Explorations in Decoding of 2D Topological Codes

A Project Report

submitted by

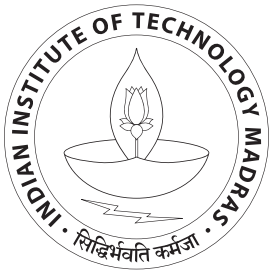
KULKARNI AMIT ANIL

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY
(ELECTRICAL ENGINEERING)

AND

MASTER OF TECHNOLOGY
(COMMUNICATION SYSTEMS)



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

JUNE 2016

THESIS CERTIFICATE

This is to certify that the thesis entitled **Explorations in Decoding of 2D Topological Codes**, submitted by **KULKARNI AMIT ANIL**, to the Indian Institute of Technology Madras, for the award of **Bachelor of Technology in Electrical Engineering and Master of Technology in Communication Systems**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.

Dr. Pradeep Sarvepalli
Research Guide
Assistant Professor
Department of Electrical Engineering
IIT Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I would like to thank Dr. Pradeep Sarvepalli for his guidance in this project, and for giving me the freedom to pursue my ideas and work in the comfort of my room, while still ensuring that I was on the right track. Special thanks to Arun Aloshious for explaining the error map from square octagonal lattice to toric codes, and for his help in debugging the code. I am also grateful to Arjun Bhagoji for his help with the toric code decoder.

I thank my colleagues and wing-mates, especially Amey, Kumar, Navneet, Sreeker, Lakshmikant and Priyadarshan who helped me have a great time in the institute and for staying with me through thick and thin. They are part of the memories I'll cherish forever.

Finally, none of this would have been possible without the constant encouragement and support from my mother, father and sister. Their unwavering faith in me has guided me through difficult times, and will continue to be the source of my motivation.

ABSTRACT

A work by Bombin, Duclos-Cianci and Paulin showed that every local translationally invariant 2D topological stabilizer code is locally equivalent to a finite number of copies of Kitaev's toric code. A paper by Bhagoji et al. gave an algorithm to map an arbitrary color code to a pair of surface codes. It was shown that the constraint of translational invariance can be relaxed, and any color code can be mapped to exactly two copies of a related surface code.

In this thesis, we study the performance of this proposed matching decoder for a square octagonal color code lattice. The decoding involves mapping the syndromes from said color code lattice to two copies of square toric codes, and then using a matching decoder on those two to estimate the most likely error path. Through a simulation, we obtained an error threshold of approximately 6% for this type of decoder.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
LDPC	Low Density Parity Check
$a\%b$	Reminder obtained after dividing a with b (modulo operator)
$\|a - b\|$	Absolute difference between a and b
BER	Bit error rate
BFS	Breadth-first search

NOTATION

σ_{xi}	Pauli operator σ_x acting on ith qubit
σ_{yi}	Pauli operator σ_y acting on ith qubit
σ_{zi}	Pauli operator σ_z acting on ith qubit

CHAPTER 1

INTRODUCTION

1.1 Introduction

Quantum systems have the ability to exist in superpositional states, allowing them to vastly outperform their classical equivalents. The biggest problem in the construction of a large, multi-qubit quantum computer is quantum decoherence.

The branch of topology is concerned with features of space that are robust against small, local disturbances like bending and stretching but not tearing or gluing. Fault-tolerant topological quantum codes are insensitive to local changes, and can prevent error propagation in quantum information while processing it.

Kitaev's toric codes require only local quantum processing for a graph embedded in a surface. This class of fault-tolerant quantum codes is well-studied, and efficient decoder implementations exist for them with error thresholds close to the optimal value of about 11% (de Queiroz [2009]). We were able to achieve a near-optimal error threshold of 10.9% in a simulation for toric codes.

Color codes are another class of topological quantum codes introduced by Bombin and Martin-Delgado [2006]. They are local, since each generator acts on a limited number of qubits. Graphical decoding algorithms for topological color codes have been studied in Wang *et al.* [2009] and a threshold of 13.3% for double-Pauli channel was reported. For a bit-flip channel, it translates to an effective error threshold of $\frac{2}{3} \times 13.3\% = 8.87\%$.

Matching decoding can be performed for arbitrary color codes by mapping them on a pair of surface codes, as proposed in Bhagoji and Sarvepalli [2015]. However, no data about its decoding performance was available.

This thesis aims at finding an error threshold estimate of this matching decoder for color codes. We study the performance of this type of decoder for color codes with square octagonal lattice. Errors are introduced on the color code lattice, and syndromes are mapped on two copies of square toric codes. We then estimate the most likely error path in color codes by decoding these two toric codes.

Our results showed an error threshold of nearly 6% for a phase-flip error model. Improvements in error threshold are possible by using soft-decision decoding instead of hard-decisions for finding the most likely path. Since the X and Z type errors on the surface code pair aren't fully independent, their correlation can be exploited to improve the error threshold further, as proposed by Delfosse and Tillich [2014].

1.2 Organization of thesis

Remainder of the thesis is organized as follows:

Chapter 2 briefly explains the necessary concepts of quantum information, quantum error correction and two classes of fault-tolerant topological error correcting codes. It serves as a quick introduction for the readers already familiar with classical error control coding theory.

Chapter 3 explains the process of matching decoding for Kitaev's toric code through an example. Brief explanations and pseudo-codes are provided where

necessary. A simulation of this matching decoder is done to obtain the error threshold close to the optimal value. Resultant plots are provided. A brief description of the process of modifying this algorithm for other periodic lattice geometries is given at the end of this chapter.

Chapter 4 talks about square octagonal color codes, and explains the process of matching decoding with an example. There are two mappings involved:

- Syndrome mapping from color codes to the surface code pair
- Getting the most likely error estimate from the matched syndromes in surface code pair

We explain both of these mappings with the help of figures and pseudo-codes with explanation where necessary. We then state a generic algorithm for this decoder, and finally the plots and error threshold obtained through a simulation of this decoder.

CHAPTER 2

BACKGROUND

I assume that the readers are already familiar with classical coding theory. The PhD thesis by Gottesman [1997] describes the concepts of quantum computation in detail. This chapter provides a quick introduction to some of the important concepts related to quantum computation.

2.1 Quantum Computation

In classical computation, information is stored in bits, each of which can assume a discrete value of either 0 or 1. If storing one number takes 64 bits, storing N numbers would linearly scale up and take N times 64 bits. Calculations are done essentially in the same way as one would by hand, and the speed advantage we get is purely due to higher processing speed of a computer. Quantum computation is different from classical in several aspects.

2.1.1 Qubits

Information in quantum computers is stored in *qubits*. Unlike classical bits that can only take discrete values, a qubit can be in a state represented as $|0\rangle$, $|1\rangle$, or a *superposition* of both denoted by $a|0\rangle + b|1\rangle$, where a and b are complex numbers. Here, $|0\rangle$ and $|1\rangle$ are two orthogonal states. Thus, the state of a single qubit comes from a complex vector space with its basis as $\{|0\rangle, |1\rangle\}$.

2.1.2 Entanglement

Another interesting property of qubits is called *entanglement*. In case of classical computation, a multi-bit state can always be decomposed into individual bits. However, this isn't necessarily true in case of qubits. Multiple qubits can form an *entangled state* that cannot be decomposed into tensor product of individual qubits.

For example, the state $\frac{1}{\sqrt{2}}(|01\rangle + |11\rangle)$ can be decomposed into a tensor product of single qubits as follows:

$$\frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) \equiv \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |1\rangle$$

However, we cannot decompose the entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ in a similar way.

With every extra qubit, the size of the basis multi-qubit vector space gets doubled. For example, basis of one-qubit vector space is $\{|0\rangle, |1\rangle\}$. For a two-qubit vector space, basis becomes $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. Thus, for every extra qubit, we can have a state with twice as many complex coefficients.

2.1.3 Unitary transformations

In case of classical bits, a single-bit operation can only modify information worth one bit. In case of qubits, we can perform what are called *unitary transformations* on the state of the qubits. We'll explain this with an example of quantum NOT (qNOT) gate.

A classical NOT gate flips its input bit, i.e., $NOT(0) = 1$ and $NOT(1) = 0$. Its quantum analog also does the same, but it flips all states in a superposition simultaneously.

$$QNOT_1(\,|000\rangle + |001\rangle + 2|010\rangle - |011\rangle - |100\rangle + 3i|101\rangle + 7|110\rangle\,)$$
$$= \quad |100\rangle + |101\rangle + 2|110\rangle - |111\rangle - |000\rangle + 3i|001\rangle + 7|010\rangle$$

Unitary transformations and superposition create possibilities that are not available for hand calculations.

2.1.4 Measurement in quantum mechanics

Measuring a quantum state $(\alpha|0\rangle + \beta|1\rangle)$ by taking projections on the basis $|0\rangle$ and $|1\rangle$, gives $|0\rangle$ with a probability of $|\alpha|^2$ and $|1\rangle$ with a probability of $|\beta|^2$. The normalization ensures that the total probability of returning a result is 1. After the measurement is performed, the state collapses into the outcome of measurement.

Note that unlike bits in classical computers, measuring a quantum state directly causes a discontinuous change of the state of the system, known as *the collapse of the state vector*. This makes it impossible to measure a quantum state and make an exact copy of it as we normally would in case of a classical state.

2.2 Pauli operators & their matrix representation

Instead of measuring a quantum system directly, we can understand it by observing the behaviour of various operators acting on the system state. For a single-qubit system, we have a set of operators called *Pauli spin matrices*.

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

All of the Pauli matrices are *Hermitian*, and thus have real eigenvalues. Moreover, they *anti-commute* with each other while also *commuting* with themselves.

$$\text{for } i \neq j: \quad \{\sigma_i, \sigma_j\} = \sigma_i\sigma_j + \sigma_j\sigma_i = 0$$

$$\text{for } i = j: \quad \sigma_i\sigma_j = \sigma_j\sigma_i = 1$$

This property will be useful to us in understanding stabilizers formalism in 2.5. The three Pauli matrices correspond to three Pauli operators X, Y and Z respectively.

2.3 Quantum errors

Same as its classical equivalent, quantum error correction codes also involve syndrome calculation, which reduces the effect of an error to one of several distinct possibilities. In most of the codes, error is either a **bit flip**, **sign (phase) flip**, or **both**.

A bit flip can be done on a state by the action of Pauli X operator, as defined in 2.2. Effect of bit-flip on a single qubit quantum state is as follows:

$$|0\rangle \rightarrow |1\rangle$$

$$|1\rangle \rightarrow |0\rangle$$

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|1\rangle + \beta|0\rangle$$

Similarly, a sign flip error can be generated on a state with Pauli Z operator. Its effect on a single qubit quantum state is as follows:

$$|0\rangle \rightarrow |0\rangle$$

$$|1\rangle \rightarrow -|1\rangle$$

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|0\rangle - \beta|1\rangle$$

The two errors can occur simultaneously on a qubit. Result is the same as what we would obtain with Pauli's Y operator, corresponding to $\sigma_y = i\sigma_x\sigma_z$. Any linear combination of the above three errors (with complex coefficients) can also occur simultaneously. Pauli operators thus form the basis of error vector space.

2.4 Quantum error correction

Classical error correction makes use of redundancy to safeguard information against noise and errors. However, copying quantum information is not possible, since that involves measuring it, leading to collapse of the state vector. Consequently, it's not possible to encode a state $\alpha|0\rangle + \beta|1\rangle$ as:

$$\alpha|0\rangle + \beta|1\rangle \rightarrow (\alpha|0\rangle + \beta|1\rangle)(\alpha|0\rangle + \beta|1\rangle)(\alpha|0\rangle + \beta|1\rangle)$$

However, we can "spread" the information of one qubit over a highly entangled state of multiple qubits. Consider this encoding scheme, where one qubit is being spread over nine:

$$|0\rangle \rightarrow |\bar{0}\rangle \equiv (|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)$$
$$|1\rangle \rightarrow |\bar{1}\rangle \equiv (|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)$$

This code is due to Shor [1996]. In order to know which one of the states the system is, here we need to measure at least three qubits. Also, as seen in 2.3, possible errors on every qubit could be σ_x, σ_y and σ_z.

A bit flip error on first qubit can be detected by first comparing the initial two qubits, and then second and third qubits. Results of these comparisons will give

us information about which one of the three bits has flipped, and we can correct that by flipping back using σ_x operator.

Let's split the nine qubits into groups of three. Now a sign flip error can be detected by first comparing the signs of two initial groups, followed by another comparison between second and third group. After we find the group with phase error, it can be corrected by changing back the signs using σ_z operator.

If a qubit has undergone both, a bit flip and a sign flip, it will still be corrected if we go through both the correction processes described above. In fact, the most general single-qubit error can be expressed as a linear combination of σ_x, σ_y, σ_z and identity matrix I, and can also be fixed in the same way.

2.5 Stabilizer formalism

Stabilizer generators are quantum equivalent of parity check matrix in classical coding theory. They allow us to "import" some classical coding schemes for direct use as a quantum code. They contain a set of mutually commuting check operators. Codespace is then defined as a set of all codes where check operators act trivially.

Specifying the stabilizer group is equivalent to fully specifying the codewords. The stabilizer formalism gives us a clean and elegant representation of quantum codes.

We will explain this further through the example of Kitaev's toric codes in Section 2.8.

2.6 Topology

Topology is the study of geometrical properties and spatial relations unaffected by the continuous change of shape or size of figures. Any two objects in space are said to be *topologically equivalent* if one can be converted into another by continuous modifications, i.e., without "tearing" or "gluing" the surface, although "stretching" or "contracting" it is fine.

There's an old joke among mathematicians that a topologist cannot tell the difference between a coffee mug and a donut. This is because both of these objects have a single hole and can be deformed from one into another without tearing or gluing the surface as shown in Figure 2.1.

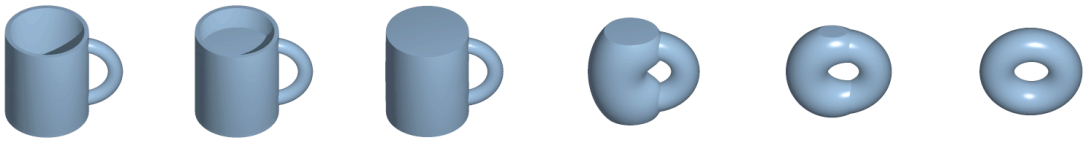


Figure 2.1: Continuous deformation of a coffee mug into a donut

The motivation behind topology is that some geometric problems depend not on the exact shape of the object, but instead on the way they are put together.

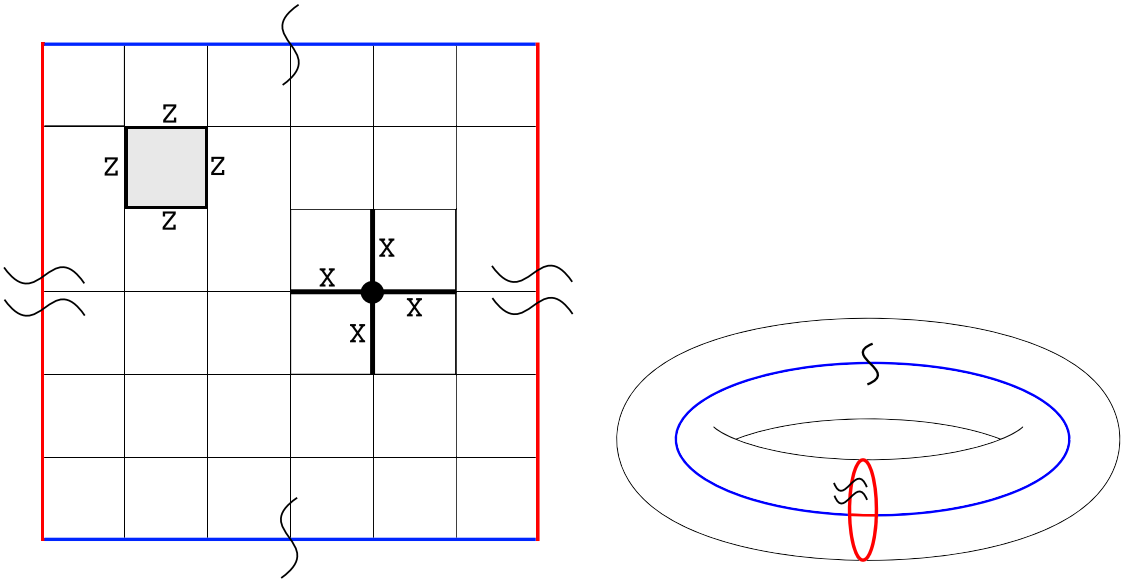
2.7 Topological error correcting codes

Topological quantum codes were first introduced by Kitaev [2003]. As topology is insensitive to continuous geometric deformations, topological error correcting codes focus on *local* detection and correction of errors. Qubits in topological error correcting codes are stored on a lattice with periodic boundary conditions. Syndrome calculation can be done *locally*, i.e., all stabilizer generators act on only

a few nearby qubits. Error correction can thus be done locally. The lattice itself can be arbitrarily large.

2.8 Kitaev's Toric Code

Kitaev's toric code is a topological quantum error correcting code, and an example of stabilizer code defined on a two-dimensional spin lattice. It is the simplest and most well studied of the quantum double models.



(a) Square lattice with periodic boundaries. The two types of stabilizers are shown.

(b) Alternate representation of toric lattice.

Figure 2.2: Representations of the lattice used in toric code

In Kitaev's toric code, the qubits are stored on the edges of a square lattice with dual periodic boundary conditions, i.e., the opposite sides of the square are identified making the lattice take form of a torus as shown in Figure 2.2b.

2.8.1 Stabilizers

Two types of stabilizer generators are defined for this code, as shown in Figure 2.2a. X-type stabilizers reside on vertices of the lattice and Z-type stabilizers on the faces (plaquettes) of the lattice.

Each X-type stabilizer is a tensor product of the four X-type operators at that vertex. Each Z-type stabilizer is a tensor product of the four Z-type operators surrounding the plaquette. It is important to note that:

- All the stabilizers commute with each other since they always cross one another an even number of times.
- A valid codeword is where every stabilizer operator commutes with the code.

Thus, an error that can be represented by a collection of stabilizer operators will commute everywhere and not change the codeword. Examples of this are *trivial cycles*, as shown in Figure 2.3.

We have $2l^2$ physical qubits for an $l \times l$ square lattice. Due to the periodicity of the lattice, total number of independent stabilizers is $l^2 - 1$ for each type of stabilizers. Due to the constraint that every stabilizer operator must commute with a valid codeword, the number of encoded qubits becomes

$$(n - g) = 2l^2 - (2(l^2 - 1)) = 2.$$

2.8.2 Syndromes

Without loss of generality, let's consider only Z-type (phase flip) errors for now. Suppose a single Z-type error occurs on an edge. Then, this will affect two X-type stabilizers that contain this edge, and make them both evaluate to -1.

Similarly, if there's a chain of Z-type errors, only the X-type stabilizers at both ends of each *connected* segment will evaluate to -1. X-type stabilizers that evaluate to -1 at a vertex are said to have a negative syndrome at that vertex.

The syndromes we obtain from the toric code are very ambiguous. Multiple different paths where errors start and end at the same points will give same syndromes. Also, note that the number of vertices with negative syndromes will always be even for a toric code.

2.8.3 Trivial and non-trivial cycles

If an error path starts and ends at the same point, it forms a cycle. As shown in figure 2.3, there are two types of cycles possible in a toric graph: a *trivial cycle* and a *non-trivial cycle*.

A **trivial cycle** forms the boundary of a set of plaquettes. This homologically trivial cycle is in fact a product of plaquette operators and thus doesn't affect the code.

A **non-trivial cycle**, on the other hand, cannot be represented as a product of plaquettes. It "winds around" the torus, as can be seen in figure 2.3. It commutes with all the stabilizers, but is not a product of them. In fact, it is the product of an encoded or logical Z operation and a plaquette stabilizer.

A non-trivial cycle damages the encoded state of the system. It has the effect of changing one codeword into another, resulting in loss of stored information.

X-type errors can form similar types of cycles on the *dual lattice*. The dual is formed by interchanging the positions of vertices and plaquettes on the primary lattice. Thus, there are two logical Z operators, Z_1' and Z_2' on the primary lattice,

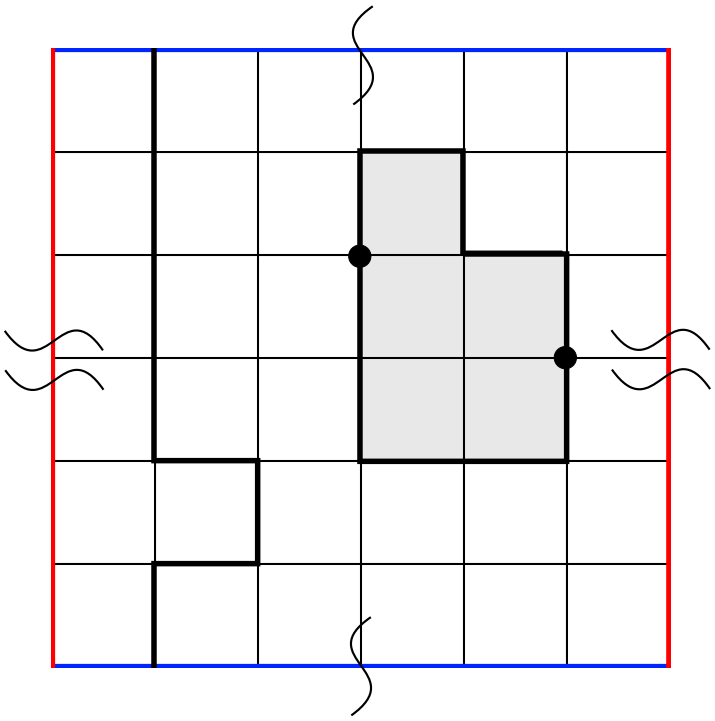


Figure 2.3: A trivial cycle (right) bounds a set of plaquettes unlike a non-trivial cycle (left). In the trivial cycle on the right, the syndrome of the top path is the same as that of the bottom path. Thus, the ambiguity in syndromes. If top path were the actual error on the lattice, and the bottom path our estimate, the encoded state is undamaged by this error.

and two logical X operators, X_1' and X_2' that come from non-trivial cycles on the dual lattice, as shown in figure 2.4.

2.8.4 Cycles from a topological perspective

As we saw in Section 2.6, a topology is insensitive to gradual changes that do not involve tearing or gluing of the surface. Kitaev's toric code is a topological code. The periodic square lattice is embedded on a torus.

Figure 2.5 shows spacial representation of a trivial cycle. A trivial cycle can be contracted to a single point without altering the topology of the torus. A non-trivial cycle cannot be reduced to a point without changing the topology.

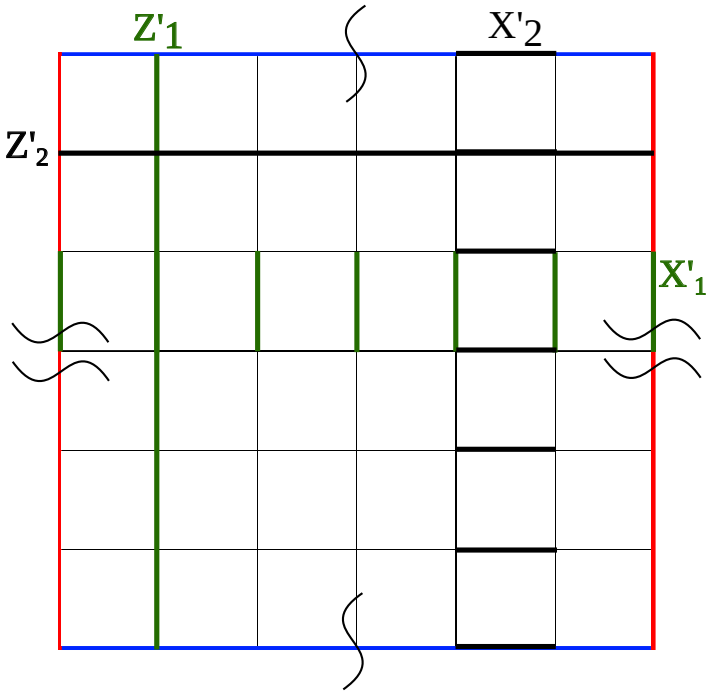


Figure 2.4: The basis for the four logical operators that act on the encoded qubits.

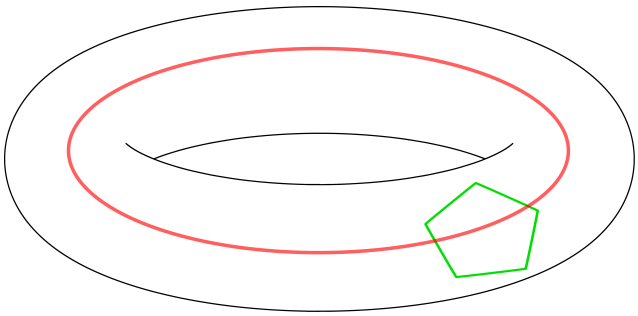


Figure 2.5: A trivial cycle (green) and a non-trivial cycle (red).

2.8.5 Code distance

The distance of a stabilizer code is the smallest weight of the Pauli operator that affects the encoded qubits, but still commutes with all the stabilizers. Thus, for a toric code on a $l \times l$ lattice, the code distance is the length of smallest non-trivial cycle, l. We can increase the code distance by just increasing the size of lattice. However, increasing the size of lattice does not affect the number of encoded qubits.

2.9 Color codes

Color codes are another class of topological codes. They are similar to surface codes except for an additional property associated with their faces, called 'color'. Often, the colors used to classify the faces are red, green and blue.

They are obtained from **trivalent 2D lattices** with 3-colorable faces, known as *2-Colexes*. Unlike the toric code where a qubit corresponds to an edge of the lattice, here each vertex represents a qubit. The generators of the stabilizer are X and Z plaquette operators. Edges of color codes can be 3-colored accordingly, based on the color of faces they connect.

A stabilizer code can be defined on a 2-Colex by a stabilizer S:

$$S = \langle B_f^X, B_f^Z \mid v \,\epsilon\, F(\Gamma) \rangle \quad \text{where} \quad B_f^\sigma = \prod_{v \epsilon f} \sigma_v$$

We will see a specific example of color codes later in Chapter 4.

CHAPTER 3

DECODER FOR TORIC CODE

The process of error estimation and correction for Kitaev's toric codes requires first calculating the syndromes at each vertex and plaquette. The points with non-zero syndromes correspond to end-points of actual errors. Then we proceed to estimate the most likely error that could have occurred, and try to negate its effect.

Unlike classical codes, however, estimated error here need not be equal to the actual error. Even if the estimated error and actual error form a trivial cycle, the encoded information is still preserved.

In this chapter, we will see the process of detection and correction of errors in Kitaev's toric code with an example.

Due to the symmetry of decoding X and Z errors, we only consider Z-type errors in this chapter. We use a *phase-flip* error model, i.e., a qubit can be affected by Z error independently with a probability p, and can similarly remain unaffected with a probability $(1 - p)$.

3.1 Error estimation

In a graph with only Z-type errors, it suffices to calculate the syndromes at every vertex using only X-type stabilizers. Figure 3.1 shows a toric code after introducing random Z-type errors. We will use this as an example to illustrate the steps involved in this process.

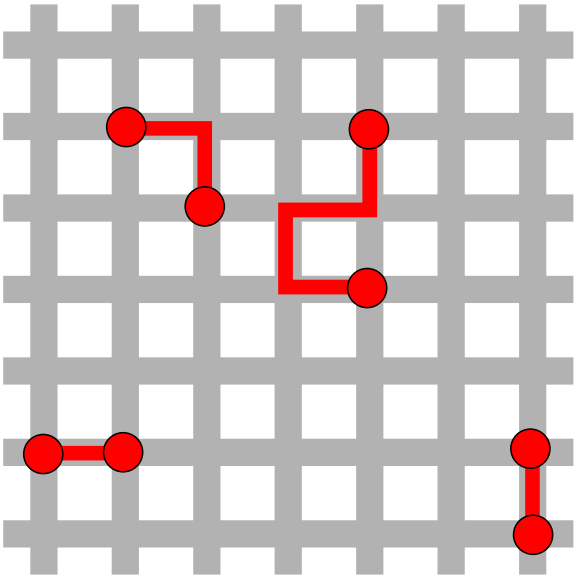


Figure 3.1: A toric code after introducing random Z errors

3.1.1 Syndrome calculation

Endpoints of error segments will anti-commute with stabilizer operators and can will be detected. Multiple distinct errors may result in equal syndromes, since the end-points of error segments remain the same. Figure 3.2 shows the endpoints of error segments detected in our example.

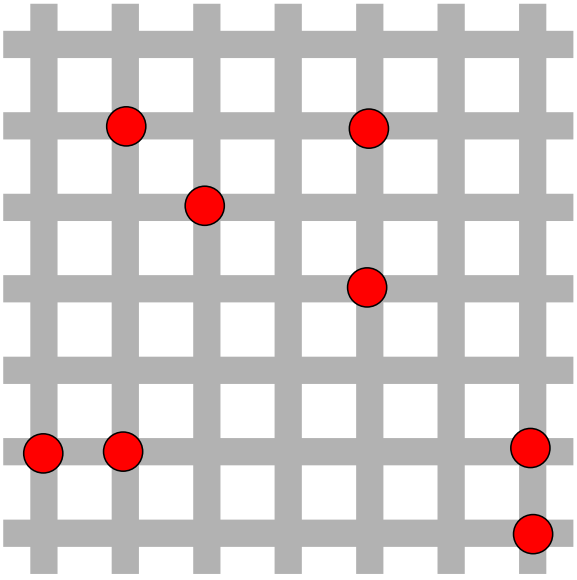


Figure 3.2: Error end-points become known after finding syndromes

3.1.2 Finding most likely error path

After syndrome calculation, we have a set of points with non-zero syndromes and are required to find an estimate of the error path.

Error probability is usually small and necessarily less than one. Probability of an error to have affected four and three physical qubits is thus p^4 and p^3 respectively, with $p < 1$. Hence, we can see that shorter error paths are more likely compared to longer ones.

Thus, the problem reduces to connecting all of these points with segments such that their total length is minimized. For this, we make use of **perfect matching**. See Appendix A for more details on perfect matching.

In our example, performing matching on Figure 3.2 results in error estimate shown with dashed lines in Figure 3.3.

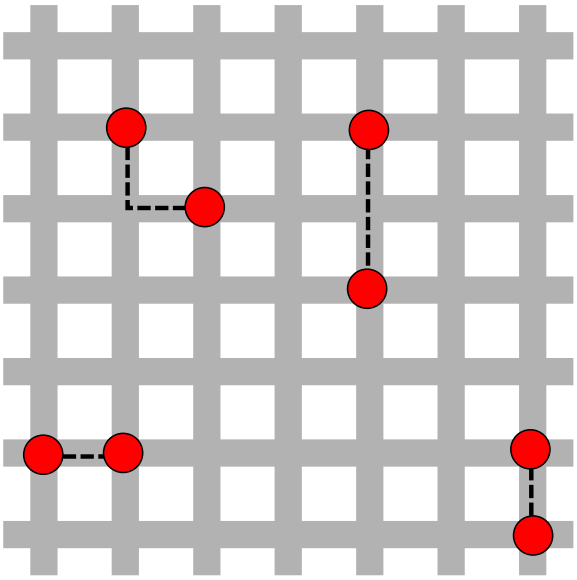


Figure 3.3: Matched vertices with most likely error estimate, $\hat{E}$, shown with dashed lines

3.2 Error correction

After obtaining the estimated error path, we flip the phase along the estimated error path to hopefully negate the effect of original error. This can result in three outcomes:

1. Non-zero syndromes exist after error correction
2. Syndromes zero after error correction, no non-trivial cycles in the graph
3. Syndromes zero after error correction, no non-trivial cycles in the graph

3.2.1 Case 1: Nonzero syndromes exist after correction

Since the number of syndrome points are always even, a perfect matching always exists for toric codes. Thus, syndromes after error correction **must** be zero everywhere. Existence of a non-zero syndrome signifies incorrect matching.

In rest of the cases, we assume syndromes are zero everywhere after error correction.

3.2.2 Case 2: No non-trivial cycles in the graph

We check for non-trivial cycles using logical operators shown in Figure 2.4. When a non-trivial cycle is present, one of these logical operators will anti-commute with it.

If the graph doesn't contain any non-trivial cycles after error correction, that means the encoded information is now error-free and decoding was successful. Note that trivial cycles may still exist in the graph, but they do not affect the encoded information in any way.

3.2.3 Case 3: Non-trivial cycles exist in the graph

If the graph contains a non-trivial cycle after error correction, that implies the encoded information contains a logical error. We declare a decoder failure in this case.

In classical coding terms, this is equivalent to a scenario where correcting an error gives us a valid but different codeword. The parity check says it's a valid codeword, but our information is lost irrecoverably.

3.2.4 Examples

In our example, adding estimated error (Figure 3.3) with original error (Figure 3.1) gives rise to two trivial cycles as shown in Figure 3.4.

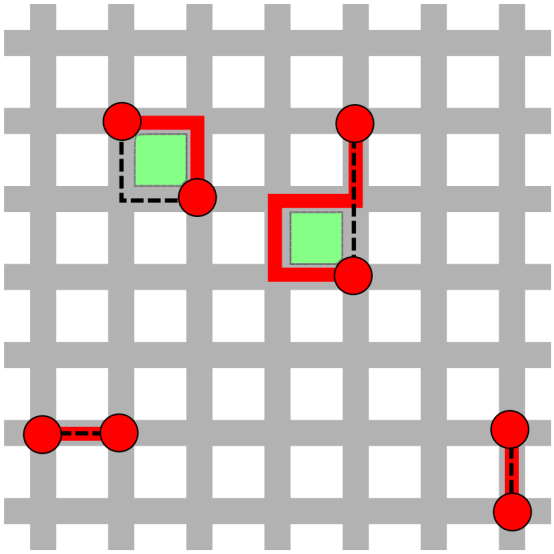


Figure 3.4: Adding estimate to the original error, $E + \hat{E}$, gives rise to two trivial loops shown in green.

Figure 3.5 shows a slightly different case, where we see a non-trivial cycle formed after error correction. Encoded information is lost irrecoverably in this case.

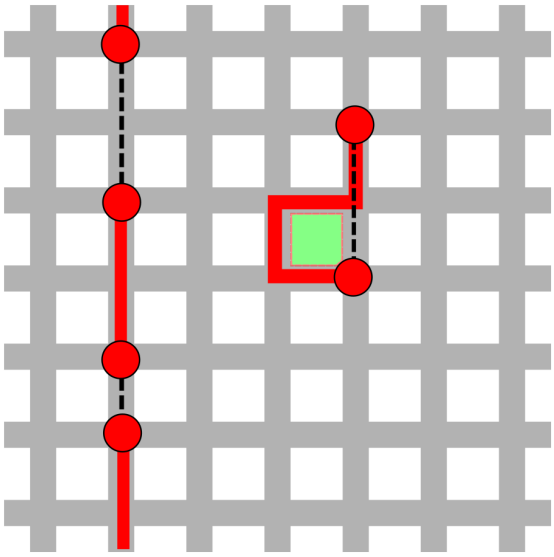


Figure 3.5: A scenario where a non-trivial cycle is formed after error correction

3.3 Algorithm for matching decoder

Algorithm 1 illustrates the simulation we ran for testing the performance of a matching decoder for Kitaev's square toric code. We have already described all the steps involved in previous sections of this chapter. Rest of the details (e.g. Manhattan matrix distance) can be found in Appendix A.

3.4 Simulation and results

Optimal decoders for Kitaev's toric code have an error threshold close to 11%. We ran a simulation in the form of a C++ program based on Algorithm 1. Boost C++ libraries (Boost [2012]) were used for this.

We were able to replicate a phase error threshold of approximately 11% through our simulation, as can be seen from figures 3.6 and 3.7.

Algorithm 1: Simulation of perfect matching decoder for Kitaev's toric code

input : error probability, p; number of trials, T; lattice length, L
output: Error rate for a given error probability

```
create a periodic square lattice of length L for toric code;
for T trials do
    introduce a random Z errors with error probability p;
    find syndromes at each vertex;
    if all syndromes are zero then
        check for logical error;
        if logical error found then
            declare decoder failure;
        else
            declare decoder success;
        end
    end
    create a graph G from all vertices with nonzero syndromes;
    foreach distinct vertex pair (u, v) in G do
        get shortest distance d(u, v) = W;
        add an edge between (u, v) with weight W;
    end
    get a minimum cost perfect matching M on graph G;
    error estimate Ê = union of shortest paths for matched vertices in M;
    add Ê to the original error E;
    check for logical error;
    if logical error present then
        declare decoder failure;
    else
        declare decoder success;
    end
end
```

return : Error rate=(number of decoder failures)/T

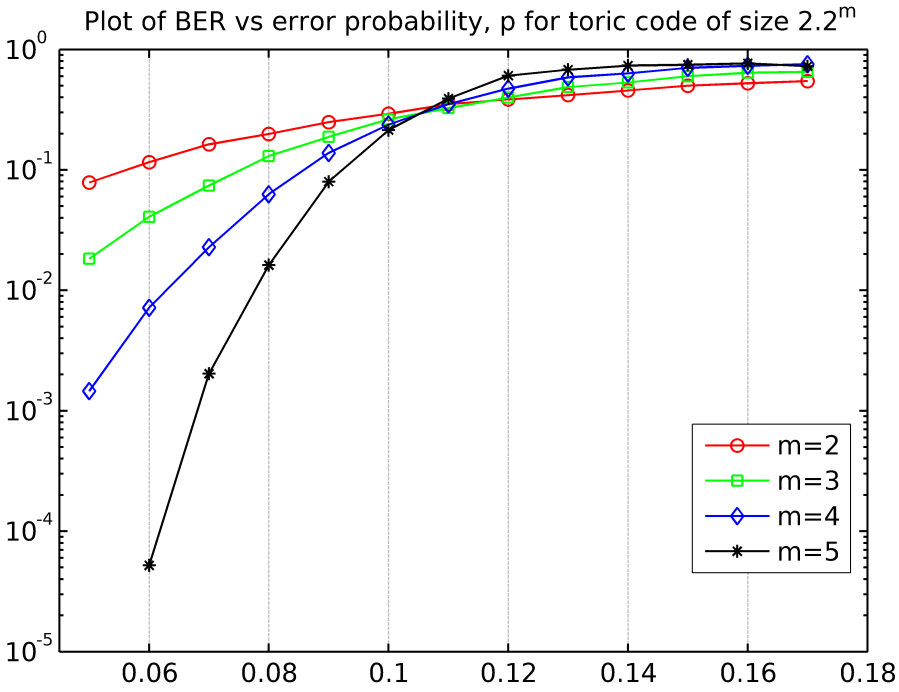


Figure 3.6: Decoding performance of standard matching decoding for toric code lattices of size 8, 16, 32 and 64. X axis is linear.

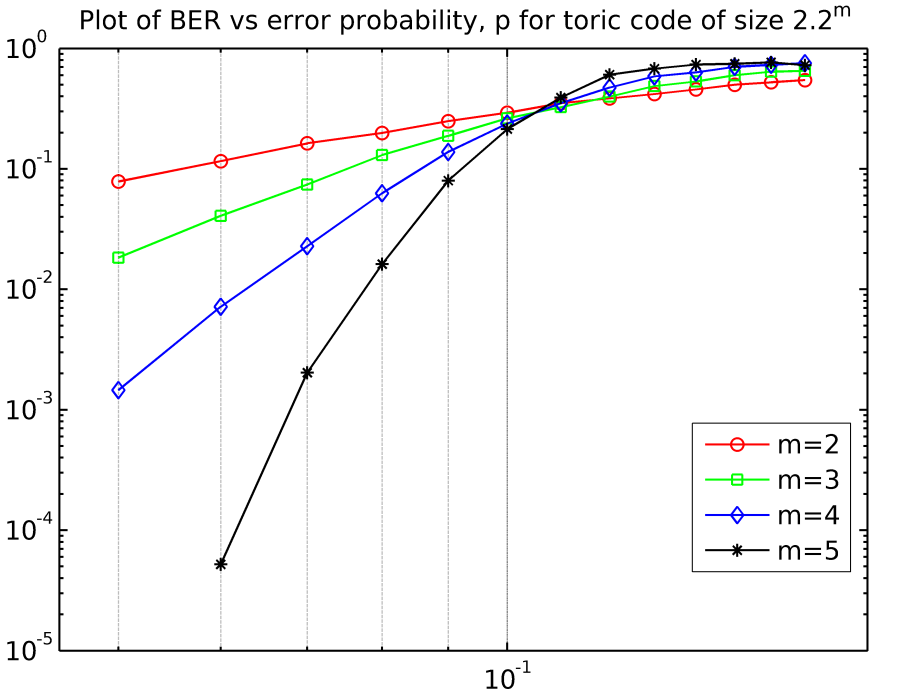


Figure 3.7: Decoding performance of standard matching decoding for toric code lattices of size 8, 16, 32 and 64 on a logarithmic scale.

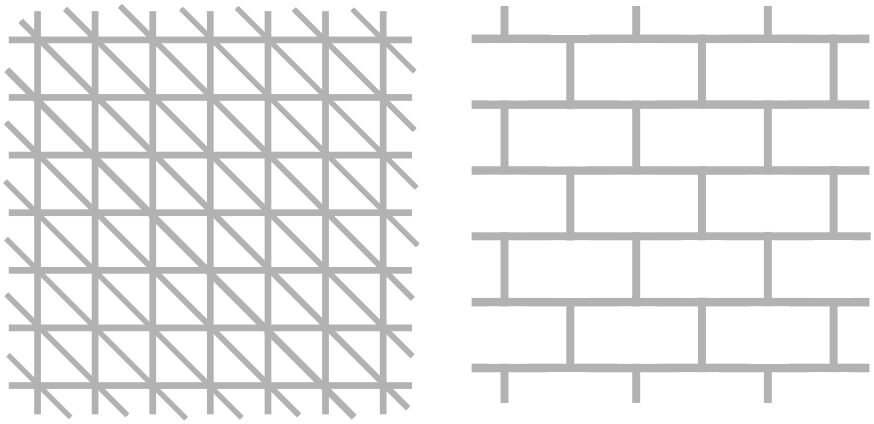
3.5 Extending the toric decoder to other geometries

Algorithm 1 can be extended for other periodic lattice geometries as well, e.g., triangular or hexagonal, by changing the shortest distance and shortest path functions.

For unweighted graphs, breadth-first search (BFS) algorithm can be used to find distance. This will work irrespective of the geometry of the lattice. Dijkstra's algorithm can be used to find distance in weighted graphs.

Geometry-specific functions can be designed and will generally perform better than BFS or Dijkstra's algorithm. A starting point in making such functions for triangular and hexagonal lattice can be a paper about shortest paths in triangular grids by Nagy (Nagy [2003]). However, the algorithm proposed in it needs to be modified in order to accommodate periodic boundary conditions of a surface code.

Figure 3.8 shows alternate representations of triangular and hexagonal lattice that might be helpful for this purpose.



(a) Triangular lattice made by adding an additional diagonal edge per cell of square lattice

(b) Hexagonal lattice made by removing alternate edges from square lattice

Figure 3.8: Possible arrangements for hexagonal and triangular lattices that may be used with minimal modifications in the existing decoder for square lattice

CHAPTER 4

A DECODER FOR COLOR CODES

A recent paper by Bhagoji and Sarvepalli [2015] gives the equivalence of arbitrary 2D color codes to surface codes, lifting the constraint of translational symmetry as presented by Bombin *et al.* [2012]. To test the performance of matching decoder of color codes, a simulation was carried out. The syndromes in square octagonal color code were mapped to two instances of toric code, decoded, and the estimated error path was mapped back to the original color code. In this chapter we will illustrate the whole process with an example. A generalized algorithm, syndrome and error mappings, details of the simulation, and results will follow.

4.1 Square octagonal lattice

The Figure 4.1 shows a 64-qubit square octagonal lattice for color code, with red, green and blue colored faces. It has similar periodic boundary conditions as that of a square toric code, making its 3D representation look like a torus.

Number of independent stabilizer generators for this code are:

$$\begin{aligned} g &= 2(|F_r| + |F_g| + |F_b|) - 4 \\ &= 2(4 + 2 + 2) - 4 = 12 \end{aligned}$$

Thus, the number of encoded qubits are $16 - 12 = 4$.

The edges connecting two red surfaces are colored red, for example, $(1, 2)$ and $(3, 4)$. Similarly, $(1, 12)$ and $(2, 3)$ are colored green and blue, respectively. Each

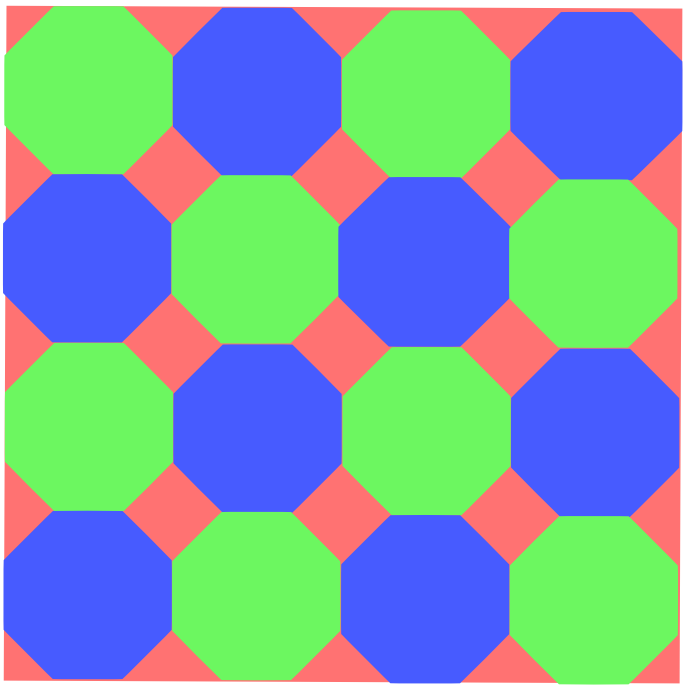


Figure 4.1: A 64-qubit square octagonal color code lattice

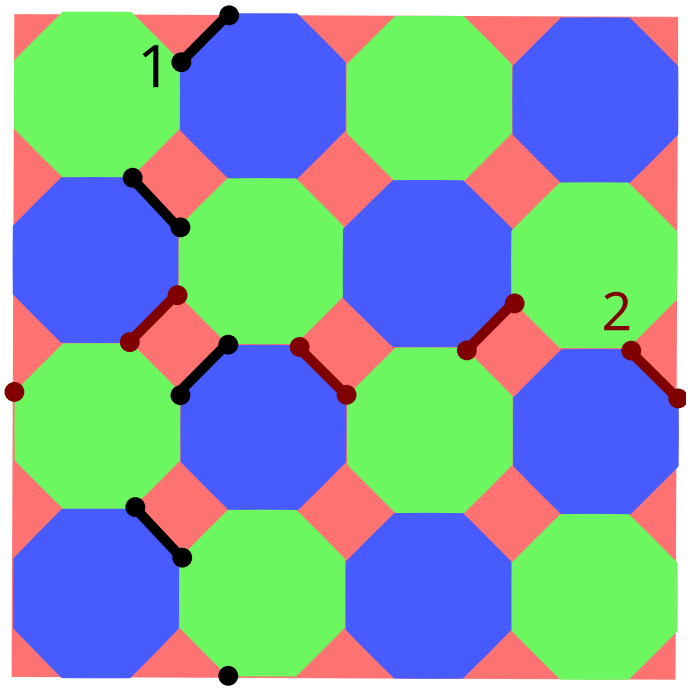
vertex in the code shown in Figure 4.1 can have an X or a Z error on it with a probability p, independently.

4.2 Logical operators

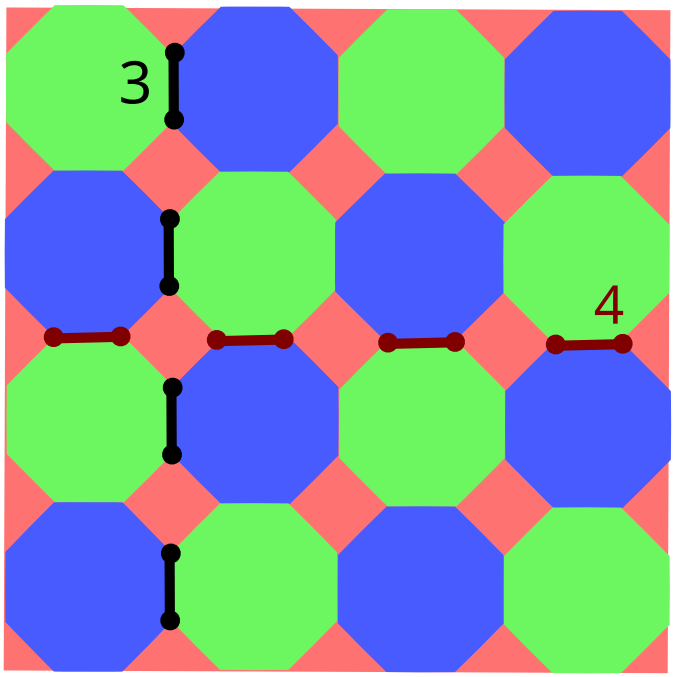
Square octagonal color code has four logical operators we need to check for during decoding, as shown in Figure 4.2.

4.3 Decoder algorithm

Algorithm 4 shows the process of decoding a square octagonal color code by mapping it to two copies of toric code. Parts of the algorithm will be explained in the subsequent subsection.



(a) All the segments in 1 and 2 must have the same color, i.e., they must connect faces of the same color.



(b) Segments in 3 and 4 should be red, i.e., connect only the red faces.

Figure 4.2: Four logical operators (numbered 1 to 4) for a square octagonal color code lattice, for a particular type of error (X or Z).

4.3.1 Mapping syndromes from color code to surface code

All the face syndromes of a square octagonal color code (C_1) are mapped to two copies of square toric lattices (S_1 and S_2). Syndromes of octagonal faces map only to S_1, while those of the square faces map to S_2 with some modifications to them.

$$C_1 \rightarrow S_1 + S_2$$

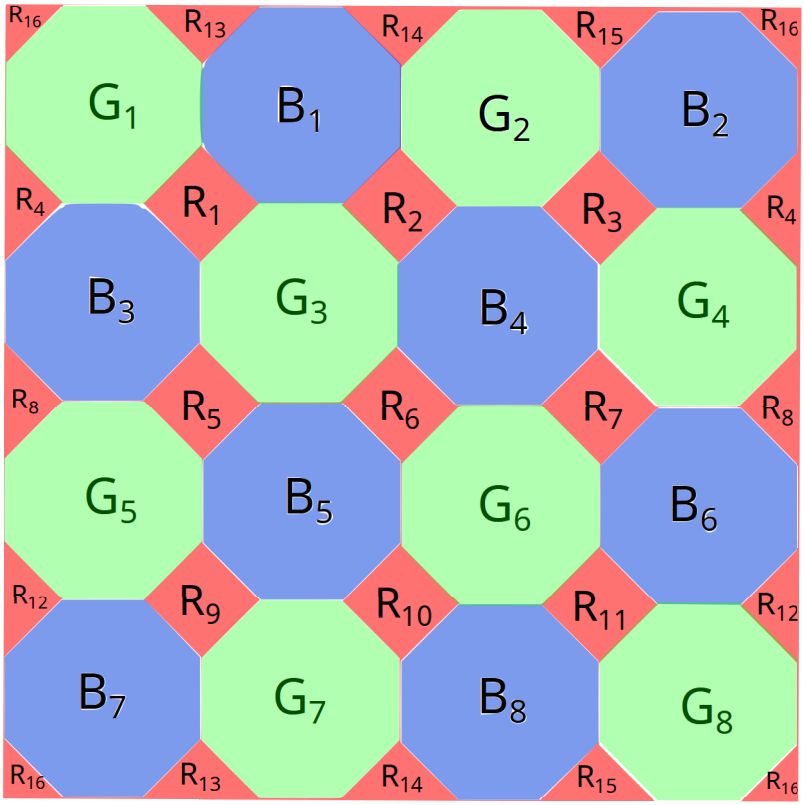
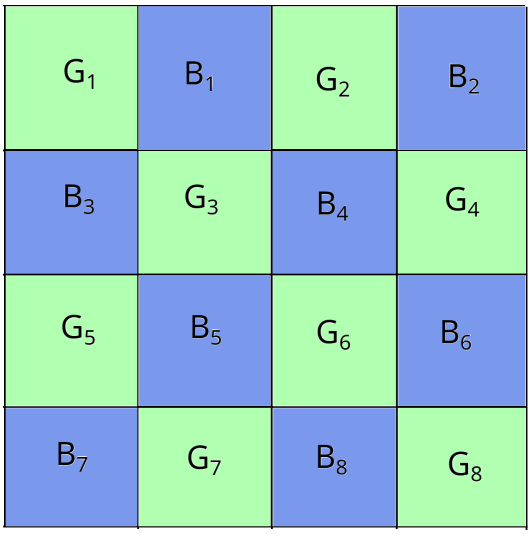
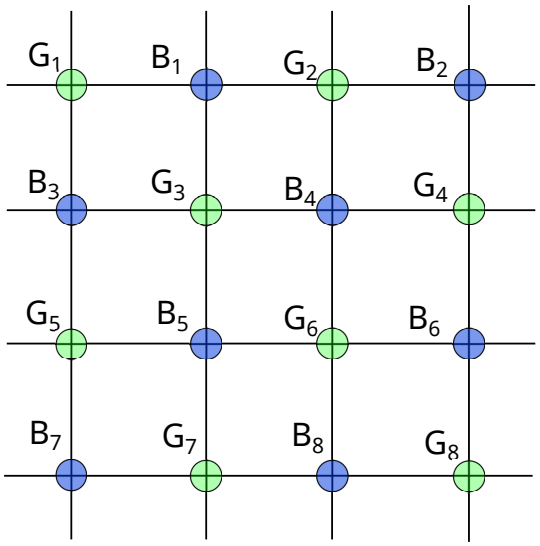


Figure 4.3: The color code C_1, with all its face syndromes G_i, R_i and B_i.

Figure 4.3 shows the face syndromes of all the faces of C_1. Figure 4.4 shows the mapping of octagonal face syndromes of C_1 to S_1. Syndromes are denoted by G_i and B_i for green and blue faces. Figure 4.5 shows the mapping for square face syndromes of C_1 to S_2.



(a) Square lattice, S_1 after syndrome mapping. Colors are shown only to identify the corresponding face.



(b) Dual of S_1. Syndromes now reside on the vertices.

Figure 4.4: Mapping of octagonal face syndromes of C_1 to S_1. Syndromes B_i and G_i are mapped directly as shown. Dual lattice is used for finding the error path using Algorithm 1.

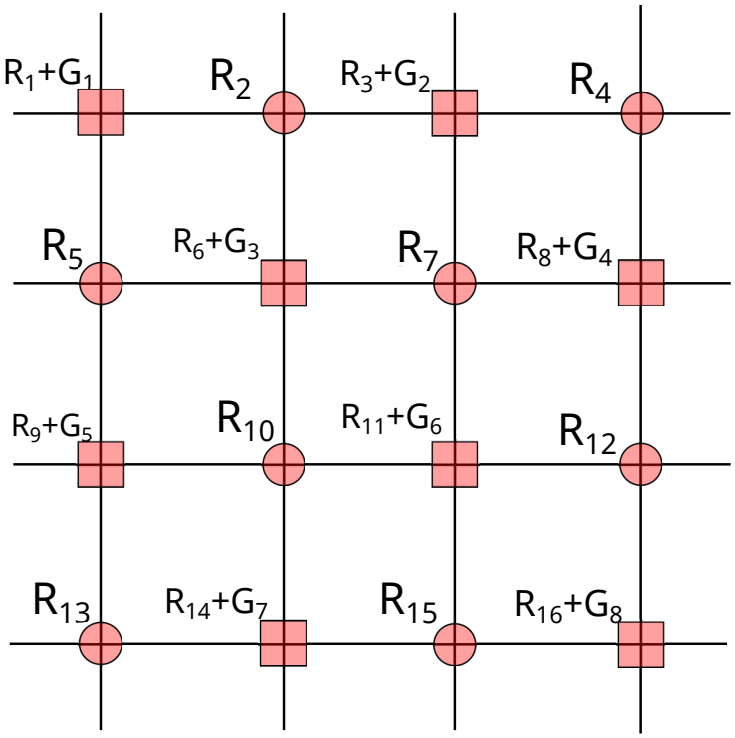
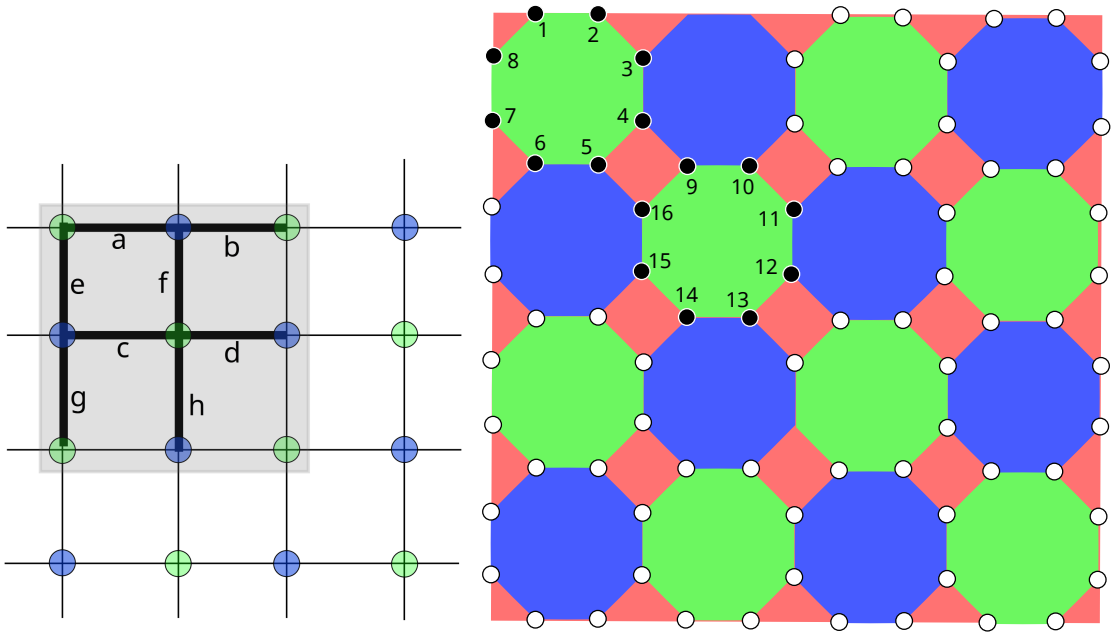


Figure 4.5: Square lattice S_2, after mapping square face syndromes from C_1. Colors are shown only to identify the corresponding face. Syndromes reside on vertices, and S_2 can directly be used for finding error path.

4.3.2 Mapping error path from surface codes to color code

After we estimate error paths through matching in graphs S_1 and S_2, we will need to lift both the estimate back to the color code lattice. We do this locally, by tiling S_1 and S_2 into sections as shown in Figure 4.6 and Figure 4.7.

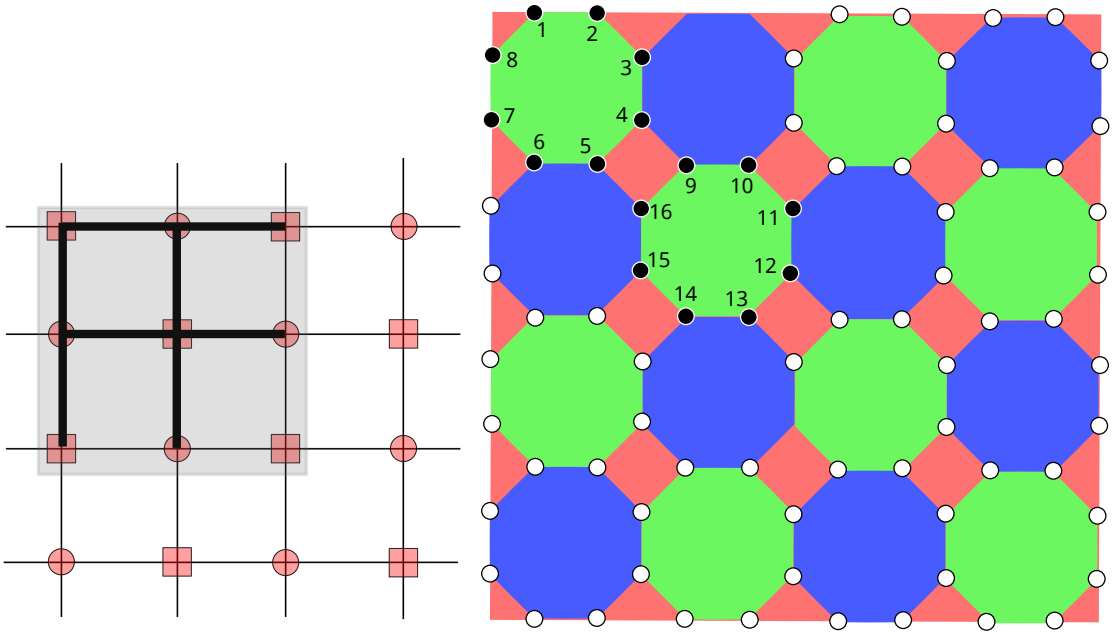


(a) A tile of S_1 (b) Corresponding vertices on C_1.

Figure 4.6: A tile of S_1 and its corresponding vertices on C_1 denoted using black dots.

4.4 Simulation results

Simulation in the form of a C++ program based on Algorithm 2 was run. Resulting BER vs error probability plots are shown in Figures 4.8 and 4.9. Phase error threshold of nearly 6% was obtained.



(a) Tiling of S_2

(b) Corresponding vertices on C_1 is shown using black dots and numberes.

Figure 4.7: A tile of S_2 and its corresponding vertices on C_1 denoted using black dots.

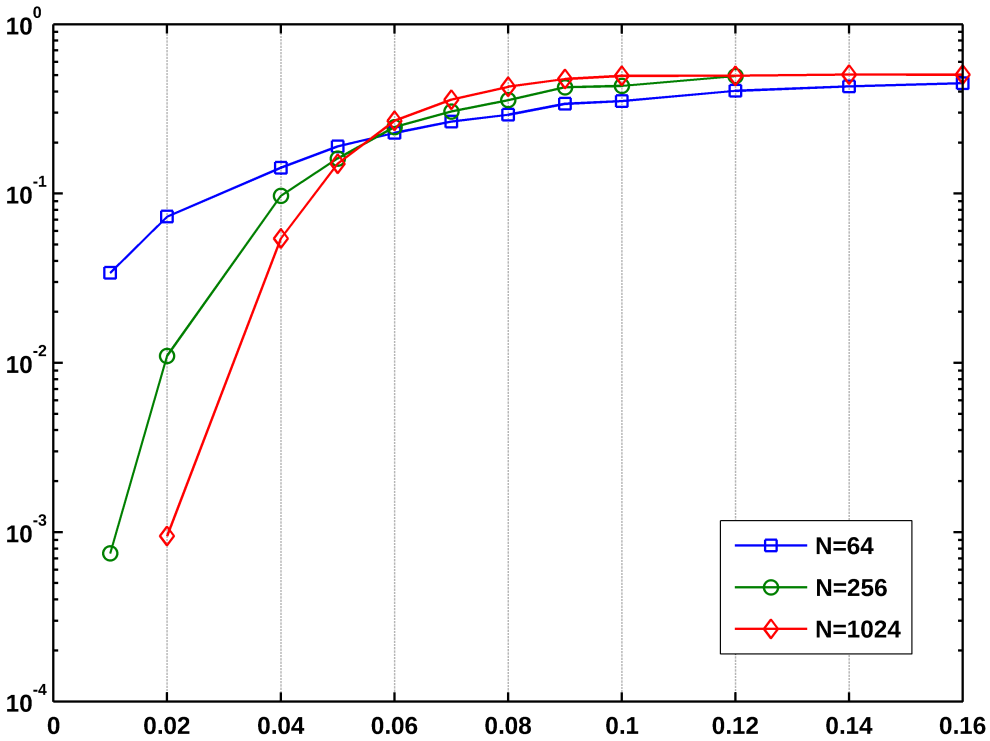


Figure 4.8: Decoding performance of standard matching decoding for color code lattices with 64, 256 and 1024 qubits. X axis is linear.

Algorithm 2: Algorithm for color code decoder using surface code mapping

input : probability of error p, number of trials T
output: bit error rate for the given p, $BER = success\_count/T$

initialize *trial_count* to zero;
create a periodic square octagonal color code lattice of length L;
while *trial_count is less than or equal to T* **do**
    increment *trial_count* by one;
    introduce random Z errors with error probability p;
    find syndromes at each octagonal (green and blue) face;
    find syndromes at each square (red) face;
    if *no negative syndromes* **then**
        check for the presence of non-trivial cycles using logical operators;
        if *a non-trivial cycle found* **then**
            increment *failure_count* and move on to the next trial;
        else
            increment *success_count* and move on to the next trial;
        end
    end
    map the syndromes to two instances of square toric lattice;
    foreach *square toric lattice* **do**
        add all vertices with negative syndrome to a vector V;
        create a weighted undirected graph G;
        foreach *pair of distinct vertices* (u, v) *in* V **do**
            get Manhattan distance between points $(u, v) = W_{uv}$;
            add vertices (u, v) to G;
            create an edge (u, v) with weight W_{uv}) in G;
        end
        get a perfect matching M by running Blossom V on graph G;
        foreach *matched vertex pair* (u, v) *in* M **do**
            choose a shortest path from u to v;
            map the shortest error path back to color code lattice;
        end
    end
    toggle X and Z errors along mapped error paths;
    re-calculate all the face syndromes;
    if *a negative face syndrome found* **then**
        declare matching error and quit the program;
    end
    check for the presence of non-trivial cycles using logical operators;
    if *a non-trivial cycle found* **then**
        increment *failure_count* and move on to the next trial;
    else
        increment *success_count* and move on to the next trial;
    end
end

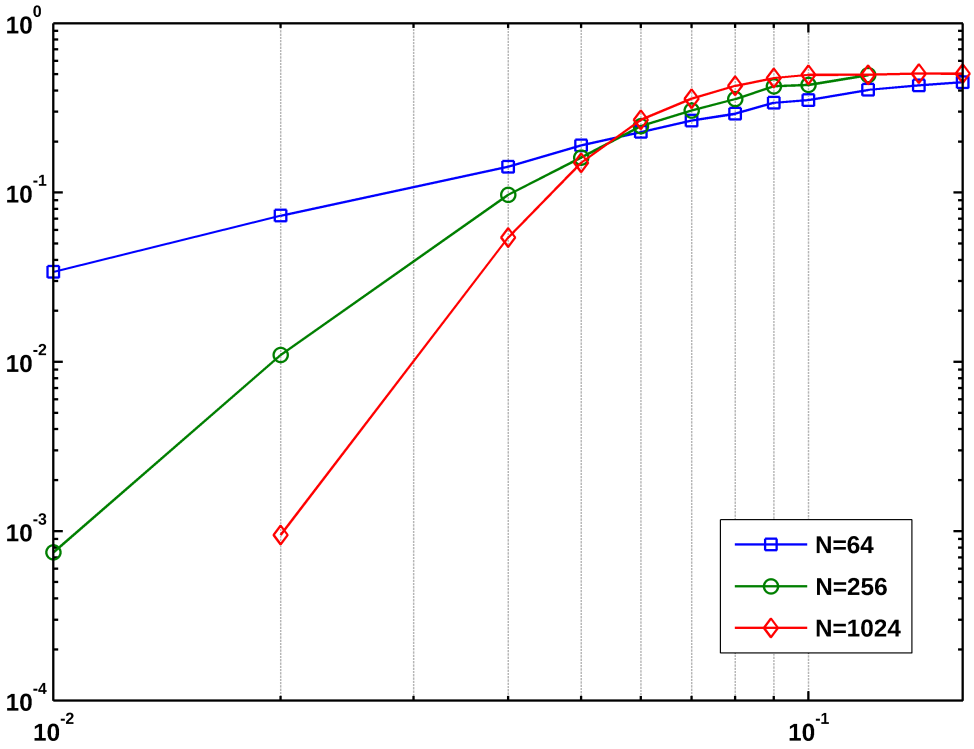


Figure 4.9: Decoding performance of standard matching decoding for color code lattices with 64, 256 and 1024 qubits on a logarithmic scale.

APPENDIX A

MISCELLANEOUS CONCEPTS

A.1 Perfect matching

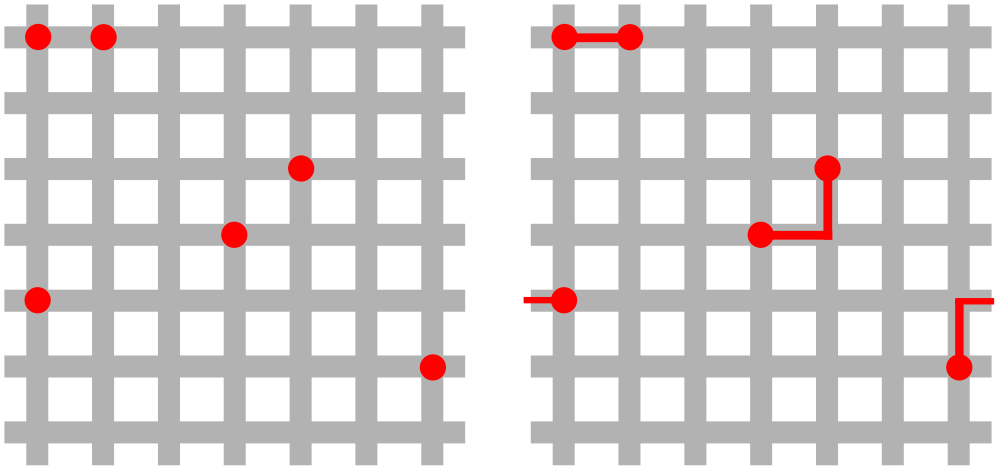
Consider a graph G, with all its vertices forming a set V. A *matching* performed on this graph gives us a set of edges M forming pairs of points from V, such that no two distinct edges in M share a common vertex. A vertex u from V is said to be *unmatched* if for a given matching M, there is no edge in M that involves u. We call it a *perfect matching* when there is no unmatched vertex in V.

Consider a graph G with vertices V. Assume that each edge e connecting points u and v from V has a non-negative weight w_e associated with it. Let W be a set of all edge weights. For a matching M, we define *cost of matching* as:

$$C = \sum_e w_e \quad ; \quad e \epsilon M$$

A matching that minimizes this cost is known as *minimum cost perfect matching*. Our problem of pairing the endpoints of error segments is essentially finding a minimum cost perfect matching for a graph $G = (V, E, W)$. Since the number of endpoints of error segments is always even, a perfect matching always exists.

For the simulation, I have used Blossom V implementation of a minimum cost perfect matching algorithm given by Kolomogorov (Kolmogorov [2009]).



(a) Points with negative syndromes, indicating end-points of error paths

(b) After performing minimum cost perfect matching

Figure A.1: An example of minimum cost perfect matching for toric code

A.2 Manhattan matrix distance

A *Manhattan matrix distance* is the shortest distance between two vertices on a square mesh, provided you travel only along the edges. Figure A.2 shows three equivalent Manhattan distances between two diagonally opposite points.

In case of toric code, probability of an error path connecting two vertices diminishes the more number of edges it contains, most likely error path being one with the least number of edges. The likelihood of an error path can thus be estimated using Manhattan matrix distance. Most likely path will correspond to one with the shortest Manhattan matrix distance between the endpoints. Note that here we will also have to take into account the periodic boundaries. Algorithm 3 shows a simple process for finding Manhattan distance in toric code.

Algorithm 3: Manhattan distance function for toric code

input : indices of two vertices, a and b; code length, L
output: Manhattan distance between a and b;

1 define $A_x = a\%L$ and $A_y = a/L$;
2 define $B_x = b\%L$ and $B_y = b/L$;
3 define $X_{dist} = min\ \{\ |A_x - B_x|,\ L - |A_x - B_x|\ \}$;
4 define $Y_{dist} = min\ \{\ |A_y - B_y|,\ L - |A_y - B_y|\ \}$;

return : $X_{dist} + Y_{dist}$

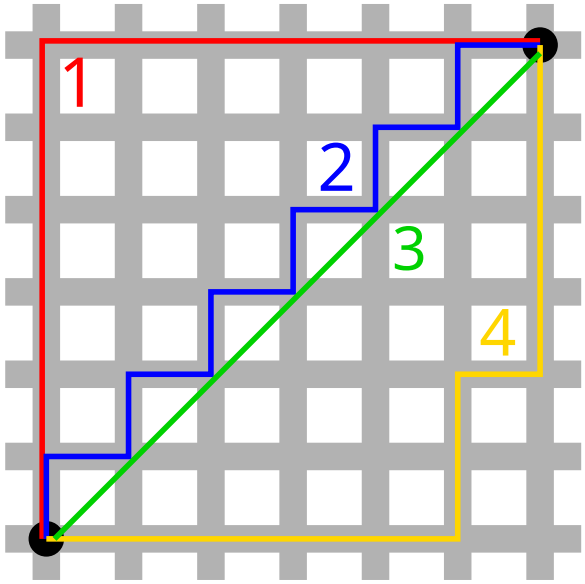


Figure A.2: Path 3 shows the straight line distance between the diagonal points. Paths 1, 2 and 4 are equivalent Manhattan distances between them.

REFERENCES

Bhagoji, A. and **P. Sarvepalli**, Equivalence of 2d color codes (without translational symmetry) to surface codes. *In 2015 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2015.

Bombin, H., **G. Duclos-Cianci**, and **D. Poulin** (2012). Universal topological phase of two-dimensional stabilizer codes. *New Journal of Physics*, **14**(7), 073048.

Bombin, H. and **M. A. Martin-Delgado** (2006). Topological quantum distillation. *Physical review letters*, **97**(18), 180501.

Boost, C. (2012). Libraries.

de Queiroz, S. (2009). Location and properties of the multicritical point in the gaussian and±j ising spin glasses. *PHYSICAL REVIEW B Phys Rev B*, **79**, 174408.

Delfosse, N. and **J.-P. Tillich** (2014). A decoding algorithm for css codes using the x/z correlations. *arXiv preprint arXiv:1401.6975*.

Gottesman, D. (1997). Stabilizer codes and quantum error correction. *arXiv preprint quant-ph/9705052*.

Kitaev, A. Y. (2003). Fault-tolerant quantum computation by anyons. *Annals of Physics*, **303**(1), 2–30.

Kolmogorov, V. (2009). Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, **1**(1), 43–67.

Nagy, B. N. (2003). Shortest paths in triangular grids with neighbourhood sequences. *CIT. Journal of computing and information technology*, **11**(2), 111–122.

Shor, P. W., Fault-tolerant quantum computation. *In Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*. IEEE, 1996.

Wang, D. S., **A. G. Fowler**, **C. D. Hill**, and **L. C. L. Hollenberg** (2009). Graphical algorithms and threshold error rates for the 2d colour code. *arXiv preprint arXiv:0907.1708*.