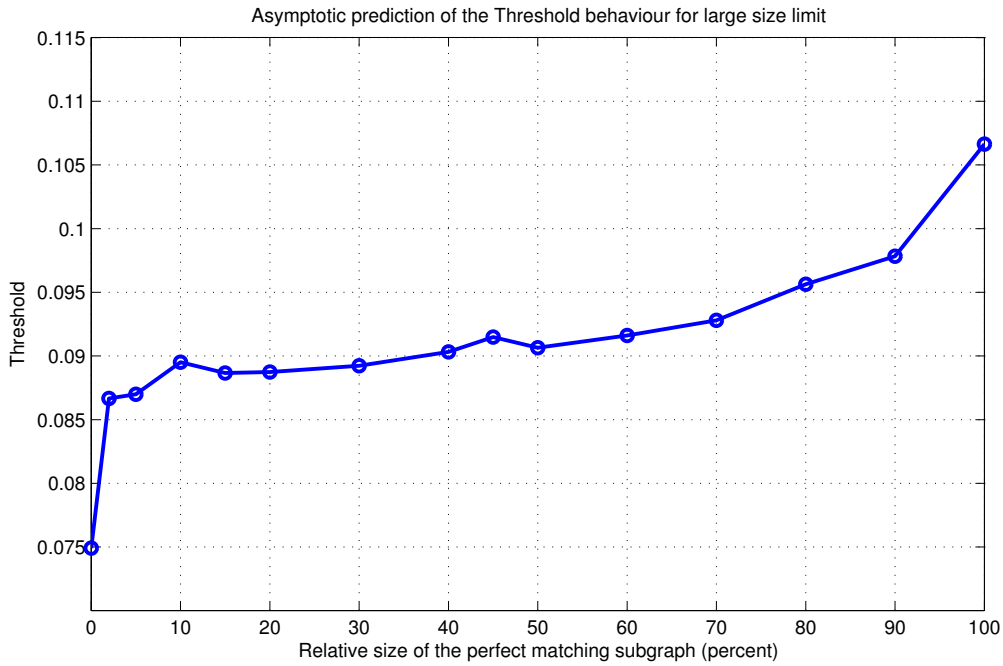
The second part of the figure with parabolic behavior suggests the non-linear increase in threshold when we change *OMSS*. Although this is interesting enough to understand that the threshold grows faster than *OMSS*, but the most interesting part of the graph is the step like jump at the beginning.

In order to study the jump, let's look at the complexity more carefully. *OMSS* is a constant multiplied to the total number of defects. So in the large size limit when we are calculating the order of complexity it doesn't really matter whether we are applying Edmonds matching to the whole system or only a fraction of it i. e. $O((OMSS \times L)^6) = O((c.L)^6) = O(L^6)$, with c being a constant. Still the Edmonds matching will dominate the complexity and hence the overall time complexity of the algorithm will be as bad as the Edmonds matching itself. This argument shows that if *OMSS* is a linear function of the lattice size then the time complexity will

not dramatically change from that of the Edmonds matching. But the good result is that the threshold increases non-linearly when we change *OMSS* linearly and the most non-linearity is observed in the beginning of the figure. We can conclude from the jump in figure 3.7 that *OMSS* does not need to be a linear function of the lattice size for us to get to a threshold of 8.6%.

Since the jump suggests that as far as *OMSS* is non-zero, in the large size limit we will get to 8.6% threshold, we can choose *OMSS* as low as possible so that the complexity of that instance of algorithm will be dominated by the sorting algorithm not the Edmonds perfect matching. Now we need to study how the threshold increases with increasing the lattice size to prove the conjecture.

Figure 3.7: Asymptotic large size limit curve



In conclusion, we have designed an algorithm that has a time complexity of order $log(L)$ and a large size limit threshold of 8.6% for a bit-flip error channel on a lattice of qubits. This result can be compared to that of [9] and [10].

The only thing that we have to note before we start using this is that the hy-

brid algorithm makes sense where the size of the remaining complete graph for Edmonds matching is at least 4 because the lower complete graph will be of size 2 which has only one possible perfect matching. So different algorithms will have the same result. This leads to the following calculation which suggests a lower bound on the size of the large lattice:

$$OMSS \times (2p_{error}L^2) \geq 4 \tag{3.4}$$

$$L_{large} \geq \lceil \sqrt{\frac{2}{OMSS \times p_{error}}} \rceil \tag{3.5}$$

3.6.4 The Threshold Jump

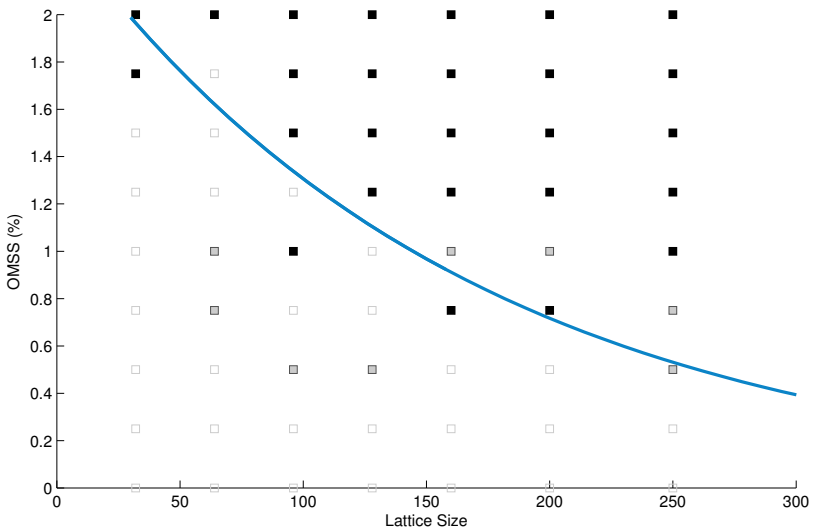
As it is illustrated in figure 3.7, there is a very steep increase in threshold when we go to OMSS greater than zero. It intuitively suggests that by turning the Edmonds perfect matching on over a very small defect subgraph we can have a significant increase in the lattice size. Now we need to study the size of this small subgraph as we increase the lattice size. We conjecture that by increasing the lattice size this small subgraph size decays exponentially and hence in the large size limit, the sorting-based matching is dominating the complexity of the whole algorithm, not the Edmonds matching.

Figure 3.8 shows the way we studied our conjecture. We fixed the error probability right in the middle of the steep jump at $p_{error} = 8.1\%$, then observed how the success probability of the code changes with the lattice size as we go from $OMSS = 0\%$ to $OMSS = 2\%$ where the jump ends. For the configurations where $p_{error} = 8.1\%$ is above their threshold we expect to get a success probability close to .5 and for the cases where $p_{error} = 8.1\%$ is below their threshold we expect to get a success probability close to 1. The exponential decay in the transition from below to above threshold in figure 3.8 proves our conjecture. However, we need to go to larger lattice sizes in order to have a better estimation of the curve fitted on the transition line.

This basically means that, at a large size limit the OMSS needed to get to a threshold of 8.6% decays exponentially, thus the complexity of the algorithm is

dominated by the sorting-heuristic.

Figure 3.8: The success probability of surface code error correction is classified in 3 groups, close to 1 (black), which happens when we are below threshold, close to .5 (white) when we are above the threshold and close to .75 (gray) when we are really close to the threshold itself. Data is gathered for a fixed error probability of $p_{error} = 8.1\%$



3.7 Approximations to WPM Algorithms and Improved Complexity

As discussed in [12] matching is one of those applied problems for which there exists a polynomial time solution but all trivial algorithms will take exponential time to solve it. One of the steps that computer scientists take to reduce the computational complexity of hard optimization problems is to try approximation methods. Approximations become very important when you can tolerate an answer close enough to the optimal answer in order to get a gain in terms of speed. Sometimes an exact algorithm to solve a problem cannot be parallelized while an approximation of it will be easily parallelised and saves a lot of time. This is another

motivation behind approximation algorithms.

In our case, through various experiments done in this study and other studies in the literature, it's easy to see that the fault tolerant quantum computation with cluster states is robust to the change of decoding algorithm and even the simplest heuristic will maintain a highly comparable threshold to that of the optimal case. This suggests that maybe there is room for improving the time complexity of decoding algorithm using the approximation methods to the Edmonds MWPM.

A super fast approximation to the matching algorithm is introduced in Holloway et al. [22] for minimum weight perfect matching in a complete graph satisfying the triangle inequality between its weights. This algorithm is highly parallelisable and has a time complexity of $O(\log^3(n))$ where n is the number of vertices in the complete graph. With our complexity calculations in part 3.4.2 this means a time complexity of $O(\log^3(L))$ where L is the linear size of the lattice.

As a future work, it would be great to run the Monte Carlo simulations with an implementation of Holloway's Algorithm and see the behavior of the threshold. But what we can conjecture is that the threshold is definitely bigger than 8.6% because this algorithm is supposed to act at least as well as the simple sorting heuristic. The result will be a poly-logarithmic algorithm with a threshold even higher than 8.6%.

Chapter 4

Conclusions

To be an error and to be cast out is a part of God's design.
— William Blake (1757 - 1827)

In summary we used the following ideas and heuristics to improve the post processing algorithm for FTQEC using surface codes:

Complete Graph The graph which is made of defects as its vertices and edges between them weighted by the length of the shortest path between two defects forms a complete graph. Finding a perfect matching on a complete graph is easier than a general random graph because all vertices are connected.

Sorting and approximating Because of the above, finding a minimum weight perfect matching on a complete graph can be approximated by heuristics as simple as sorting all the edges and always picking the shortest one.

Parallelising These approximation algorithms are highly parallelisable which can help us get a much lower time complexity.

Hybrid Algorithms Because of the robustness of the lattice against trivial cycles we only need to be careful when we are decoding long range defects. So we can apply our approximation to the bigger chunk of defects which are numerically proven to be close to each other and then apply the slow optimal algorithm to the remaining long range defects which are again proven

numerically to form a small portion of the defects, particularly for the large lattice sizes.

Using all these ideas we proposed a family of decoding algorithms with a sliding parameter *OMSS* which can give you any threshold between 8.6% to the optimal 10.5% by adding more complexity. The complexity of getting the 8.6% threshold is $\log(L)$ for large lattice sizes. Please note that the same idea of hybrid algorithms can be used for other approximation algorithms like [10] or it can be implemented with [22] to both improve complexity and threshold.

The most important fact about our proposed approach is its compatibility with the nature of fault tolerant quantum computation with surface codes. Although threshold is a parameter to compare different methods, but the surface codes are not expected to operate anywhere close to the threshold. In a real scenario the surface code will run on a large lattice and for very small error probabilities. Our heuristic fails where long and short chains are close to each other on the surface. However, in a lower-error regime we expect most of the error chains to be very short (almost 1 or 2 errors) and distanced from each other (almost no long chains and well-distanced). Thanks to this and topological robustness of the surface codes, our sorting-based matching is expected to work really well even without hybrid or approximation methods considered. In fact, this is part of the study we want to propose as a very important future work. we can extend and continue this research in certain directions including but not limited to the following:

- Simulate and compare the behavior of the proposed algorithm in low-error regimes, especially compare the quantum resources needed for each of the different error correcting methods.
- Develop the complete graph MWPM approximation algorithm to measure the potential improvement in threshold and complexity.
- Reduce the operational complexity of the algorithm by doing the error matching on the lattice data structure instead of extracting a new graph out of it.

Bibliography

[1] D. Aharonov and M. Ben-or. Fault-tolerant quantum computation with constant error rate, 1999. → pages 1, 15

[2] S. Boixo, T. F. Ronnow, S. V. Isakov, Z. Wang, D. Wecker, D. A. Lidar, J. M. Martinis, and M. Troyer. Evidence for quantum annealing with more than one hundred qubits. *Nat Phys*, 10:218–224, 2014. ISSN 1745-2473. doi:10.1038/nphys2900. → pages 1, 4

[3] S. Bravyi and A. Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Phys. Rev. A*, 71:022316, Feb 2005. doi:10.1103/PhysRevA.71.022316. URL http://link.aps.org/doi/10.1103/PhysRevA.71.022316. → pages 63

[4] S. B. Bravyi and A. Y. Kitaev. Quantum codes on a lattice with boundary. 1998. → pages 49

[5] H. J. Briegel, D. E. Browne, W. Dur, R. Raussendorf, and M. Van den Nest. Measurement-based quantum computation. *Nat Phys*, 2009. → pages 12

[6] T. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3 edition, 2009. ISBN 9780262533058. → pages 85

[7] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002. doi:http://dx.doi.org/10.1063/1.1499754. URL http://scitation.aip.org/content/aip/journal/jmp/43/9/10.1063/1.1499754. → pages x, 49, 59, 67, 73, 75, 76, 87

[8] S. J. Devitt, W. J. Munro, and K. Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013. URL http://stacks.iop.org/0034-4885/76/i=7/a=076001. → pages 25, 34

[9] G. Duclos-Cianci and D. Poulin. Fast decoders for topological quantum codes. *PRL*, 104, 2010. → pages ii, 2, 73, 77, 78, 94

[10] G. Duclos-Cianci and D. Poulin. A renormalization group decoding algorithm for topological quantum codes. In *ITW*, 2010. → pages ii, 77, 78, 94, 99

[11] G. Duclos-Cianci and D. Poulin. Fault-tolerant renormalization group decoder for abelian topological codes. In *QIC*, volume 14, pages 721–740, 2014. → pages 2, 88

[12] J. Edmonds. Paths, trees, and flowers. *CJM*, 17, 1965. → pages 2, 69, 96

[13] E. Farhi, J. Goldstone, S. Gutmann, and S. Michael. Quantum computation by adiabatic evolution. 2000. URL arXiv:quant-ph/0001106v1. → pages 13

[14] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda. A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292(5516):472–475, 2001. doi:10.1126/science.1057726. URL http://www.sciencemag.org/content/292/5516/472.abstract. → pages 13

[15] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982. ISSN 0020-7748. doi:10.1007/BF02650179. URL http://dx.doi.org/10.1007/BF02650179. → pages 13

[16] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012. doi:10.1103/PhysRevA.86.032324. URL http://link.aps.org/doi/10.1103/PhysRevA.86.032324. → pages 7, 49, 51, 66

[17] D. S. Garey, Michael R.and Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0716710455. → pages 3, 4

[18] D. Gottesman. *Stabilizer codes and quantum error correction*. PhD thesis, Caltech, 1997. → pages 46, 47, 49

[19] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM. ISBN 0-89791-785-5. doi:10.1145/237814.237866. URL http://doi.acm.org/10.1145/237814.237866. → pages 1, 4

[20] R. Harris, J. Johansson, A. J. Berkley, M. W. Johnson, T. Lanting, S. Han, P. Bunyk, E. Ladizinsky, T. Oh, I. Perminov, E. Tolkacheva, S. Uchaikin, E. M. Chapple, C. Enderud, C. Rich, M. Thom, J. Wang, B. Wilson, and G. Rose. Experimental demonstration of a robust and scalable flux qubit. *Phys. Rev. B*, 81:134510, Apr 2010. doi:10.1103/PhysRevB.81.134510. URL http://link.aps.org/doi/10.1103/PhysRevB.81.134510. → pages 1, 5

[21] W. K. Hensinger, S. Olmschenk, D. Stick, D. Hucul, M. Yeo, M. Acton, L. Deslauriers, C. Monroe, and J. Rabchuk. T-junction ion trap array for two-dimensional ion shuttling, storage, and manipulation. *Applied Physics Letters*, 88(3):034101, 2006. doi:http://dx.doi.org/10.1063/1.2164910. URL http://scitation.aip.org/content/aip/journal/apl/88/3/10.1063/1.2164910. → pages 1, 5

[22] N. W. Holloway, S. Ravindran, and A. M. Gibbons. Approximating minimum weight perfect matchings for complete graphs satisfying the triangle inequality. In *Graph-Theoretic Concepts in Computer Science*, volume 790 of *Lecture Notes in Computer Science*, pages 11–20, 1994. → pages 97, 99

[23] A. Y. Kitaev. Fault tolerant quantum computation by anyons. *Annals Phys.*, 303:2–30, 2003. doi:10.1016/S0003-4916(02)00018-0. → pages 58

[24] V. Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1): 43–67, 2009. ISSN 1867-2949. doi:10.1007/s12532-009-0002-8. URL http://dx.doi.org/10.1007/s12532-009-0002-8. → pages 2, 69

[25] A. Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2 (5), 2014. ISSN 2296-424X. doi:10.3389/fphy.2014.00005. → pages 1, 4

[26] O. Mandel, M. Greiner, A. Widera, T. Rom, T. W. Hänsch, and I. Bloch. Coherent transport of neutral atoms in spin-dependent optical lattice potentials. *Phys. Rev. Lett.*, 91:010407, Jul 2003. doi:10.1103/PhysRevLett.91.010407. URL http://link.aps.org/doi/10.1103/PhysRevLett.91.010407. → pages 1, 5

[27] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'Brien. Experimental realization of shor's quantum factoring algorithm using qubit recycling. *Nat Photon*, 6:773–776, 2012. doi:10.1038/nphoton.2012.259. → pages 1, 15

[28] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. The Cambridge University Press, 1 edition, 2000. ISBN 9780521635035. → pages viii, 10, 23, 24, 25, 34, 38, 44

[29] T. Ohno, G. Arakawa, I. Ichinose, and T. Matsui. Phase structure of the random-plaquette Z_2 gauge model: accuracy threshold for a toric quantum memory. *Nuclear Physics B*, 697:462–480, Oct. 2004. doi:10.1016/j.nuclphysb.2004.07.003. → pages 87

[30] C. Pomerance. A tale of two sieves. *Notices Amer. Math. Soc*, 43: 1473–1485, 1996. → pages 14

[31] D. M. W. Powers. Parallelized quicksort and radixsort with optimal speedup. In *Proceedings of International Conference on Parallel Computing Technologies*, 1991. → pages 85

[32] D. M. W. Powers. Parallel unification: Practical complexity. In *Australasian Computer Architecture Workshop, Flinders University*, 1995. → pages 85

[33] R. Raussendorf and J. Harrington. Fault-tolerant quantum computation with high threshold in two dimensions. *Phys. Rev. Lett.*, 98:190504, May 2007. doi:10.1103/PhysRevLett.98.190504. URL http://link.aps.org/doi/10.1103/PhysRevLett.98.190504. → pages 16, 49, 61, 62

[34] R. Raussendorf, J. Harrington, and K. Goyal. A fault-tolerant one-way quantum computer. *Annals of Physics*, 321(9):2242 – 2270, 2006. ISSN 0003-4916. doi:http://dx.doi.org/10.1016/j.aop.2006.01.012. URL http://www.sciencedirect.com/science/article/pii/S0003491606000236. → pages

[35] R. Raussendorf, J. Harrington, and K. Goyal. Topological fault-tolerance in cluster state quantum computation. *New Journal of Physics*, 9(6):199, 2007. URL http://stacks.iop.org/1367-2630/9/i=6/a=199. → pages 49, 62, 76, 87

[36] P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134, Nov 1994. doi:10.1109/SFCS.1994.365700. → pages 1, 4, 14

[37] K. M. Svore, D. P. Divincenzo, and B. M. Terhal. Noise threshold for a fault-tolerant two-dimensional lattice architecture. *Quantum Info. Comput.*, 7(4):297–318, May 2007. ISSN 1533-7146. URL http://dl.acm.org/citation.cfm?id=2011725.2011727. → pages 51

[38] T. Tanamoto, Y.-x. Liu, S. Fujita, X. Hu, and F. Nori. Producing cluster states in charge qubits and flux qubits. *Phys. Rev. Lett.*, 97:230501, Dec 2006. doi:10.1103/PhysRevLett.97.230501. URL http://link.aps.org/doi/10.1103/PhysRevLett.97.230501. → pages 5

[39] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936. doi:10.1112/plms/s2-42.1.230. → pages 13