Portes tolérantes aux fautes pour les codes produits d'hypergraphes

Fault-tolerant gates on hypergraph product codes

par

Anirudh Krishna

Thèse présentée au département de physique

en vue de l'obtention du grade de docteur ès sciences (Ph.D.)

FACULTÉ des SCIENCES

UNIVERSITÉ de SHERBROOKE

Sherbrooke, Québec, Canada, 31 Janvier 2020

Le 31 Janvier 2020

le jury a accepté la thèse de Monsieur Anirudh Krishna dans sa version finale.

Membres du jury

Professeur David Poulin
Directeur de recherche
Département de physique

Professeur Louis Taillefer
Membre interne
Département de physique

Professeur Robert Koenig
Membre externe
Département de Mathematique
Technische Universität München

Professeur Ion Garate
Président-rapporteur
Département de physique

To Kay, Bee and Su

Sommaire

L'un des défis les plus passionnants auquel nous sommes confrontés aujourd'hui est la perspective de la construction d'un ordinateur quantique de grande échelle. L'information quantique est fragile et les implémentations de circuits quantiques sont imparfaites et sujettes aux erreurs. Pour réaliser un tel ordinateur, nous devons construire des circuits quantiques tolérants aux fautes capables d'opérer dans le monde réel. Comme il sera expliqué plus loin, les circuits quantiques tolérant aux fautes nécessitent plus de ressources que leurs équivalents idéaux, sans bruit.

De manière générale, le but de mes recherches est de minimiser les ressources nécessaires à la construction d'un circuit quantique fiable. Les codes de correction d'erreur quantiques protègent l'information des erreurs en l'encodant de manière redondante dans plusieurs qubits. Bien que la redondance requière un plus grand nombre de qubits, ces qubits supplémentaires jouent un rôle de protection: cette redondance sert de garantie. Si certains qubits sont endommagés en raison d'un circuit défectueux, nous pourrons toujours récupérer l'informations.

Préparer et maintenir des qubits pendant des durées suffisamment longues pour effectuer un calcul s'est révélé être une tâche expérimentale difficile. Il existe un écart important entre le nombre de qubits que nous pouvons contrôler en laboratoire et le nombre requis pour implementer des algorithmes dans lesquels les ordinateurs quantiques ont le dessus sur ceux classiques. Par conséquent, si nous voulons contourner ce problème et réaliser des circuits quantiques à tolérance aux fautes, nous devons rendre nos constructions aussi efficaces que possible. Nous devons minimiser le *surcoût*, défini comme le nombre de qubits physiques nécessaires pour construire un qubit logique. Dans un article important, Gottesman a montré que, si certains types de codes de correction d'erreur quantique existaient, cela pourrait alors conduire à la construction de circuits quantiques tolérants aux fautes avec un surcoût favorable. Ces codes sont appelés codes éparses.

La proposition de Gottesman décrivait des techniques pour exécuter des opérations logiques

sur des codes éparses quantiques arbitraires. Cette proposition était limitée à certains égards, car elle ne permettait d'exécuter qu'un nombre constant de portes logiques par unité de temps. Dans cette thèse, nous travaillons avec une classe spécifique de codes éparses quantiques appelés codes de produits d'hypergraphes. Nous montrons comment effectuer des opérations sur ces codes en utilisant une technique appelée déformation du code. Notre technique généralise les codages basés sur les défauts topologiques dans les codes de surface aux codes de produits d'hypergraphes. Nous généralisons la notion de perforation et montrons qu'elle peut être exprimée naturellement dans les codes de produits d'hypergraphes. Comme cela sera expliqué en détail, les défauts de perforation ont eux-mêmes une portée limitée. Pour réaliser une classe de portes plus large, nous introduisons un nouveau défaut appelé trou de ver basé sur les perforations. À titre d'exemple, nous illustrons le fonctionnement de ce défaut dans le contexte du code de surface.

Ce défaut a quelques caractéristiques clés. Premièrement, il préserve la propriété éparses du code au cours de la déformation, contrairement à une approche naïve qui ne garantie pas cette propriété. Deuxièmement, il généralise de manière simple les codes de produits d'hypergraphes. Il s'agit du premier cadre suffisamment riche pour décrire les portes tolérantes aux fautes de cette classe de codes. Enfin, nous contournons une limitation de l'approche de Gottesman qui ne permettait d'effectuer qu'un certain nombre de portes logiques à un moment donné. Notre proposition permet d'opérer sur tous les qubits encodés à tout moment.

Summary

One of the most exciting challenges that faces us today is the prospect of building a scalable quantum computer. Implementations of quantum circuits are imperfect and prone to error. In order to realize a scalable quantum computer, we need to construct fault-tolerant quantum circuits capable of working in the real world. As will be explained further below, fault-tolerant quantum circuits require more resources than their ideal, noise-free counterparts.

Broadly, the aim of my research is to minimize the resources required to construct a reliable quantum circuit. Quantum error correcting codes protect information from errors by encoding our information redundantly into qubits. Although the number of qubits that we require increases, this redundancy serves as a buffer – in the event that some qubits are damaged because of a faulty circuit, we will still be able to recover our information.

Preparing and maintaining qubits for durations long enough to perform a computation has proved to be a challenging experimental task. There is a large gap between the number of qubits we can control in the lab and the number required to implement algorithms where quantum computers have the upper hand over classical ones. Therefore, if we want to circumvent this bottleneck, we need to make fault-tolerant quantum circuits as efficient as possible. To be precise, we need to minimize the *overhead*, defined as the number of physical qubits required to construct a logical qubit. In an important paper, Gottesman showed that if certain kinds of quantum error correcting codes were to exist, then this could lead to constructions of fault-tolerant quantum circuits with favorable overhead. These codes are called quantum Low-Density Parity-Check (LDPC) codes.

Gottesman's proposal described techniques to perform gates on generic quantum LDPC codes. This proposal limited the number of logical gates that could be performed at any given time. In this thesis, we work with a specific class of quantum LDPC codes called hypergraph product codes. We demonstrate how to perform gates on these codes using a technique called code deformation. Our technique generalizes defect-based encodings in

the surface code to hypergraph product codes. We generalize puncture defects and show that they can be expressed naturally in hypergraph product codes. As will be explained in detail, puncture defects are themselves limited in scope; they only permit a limited set of gates. To perform a larger class of gates, we introduce a novel defect called a wormhole that is based on punctures. As an example, we illustrate how this defect works in the context of the surface code.

This defect has a few key features. First, it preserves the LDPC property of the code over the course of code deformation. At the outset, this property was not guaranteed. Second, it generalizes in a straightforward way to hypergraph product codes. This is the first framework that is rich enough to describe fault-tolerant gates on this class of codes. Finally, we circumvent a limitation in Gottesman's approach which only allowed a limited number of logical gates at any given time. Our proposal allows to access the entire code block at any given time.

Acknowledgements

I am extremely fortunate to have had the chance to work with David Poulin as a Ph.D. student. David is a role model both as researcher and adviser. His expertise in a wide range of topics, and interest in both concrete and abstract problems is an inspiration. In addition, David has been patient, kind and generous. I would like to express my sincerest thanks to David for this opportunity.

Inria was my home-away-from-home for a significant duration of my studies. I would like to thank Jean-Pierre Tillich for his guidance during my internship at Inria. Thanks to Jean-Pierre, I gained a special appreciation for classical coding theory. Jean-Pierre's attention to detail also helped me write better.

My path down quantum error correction started with an internship with Daniel Gottesman. My interest in quantum LDPC codes began over lunch time conversations with Daniel at the Blackhole bistro at Perimeter. I'd like to thank Daniel for introducing me to one of the topics that became the focus of my Ph.D.

I'd like to thank Barbara Terhal and Alexandre Blais for inviting me to collaborate with their groups. I learned a lot working on these projects and this helped guide the course of my research. I'd also to like to thank Stephen Bartlett, Steve Flammia, and Ken Brown for inviting me to visit. Each of these visits were impactful, and I enjoyed discussions with you.

Over the past few years, I've had the opportunity to collaborate with inspiring people around the world. My Ph.D. wouldn't have been the same without Nikolas Breuckmann, Christophe Vuillot, Ben Criger; Anthony Leverrier, Antoine Grospellier, Vivien Londe; Tomas Jochym O'Connor; Markus Kesselring; Narayanan Rengaswamy, and Mike Newman. I've learned a lot from all of you.

Sherbooke has been my home for the past three years and it wouldn't have been complete without the team at Sherbrooke who served as friends and counsel. Thank you to Pavithran

Iyer, Andrew Darmawan, Guillaume Dauphinais, Colin Trout, Jessica Lemieux, Maxime Tremblay, Benjamin Bourassa, Yehua Liu, Shruti Puri, Jonathan Gross, Thomas Gobeil and Thomas Baker.

Outside my work environment, I'd like to thank Zoe Xuan Qin for bringing energy and happiness to my life. I would also like to thank Jalaj Upadhyay for being a friend and mentor.

Finally, none of this would have been possible without my family. My parents have been pillars of support, and my brother has been my North star. Regardless of what endeavor I had chosen to pursue, I couldn't have done it without your support. Thank you.

Contents

Sommaire ii

Summary iv

Introduction 1

1 Classical Error Correction 5
- 1.1 Classical error correction . . . 5
- 1.2 Expander codes . . . 8
- 1.3 Chapter summary . . . 16

2 Quantum Error Correction 18
- 2.1 Quantum error correction . . . 19
- 2.2 CSS codes . . . 22
- 2.3 Universal gate sets . . . 24
- 2.4 Transversal gates . . . 25
- 2.5 State injection . . . 26
  - 2.5.1 Magic state distillation . . . 27
- 2.6 Chapter Summary . . . 29

3 The Surface Code 30
- 3.1 Definition . . . 31
- 3.2 Decoding the surface code . . . 35
- 3.3 Defects on the surface code . . . 38
- 3.4 Clifford gates . . . 45
  - 3.4.1 Measurements . . . 46
  - 3.4.2 Resource state preparation . . . 50
- 3.5 Chapter summary . . . 51

4 Hypergraph Product Codes 53

4.1 Background and notation . . . 55
4.1.1 Classical and quantum codes . . . 55
4.1.2 The hypergraph product code . . . 56
4.2 The small-set-flip algorithm . . . 59
4.3 Chapter summary . . . 64

5 Defects on Hypergraph Product Codes **65**
5.1 Punctures . . . 66
5.1.1 Definition . . . 66
5.1.2 Logical Pauli operators for punctures . . . 70
5.2 Wormholes . . . 79
5.2.1 Measurements and hybrid stabilizers . . . 81
5.2.2 Logical Pauli operators for wormholes . . . 84
5.3 Code deformation . . . 87
5.3.1 Non-mixing . . . 87
5.3.2 Code deformation on the hypergraph product code . . . 91
5.3.3 Measurements and traceability . . . 92
5.3.4 Resource states . . . 94
5.3.5 Point-like punctures . . . 96
5.4 Chapter summary . . . 98

Conclusion **99**

Bibliographie **101**

List of Tables

5.1 Summary of properties of punctures. 70

List of Figures

1.1 Factor graph of the repetition code. . . . 9
1.2 Schematic of a factor graph 11

2.1 Quantum codes via the CSS construction. . . . 23
2.2 Error propagation in a circuit. . . . 25
2.3 State injection 27

3.1 The surface code. . . . 32
3.2 X stabilizers of the surface code. . . . 33
3.3 Stabilizers of the surface code. . . . 34
3.4 Logical operators of the surface code. . . . 35
3.5 Errors, stabilizers and logical operators on the surface code. . . . 35
3.6 Half a stabilizer confounds a local decoder. . . . 36
3.7 Illustrating minimum weight matching. . . . 37
3.8 Smooth punctures. . . . 40
3.9 Rough punctures. . . . 41
3.10 Encoding logical qubits in punctures. . . . 42
3.11 Braiding punctures on the surface code. . . . 42
3.12 Creating a wormhole. . . . 43
3.13 Lattice-free representation of wormhole. . . . 44
3.14 Stabilizer and logical generators of the wormhole. . . . 45
3.15 A measurement-based circuit to perform controlled-Z. . . . 46
3.16 The braiding operation. . . . 47
3.17 The stitching operation. . . . 48
3.18 The Y operator of a wormhole is not traceable. . . . 49
3.19 A product of Y operators of the wormhole are traceable. . . . 50

4.1 `flip` fails on hypergraph product codes. . . . 59
4.2 Results of simulation of $5, 6$ codes. . . . 63

5.1 Schematic of factor graphs. 67
5.2 Schematic for a puncture on a hypergraph product code. 68
5.3 Panels (a) and (b) feature a smooth puncture defined on a lattice with only smooth boundaries. The strings of X operators defined only on VV qubits (as in panel (a)) or only on CC qubits (as in panel (b)) are equivalent. Panels (c) and (d) feature a lattice with smooth and rough boundaries, with a smooth punctured carved out from the inside. The logical X shown running from the smooth puncture to the boundary in panels (c) and (d) are equivalent up to an embedded logical X. 77
5.4 Schematic for a wormhole on hypergraph product codes. 81
5.5 A pair of punctures used to encode a single logical qubit on the surface code. 90
5.6 A measurement-based circuit to perform controlled-Z. 91
5.7 A puncture crossing its own path. 94
5.8 A point-like puncture. 97

Introduction

The last decade has witnessed tremendous progress in quantum computation. Quantum computers may be capable of solving certain kinds of problems faster than their classical counterparts. However, we are still far from these applications as constructing a quantum computer is an enormously difficult problem. Quantum information is plagued by noise which impedes our ability to prepare and coherently maintain quantum bits (qubits). The domain of quantum error correction and fault tolerance emerged as a response to this problem and forms a pillar of research in quantum computation.

The article [1] presents the accuracy of one- and two-qubit gates in some devices (see table 1 in [1]). Even with the smallest error rate of 0.01% reported in table 1, a quantum algorithm containing more than $10,000$ logical gates would very likely contain an error. More generally, if the probability of failure of a circuit component is a constant, we are certain to fail as we build larger and larger circuits. Each architecture comes with its own set of advantages and disadvantages. The physical incarnation of the error mechanisms will depend on the experimental system in question. At this early stage, it is difficult to bet on one approach. For this reason, our discussion will abstract away the specifics of particular architectures. Although the actual values will change, errors are not something that will entirely be removed from our systems. It is highly unlikely that we will ever be able to engineer systems to the degree of precision required to run long quantum algorithms on raw qubits. This is disheartening because it seems to imply that quantum computation is impossible.

Quantum error correction is a software method used to construct arbitrarily accurate virtual qubits from underlying faulty physical qubits. In addition to error correction, a quantum computer must also manipulate the encoded information, and these logical operations must be executed in a way that avoids error propagation. The *threshold theorem* guarantees that as long as the probability that a circuit component fails is below some threshold, we can increase the length of the circuit arbitrarily [2, 3, 4, 5, 6]. At the heart of the threshold theorem are objects called error correcting codes. Error correcting codes are ways of storing

information redundantly and we shall review them in chapter 2. This redundancy serves as a buffer – in the event that some physical qubits are corrupted, we can still retrieve the information. Once information is encoded in an error correcting code, we cannot point to any one region and ask if the information resides there. Rather, information is stored in non-local degrees of freedom in such a way that no local read-out can discern the encoded information. By the same token, local errors cannot damage stored information. However making systems robust comes at a cost of increasing the number of qubits we have to control.

Optimizing the storage capacity of quantum error correcting codes is one way to minimize the qubits we need to perform quantum computation. The other concern is the number of extra qubits we need to perform measurements. In addition to the qubits required for storage, we also need extra qubits, i.e. an ancillary system, for performing measurements. Quantum error correcting codes necessitate performing joint measurements on sets of data qubits. Based on the outcome of these measurements, we can deduce the error and thereby perform error correction. The size of this ancilla system could grow considerably based on the technique that we use. These trade-offs will depend on the quantum error correcting code we choose.

Not all quantum error correcting codes are created equal. Some codes offer better protection than others for fixed cost (as measured by figures of merit we shall discuss later). They also permit simple measurement protocols.

Raussendorf and collaborators [7, 8, 9] discovered a fault-tolerant scheme that is naturally defined by local interactions in a two-dimensional geometry and has a relatively large error threshold. In this scheme, logical qubits are encoded in Kitaev's surface code [10, 11] and some logical gates can be implemented by topologically protected operations. Another advantage of the surface code is that performing joint measurements on the data qubits can be done using only a constant number of ancilla qubits. This topological architecture has been the object of intense theoretical studies and is now being pursued by major experimental groups and corporations worldwide (see for example [12, 13] and references contained therein). We shall cover these properties in chapter 3. Experimental demonstrations of fault tolerance will certainly be a milestone experiment in the next few years. The success of the surface code architecture can on the one hand largely be attributed to its simplicity and relatively good performance, but on the other hand this architecture has benefited from over a decade of theoretical development and optimization. In fact, this single architecture has probably received far more theoretical inputs than all other coding schemes combined, so it should not come as a surprise that it has caught the attention of experimentalists.

It is unclear that the surface code will be the architecture we choose to use in the long run. The article [14] shows resource estimates for implementing Shor's algorithm using the surface code (see table 1 in [14]). We can see that even with optimistic assumptions, it will take anywhere between a million and hundred million physical qubits to run the algorithm in a year. When we struggle to control tens of qubits today, these numbers do not appear to be in the realm of the attainable. However we could also interpret these numbers to mean that the surface code is perhaps not the best architecture suited for scalable quantum computation. What class of codes can replace the surface code while maintaining some or all of the features that make it appealing? Does there exist another local code capable of storing more logical qubits and simultaneously able to protect qubits just as well?

Unfortunately, all quantum error correcting codes that can be laid out on a table top with only nearest neighbor connections are limited by construction. Results like [15, 16] place severe restrictions on local quantum error correcting codes. If we wish to increase the capacity of quantum error correcting codes, something has to give.

Motivated by this state of affairs, this thesis will emphasize one particular alternative to the surface code, namely quantum Low-Density Parity-Check (LDPC) codes. Although we will discuss the surface code to help build intuition for LDPC codes, this thesis is *not* a review of the surface code. For a review of quantum error correction from a different perspective, see [17]. The main similarity between the surface code and LDPC codes is that error correction requires only joint measurement on a sparse set of qubits, hence the name. This is an important feature because the experimental complexity of performing multi-qubit measurements generally scales with the number of involved qubits, so it is desirable to keep that number low. In addition to this sparsity constraint, the measurements in the surface code only involve qubits which are geometrically near each other, and this restriction is dropped in more general LDPC codes.

The upshot of this relaxation is a lower *encoding overhead*: much fewer physical qubits are needed to encode a given number of logical qubits to some desired logical accuracy. This is a very desirable feature, but it comes at the cost of using non-local multi-qubit measurements. While this is experimentally very challenging, an in-depth study of the benefits of LDPC codes is needed before deciding if they are worth the additional experimental efforts. This thesis is a step in that direction.

Once quantum information has been encoded in an error correcting code, we need to find ways of manipulating this information to perform computation. These operations must be performed such that in the event of an error on one qubit, the operations do not spread the error to other qubits. Such an avalanche of errors would be disastrous as the quantum

error correcting code can only buffer against a limited number of errors. Thus we seek to perform gate operations in a *fault-tolerant* manner. We could approach this in many ways. For instance, one way to do so would be to directly perform the unitary gate on the error correcting code. If these gates can be implemented in a short duration, it minimizes the amount of time available for potential errors to propagate to many qubits. Whether or not such gates exist depends on the quantum error correcting code, and symmetries it may possess. We shall say more about such codes in chapter 2.

Rather than take this approach, we shall use a framework called *code deformation*. Code deformation involves modifying the code gradually and this transformation eventually returns to the code that we started with. The aim is for this sequence of gradual transformations to have a non-trivial logical effect on the code. We present the first framework to perform fault-tolerant gates on quantum LDPC codes. We shall focus on a particular class of quantum codes called hypergraph product codes [18]. Discovered by Tillich and Zémor in 2009, this class of codes offer an easy way to produce quantum LDPC codes using classical ones. As we use code deformation to perform gates, we run into a non-trivial problem. It is unclear whether the codes that we encounter over the course of code deformation will also be LDPC. It is already a difficult problem to construct quantum LDPC codes, and finding a set of adiabatically connected quantum LDPC codes is challenging at the outset. However the defect-based techniques that are described in chapter 5 demonstrate that this is indeed possible.

Outline of the thesis: We begin by introducing some fundamental ideas in classical error correction in chapter 1. We then overview quantum error correction in chapter 1. These chapters lay out some of the motivation and key ideas in this thesis. We then proceed to describe the surface code in chapter 3. We use the surface code to introduce wormhole defects, and illustrate how these defects work. Some of the material in this chapter appears in [19]. We extend this to hypergraph product codes in the following chapters. We review the definition of the hypergraph product code in 4. Finally in chapter 5 we describe the framework to perform gates on hypergraph product codes. The material described in this chapter appears in [20]. The articles [19] and [20] are the main contributions to this thesis.

There are other articles that I have contributed to over the course of my doctorate studies that pertain to different aspects of quantum error correction and fault tolerance. These articles will be summarized in the appropriate section and will be highlighted as author contributions.

Chapter 1

Classical Error Correction

The theory of classical error correction is the backbone of the theory of quantum error correction. Kick started by Richard Hamming [21], classical error correction studies how to reliably transmit information across unreliable transmission. In contrast to the work by Shannon [22], Hamming's work is less about existence proofs and more about concrete, achievable codes. It has a wide array of applications, from satellite communications [23] to WiFi.

We begin by introducing the basics of error correcting codes, and some salient ideas that carry over to the quantum realm. With an eye towards covering some recent developments in quantum error correction, we review relevant material on expander codes. We then briefly discuss decoding strategies for LDPC codes and introduce the famous Sipser-Spielman decoder [24]. The interested reader is pointed to the textbook by Richardson and Urbanke [25] for more details.

1.1 Classical error correction

Assume that Alice wished to transmit a bit, either 0 or 1 to Bob. They can only communicate with each other via a noisy channel $\mathcal{N}$ which flips bits with probability p. For instance, perhaps they store their information in the spin degree of freedom of an atom which could spontaneously flip. Thus if Alice sends Bob a bit $x = 0$ it is possible with probability p that Bob receives a $y = 1$.

A simple way to overcome this problem is to send the information *thrice* – Alice instead sends Bob 000 if she wishes to send 0 and 111 if she wishes to send 1. If a single bit was

flipped, then Bob can still recover the information by informed guessing. To deduce the message, Bob will simply take a majority vote.

This is the simplest example of a *linear* code, i.e. the space of codewords can be expressed as a linear subspace. All spaces that we shall deal with for the rest of this chapter are defined over the field $\mathbb{F}_2$. This field has only two elements $\{0, 1\}$ and is equipped with addition operation modulo 2 ($1 + 1 = 0 \pmod 2$). We shall use $\mathbb{F}_2^n$ and $\mathbb{F}_2^{m\times n}$ to denote a vector space of n elements and the space of $n \times m$ matrices over $\mathbb{F}_2$ respectively.

Linear codes can be expressed in terms of a generator matrix, which in this case is $G := (1, 1, 1)^t \in \mathbb{F}_2^{1\times 3}$. If Alice wished to send the message $m \in \mathbb{F}_2$, then she transmits $x = Gm$. Bob receives some potentially corrupted word $y \in \mathbb{F}_2^3$. In other words, for some vector $e \in \mathbb{F}_2^3$, Bob receives the word $y := x + e$ when Alice transmits x. To recover the transmitted word, Bob will check successive bits of y to see if they have the same value. If two successive bits do not have the same value, then Bob has detected an error. He can attempt to undo it but which of the two bits that he's checked are erroneous?

This checking process is represented using a parity check matrix H which in this case, is defined as

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} .$$

The parity check matrix obeys $HG = 0 \pmod 2$. Each row of the parity check matrix H shall be referred to as a check. The first row of H corresponds to the first check to see if the first two bits are the same, and the second row corresponds to the check to see if the last two bits are the same.

The relation between the rank of H and the number of codewords in the codespace is a matter of simple linear algebra. Suppose the codespace carried k bits. Then the parity check matrix has to have $n - k$ independent rows. In general, it could have $m \geq n - k$ rows as some of the rows could be redundant.

Bob computes the error syndrome $s = \mathrm{H}\, y$, which would be all 0 in the absence of errors. Suppose a codeword x is affected by an error e, i.e. we receive the message $y = x + e$. The corresponding syndrome is $s = \mathrm{H}\, y = \mathrm{H}\, x + \mathrm{H}\, e = \mathrm{H}\, e$. Hence the syndrome provides direct information about the error e.

Suppose y has been affected by a single error, i.e. e has exactly one 1. It can easily be seen that a flip of the first bit would produce the syndrome $\mathrm{H}(1, 0, 0)^t = (1, 0)^t$, a flip of the

second bit would produce $\mathrm{H}(0,1,0)^t = (1,1)^t$, while a flip of the last bit would produce $\mathrm{H}(0,0,1)^t = (0,1)^t$. Since these are unique, the syndrome can diagnose each single-bit error, which Bob can correct by flipping the same bit again. This technique has its limits. Indeed, if the first two bits are flipped, the syndrome will be the same as if the third bit had been flipped, $\mathrm{H}(1,1,0)^t = H(0,0,1)^t$. Bob is therefore not able to discriminate these two cases, leading to a failure to recover the transmitted information. This reduces the probability of error from p to $O(p^2)$.

The weight of a string $u \in \mathbb{F}_2^n$ is the number of non-zero elements that appears in u. The distance d of the code is the minimum weight of an element u in the codespace. This represents the minimum number of bits that need to be flipped in order to jump from one codeword in the space to another. Thus if d single-bit errors were to accumulate, we could have a logical error. A linear code over n physical bits carrying k logical bits and distance d is denoted $[n,k,d]$.

If we wished to make communication more robust, it would be natural to generalize this model. By repeating the bit we wish to transmit n times, we can increase the probability that Bob correctly decodes the message. To leading order, the probability of error will fall as $O(p^{n/2})$ and eventually this can be as low as desired. However, there is a danger in letting the repetition code serve as the basis for our intuition. In particular, we are still only able to encode 1 bit regardless of how many bits we transmit. Can we do better? Can we pack more than 1 message into a block of n bits?

In a landmark paper in 1948, Claude Shannon answered this very question [22]. The *rate* is a figure of merit for the 'packing efficiency', defined as the ratio of the number of message bits k we can transmit in a given block of n physical bits. The rate of the repetition code is vanishing – it encodes 1 bit into n bits. To state his result informally, Shannon proved that we can transmit information at a *constant* rate, while at the same time achieving error-free communication. In other words, there is a 'wholesale' effect that comes into play. The more physical bits we transmit, the more messages we can pack into this block, and the more reliable the transmission becomes.

In general a classical error correcting code is parameterized as an $[n,k,d]$ code: n is the number of bits, k is the number of encoded bits, and d is the minimum distance, defined as the minimum number of bits that need to be flipped to map one codeword to another codeword. It follows from this definition that a code of minimum distance d can correct up to $\frac{d-1}{2}$ bit flip errors. The repetition code above is a $[3,1,3]$ code. A good code is one for which k and d scale proportionally to n. In other words, this facilitates a 'wholesale effect' – the more the physical bits we use, the more logical bits we can encode. The cost of encoding

a logical bit thus becomes constant. At the same time, the number of errors this code can handle also increases.

1.2 Expander codes

We have seen that when given a parity check matrix H and a transmitted codeword x, the received word y can differ from transmitted codeword by some error e, i.e., $y = x + e$. The error syndrome $s = \mathrm{H}\, y = \mathrm{H}(x + e) = \mathrm{H}\, e$ (recall that $\mathrm{H}\, x = 0$ for all codewords) gives us partial information about the error. But since e is a string of n bits and s contains only $n - k$ bits, it is not sufficient to uniquely determine the error e. Thus, *decoding* entails guessing the error e given the syndrome s. It can formally be expressed as a Bayesian inference problem, but this problem is generally too hard to solve exactly [26] and therefore some constraints on the H must be imposed and/or approximations must be made.

The theme of modern coding theory is to find a good decoding algorithm first and then work backwards to design a code that optimizes performance under this decoding algorithm. Thus, much like the wand picking the wizard in Harry Potter, it is the decoding algorithm that has chosen the code rather than the other way around. These trends have been very successful and have resulted in efficient, capacity-achieving classical codes [27, 28, 29]. These are not mere theoretical curiosities. The website `www.ldpc-decoder.com` has compiled a list of the myriad uses of LDPC codes such as in the 802.11n WiFi standard. Polar codes will be used in parts of 5G communications [30].

In a random parity check matrix, a check could be connected to up to $O(n)$ bits on average. If we noticed that the i-th check was dissatisfied, we learn that at least one of the bits involved in this check has an error, but this provides very little information about any particular bit being flipped. In fact, the probability that any given bit in the support of the check is affected falls exponentially as the size of the check increases (The support of a check is the set of bits that the check acts on). On the other hand if we limited the number of bits in the support of a check to grow very slowly or even remain constant, then we can gain a lot more information about the location of the error. An LDPC code refers to a code family where each check only involves a constant number of bits as a function of n, the block size.

LDPC codes are designed for a set of graph-based algorithms which fall under the broad umbrella of so-called *belief propagation* algorithms. Although the exact rules could vary, belief propagation refers to a whole host of algorithms which involve nodes on a graph passing messages to their neighbors. The Sipser-Spielman decoder we shall discuss here is

a version of a decoding algorithm where the nodes pass bits to each other. In general, they could pass real values, or even probability distributions.

We can represent a code $\mathcal{C}$ by its *factor graph*, also called *Tanner graph*. The factor graph $\mathcal{G}$ is a *bipartite* graph meaning that it has two sets of nodes V and C. Let $\mathcal{G} = (V \cup C, E)$ be a bipartite graph, then the edges are $E \subseteq V \times C$. The sets $V = \{1, ..., n\}$ and $C = \{1, ..., m\}$ correspond to the bits and checks in the code. Every bit in the error correcting code is assigned a node $v \in V$, also known as a variable node, and every check in the code is associated to a node in $c \in C$, also known as a check node. All edges in the graph are between a node $v \in V$ and a node in $c \in C$ – never between two nodes in V or two nodes in C. We draw an edge between check node c and the variable node v if v is in the support of c. Equivalently, if the code has parity check matrix H, its factor graph has an edge between c and v if and only if $\mathrm{H}[c, v] = 1$.

As an example, consider the factor graph of the $[3, 1, 3]$ repetition code above. We choose

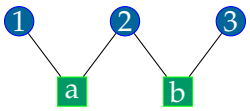


Figure 1.1 Factor graph of the repetition code.

to denote variable nodes by blue circular nodes and check nodes by green square nodes. Furthermore variable nodes have been indexed using numbers whereas checks have been indexed using lower-case letters. Given a word $x \in \mathbb{F}_2^n$, we shall associate the bit x_v with the variable node $v \in V$.

In this section we shall explore codes that are linked to graphs called expander graphs. Intuitively, an expander graph is one for which any small portion of the graph seems to be growing very quickly. If you were to stand on this set of nodes and look out, you'd find that the set of nodes connected to small portion of the graph is bigger than that portion itself. There is a joke that there are as many definitions of expanders as there are people that use them [25]. The definition we use here, as well the exposition of expander codes mirror that of Prof. Madhu Sudan [31].

We begin by defining the neighborhood of a graph – given a subset of nodes S, the neighborhood refers to all the nodes that are connected to S. The following definitions apply to a bipartite graph $\mathcal{G} = (V \cup C, E)$ such that $|V| = n$ and $|C| = m$. The set of edges $E \subseteq V \times C$ is a subset of $V \times C$. Thus edges are of the form (u, c) where $u \in V$ and $c \in C$ with the left part representing the bits, and the right part representing the checks.

For ease of exposition, we shall assume that the graph is *bi-regular*: all the variable nodes

have degree Δ_V and all the check nodes have degree Δ_C. The degree refers to the number of edges emanating from a given node.

Definition 1 (Neighborhood)**.** *For $S \subseteq V \cup C$, the neighborhood $\Gamma(S)$ is the set*

$$\Gamma(S) = \{a | \ \exists b \in S : (a,b) \text{ or } (b,a) \in E\} \ .$$

An expander graph is one for which the size of the neighborhood of S is bigger than the size of S.

Definition 2 (Expander)**.** *Let γ, δ be some constants. $\mathcal{G}$ is a (γ, δ)-expander if for $S \subseteq V$,*

$$|S| \leq \delta n \implies |\Gamma(S)| \geq \gamma |S| \ .$$

The notation $|S|$ here refers to the size of the set S. A good expander is one with large γ. Notice that there is a directionality to this expansion. Expanding from C to V is easy because there are more nodes in V. It is not all that surprising if a small set of nodes in C is connected to more nodes in V. On the other hand, expanding into C is non-trivial because there are fewer nodes and we risk overlap. From the point of view of an error correcting code, the expansion property lower bounds the number of check nodes that will detect an error pattern S. The larger γ, the more checks that detect the error, which in turn makes it more likely for us to flag the error. Contrast this with the repetition code where regardless of how large a connected error is, only two checks ever see it. This expansion is conditional on the parameter δ – we are guaranteed expansion only if we consider a set of size less than some fraction δ of vertices in V. We shall that this parameter is intimately related to the distance of codes defined on this graph.

Of course, the risk with this intuition is that we could be over-counting – some vertices in S may have common neighbors in $\Gamma(S)$. Recall that the arithmetic of a check is performed mod 2. If a check is connected to an even number of erroneous bits, then it will not be UNSAT. It is thus informative to know the number of unique neighbors in the neighborhood.

Definition 3 (Unique neighborhood)**.** *For $S \subseteq V$, the unique neighborhood $\Gamma^+(S) \subseteq C$ is*

$$\Gamma^+(S) = \{c \in C | \ \exists \text{ unique } u \in S : (u,c) \in E\} \ .$$

Since each check only counts the parity of its neighborhood, an error could potentially go undetected if the error touches each check an even number of times. The unique neighbor-

hood guarantees that since the check is only connected to a single element of S, it cannot be 'turned off' by another error within S.

A unique expander is then naturally defined as follows.

Definition 4 (Unique expander). *$\mathcal{G}$ is a $(\widetilde{\gamma}, \delta)$-unique expander if for $S \subseteq V$,*

$$|S| \leq \delta n \implies |\Gamma^+(S)| \geq \widetilde{\gamma}|S| \,.$$

The key property of expander graphs is that if $\mathcal{G}$ is a good expander, then it is also a good unique expander.

Lemma 5. *Assume $\gamma > \Delta_V/2$. If $\mathcal{G}$ is a (γ, δ)-expander, then G is a $(2\gamma - \Delta_V, \delta)$-unique expander.*

Proof. Consider the subgraph of $\mathcal{G}$ induced by $S, \Gamma(S)$ and the edges that run between these sets. Partition the space $\Gamma(S)$ into $\Gamma^+(S)$ the set of unique neighbors and $T := \Gamma(S) \setminus \Gamma^+(S)$, the set of non-unique neighbors.

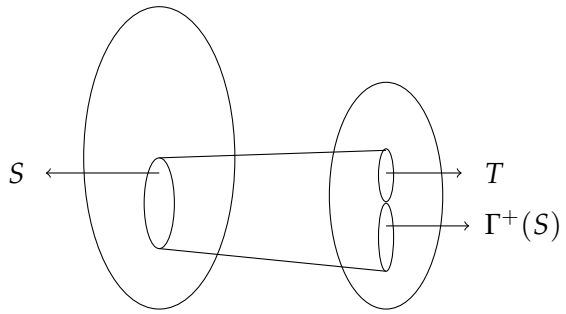


Figure 1.2 A representation of the set $S \subseteq V$ and its neighborhood.

Note that we must have

$$\left(|\Gamma^+(S)| + |T|\right) = \Gamma(S) \geq \gamma|S| \,. \tag{1.1}$$

On the other hand we can count the number of edges leaving both S and $\Gamma(S)$ to conclude that

$$(|\Gamma^+(S)| + 2|T|) \leq \Delta_V|S| \,. \tag{1.2}$$

Subtracting eq. (1.2) from twice eq. (1.1) yields the desired bound. □

This lemma is at the heart of this analysis. It is important because it implies that the distance of the code is lower bounded.

Lemma 6. *Let $\mathcal{G}$ be a (γ, δ)-expander such that $\gamma \geq \Delta_V/2$. Then the distance of the associated code $\mathcal{C}(\mathcal{G})$ is δn.*

Proof. The distance of a code is the weight of the smallest codeword. Codewords by definition will not be detected by any checks. Let e be an error such that its support is $S \subseteq V$. Therefore at least one check will flag the pattern corresponding to $|S|$. □

Expander graphs are interesting because they have a deceptively simple decoding algorithm called `flip` [24].

For the rest of the analysis we will require that $\gamma \geq 3\Delta_V/4$. Let x be the transmitted codeword and y be the received word. A check node c is said to be unsatisfied (or UNSAT) if the syndrome s_c

$$s_c := \sum_{v \in \Gamma(c)} x_v = 1 \pmod 2 ,$$

and satisfied (or SAT) otherwise. Equivalently, we shall say that the c-th syndrome bit is 1.

The decoding algorithm `flip` is shown in algorithm 1. The first few lines of the algorithm

Algorithm 1 `flip`

Input: Received word $y \mathbb{F}_2^n$, syndrome $s(e) \in \mathbb{F}_2^m$.
Output: $w \in \mathbb{F}_2^n$

$w := y$ ▷ Update w iteratively
$\mathcal{F} = \varnothing$ ▷ Flippable vertices
for $u \in V$ **do** ▷ Setup phase: update variable nodes
    If u has more UNSAT than SAT neighbors, add it to $\mathcal{F}$
end for
while $\exists u \in \mathcal{F}$ **do** ▷ Flip while flippable vertices exist
    flip w_u
    Update $\Gamma(u)$, and decide whether they are UNSAT
    Update $\Gamma(\Gamma(u))$ and decide whether they are in $\mathcal{F}$
end while
Return w

are the setup phase. We shall iteratively maintain the word w until we arrive at the answer. If the algorithm terminates with no UNSAT checks, then we return w as the transmitted

codeword. The precomputation to obtain the syndrome is linear time. Indeed, there are $m = O(n)$ check nodes and each has constant degree, so we can compute the syndrome vector $s(e)$ in linear time.

The setup cost is linear time. There are n variable nodes and we can decide whether each node ought to be added to $\mathcal{F}$ in constant time as it only has a constant number of neighbors.

Next, we study the main part of the algorithm within the while loop. Each step of the while loop is rather simple and takes constant time. How long does the while loop run? The number of unsatisfied checks is decreasing monotonically at each step. This immediately lets us prove the following claims.

Claim 7. *Let $\mathcal{G} = (V \cup C, E)$ be a bipartite graph with $|V| = n$ and $|C| = m$ be a (γ, δ)-expander graph. Let the total number of errors be n_e, i.e. if $y = x + e$, then $n_e = \mathrm{wt}(e)$. If we decode using* `flip`*, then*

1. *the decoding time is less than m.*
2. *the decoding time is less than $\Delta_V n_e$.*

Proof. We shall prove each claim in turn.

1. This is evident because the number of unsatisfied checks is monotonically decreasing. Since there are at most m checks, the algorithm will take time at most m.
2. If there are n_e errors, then the number of unsatisfied checks is at most $\Delta_V n_e$. As above, we make use of the fact that the number of unsatisfied checks is monotonically decreasing to arrive at the claim.

□

When does the `flip` algorithm return to the right codeword?

Claim 8. *Let $\mathcal{G} = (V \cup C, E)$ be a bipartite graph with $|V| = n$ and $|C| = m$ be a (γ, δ)-expander graph such that $\gamma \geq 3\Delta_V/4$. Let the total number of errors be n_e, i.e. if $y = x + e$, then $n_e = \mathrm{wt}(e)$. If $n_e < \delta n/(\Delta_V + 1)$, then the decoding algorithm is guaranteed to succeed.*

Proof. At each step of the algorithm, only a single bit is flipped. The weight of the error e is initially n_e. Since the algorithm is guaranteed to conclude before $\Delta_V n_e$ steps, then the weight of the error can at most become $(\Delta_V + 1)n_e$.

For the sake of contradiction, suppose that the algorithm has reached its last step. There are still uncorrected errors, but `flip` finds no more variable nodes to flip. In other words, there exist no variable nodes u such that they are connected to more UNSAT checks than SAT checks. However, since the weight of the error set is now at most $(\Delta_V + 1)n_e < \delta n$, we can invoke the unique expander property. Therefore, if we let S denote the support of the error e, then it follows from lemma 5 that

$$\begin{aligned} |\Gamma^+(S)| &\geq (2\gamma - \Delta_V)|S| \\ &\geq (\Delta_V/2)\,|S|\,. \end{aligned}$$

The second inequality follows from the assumption that $\gamma \geq 3\Delta_V/4$. Since the neighborhood of S contains at most $\Delta_V|S|$ elements, this means that there exists at least one variable node in S that has more unsatisfied neighbors than satisfied neighbors. Therefore this cannot be the penultimate step as there is at least one more variable node to flip. □

This shows that the decoding algorithm terminates, and terminates on a codeword. How do we know that this is the *right* codeword? By assumption, the total number of errors on the word never exceeds $(\Delta_V + 1)n_e < \delta n$. However, since the distance of the code is δn, we could not have mapped to the wrong codeword. In other words, if the algorithm had mapped us to some other word $x' \in \mathcal{C}$, then we would have $h(x, x') \geq \delta n$. This is impossible.

The `flip` algorithm takes as input a corrupted codeword y and returns the codeword x that was most likely transmitted. We have assumed that computing the syndrome in the setup phase of the algorithm could be done perfectly. When we consider the quantum equivalent, we shall see that we cannot directly peek at the quantum state, and our measurements might be imperfect. We will need to perform error correction in a *fault-tolerant* manner to ensure that we can overcome this. Although we will not cover the quantum proof because it is complicated, we offer a simpler, classical analogy.

In the face of syndrome errors, it will turn out that we cannot perform perfect error correction. Rather, we will hope to merely *reduce* the number of failures on the code based on the number of errors on the syndromes. Buried within Spielman's construction for efficient encoders [32] based on expander graphs is a proof that there exist good *error-reduction codes* (see lecture 18 of Prof. Sudan's lectures). This construction merely uses `flip` again, showing that this simple algorithm suffices to reduce the error. In this generalized setting, `flip` will accept as input both the corrupted codeword, as well as the corrupt syndrome and output the best possible word.

The construction begins with a n bit codeword $x \in \mathbb{F}_2^n$. After transmission, we receive $y := x + e$, for some error $e \in \mathbb{F}_2^n$. Let $S = \{v | y_v \neq x_v\} \subseteq V$ be the set of indices corresponding to error locations. Similarly, let $f \in \mathbb{F}_2^m$ be the error on syndromes, i.e. the incorrect syndrome $s'(y)$ is defined as $s'(y) = s(y) + f$ for. Let $E \subseteq C$ be the subset of check nodes whose syndromes are inferred incorrectly, i.e.

$$E = \left\{ c \middle| f_c = 1 \right\} .$$

In other words, these could correspond to either check nodes that are supposed to detect an error but do not, or check nodes that detect a phantom error.

We shall show the following result.

Lemma 9. *Let $\mathcal{G} = (V \cup C, E)$ be a $(\Delta, 2\Delta)$ bi-regular (γ, δ)-expander. Suppose $\Delta > 8$ and $\gamma > \frac{7}{8}\Delta$. Upon transmission of x, let the received word be $y := x + e$ and the syndrome be $s'(y) = s(y) + f$. If the total number of errors is $n_t \leq \delta n/(\Delta + 1)$, then $\texttt{flip}(y, s')$ outputs a word w such that the Hamming weight of $w - x$ is upper bounded by $8|E|/\Delta$.*

Proof. Recall that the current state of the `flip` algorithm is labeled $w \in \mathbb{F}_2^n$. Its syndromes corresponding to the c-th constraint is denoted $s_c(w)$ is said to be correct or SAT if

$$s_c(w) = \sum_{u \in \Gamma(c)} w_u \pmod 2 .$$

The idea here shall be that rather than iterate till all checks are SAT, we merely iterate till every message bit is adjacent to more SAT constraints than UNSAT.

Let $n_t < \delta n/(\Delta + 1)$ be the total number of errors in both the codeword and the syndrome. The initial number of UNSAT constraints $\leq \Delta n_t$ so the algorithm terminates in (at most) Δn_t iterations.

Over the course of the algorithm, each iteration flips at most one bit. Since the total number of iterations is upper bounded, the total number of message errors is upperbounded by $(\Delta + 1)n_t < \delta n$.

Note that $\Gamma^+(S) \setminus E \subseteq \text{UNSAT} \subseteq \Gamma(S) \cup E$. The first containment follows because the unique neighborhood of S will ideally detect the error pattern e. However, because the syndromes could have errors, it is only $\Gamma^+(S) \setminus E$ that is guaranteed to be UNSAT. In other words, this is the set of checks that ought to detect an error, but do not.

The second containment follows because the set of UNSAT checks either correctly flags a real error in S or is detecting a phantom error and therefore in E.

This implies the following inequality: $|\text{UNSAT} \cap \Gamma(S)| \geq (2\gamma - \Delta)|S| - 2|E|$. This follows because for two sets A, B, we have the identity $|A \cap B| = |A| + |B| - |A \cup B|$. Applying this above, we have

$$\begin{aligned} |\text{UNSAT} \cap \Gamma(S)| &= |\text{UNSAT}| + |\Gamma(S)| - |\text{UNSAT} \cup \Gamma(S)| \\ &= |\text{UNSAT}| + |\Gamma(S)| - |\Gamma(S) \cup E| \\ &\geq |\Gamma^+(S) \setminus E| - |E| \\ &\geq (2\gamma - \Delta)|S| - 2|E| \,. \end{aligned}$$

This set is what we ultimately care about because it is the set that will determine if `flip` will terminate; if

$$\frac{|\text{UNSAT} \cap \Gamma(S)|}{|S|} > \frac{\Delta}{2}$$

then we are not done, as this means that at least one element of S has more than $\Delta/2$ unsatisfied neighbors. Equivalently, if the algorithm has stopped, then

$$\begin{aligned} 2\gamma - \Delta - 2\frac{|E|}{|S|} &\leq \frac{\Delta}{2} \\ 2\frac{|E|}{|S|} &\geq 2\gamma - \frac{3\Delta}{2} \geq \frac{\Delta}{4} \\ |S| &\leq \frac{8}{\Delta}|E| \,. \end{aligned}$$

The second chain of inequalities follows because $\gamma \geq 7\Delta/8$. Thus if $\Delta > 8$, we are guaranteed that the number of residual errors on the codeword is upper bounded. □

1.3 Chapter summary

In this chapter, we reviewed the basic definitions of classical error correcting codes over the binary alphabet. We defined the central characteristics of a linear code – the number of bits n, the number of encoded bits k and the distance d. The distance was defined as the minimum number of bits that had to undergo an error for us to jump from one codeword in the codespace to another. We understood that codes could be defined using the parity check matrix. The parity check matrix checks that each codeword obeys certain linear constraints.

We proceeded to discuss an important class of classical codes called expander codes. We reviewed the definition of a bipartite graph $\mathcal{G}$ and how to associate a code to a graph. We noted that if the graph $\mathcal{G}$ is an expander graph, then many useful properties follow. Firstly, the distance of the associated code immediately follows from the expansion property. Furthermore, expansion also implies that the associated code possesses a simple decoding algorithm. This algorithm is called `flip` and simply flips bits to minimize the weight of the syndrome. We showed that this simple algorithm does indeed work if the graph $\mathcal{G}$ is a good expander. Finally we concluded by discussing how `flip` is fault tolerant. In other words, even if some of the syndrome bits are corrupted, `flip` succeeds in reducing the number of errors.

Chapter 2

Quantum Error Correction

How do we construct a quantum computer if circuit components are not perfect? As the size of the circuit increases, so does the likelihood of an error. The threshold theorem, one of the crown jewels of the theory of quantum computation, guarantees that we can in fact execute quantum computations of arbitrary length.

The key idea behind fault tolerance is to use *quantum error correcting codes*. Quantum error correcting codes serve as a buffer against noise – so long as too many errors do not accumulate, we can still recover the quantum information they encode. Error correcting codes thus permit us to simulate the circuit that we want to implement using another circuit that is more robust. This can then be repeated - we can encode the simulation in a simulation and reduce the error further until the desired level of robustness is achieved. The process is called concatenation and is the ingredient behind the first proofs of fault tolerance.

At present, we believe that concatenation by itself will not suffice to build a quantum computer as the number of times we can reasonably concatenate a code is limited. Most efforts are focused on the surface code architecture, which in its simplest form does not involve concatenation. Another important difference is that the *stabilizer generators* of both surface codes and LDPC codes which we shall study later act on a constant-bounded number of qubits. These are much smaller than the number of errors t that the codes can correct. This contrasts with concatenated codes where each stabilizer acts on $\mathcal{O}(n) \gtrsim t$ qubits, so extra care must be taken when measuring a stabilizer in order not to create an uncorrectable error in the process. Surface codes and LDPC codes thus have simpler syndrome extraction circuits. For a complete review of the fundamentals, see [33].

In this chapter we shall lay the foundation for what follows. We start by defining quantum error correcting codes and describing necessary and sufficient conditions for a quantum

code to be effective. Over the course of the chapter, we shall discuss guidelines for code construction. These guidelines, sometimes known as no-go results, are fundamental limits to quantum error correcting codes. We shall then overview techniques to manipulate encoded information and what we require to attain a universal gate set.

2.1 Quantum error correction

A quantum error correcting code is a way of storing quantum information redundantly. Broadly, a quantum error correcting code $\mathcal{C}$ is a subspace of n-qubit states with some important properties. Namely, these n qubits could be subject to some quantum noise channel $\mathcal{E}$. A good code will allow us to recover the information stored in $\mathcal{C}$ by undoing the effect of the noise channel $\mathcal{E}$.

At first glance, quantum error correction appears to be a wholly different enterprise than its classical counterpart. Quantum states could exist in coherent superpositions and measurement could potentially collapse the state. Further complicating the matter, errors on a qubit could be continuous rotations and therefore a continous set of errors against which to defend. However, the miracle of quantum error correction guarantees that it is possible to address these issues. We discuss each of them in turn.

Stabilizer group: A code $\mathcal{C}$ is 2^k-dimensional if it maps a k-qubit state $|x\rangle$, where $x \in \{0,1\}^k$ is some k bit string, to an n-qubit state $|\overline{x}\rangle$ for some $n > k$. Such a code space is spanned by vectors $\{|\overline{x}\rangle\}_{x\in\{0,1\}^k}$, where $|\overline{x}\rangle$ denotes the encoded version of the k qubit state $|x\rangle$. Each codeword $|\overline{x}\rangle$ is itself the superposition of several computational basis states. We first address how to construct such states such that it facilitates measurement of the code space without collapsing the superposition. Codes shall be defined as the common eigenspace of a set $\mathcal{S}$ of commuting Pauli operators on n qubits.

$$\mathcal{C}(\mathcal{S}) = \{|\psi\rangle \,|\, |\psi\rangle \in \mathbb{C}^{2^n}, S\,|\psi\rangle = |\psi\rangle \,\forall S \in \mathcal{S}\}\,. \tag{2.1}$$

The set $\mathcal{S}$ is referred to as the stabilizer group and the generators of this group are called the stabilizer generators. Although the state ψ is in superposition, the stabilizer conditions stipulate that the code state can be measured without collapsing the state because it is an eigenstate of the stabilizer operator. In other words, the operators in $\mathcal{S}$ do not form a complete set of observables, so specifying their eigenvalue leaves some degenerate subspace where information can be stored. Equivalently, the code space $\mathcal{C}$ can be specified by $n-k$ independent stabilizer generators $\{S_i\}_{i=1}^{n-k}$.

If the codewords of $\mathcal{C}$ are corrupted due to some noise channel $\mathcal{E}$, when can we recover the encoded quantum information? The fundamental theorem of quantum error correction due to Knill and Laflamme [34] states the conditions under which a quantum channel can be error-corrected by a specific code $\mathcal{C}$.

Theorem 10 (Knill-Laflamme)**.** *Let $\mathcal{E}$ be a noisy channel whose Kraus operators are $\{E_i\}_{i=1}^m$, i.e. the action of $\mathcal{E}$ on an n-qubit density matrix ρ is described as*

$$\mathcal{E}(\rho) = \sum_i E_i \rho E_i^\dagger \ .$$

The code $\mathcal{C}$ is robust against the noise channel $\mathcal{E}$ if and only if

$$\langle \overline{x} | \, E_i^\dagger E_j \, | \overline{y} \rangle = c_{ij} \delta_{xy} \ , \tag{2.2}$$

where $C = c_{ij}$ is some matrix.

The Knill-Laflamme condition helps us understand how we can correct against a very large class of (potentially continuous) errors. A key feature of eq. 2.2 is that the constants c_{ij} do not depend on the codewords. A set of errors that obeys the Knill-Laflamme condition forms a linearly closed vector space, in the sense that if the set $\{E_i\}$ obeys 2.2, then so does the set $\{F_j\}$ where each operator F_j is a linear combination of the operators $\{E_i\}$. Since the Pauli operators form an operator basis, any error E_i can be decomposed as a linear combination of Pauli operators. We can therefore construct codes that correct a large number of Pauli errors, and by linearity this will extend to any error that is a linear combination of the correctable Pauli errors.

Of course, the quantum error correcting code is limited and cannot correct against all Pauli errors. One useful way of classifying correctable Pauli errors is by their weights, defined as the number of qubits on which the Pauli operator acts non-trivially. The minimum weight of an error E such that the Knill-Laflamme condition is violated represents the *distance* d of the code $\mathcal{C}$. It represents the minimum number of qubits that need to be affected to map one codeword to another. It follows that the Knill-Laflamme condition 2.2 will hold for any set of errors of weight bounded by $\frac{d-1}{2}$. By linearity, 2.2 will also hold for any set of Kraus operators that are linear combinations of Pauli operators of weight bounded by $\frac{d-1}{2}$. Henceforth we focus on Pauli errors.

How to perform quantum error correction: Suppose a code state $|\psi\rangle \in \mathcal{C}$, undergoes a Pauli error E, $|\psi\rangle \to |\psi'\rangle = E\,|\psi\rangle$. Error correction proceeds by measuring a set of stabilizer generators $\{S_i\}_{i=1}^{n-k}$ on the state $|\psi'\rangle$. In the absence of errors, each of these measurements

returns the outcome $+1$ by definition of the code space $\mathcal{C}$. In the presence of errors however, we get

$$S_i \left|\psi'\right\rangle = S_i E \left|\psi\right\rangle = \left\{ \begin{array}{rcrcll} ES_i \left|\psi\right\rangle & = & E\left|\psi\right\rangle & = & \left|\psi\right\rangle' & \text{if } ES_i = S_i E \\ -ES_i \left|\psi\right\rangle & = & -E\left|\psi\right\rangle & = & -\left|\psi\right\rangle' & \text{if } ES_i = -S_i E. \end{array} \right\} := s_i(E) \left|\psi'\right\rangle . \tag{2.3}$$

These are the only two possibilities since Pauli operators either commute or anti-commute. In either case, the measurement outcome reveals the error syndrome bit $s_i(E)$ which encodes the commutation relation of the stabilizer generator S_i and the error E that afflicted the system. The collection of all measurement outcomes $(s_1, s_2, \ldots s_{n-k})$ is called the error syndrome, and is used to determine the error E that occurred. The syndrome does not uniquely identify the error, so given an a priori distribution on the possible errors $p(E)$, statistical inference is used to identify the most likely error. The algorithm that takes as input an error syndrome and returns an error guess is called a *decoder*. Note that multiple errors could have the same syndrome. We say that errors E and E' are degenerate if for some stabilizer element $S \in \mathcal{S}$,

$$EE' = S . \tag{2.4}$$

Henceforth, such a code shall be called an $[\![n, k, d]\!]$ code. In other words, it uses n physical qubits to encode k logical qubits and is capable of protecting the encoded information against $(d-1)/2$ errors.

Logical operators: The logical operators $\mathcal{L}$ of a stabilizer code map codewords of the code $\mathcal{C}$ to other codewords of the code $\mathcal{C}$. Thus the logical operators map the codespace to itself, although their action on any individual codeword may be non-trivial. All the operators $L \in \mathcal{L}$ obey $L\mathcal{S} = \mathcal{S}L$, i.e. they commute with the codespace. Logical operators thus correspond to undetectable operators – if the code is afflicted by an error corresponding to $L \in \mathcal{L}$, we cannot detect it. Thus the distance d of the quantum error correcting code $\mathcal{C}$ can also be defined as the minimum weight of the logical operators in $\mathcal{L}$. In what follows, this set may sometimes also be represented as $\mathcal{N}(\mathcal{S})$ or the *normalizer* of the stabilizer group. We focus on the logical X and Z operators of this space. Linear combinations and products of these operators generate the entire logical space, and clearly commute with the stabilizers.

For simplicity, suppose the code only carried a single logical qubit, then there is only one pair of logical X and Z operators. Denoted by $\overline{X}$ and $\overline{Z}$, their action on the codespace can

be described as follows. Let $|\overline{0}\rangle$ denote the encoded logical $|0\rangle$ and $|\overline{1}\rangle$ denote the encoded logical $|1\rangle$. Then we have

$$\begin{aligned} \overline{X}\,|\overline{0}\rangle &= |\overline{1}\rangle & \overline{Z}\,|\overline{0}\rangle &= +|\overline{0}\rangle \\ \overline{X}\,|\overline{1}\rangle &= |\overline{0}\rangle & \overline{Z}\,|\overline{1}\rangle &= -|\overline{1}\rangle\ . \end{aligned}$$

These logical operators obey the anti-commutation relations $\overline{XZ} = -\overline{ZX}$.

In general, if the code carries k logical qubits, then the logical operators of the code can be described using $2k$ logical operators or equivalently, k pairs of $\overline{X}_i$ and $\overline{Z}_i$ operators, where the subscript i indicates which logical qubit the operators correspond to. Each pair $\overline{X}_i$ and $\overline{Z}_i$ obeys the corresponding anti-commutation relations.

2.2 CSS codes

CSS codes are a template to form quantum codes from classical codes that obey certain constraints [35, 36]. They are composed of two binary linear codes $\mathcal{C}_Z = [n, k_1, d_1]$ and $\mathcal{C}_X = [n, k_2, d_2]$ such that $\mathcal{C}_Z^\perp \subseteq \mathcal{C}_X \Leftrightarrow \mathcal{C}_X^\perp \subseteq \mathcal{C}_Z$. The space $\mathcal{C}_X^\perp$ and $\mathcal{C}_Z^\perp$ refer to the spaces of vectors that are orthogonal to all vectors in $\mathcal{C}_X$ and $\mathcal{C}_Z$ respectively. The resulting quantum code only contains stabilizers whose elements are all X or all Z. The constraints help guarantee that the resulting stabilizers commute. Suppose the parity check matrices of the codes $\mathcal{C}_X$ and $\mathcal{C}_Z$ are H_X and H_Z respectively. Let G_X and G_Z be the generator matrices for these spaces respectively.

To construct the code, we

1. map the i^{th} row of H_Z to Z stabilizer generator S_i^Z by mapping 1's to Z and 0's to identity; and
2. map the j^{th} row of H_X to X stabilizer generator S_j^X by mapping 1's to X and 0's to identity.

This is depicted in fig. (2.1) where the red regions map to the stabilizer generators.

For $e, f \in \mathbb{F}_2^n$, let $X(e) = \otimes_j X^{e_j}$ and $Z(f) = \otimes_j Z^{f_j}$. Then

$$\begin{aligned} Z(f)\,|w\rangle &= (-1)^{\langle f,w\rangle}\,|w\rangle \\ X(e)\,|w\rangle &= |w \oplus e\rangle\ . \end{aligned}$$

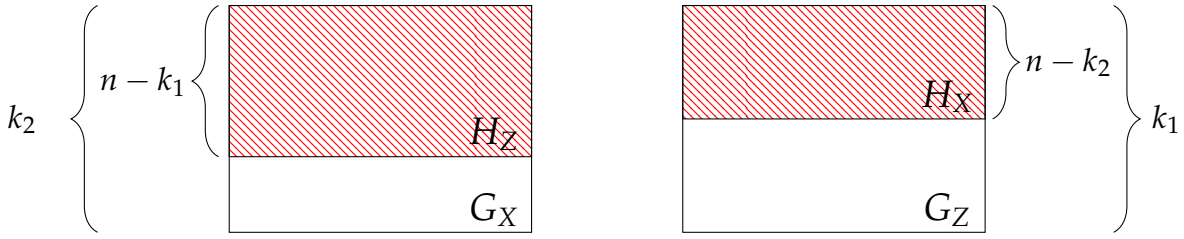


Figure 2.1 Quantum codes via the CSS construction. Two parity check matrices and codes are shown. Red regions map to stabilizer generators.

The notation $\langle f, w\rangle = \sum_i f_i w_i$ denotes the inner product between the strings f and w. With this correspondence between the space of vectors over $\mathbb{F}_2^n$ and Pauli operators, it is easy to verify when two Pauli operators corresponding to $X(e)$ and $Z(f)$ commute. We have the condition $[X(e), Z(f)] = 0$ if and only if $\langle e, f\rangle = 0 \pmod 2$.

The codewords correspond to cosets of $\mathcal{C}_Z/\mathcal{C}_X^\perp$ and hence the code dimension is $k := \dim\left(\mathcal{C}_Z/\mathcal{C}_X^\perp\right) = \dim\left(\mathcal{C}_X/\mathcal{C}_Z^\perp\right)$.

The distance of a CSS code is then expressed as $d = \min\{d_X, d_Z\}$ where

$$d_X = \min_{e\in\mathcal{C}_Z\setminus\mathcal{C}_X^\perp} \mathrm{wt}\,(e) \qquad\qquad d_Z = \min_{f\in\mathcal{C}_X\setminus\mathcal{C}_Z^\perp} \mathrm{wt}\,(f)\ .$$

This corresponds to the minimum weight of an undetectable error of X and Z type respectively.

Guidelines for code construction # 1: BPT bound

We have so far not discussed the geometry of our quantum error correcting code. It would be convenient if we could lay out our qubits in 2 dimensions, i.e. on a table top. Furthermore, it would be nice if we did not need to engineer too many connections – if each stabilizer was only connected to a handful of qubits. Finally, we might want to minimize 'wire crossings', i.e. such that all the qubits in the support of a stabilizer are right next to it. Unfortunately, such codes are highly restrictive.

The BPT bound, named after its discoverers Bravyi, Poulin and Terhal, states that the surface code captures everything we need to know about local codes (up to constant factors). They proved that any $[\![n, k, d]\!]$ code in 2 dimensions defined by local stabilizer generators obeys the following bound:

$$kd^2 \leq O(n)\ .$$

Ignoring constants, this bound is saturated by letting $d = O(\sqrt{n})$. In the next chapter, we shall discuss one such code, called the surface code, which saturates this bound. If we want to get more bang for our buck, we might need to give up the constraint of locality.

2.3 Universal gate sets

A fault-tolerant computational scheme must be realized with a discrete set of universal gates. While physical gates can be tuned continuously, logical gates must belong to a discrete set to be correctable. Suppose for instance that the logical gate $R_x(\theta) := e^{-i\theta\sigma_x}$ was permitted for any real value of θ in some fault-tolerant scheme. If some small error in the execution of the gate yielded the transformation $R_x(\theta + \epsilon)$ instead, this would be indistinguishable from an ideal circuit implementing a different logical gate, so this physical error would be promoted to an uncorrectable logical error.

Each gate in this discrete set can be corrected to any desired accuracy given an appropriate error-correcting code. If the gate set is universal, then any logical gate belonging to a continuum can be *approximated* by a suitable sequence of gates from the discrete universal set. This is called *gate synthesis*.

The most common path to universality is formed of the CNOT, the Hadamard H and phase $S := \sqrt{\sigma_z} = R_x(\frac{\pi}{4})$ which together generate the Clifford group. This is a finite maximal subgroup of the unitary group on n qubits. The Clifford group is the automorphism group of the Pauli group and its elements are completely characterized by their action on the generators of the Pauli group, i.e. on X and Z. For instance, the action of the Clifford group generators are

$$\begin{aligned} HXH^\dagger &= Z & \mathsf{CNOT}(XI)\mathsf{CNOT}^\dagger &= XX \\ HZH^\dagger &= X & \mathsf{CNOT}(IX)\mathsf{CNOT}^\dagger &= IX \\ PXP^\dagger &= iXZ & \mathsf{CNOT}(ZI)\mathsf{CNOT}^\dagger &= ZI \\ PZP^\dagger &= Z & \mathsf{CNOT}(IZ)\mathsf{CNOT}^\dagger &= ZZ \end{aligned} \tag{2.5}$$

It is not universal and in fact Clifford circuits acting on computational basis state inputs and Pauli measurements can be efficiently simulated classically [37, 38]. The fact that it is maximal means that adding any gate to this set produces a universal set of generators.

Common choices for this last gate are $T := \sqrt{S} = R_z(\frac{\pi}{8})$, the Toffoli gate control-control-not, or the controlled-S gate. These all belong to the third level of the Clifford hierarchy, which means that for any Pauli operator P and any one of these third-level gate U, the combination $UPU^\dagger$ is a Clifford gate. This will be important for state injection. In the case of T, efficient compilation algorithms are known [39] that take as input an arbitrary single-qubit gate U, and output a sequence of T and H of length $3\log(\frac{1}{\epsilon})$ that synthesize U to accuracy ϵ.

2.4 Transversal gates

One of our main concerns when designing a fault-tolerant scheme is error propagation. If a qubit has suffered an error and is then involved in a two-qubit gate, then after the gate both qubits are potentially erroneous. If this occurred in a distance $d = 3$ code, then the

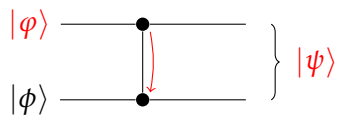


Figure 2.2 The state φ contains an error (indicated in red). After φ and ϕ interact via a 2-qubit gate, the resulting 2-qubit state ψ also potentially contains a 2-qubit error (indicated in red).

two-qubit gate has promoted a single-qubit correctable error to a two-qubit uncorrectable error.

One way to avoid this problem is to never couple two qubits from the same code block. *Transversal gates* acting on single logical qubit are thus tensor products of single-qubit gates $\bigotimes_{j=1}^{n} U_j$. Transversal gates involving logical qubits from two distinct code blocks are tensor products of two-qubit gates $\bigotimes_{k=1}^{n} U_{j_k^1, j_k^2}$ where the first qubit of a pair (j_k^1, j_k^2) belongs to the first code block and the second qubit belongs to the second code block, and each qubit appears in a single pair. While such a two-qubit transversal gate can transform a single-qubit error in a two-qubit error, these two errors will belong to different code blocks, so they will be correctable by the respective code on which they act.

Note that this definition of transversality was motivated by distance 3 codes which can correct at most one error. It ensures that all weight-one errors remain correctable, so if the physical error rate is ϵ, the effective logical error rate will be $\mathcal{O}(\epsilon^2)$. For a code with a larger minimum distance, we could demand that a generalized transversal gate increases the weight of an error by at most some constant a. Then, because the code can correct all

errors of weight at most $\frac{d-1}{2}$, the logical error rate of such a generalized transversal gate would be $\mathcal{O}(\epsilon^{\frac{d-a-1}{2}})$.

We also add that all considerations of transversal gates have either involved codes with a single logical qubit, or when the code contains k qubits, that the transversal gate has the same action on all qubits.

Guidelines for code construction # 2: transversal logicals are not universal.

Transversal gates are ideal in a fault-tolerant setting because they prevent errors from spreading. The goal would be to perform a universal set of gates fault-tolerantly using only transversal gates. Unfortunately, this is untenable due to a result by [40, 41]. Performing only transversal gates, we cannot achieve universality.

This result does not hold if we can perform measurements and adaptively perform gates. Since we have to do so for performing quantum error correction, this is not an unreasonable assumption. [42, 43].

Guidelines for code construction # 3: short-depth circuits are restricted.

Transversal gates may not be the only way to achieve fault tolerance. Perhaps one can restrict the spread of errors using only circuits of short-depth. If we considered the 'light cone' around any input to the circuit, then it does not spread very far before the circuit terminates. In this way, even if an error occurred it would not be able to grow to the point it creates a logical error.

Unfortunately, short-depth circuits are also restrictive [16, 44]. If we consider their action on 2-dimensional local codes, it turns out that these gates can only perform gates in the Clifford group. If we want to find a universal gate set, we need to find alternative techniques.

2.5 State injection

There exist a number of quantum error correction schemes that can realize Clifford operations, either transversally or through some other fault-tolerant method. If we can only perform Clifford gates and Pauli measurements, then we can achieve universality using state injection. Here we demonstrate single-qubit teleportation [45]; see also the paper by Gottesman and Chuang [46]. Let $|A\rangle = T|+\rangle = \left(|0\rangle + e^{i\pi/4}|1\rangle\right)/\sqrt{2}$. The following circuit takes as input state $|A\rangle$ and otherwise uses only Clifford operations to implement the T gate on any state $|\psi\rangle$.

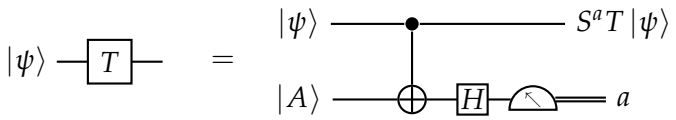


Figure 2.3 Performing the gate T by injecting $|A\rangle$. The measurement is performed in the computational basis.

Thus under the action of the CNOT gate, the state ψ evolves as follows:

$$|\psi\rangle\,|A\rangle \xrightarrow{\text{CNOT}} T\,|\psi\rangle\,|+\rangle + ST\,|\psi\rangle\,|-\rangle \ .$$

Upon performing an X measurement, we either obtain the state $T\,|\psi\rangle$ if the outcome is $+$; this is the desired output state. On the other hand, we obtain $ST\,|\psi\rangle$ if the outcome is $-$. However, this is not a problem because the S gate is part of the Clifford group, and therefore can be undone easily.

2.5.1 Magic state distillation

The magic state $|A\rangle$ combined to Clifford group transformations complete a universal set. As we just described, using a noisy input state $|A\rangle$ will result in a noisy output $T\,|\psi\rangle$, so we need a high-fidelity magic state. Preparing a high-fidelity magic state $|A\rangle$ may be possible in some qubit architectures. However, recall that the Clifford operations are acting on encoded quantum information, so we also need the magic state to be encoded in an error correcting code. In order to produce sufficiently reliable encoded magic states, we use a sub-routine called magic state distillation.

The basic idea of magic state distillation is the following. There exist Clifford circuits that take as input several low-fidelity magic states and output a smaller number of higher fidelity ones. There are several methods to achieve this (see for example [47, 48, 49, 50, 51, 52]), for instance using a quantum error-correcting code that admits a transversal T gate that are realized using state injection. This distillation procedure is realized on encoded data, and since it only requires Clifford operations, it requires a coding scheme with fault-tolerant Clifford. The only remaining difficulty is to prepare some initial encoded magic states $|A\rangle$. This will be done in a faulty manner, e.g. by preparing an unencoded magic state and then gradually encoding it, or again by teleporting an unencoded state in a code. Either way, we obtain low-fidelity magic states, which can then be distilled using protected Clifford circuits.

Bravyi and Haah [53] introduced a framework called tri-orthogonality to describe quantum

error correcting codes that possess certain symmetries. These symmetries imply that if a transversal T gate were applied on the physical qubits of the quantum error correcting code, then it would act as the logical transversal T on all the encoded logical qubits.

Author contributions: Working within this framework, I have studied the process of magic state distillation together with Dr. Jean-Pierre Tillich of Inria, Paris. Here I shall provide a brief summary of my results.

As explained above, magic state distillation requires quantum error correcting codes that supports T gates. In [54], we presented a scheme for magic state distillation using punctured polar codes. This is a systematic construction of a family of tri-orthogonal codes based on polar codes. Our results build on some recent work by Bardet et al. [55] who discovered that polar codes can be described algebraically as decreasing monomial codes. The advantage of this framework is that it allows us to cast the conditions of tri-orthogonality in a purely algebraic manner. This considerably simplifies the process of proving that these codes possess the desired symmetries. Using this powerful framework, we construct tri-orthogonal quantum codes that can be used to distill magic states for the T gate. We propose slightly tweaking the work of Bravyi and Haah, and use a decoder to decode the quantum code at the end of the process. This is in contrast to the original proposal which requires that we project on to the codespace of the code. An advantage of these codes is that they permit the use of the successive cancellation decoder whose time complexity scales as $O(N \log(N))$. We supplement this with numerical simulations for the erasure channel and dephasing channel.

An alternative to performing decoding is to concatenate the code and thereby obtain a code family. The overhead of magic state distillation is defined as the ratio of the number of input to the number of output states. In this scenario, to achieve a target error rate of ϵ, the overhead scales asymptotically as $O\left(\log^{\gamma}\left(1/\epsilon\right)\right)$ [53]. Bravyi and Haah conjectured that γ is lower bounded by 1 [53], implying a bound on the efficiency of magic state distillation. In a breakthrough article, Hastings and Haah [56] recently demonstrated this conjecture to be false by explicitly constructing schemes with sub-logarithmic overhead. Using large Reed-Muller codes, they show that $\gamma \approx 0.6779$ is achievable. This raises the question of whether γ is bounded away from 0. In an article published in Physical Review Letters, we answered this question in the affirmative by showing γ can be arbitrarily close to 0 for quantum error correcting codes comprised of qudits (d-dimensional quantum systems) [57]. The intuition stems from classical coding theory where tri-orthogonality is well studied [58]. This work builds on work by Dr. Shawn X. Cui, Dr. Daniel Gottesman and myself [59] that classify gates on d-dimensional quantum systems.

Although state injection leads to an efficient universal gate set in principle, state distillation requires very large overheads and so they are still an active area of research. Schemes that circumvent magic state distillation are also still being developed [43, 42, 60, 61].

2.6 Chapter Summary

In this chapter, we reviewed some fundamental ideas of the theory of quantum error correction. We began by defining a code using a *stabilizer* group. By measuring operators in the stabilizer, we do not collapse codewords that may be in superposition. An error correcting code can be used to transmit information across a noisy channel if and only if it satisfies the Knill-Laflamme conditions. Intuitively, the conditions state that if errors do not map codewords to each other, then we can undo the action of the error. This also implied that if we can correct a certain class of errors, we can correct a linear combination of that class. In turn this implies that it suffices to deal with Pauli errors. We defined the distance of the code as the minimum weight of an error that is undetectable. These parameters could be used to refer to a stabilizer code as an $[\![n, k, d]\!]$ code. At this juncture, we saw the first guideline for code construction. The Bravyi-Poulin-Terhal bound places strong restrictions on 2-dimensional codes.

With this groundwork, we proceeded to discuss how to perform gates on quantum error correcting codes. The trick is to be able to do so without decoding the code and thereby leaving quantum information vulnerable. We introduced the Clifford gates as the set of gates generated by the CNOT, Hadamard and phase gates. We discussed transversal gates and how this architecture limits the spread of errors in an error correcting code. We then saw some more guidelines for code construction which restricted local quantum error correcting codes. The Bravyi-Koenig bound placed restrictions on the kinds of gates that be performed with short-depth circuits on 2-dimensional codes. The Eastin-Knill bound stated that transversal gates alone are insufficient to achieve a universal gate set.

We concluded by discussing state injection. Using non-stabilizer resource states and the primitive of teleportation, we were able to perform the T gate. Together with Clifford gates and Pauli measurements, we can in principle achieve universal quantum computation. T gates are however noisy and require some pre-processing before they can be injected into the quantum circuit. This then led us to discuss magic state distillation, and how to use quantum error correcting codes to prepare high-fidelity resource states.

Thus equipped we are ready to discuss a specific quantum error correcting code in the next chapter – the surface code.

Chapter 3

The Surface Code

The surface code is arguably the most studied quantum error correcting code. From the theorist's perspective, it plays a central role in the theory of quantum error correction and topological order. As we shall see, its stabilizer is composed entirely of local terms of low weight and yet, the state of the system cannot be discerned by local measurements alone. Featuring simple interconnectivity between qubits and a simple syndrome extraction circuit, it is an ideal candidate for implementations. In addition, numerical simulations indicate that it performs well under noise. For this reason, it is a blueprint for the architecture of some quantum computers.

The surface code however is not without its faults. In some sense, it is a quantum version of the repetition code, a statement which we shall make formal in this chapter. The consequence of this is that it can only encode 1 qubit regardless of the number of physical qubits used to construct it. Given that the number of qubits we can coherently control is a major bottleneck for scalable quantum computation, this is troublesome. Quantum LDPC codes attempt to overcome the shortcomings of the surface code while preserving the features that make it appealing. We shall study these generalizations in chapter 4, but present the surface code first as it forms the basis for our intuition for many of these generalizations.

We begin by demonstrating how the surface code emerges from the repetition code. This is not the traditional way of describing the surface code, nor is it the most intuitive or pedagogical. We choose this approach as it sets the stage for the generalization to quantum LDPC codes in the next chapter. We shall then proceed to discuss how to decode the surface code efficiently. We shall then study how qubits are encoded in *defects* of the surface code. We introduce a defect called a wormhole that aid in processing encoded information and allow us to perform Clifford gates on the qubits they encode. Wormhole defects are

created by performing entangling measurements between two (potentially) separated sectors of a lattice. If we visualize the motion of excitations on the surface of the code, then a particle that enters via one mouth of the wormhole emerges via the other. In this sense, the geometry that these excitations witness is a direct consequence of the entanglement, and is whence these defects get their name. Together with state-injection and magic states (see chapter 2) we can obtain a universal set of gates. The following chapter on quantum LDPC code will follow exactly the same structure and will borrow many of the ideas and intuitions that we present here.

Author contributions: Some of the material presented in this chapter appears in [19] and [20]. Specifically, section 3.3 discusses how to express defects on the surface code as a graph product. This was introduced in [20]. The subsection 3.3 on wormhole defects contains material presented in [19].

3.1 Definition

The surface code [11] is a quantum error correcting code defined on a square lattice. The lattice serves to define a stabilizer group $\mathcal{S}$, and the code space is the joint $+1$-eigenspace of this stabilizer $\mathcal{S}$.

The elements of the code can be defined using two (classical) repetition codes. For illustration, we begin with the smallest repetition code of length 3, a $[3, 1, 3]$ repetition code. The surface code generated using two copies of the repetition code $[3, 1, 3]$ is shown in fig. 3.1(a) below. The two repetition codes are labeled $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively and have associated factor graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively. The factor graph $\mathcal{G}_1$ of code $\mathcal{C}_1$ runs vertically on the left of the diagram and the factor graph $\mathcal{G}_2$ of code $\mathcal{C}_2$ runs horizontally on the bottom of the diagram. As before, variable nodes are denoted by blue circles and indexed by numerals whereas check nodes are denoted by green squares and indexed by capital letters.

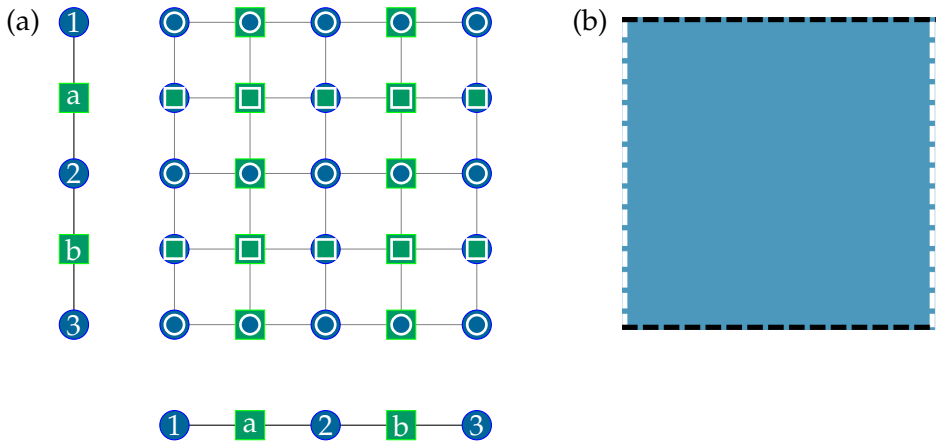


Figure 3.1 (a) The surface code that results from a product of two $[3, 1, 3]$ repetition codes. Nodes from $\mathcal{C}_1$ are inside. Nodes from $\mathcal{C}_2$ are outside. (b) Lattice-free representation of the code.

In what follows, we shall use a lattice-free representation, where we discard the grid for visual simplicity. The lattice-free representation will be useful for conceptual clarity without getting lost in the elements of the grid. This is shown in fig. 3.1(b), and we shall return to this representation shortly.

How do we obtain a quantum error correcting code from these two classical codes? The qubits of the resulting quantum code are obtained as *products* of two variable nodes or two check nodes.

Thus two nodes of the same type yield a qubit. To distinguish them, we call these qubits variable-variable or VV qubits and check-check or CC qubits respectively. Similarly, two nodes of opposite types yield stabilizers of X or Z stabilizer generators. We have chosen a convention where a variable times a check represents an X stabilizer and a check times a variable represents a Z stabilizer.

The connectivity between the qubits and the stabilizers is inherited from the classical codes. For instance consider the action of X stabilizer generators on qubits of VV type as in fig. 3.2(a). These inherit the connectivity of the underlying graph $\mathcal{G}_2$. In other words, for each variable node in $\mathcal{G}_1$, we consider its product with $\mathcal{G}_2$. Similarly, the action of the X stabilizers

on qubits of CC type, depicted in fig. 3.2(b), is inherited from the graph $\mathcal{G}_1$. For each check node in $\mathcal{G}_2$, we consider its product with $\mathcal{G}_1$. Together these generate the set of X stabilizers. In a similar manner, we can obtain the Z stabilizer generators.

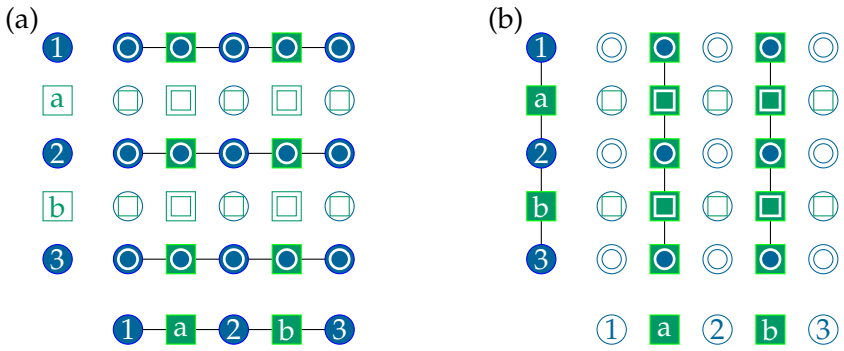
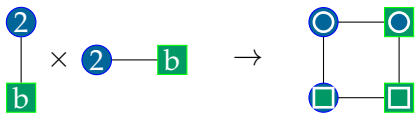


Figure 3.2 Breaking down the X stabilizer. We see the action of X stabilizer generators on VV qubits on the left and CC qubits on the right. Filled nodes represent nodes involved in the stabilizer generator, and are set against a backdrop of transparent nodes representing the rest of the nodes in the factor graph that are not involved.

By overlaying these diagrams, we obtain fig. (3.1). In other words, this *graph product* is obtained by copying the Tanner graph $\mathcal{G}_1$ on every vertical column of the lattice and the Tanner graph $\mathcal{G}_2$ on every horizontal line of the lattice. To ensure that these operators constitute a code, we see that any pair of X and Z stabilizer generators commute. This is because they either do not meet at all, or if they do, they meet exactly twice. For example consider the subgraph defined by the variable node 2 and its neighbor B in both graphs $\mathcal{G}_1$ and $\mathcal{G}_2$.



As can be seen from this diagram, every adjacent pair is forced to meet at two qubits and therefore they commute.

Notice that the variable nodes on the boundary of the classical codes are different from those in the interior since they are only connected to one check node and not two. Since the quantum code inherits connectivity from the classical code, this implies that all stabilizer generators are not created equal. As illustrated in fig. 3.3, those on the boundary are only connected to 3 qubits, and are different from stabilizers in the bulk which are all connected to 4 qubits.

On the top and bottom boundaries, the X stabilizers are broken whereas on the left and

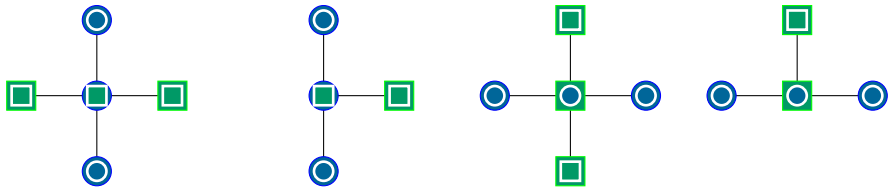


Figure 3.3 Stabilizers of the surface code: (a) Z stabilizer generator in the bulk. (b) Instance of broken Z stabilizer generator in the boundary. (c) X stabilizer generator in the bulk. (d) Instance of broken X stabilizer generator in the boundary.

right boundaries, the Z stabilizers are broken. These boundaries are labeled *rough* and *smooth* respectively.

Stabilizers on the boundaries are interesting because of how they detect errors. Each qubit in the bulk is flanked by two Z stabilizers generators and two X stabilizer generators. Therefore if it is afflicted by an X or Z error respectively, there are two stabilizers on either side that detect it. On the other hand, since qubits on the boundary are only adjacent to a single stabilizer, an error there is potentially only detected by a single stabilizer depending on the type of error.

The fact that we have created copies of codes $\mathcal{C}_1$ and $\mathcal{C}_2$ also implies a neat structure for the logical operators. Recall that the codeword of the repetition code $[3,1,3]$ is the string $(1,1,1) \in \mathbb{F}_2^3$. These have been circled in red in fig. 3.4 below on the factor graphs $\mathcal{G}_1$ and $\mathcal{G}_2$. Each of these strings corresponds to a logical operator on the quantum code. The X logical operator, shown in fig. 3.4(a), is obtained from the codeword of $\mathcal{C}_2$. Similarly, the Z logical operator, shown in fig. 3.4(b), is obtained from the codeword of $\mathcal{C}_1$. Note that the two logical operators meet on the VV qubit $(2,2)$ and therefore anti-commute as desired. Single-qubit X operators have been depicted as white circles whereas single-qubit Z operators have depicted in black circles. This notation carries over to the lattice-free representation shown in fig. 3.4(c).

If a bit-flip error affecting ℓ consecutive bits occurs in a repetition code, it will only be detected by the parity-checks at its endpoints in the bulk, and will be undetected on the boundary. The syndromes associated to endpoints of error segments are called domain walls in physics. Similarly, we can see that the syndrome associated to an error string on the surface code will reside on the ends of the string. Moreover, an X type error string leaves no syndrome on a smooth boundary while a Z type error string leaves no syndrome on a rough boundary.

In the lattice-free representation, the crosses on either end of the string represent the syndromes that detect the error chain. These objects can themselves be regarded as defects on

the surface. We can move the defects by extending the error chain in any direction we desire. Furthermore, we can eliminate a pair of these defects by bringing them together and *fusing* them. We shall return to this idea of moving defects when we discuss implementing Clifford gates.

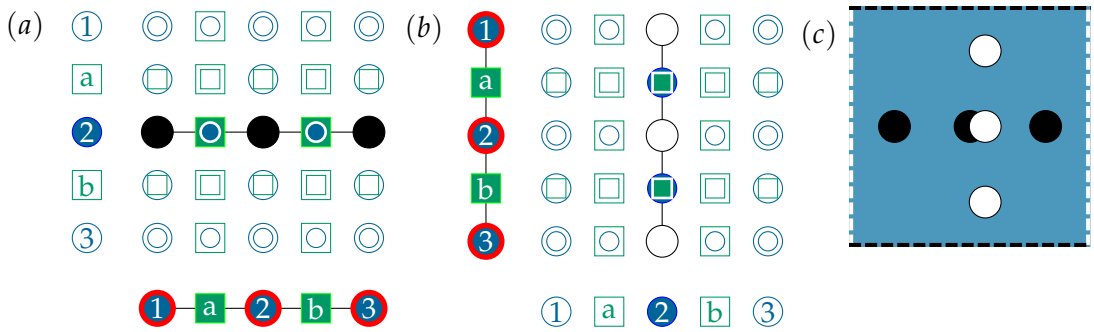


Figure 3.4 Logical operators of the surface code. (a) shows how the logical Z operator emerges from the codeword of $\mathcal{C}_2$. (b) shows how the logical X operator emerges from the codeword of $\mathcal{C}_1$. (c) shows both logical operators in the lattice-free representation.

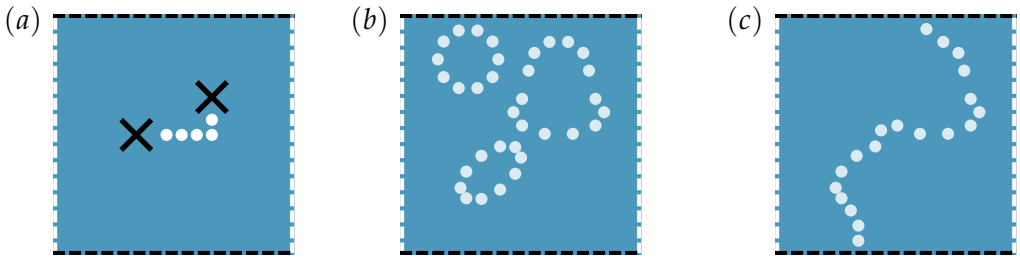


Figure 3.5 (a) Strings of errors can be detected by the syndromes on either end of the string. (b) Contractible loops of X operators. Their action is trivial because they can be expressed as products of X stabilizer generators. (c) Strings equivalent to the X logical operator.

3.2 Decoding the surface code

The surface code possesses an efficient decoding algorithm called *matching*. As was pointed out earlier, any error chain on the surface code always creates pairs of syndromes, one on either end of the chain, or the chain can end on a boundary without leaving any trace. When we perform a measurement of the stabilizers, the generators corresponding to the ends of error chains will be unsatisfied.

Let us first consider the action of a local decoder to demonstrate the innovation of matching using a simple example. What we mean by a *local decoder* is one that makes a recommendation for each qubit individually: has this qubit suffered a X, Y, Z error or no error at all? Consider the X stabilizer generators in fig. 3.6. If the two of them are unsatisfied, there are two possible errors that are likely. Both of these corresponds to half a Z stabilizer generator shown in the center of the diagram. The first is the error chain indicated using a solid red line, and the other is the error chain indicated via the dashed red line (there are other errors that are possible with the same syndrome but they have a higher weight and are thus less likely). By symmetry, it is clear that a local decoder will not be able to decide between these two error chains and will get stuck.

The way out of this conundrum is to realize that in this case, we can pick either chain. Suppose the actual error is really the solid red line; if the decoder picks this line, and undoes the errors along it, then we have undone the error. On the other hand if the decoder picks the dashed red line, we have in effect applied the entire cycle corresponding to the product of the solid and dashed lines. This is a Z stabilizer generator. Recalling eq. 2.1 which stated that if $S \in \mathcal{S}$ is a stabilizer, and $|\psi\rangle$ is in the codespace, then $S|\psi\rangle = |\psi\rangle$. Thus applying the stabilizer will also undo the error.

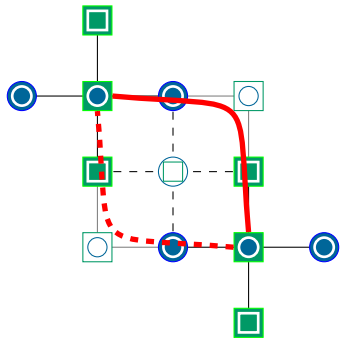


Figure 3.6 Half a stabilizer confounds a local decoder. Two distinct X stabilizers on either side of a Z stabilizer that flag an error. The error could be half the Z stabilizer; the red paths (one dashed and one solid) mark potential errors that are equally likely.

With this intuition, Edmond's maximum matching algorithm, or simply `matching` is an algorithm that can be used to address decoding in general. It accepts as input a set of vertices and the distance between all pairs and attempts to come up with a *matching* such that the total length of the paths between pairs is minimized. As a simple way to decode, we can perform the X and Z decoding separately.

In order to perform decoding we feed to the `matching` algorithm the X or Z syndromes on the surface code, together with their distances to each other and the boundaries of the

code. To be precise, for every pair of stabilizer generators S_1 and S_2, the distance $d(S_1, S_2)$ between them is defined as the minimum number of qubits we need to traverse in order to get from one to the other. We then need to feed to the `matching` algorithm the quantity $2L - d(S_1, S_2)$ since the algorithm maximizes the total path length, and we desire the minimum for decoding. The complexity of the matching algorithm scales as $O(a^3)$ when there are a defects and so the algorithm is efficient. We shall not cover the details of the algorithm here. Fig. 3.7(a) shows two syndromes and 3.7(b) shows the output of the minimum weight matching decoder.

What are the limits of the matching algorithm? There exist problematic physical errors which could lead to a logical error. Consider an error chain that starts from a smooth boundary and stretches more than half-way across the lattice as in fig. 3.7(c). The shortest path in this case in this case is to match to the opposite boundary which in turn will create a logical error. Thus in the worst case scenario, an error whose weight is half the length of the lattice could lead to a logical error.

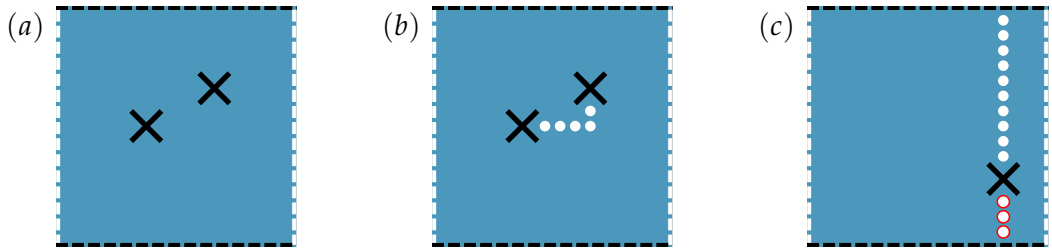


Figure 3.7 Illustrating minimum weight matching. (a) shows two syndromes marked by black crosses. (b) shows the error that matching deduces. Even if this is not the actual error, the product of the real error and the actual error is a stabilizer and therefore returns to the correct codeword. (c) shows a syndrome that could lead to a logical error. The actual error is marked using white circles but matching will deduce that the error is equivalent to the path of red circles. The product of the actual error and the deduced error is a logical operator.

The state of the art decoding algorithms for decoding the surface code currently are tensor network inspired [62]. There exist algorithms that deal better with more general types of noise [63] and function in linear time [64, 65]. The renormalization group (RG) decoder is another approach to decoding the surface code that can decode in log time [66]. There are also parallelized algorithms that can work faster [67]. There exist sophisticated decoders which take into account that there may be correlations between X and Z errors on the surface code [68], but this simple decoding procedure ignores such correlations. Recently Delfosse and Nickerson [69] also came up with an almost linear time decoder for topological codes that also applies to the surface code.

3.3 Defects on the surface code

How do we manipulate encoded information? Rather than use transversal gates to process the logical qubits of the surface code, we shall find it convenient to encode information in *punctures*. Punctures are defects on the surface code that are created by carving out a portion of our code [8]. What this means is that the qubits in the interior of the puncture will no longer be measured and the stabilizers incident to these qubits will be turned off. This is realized by measuring single-qubit Pauli operators within the interior of the puncture which serves to project the qubits in a product state uncorrelated to the rest of the lattice. Since the stabilizer is defined by a set of commuting Pauli operators, stabilizers that anti-commute with this measurement can no longer be a part of the code, and are thus discarded. The stabilizers within the puncture that do not anti-commute with the measurement will become redundant. On the other hand, stabilizers on the boundary will be broken, i.e. they will lose their support within the interior of the puncture. Punctures are classified based on how these boundary stabilizers are broken. Much like the boundaries of the surface code itself, punctures come in two varieties: smooth and rough punctures.

Z (or X) stabilizers do not straddle the boundary of a smooth (or rough) puncture; they are broken cleanly. On the other hand, the X (or Z) stabilizer generators on the boundary of a smooth (or rough) puncture will be broken. Rectangular punctures are the building blocks for punctures of other shapes. These in turn can be constructed using the graph product of two segments of the factor graphs of the repetition codes. To make these ideas concrete, we deal with an example below.

These punctures will carry logical qubits and thus the absence of a patch of the surface code can itself be treated as an entity. By moving these patches around the surface, we can perform non-trivial operations on the associated qubits.

Smooth punctures: A smooth puncture is created by measuring X on the support of a set of X stabilizer generators. By this we mean that each qubit in the specified support is measured *individually*. How do we specify the set of X stabilizers? A smooth puncture is chosen by two sub-graphs such that the product does not contain any Z stabilizers of weight 3. In other words, no Z stabilizer generators are broken by the puncture. This requires the following ingredients:

1. a sub-graph $\mathcal{G}_1' \subseteq \mathcal{G}_1$ such that the starting and ending points are both variable nodes.
2. a sub-graph $\mathcal{G}_2' \subseteq \mathcal{G}_2$ such that the starting and ending points are both check nodes.

With these ingredients, we can use the following recipe to form the puncture.

1. For every variable node in $\mathcal{G}_2'$, form a product with $\mathcal{G}_1'$.
2. For every check node in $\mathcal{G}_1'$, form a product with $\mathcal{G}_2'$.
3. Overlay these two graph products.

The qubits that are removed are within the interior of the puncture. The boundary qubits are not removed.

Fig. 3.8(a) illustrates these ideas using two $[5, 1, 5]$ repetition codes. The ingredients are:

1. The sub-graph $\mathcal{G}_1'$ is the segment between variable nodes 2 and 4.
2. The sub-graph $\mathcal{G}_2'$ is the segment between check nodes b and d.

These are combined using the recipe above. Notice that no Z stabilizer generators are broken across the puncture.

In creating this smooth puncture, we create a logical qubit. The logical operators are shown in fig. 3.8(b). The logical Z operator is a loop of Z operators that encircles the puncture. The loop of X operators encircling a smooth puncture is still part of the stabilizer. The conjugate is a chain of X operators that runs between the boundary of the smooth puncture and the smooth boundary of the lattice.

This chain operator need not terminate on the boundary of the lattice; it could also terminate on the smooth boundary of another smooth puncture. In this way, two smooth punctures can be used to encode a single logical qubit.

Rough punctures: A rough puncture is created by measuring Z on the support of a set of Z stabilizer generators. Again, we mean that each qubit in the specified support is measured *individually*. A rough puncture is chosen by two sub-graphs such that the product does not contain any X stabilizers of weight 3. No X stabilizer generators are broken by the puncture. This requires the following ingredients:

1. a sub-graph $\mathcal{G}_1' \subseteq \mathcal{G}_1$ such that the starting and ending points are both check nodes.
2. a sub-graph $\mathcal{G}_2' \subseteq \mathcal{G}_2$ such that the starting and ending points are both variable nodes.

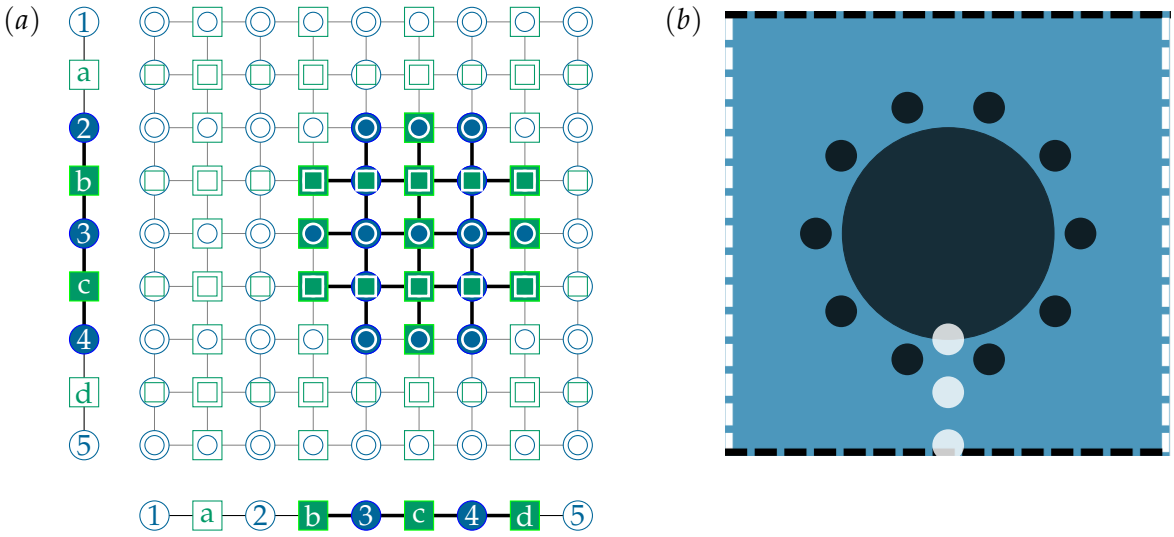


Figure 3.8 Smooth puncture created by $\mathcal{G}_1'$ and $\mathcal{G}_2'$.

This is the opposite of the constraints on smooth punctures and in this sense the punctures are dual to each other. With these ingredients, the construction proceeds as above.

With these ingredients, we can use the following recipe to form the puncture.

1. For every check node in $\mathcal{G}_2'$, form a product with $\mathcal{G}_1'$.
2. For every variable node in $\mathcal{G}_1'$, form a product with $\mathcal{G}_2'$.
3. Overlay these two graph products.

Fig. 3.9(a) illustrates these ideas using two $[5, 1, 5]$ repetition codes. The ingredients are:

1. The sub-graph $\mathcal{G}_1'$ is the segment between check nodes a and c.
2. The sub-graph $\mathcal{G}_2'$ is the segment between variable nodes 2 and 4.

These are combined using the recipe above. Notice that no X stabilizer generators are broken across the puncture.

In creating this rough puncture, we create a logical qubit. The logical operators are shown in fig. 3.9(b). The logical X operator is a loop of X operators that encircles the puncture. The loop of Z operators encircling a rough puncture is still part of the stabilizer. The conjugate is a chain of Z operators that runs between the rough boundary of the puncture and the rough boundary of the lattice.

This chain operator could also terminate on the boundary of another rough puncture. Thus two rough punctures could be used to encode a single logical qubit.

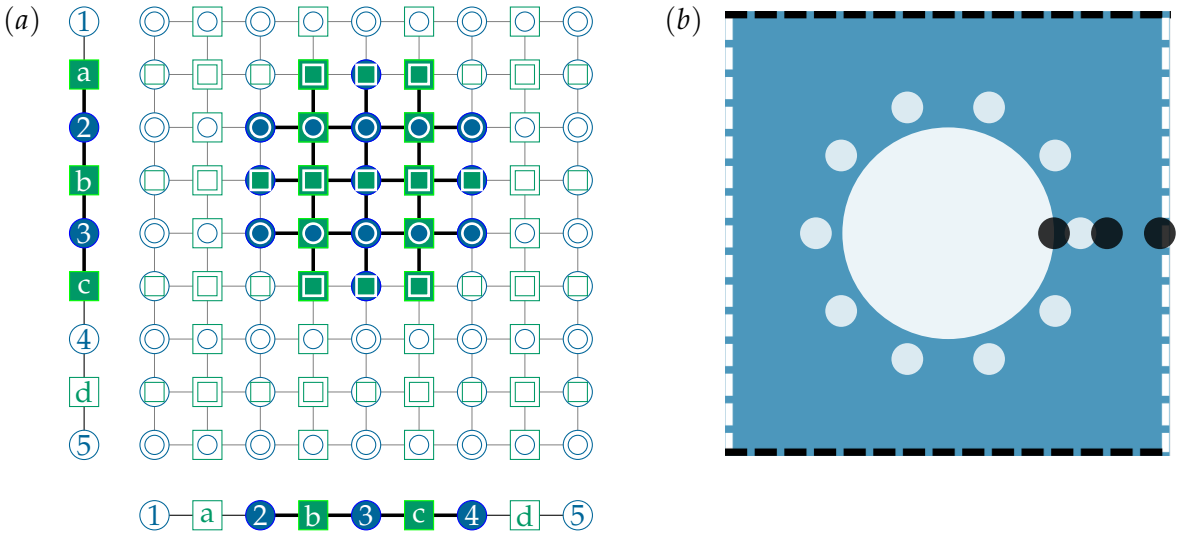


Figure 3.9 Rough puncture created by $\mathcal{G}_1'$ and $\mathcal{G}_2'$.

Movement: Encoding qubits in punctures is useful because it permits us to manipulate the encoded information by *deforming* the code. Code deformation refers to a process which can be broken down into a series of elementary steps. Each elementary step changes the error-correcting code that supports the logical qubit. To ensure that we still remain in the codespace, the initial and final codes must be the same. When the entire process is complete, the logical qubits undergo a logical transformation. Each step modifies the code by a little, so that induced local errors remain local. These local errors can be corrected and this way, we do not induce a logical error.

Visually, each elementary step can be represented as punctures moving across the lattice. Assume that we encode two logical qubits, each in a pair of punctures as shown in fig. 3.10. The two smooth punctures encode one logical qubit and the two rough punctures encode another logical qubit.

What does it mean for a puncture to move? If we imagine sliding the sub-graphs $\mathcal{G}_1'$ or $\mathcal{G}_2'$ that define either smooth puncture, then the puncture on the surface code moves either vertically or horizontally. By performing the measurements dictated by the new sub-graphs, we can extend or shrink the puncture as desired. In doing so, the loop type logical operator recenters itself around the puncture. The chain type operator on the other hand leaves a trail behind as the puncture moves.

This property can be used to perform two-qubit gates on logical qubits encoded in smooth

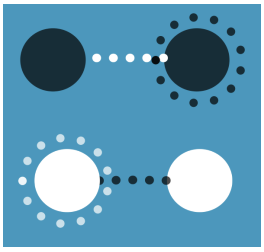


Figure 3.10 Encoding a logical qubit in a pair of smooth or rough punctures.

and rough punctures and the operation is called *braiding*. Braiding qubits encoded in smooth and rough punctures performs the CNOT gate between them. As depicted in fig. 3.11, this can be proved by looking at the transformation of the associated logical operators.

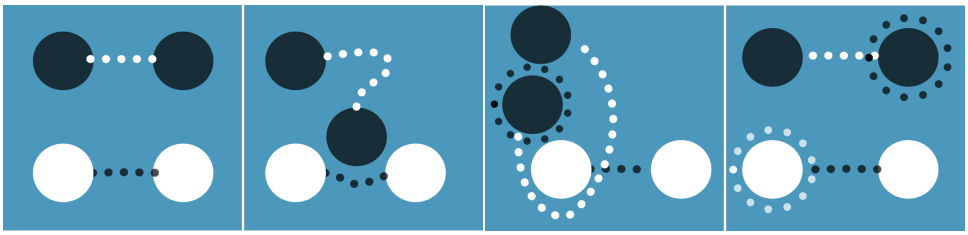


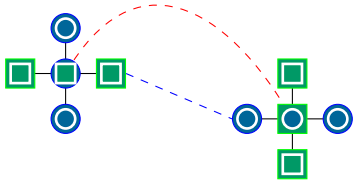
Figure 3.11 Braiding two logical qubits encoded in pairs of rough and smooth punctures respectively. We recognize the transformation $XI \to XX$, $IX \to IX$, $ZI \to ZI$, and $IZ \to ZZ$ that define a CNOT from eq. 2.5.

Wormholes: If the code deformation remains within the family of stabilizer codes, the entire sequence is described by Clifford operations (Pauli measurements), and so it can only induce logical Clifford group transformations. Moreover, graph product codes and their punctures described here are CSS codes, meaning that their stabilizer generators are either all Z or all X. If the code deformation remains within the CSS family, the resulting logical gates cannot interchange X and Z type logical operators. Thus, code deformation within the graph product formalism cannot achieve the full Clifford group.

There exists several ways to perform all gates in the Clifford group on the surface code. Some involve modifying the code or techniques such as lattice surgery [70, 71, 72, 73]. With an eye towards generalizations to fault-tolerant gates on LDPC codes, we shall present one method here. The key idea behind the technique we present is to break the CSS nature of the code. Following [74, 75, 76, 77, 78], this has been a useful trick to obtain the rest of the Clifford group.

Recall that we created a puncture by measuring single-qubit Pauli operators. These measurements anti-commuted with the existing stabilizers, forcing us to replace them. Instead

we shall modify our measurements to two-qubit operators obtained as tensor products of one X and one Z. In turn, these anti-commute with both X and Z stabilizers. To remedy this anti-commutation, we can replace the pair of anti-commuting operators by their product. In the figure below are one Z stabilizer and X stabilizer (plus their supports). We select one qubit in each of their supports; these are connected via the blue line. We perform an entangled measurement on them – we measure X on the support of the Z stabilizer and Z on the support of the X stabilizer. This pair of stabilizers is replaced by its product, indicated by the red line connecting the two check nodes.



We now use this idea to create a defect called a *wormhole*. We measure two-body operators along the boundaries of punctures which in turn results in the entangled stabilizers shown in fig. 3.12. This creates two entangled punctures that are spatially separated that we refer to as the mouths of the wormhole.

Each hybrid stabilizer is a product of one plaquette and one vertex generator and thus the weight of the resulting stabilizers is independent of the size of the punctures and the size of the underlying surface code. These new hybrid stabilizers have weight 8; this can be reduced by spreading the weight among some of the local checks adjacent to these stabilizers using CNOT gates.

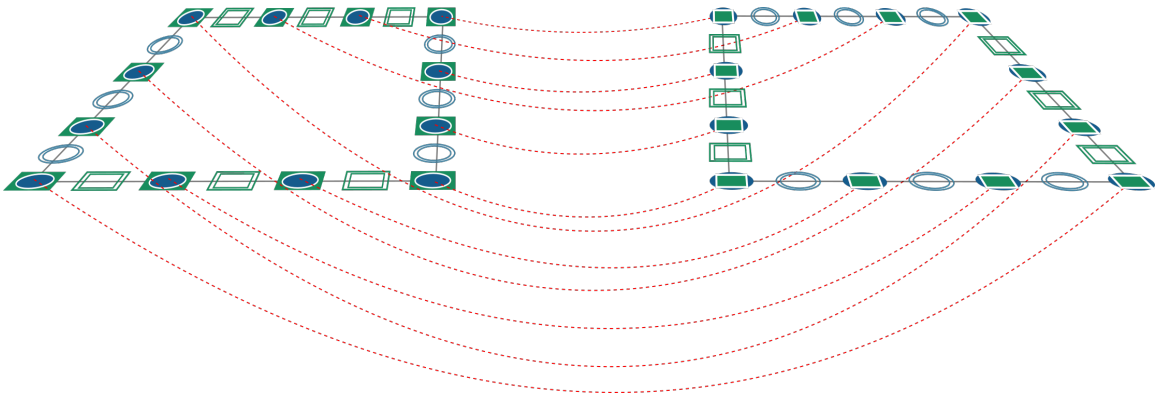


Figure 3.12 Creating a wormhole by measuring two-body Pauli operators along the boundary of a puncture. The dashed-red lines indicate stabilizers that have been paired.

A simplified version of this figure is presented in the lattice-free version in fig. 3.13. The two white circles represent the mouths of the wormhole. The wire mesh underneath the lattice represents the entanglement between these two patches. The mesh is merely a visual aid and does not represent an extension of the lattice.

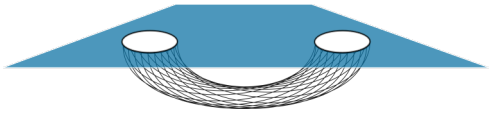


Figure 3.13 Side-view of lattice free representation of the wormhole.

Why are hybrid stabilizers useful? A puncture that enters the mouth of a wormhole is teleported from one region of the surface code to another. Furthermore, a rough puncture is transformed to a smooth puncture and vice versa. The mouths of a wormhole are capable of absorbing and emanating both smooth and rough punctures. From a topological point of view, the rough and smooth mouths are indistinguishable. For this reason, we drop the color of the mouths and without loss of generality, depict both mouths in white.

In this way, wormholes can transform information encoded in punctures, but they can also encode two logical qubits. The wormhole is an eigenstate of some set of stabilizers shown in fig. 3.14 (a) and (b). The first stabilizer shown in fig. 3.14(a) is merely the product of all the hybrid stabilizers obtained above. The second stabilizer shown in fig. 3.14(b) is merely the product of the entangling measurements. The stabilizers associated with the wormhole are thus non-local, with one loop around each mouth. As depicted in the two left panels of fig. 3.14, one element of this pair is a string of X operators whereas the other is a string of Z operators.

Figure 3.14 (c) represents the logical Z operator as a loop of physical Z operators that encircle one mouth and the conjugate logical X operator is a pair of strings, one of X type and another of Z type that run to the mouths. We assume that the strings terminate at a 'sink' wormhole elsewhere on the lattice. The logical operators of the second logical qubit are shown at fig. 3.14 (d) and are obtained by swapping the two mouths of the wormhole.

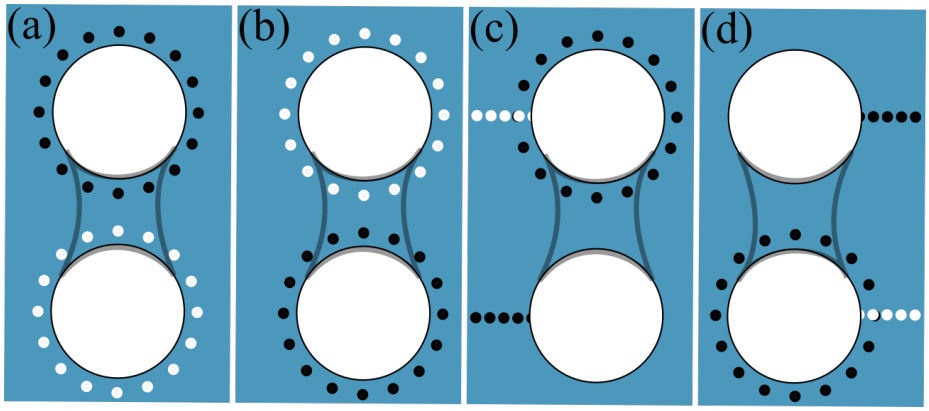


Figure 3.14 Stabilizer and logical generators of the wormhole. (a), (b) represent the stabilizer generators of the wormhole. (c), (d) represent the logical operators of each logical qubit. The logical Z is a loop of Z operators encircling the mouth of a wormhole. The logical X is comprised of two strings that runs between the two mouths of the wormholes, one string of Z operators and another string of X operators.

3.4 Clifford gates

To perform single-qubit Clifford gates, we shall use the assistance of an ancilla qubit. This ancilla qubit will always be a wormhole. We begin with two lemmas which summarize what measurements we require to perform Clifford gates on a qubit of interest. These results apply in general, even outside of logical qubits encoded in the surface code.

Suppose we have two qubits, labelled 1 and a, denoting the qubit of interest and the ancilla respectively. The first lemma summarizes what exactly is needed in order to perform single-qubit Clifford gates on qubit 1.

Lemma 11. *Let A and B be two (distinct) non-trivial single-qubit Pauli operators. Let P and Q be two Pauli operators, not necessarily distinct. The two-qubit measurements A_1P_a and B_1Q_a, together with all single-qubit Pauli measurements on qubit a are sufficient to generate the single-qubit Clifford group on qubit* 1.

Proof. A logical Clifford operation will proceed in three steps. Without loss of generality, let $P = Q = A$ and consider the measurement of A_1P_a.

1. Initialize qubit a by preparing it in the B basis.
2. Next, perform a joint measurement $A_1P_a (= A_1A_a)$ of qubits 1 and a.
3. Finally, measure qubit a in the basis $C (\neq A \neq B)$.

The following flowchart tracks the transformation of the generators of the associated stabilizer and normalizer groups, $\mathcal{S}$ and $\mathcal{N}$.

$$\begin{aligned} \mathcal{S} &= \{B_a\} \rightarrow \{A_1 A_a\} \rightarrow \{C_a\} \\ \mathcal{N} &= \{B_1, C_1\} \rightarrow \{B_1 B_a, C_1 B_a\} \rightarrow \{C_1 C_a, B_1 C_a\}\,. \end{aligned}$$

We have used the fact that Pauli operators are cyclic, i.e. the product of any two distinct operators yields the third (up to a phase). Up to stabilizer, the result of this transformation is to map B to C and vice-versa. The result follows. □

To perform two-qubit Clifford gates, we can perform controlled operations that do not require Y states. This is illustrated by the circuit in fig. 3.15 below.

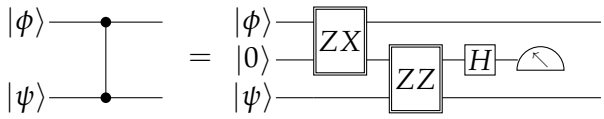


Figure 3.15 A measurement-based circuit to perform controlled-Z. We introduce an ancilla prepared in the $|0\rangle$ state. The double-boxes indicate a non-destructive projective measurement. The labels PQ on these measurements indicate that the projection is performed along the $+1$ and -1 eigenstates of the 2-qubit Pauli operator PQ. Finally, we perform a Hadamard and destructively measure the ancilla qubit in the computational basis.

We now need to demonstrate how to perform the desired Pauli measurements on the wormhole and logical qubit.

3.4.1 Measurements

One might imagine that we can perform fault-tolerant gates on wormholes by moving the mouths around one another akin to braiding. However, this will require a forbidding amount of non-local connectivity. Every qubit might need to be connected to every other qubit in case we decide to move the mouth of the wormhole to that location.

Instead, we shall use a qubit encoded in either a pair of smooth or rough punctures. This qubit, referred to as the needle, can be used to *stitch* logical operators of interest as we shall demonstrate. It will therefore not require any more long-range connectivity beyond what is required to initialize the wormholes. We can classify logical operators of the needle based on whether they can be measured fault tolerantly. Note that the needle operators X

and Z can be measured fault tolerantly. If we need to measure the string-like operator that runs between punctures for instance, we could make the punctures larger and bring them closer together. Alternatively to measure the loop-type operator, we can move the punctures apart, make them small and measure the boundary. The X and Z operators will thus be referred to as needle-measurable operators. On the other hand, Y is not needle-measurable. This is because it will require both shrinking the punctures and bringing them closer together.

The needle will then be used to facilitate measurements on the ancilla wormholes. There are different ways to perform the desired joint-operations on the needle and qubits encoded in the wormhole. Braiding the needle around one mouth of a wormhole results in the controlled-Z operation between the needle and an encoded qubit. The evolution of the logical X of the puncture is shown in fig. 3.16. Since the wormhole is traversable, a

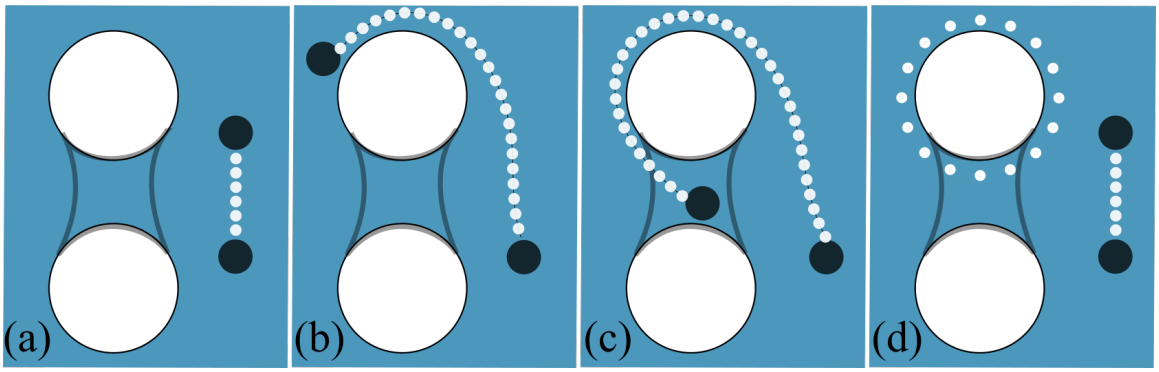


Figure 3.16 The braiding operation: Braiding the needle around the wormhole results in a controlled-Z operation between the needle and encoded qubit.

puncture can enter one mouth of the wormhole and emerge via the other. We call this operation *stitching*. Stitching results in the controlled-X operation between the needle and the encoded qubit. The evolution of the logical X of the puncture is shown in fig. 3.17.

We shall say that an operator Q is traceable if there exists a way to map a needle-measurable operator P to $P \otimes Q$. We can then measure Q by measuring the needle-measurable operator P. We now explain how this procedure will work.

Let A denote a logical qubit, or sets of logical qubits whose state we wish to measure. This could refer to a set of logicals on the wormhole, or an embedded logical, or some combination thereof. A logical operator Q_A on system A is said to be traceable if there exists a

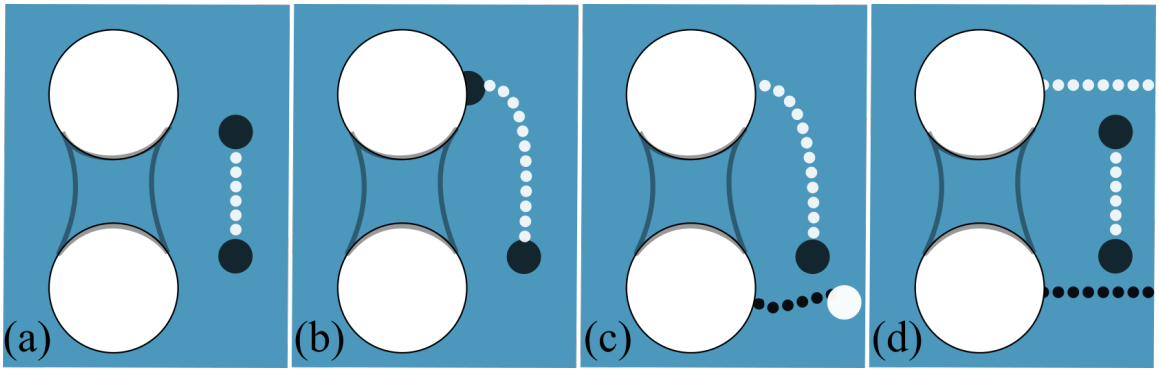


Figure 3.17 The stitching operation: Passing the puncture through the mouth of the wormhole results in the controlled-X between the needle and encoded qubit. Between panels (c) and (d), the rough puncture goes through the sink wormhole far away, is transformed and then returns to its initial position.

needle-measurable operator P_a and a unitary operation U that maps P_a to $Q_A P_a$, i.e.

$$U^\dagger P_a U = Q_A P_a \, .$$

Such operations will be used to measure traceable operators Q_A in order to effect measurements of logical operators in lemma 36.

We now outline why this is a useful operation. Suppose the ancilla is prepared in a $+1$-eigenstate of P_a labelled ϕ_a, and let ψ_A be the state of system A. Let U be the unitary operation that maps P_a to $Q_A P_a$. Suppose we perform the operation corresponding to U, and proceed to measure P_a. If the measurement outcome is labelled $m \in \{0,1\}$, the state after the measurement is described as

$$(\mathbb{1} + (-1)^m P_a) U (|\psi\rangle_A \otimes |\phi\rangle_a) \, ,$$

up to normalization. We may now apply $U^\dagger$ to obtain the state (up to normalization)

$$\begin{aligned} &(\mathbb{1} + (-1)^m U^\dagger P_a U) \, |\psi\rangle_A \otimes |\phi\rangle_a \\ =&(\mathbb{1} + (-1)^m Q_A P_a) \, |\psi\rangle_A \otimes |\phi\rangle_a \, . \end{aligned}$$

We may now discard the system a, leaving us with the system A. The measurement out-

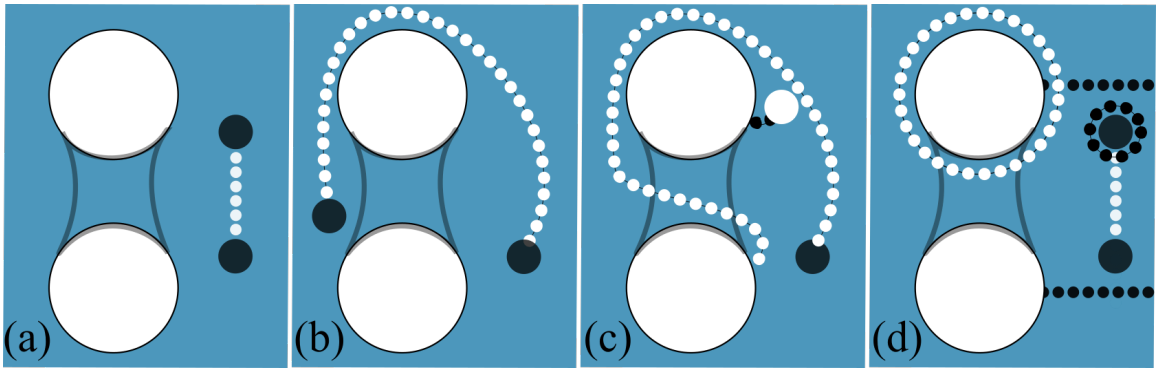


Figure 3.18 The Y operator of a wormhole is not traceable.

come m can itself be described as

$$\begin{aligned} m &= \mathrm{Tr}\left(U(\psi_A \otimes \phi_a)U^\dagger P_a\right) \\ &= \mathrm{Tr}\left((\psi_A \otimes \phi_a)U^\dagger P_a U\right) \\ &= \mathrm{Tr}\left(\psi_A Q_A\right) \ . \end{aligned}$$

In effect, we have performed a measurement of Q_A on system A using an ancillary system a. So long as there exists an effective way to prepare the needle in an eigenstate ϕ_a of the needle-measurable operator P_a, and a traceable operation U, then we can perform the desired measurements on system A.

Unfortunately, logical Y operators are not traceable as the Y operator crosses itself. This is shown in fig. 3.18. Recall that we needed to perform a fault-tolerant measurement of an operator P in order to measure a traceable operator Q. Furthermore note that a single-qubit Y operator corresponding to a wormhole (in the standard basis we have defined) is not needle-measurable. Looking at fig. 3.18, we see that the X string that enters the mouth of the wormhole crosses a Z string that encircles the mouth, as it should to yield the correct anti-commutation relations. However this implies that if we consider the logical Y operator obtained as a product of the X and Z operators, it crosses itself at this point of overlap. What this means is that the operator in panel (d) of fig. 3.18 is the product $Y \otimes Y$. Since Y on the wormhole is not measurable, we cannot perform the single-qubit Y measurement in a fault-tolerant way.

This is the final ingredient required to perform logical single-qubit Clifford gates as stipulated by lemma 11. The lack of a Y measurement is therefore problematic but can be remedied by noting that although a Y operator cannot be measured, the operator $Y \otimes Y$ can indeed be measured. This is shown in lemma 12 below. Provided a resource state of an

ancilla qubit a prepared in the Y basis, suppose we wanted to measure Y on a qubit, labeled qubit 1. We can measure $Y_1 \otimes Y_a$ since a product of Y operators traceable.

Lemma 12. *Let the ancilla be comprised of two qubits labelled a and b such that one of its stabilizer generators is $I_a Y_b$. It is possible to apply the measurement $Y_a I_b$ on qubit a without affecting the state of the generator $I_a Y_b$.*

Proof. Let a and b refer to the qubits encoded in a wormhole. However, the product $Y_a Y_b$ is traceable as shown in fig. 3.19. This is because the operator does not intersect itself.

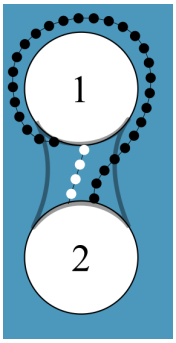


Figure 3.19 The product $Y_a Y_b$ does not cross itself and is therefore traceable.

This can be used to measure $Y_a I_b$ by initializing the wormhole in a state such that $I_a Y_b$ is a stabilizer generator. We can then measure $Y_a Y_b$, which up to action of an element of the stabilizer, is equivalent to $Y_a I_b$. The generator $I_a Y_b$ commutes with the measurement and is therefore unaffected. It can therefore be used for the next gate as well and in this sense, the gate is catalytic. □

3.4.2 Resource state preparation

We have outlined what it takes to perform Clifford gates on qubits encoded in wormholes. This requires the resource state of an ancilla qubit prepared in the Y basis. We only need to do this once, and then for the rest of the computation this Y state can serve as a catalyst.

To obtain a universal gate set, we also require a T gate which we can obtain via state injection.

Both these resource states, the Y and the T, can be prepared using non-fault-tolerant techniques. For instance, suppose we use two rough punctures to encode a qubit in the $|+\rangle$ state. We could bring the punctures close together such that the Z operator is very short,

say just one unit. The physical T gate on this qubit is then the logical T gate. This process is not fault-tolerant because the Z operator is short and therefore the logical qubit is vulnerable to dephasing errors. To obtain reliable Y states we can bring two punctures close together, and shrink the size of the puncture to measure the logical Y operator. This too is non-fault-tolerant, but it only needs to be done once. These resource states can be distilled to obtain high-fidelity states. The T states are consumed during the computation and the Y states are used catalytically. We shall expand on this discussion in chapter 5.

3.5 Chapter summary

In this chapter, we introduced the surface code. We began by defining its stabilizer generators, and found that they were all local in 2 dimensions. In other words, the stabilizers of the surface code can be laid out on a table top. We pointed out that the surface code could be expressed as the product of two repetition codes. We discussed some fundamental properties of the surface code, and understood its logical operators. Next we highlighted that the surface code has an efficient decoder – the minimum weight perfect matching algorithm. This decoder was able to overcome the shortcomings of a local decoder, i.e. one that made decisions for each qubit individually.

We then proceed to discuss defects on the surface code. These are modifications to the surface code that allow us to encode qubits, but in addition also perform logical operations on these qubits. A puncture defect is defined by specifying a region where we stop measuring stabilizers. The advantage of these defect based encodings is that they facilitate the CNOT gate on qubits encoded in smooth and rough punctures respectively. These punctures were themselves expressed as emerging from a graph product. It was shown that they were graph product of sub-codes of the repetition code. If we enforce certain boundary constraints on these sub-codes, then the resulting defects on the surface code also inherit certain properties. This let us classify punctures into two types – smooth and rough. Smooth punctures were those punctures where Z stabilizers are not broken across the boundary; X stabilizers on the other could be broken. On the other hand, a rough puncture is one where X stabilizers are not broken across the boundary; Z stabilizers could be broken.

In order to perform all Clifford gates on qubits encoded in defects, we generalized these defects. It was noted that braiding punctures alone led to operations which mapped X operators to X operators and Z operators to Z operators. This alone is insufficient to generate the entire Clifford group. Using two-qubit measurements, we were able to create defects

that are entangled across two spatially separated sectors of the lattice. Wormhole defects are also capable of storing qubits. Furthermore, using ancilla-assisted techniques, we were able to perform all Clifford gates on qubits encoded in these qubits. This required the use of two resource states. First, we used logical Y resource states that functioned as catalysts. Using these states we could perform all gates in the Clifford group. This scheme then consumed T states to achieve universality.

In the next chapter, we shall study generalizations of the surface code that could overcome the restrictions of working with a local code in 2 dimensions.

Chapter 4

Hypergraph Product Codes

The *overhead* is a figure of merit that represents the number of qubits we require to simulate a system. These simulations are characterized by a tug-of-war between the size of the circuit and the accuracy of the result. Larger problem instances require larger circuits and therefore larger error correcting codes; this however creates more opportunities for errors to accumulate. To compensate for this increased error accumulation, we must lower the logical fault rate. By choosing our quantum error correcting code appropriately, we can increase the code size n logarithmically with the size of the logical circuit and still find that the global logical error rate decreases exponentially in the code size n. One drawback of topological codes is their ability to store logical qubits. Indeed, each logical qubit in a topological code is encoded in a distinct block of size n. The number of physical qubits required to simulate a single logical qubit therefore increases with the size of the quantum computer. It is imperative to explore alternative quantum error correcting codes. This is a tall order because the surface code has many features that make it favorable. It is difficult to come up with something simpler and that works better; something has to give.

In contrast, quantum *low density parity check* (LDPC) codes [79, 6] are generalizations of topological codes that overcome this increasing encoding overhead. LDPC codes refer to families of codes where all qubits (data qubits and ancilla qubits for readout) are only connected to a constant number of other qubits. This constant is independent of the block size and thus simplifies the process of syndrome extraction. In [6], Gottesman proposed a construction that combines techniques for efficient syndrome extraction with ideas to perform logical gates on block codes. The result was a conditional statement: if 'good' quantum LDPC codes exist, the number of physical qubits required to simulate a logical qubit becomes a constant, independent of the size of the quantum computer.

The difference between LDPC codes and topological codes is that the connectivity need no longer be spatially local. By sacrificing locality, these codes overcome one of the shortcomings of the surface code and this permits a 'wholesale effect'. Increasing the size n of the code lets us encode k qubits, where k can increase with n. We can find codes for which the logical error probability decreases exponentially with n with a fixed k/n ratio. This is to be contrasted with topological codes that also achieve an exponential error suppression with n, but with $k = 1$ and hence vanishing encoding rate $1/n$.

Good LDPC codes are elusive. It is hard to enforce the commutation relations between stabilizers of a quantum code while simultaneously maintaining low connectivity. Topological codes use topology to achieve this, but there are strong constraints on the number of logical qubits they can simulate and how effectively they can do so [80, 15]. These constraints are a consequence of *locality*. Although no doubt simpler to engineer, locality is not a fundamental constraint. There exist techniques that permit qubits that are not adjacent to share entanglement in various architectures [81, 82, 83, 84]. This motivates theoretical investigations of LDPC codes that are not constrained by locality. Only with a complete understanding of the potential benefits of LDPC codes will we be able to decide if they are worth the extra experimental effort.

One of the leading candidates for LDPC codes are the so-called hypergraph product codes which eschew topology, and instead engineer commutation relations using algebraic/ graph-theoretic techniques [18]. The hypergraph product is itself not a code family, but rather a technique to construct quantum codes from classical codes. If we input two classical codes to the hypergraph product machinery, the resulting quantum code inherits properties of the classical codes. Importantly, if the classical code families are LDPC then the quantum code families will also be LDPC. Furthermore, if the classical code families have a code size k scaling linearly in the block size n, then so does the code size of the quantum code family. With regards to distance, the hypergraph product code construction yields quantum codes with distance scaling as the square-root of the block size. Up to constants, this is the same functional dependence between the distance and the block size as the surface code (but it applies to all the logical qubits). So in short, quantum LDPC code achieve the same error suppression as topological codes, but do so at a constant encoding rate. Since the inception of hypergraph product codes, there have been variants such as [85] and related classes of codes called homological product codes [86, 87]. We shall not cover these codes in this thesis.

Do these codes perform well in realistic scenarios? Do there exist efficient techniques to decode the hypergraph product code? Can they perform well when the syndrome measurements are imperfect? In this chapter, we will answer these questions in the affirmative.

These results demonstrate that hypergraph product codes are efficient fault-tolerant quantum memories.

4.1 Background and notation

4.1.1 Classical and quantum codes

We begin by reviewing some ideas that were introduced in chapter 1, and also introducing some new notation.

Classical codes: Recall that a classical code $\mathcal{C} = [n, k, d] \subseteq \mathbb{F}_2^n$ over n bits is the (right-)kernel of a matrix $\mathrm{H} \in \mathbb{F}_2^{m \times n}$, known as its parity check matrix. The code dimension k is the dimension of the kernel, $\dim(\ker(\mathrm{H}))$, and d is the minimum Hamming distance between a pair of vectors in $\mathcal{C}$. In general, we could have redundant checks and so $m \geq n - k$. We let $\mathrm{rs}(\mathrm{H})$ denote the row-span of the matrix H and H^t be the transpose of H. Each row of H encodes a parity constraint that we refer to as a check and we label $c \in C$. Likewise we label the columns of H by variable indices $v \in V$. Let $\mathbb{1}_V, \mathbb{1}_C$ denote the identity on $\mathbb{F}_2^V$ and $\mathbb{F}_2^C$ respectively. For $u, v \in \mathbb{F}_2^n$, we let $\langle u, v \rangle = \sum_{i=1}^n u_i v_i$ denote the inner product between them. Finally, for an element $u \in \mathbb{F}_2^V$, we let $\mathrm{supp}\{u\}$ denote the set of locations where u has non-trivial support. This naturally extends to other spaces. The image $\mathrm{im}(A)$ of a matrix $A \in \mathbb{F}_2^{m \times n}$ is the column-span of A in $\mathbb{F}_2^m$.

An LDPC code is a code family $\{\mathcal{C}_n\}_n$ such that the number of bits in the support of a check is upper bounded by a constant as a function of the block size n, as are the number of checks a bit is connected to [25]. In other words, the number of non-zero elements in each row and column of the parity-check matrix is bounded by a constant with respect to the block-size n. The factor graph $\mathcal{G}(\mathcal{C})$ of a code $\mathcal{C}$ lets us infer properties of the code $\mathcal{C}$ from the properties of the graph. The graph $\mathcal{G}(\mathcal{C}) = (V \cup C, E)$ is a bipartite graph, where $V = [n]$ and $C = [m]$. We draw an edge $e = (v, c)$ between check node $c \in C$ and variable node $v \in V$ if and only if $\mathrm{H}_{cv} = 1$. Given a subset $P \subseteq V \cup C$, the neighborhood of P is denoted $\Gamma(P)$ and is defined as

$$\Gamma(P) = \{q | (q, p) \text{ or } (p, q) \in E \text{ for } p \in P\} .$$

Quantum codes: Let $\mathcal{P} = \{\pm 1\} \times \{\pm i\} \times \{I, X, Y, Z\}$ denote the Pauli group and $\mathcal{P}_n = \mathcal{P}^{\otimes n}$ denote the n-fold tensor product of the Pauli group. Recall from chapter 2 that a

quantum error correcting code is specified by a group $\mathcal{S} \subseteq \mathcal{P}_n$. The set of logical operators is then defined as $\mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$, where $\mathcal{N}(\mathcal{S})$ denotes the normalizer of the stabilizer group.

Just like the individual rows of the classical parity check matrix generate a linear space of constraints, we can choose a generating set of checks for the stabilizer group $\mathcal{S}$. Much like its classical counterpart, the factor graph $\mathcal{G}_Q$ can be used to represent the stabilizer generator and provides a visual representation of $\mathcal{Q}$. The only difference is that the edges could carry labels of Pauli elements X, Y or Z, indicating the action of a check on a qubit.

4.1.2 The hypergraph product code

The hypergraph product code is a way to construct a quantum code $\mathcal{Q}$ given two classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$. As an example, the surface code was obtained as the product of two repetition codes. This can naturally be extended to families of classical codes. If the two classical code families are LDPC, then so is the resulting quantum code family. The resulting code is a CSS code [35, 36], i.e. the stabilizer generators are either products of only X operators or only Z operators. For simplicity, we shall consider the graph product of a code $\mathcal{C}$ with itself.

Graph-theoretic description: Let $\mathcal{G} = (V \cup C, E)$ be a bipartite graph. Let $\mathcal{Q}$ denote the quantum code obtained from the hypergraph product of $\mathcal{G}$ with itself. The factor graph $\mathcal{G}_\mathcal{Q}$ of $\mathcal{Q}$ is denoted as $\mathcal{G}_\mathcal{Q} = \mathcal{G} \times \mathcal{G}$. Its nodes are partitioned as follows:

1. qubits $V \times V \cup C \times C$;
2. X stabilizers $V \times C$;
3. Z stabilizers $C \times V$.

This representation highlights that this construction yields two kinds of qubits – those emerging from the product of two variable nodes (VV nodes) and those emerging from the product of two check nodes (CC nodes). We draw an edge between (a_1, a_2) and (b_1, b_2) in $V \cup C \times V \cup C$ if either $(a_1, b_1) \in E$ and $a_2 = b_2$ or if $(a_2, b_2) \in E$ and $a_1 = b_1$.

Algebraic description: Let $\mathrm{H} \in \mathbb{F}_2^{m \times n}$ define the codes $\mathcal{C} = [n, k, d]$ and $\widetilde{\mathcal{C}} = [m, \widetilde{k}, \widetilde{d}]$ as

$$\mathcal{C} = \ker(\mathrm{H}) \quad \widetilde{\mathcal{C}} = \ker\left(\mathrm{H}^t\right) . \tag{4.1}$$

The X and Z stabilizers of the code are specified via their symplectic representation [88]. The parity check matrices of the quantum code are denoted H_X and H_Z respectively, where

$$\mathrm{H}_X = \left(\mathbb{1}_V \otimes \mathrm{H} \,|\, \mathrm{H}^t \otimes \mathbb{1}_C\right) \qquad \mathrm{H}_Z = \left(\mathrm{H} \otimes \mathbb{1}_V | \mathbb{1}_C \otimes \mathrm{H}^t\right) \ . \tag{4.2}$$

In this expression, VV nodes are in the left partition while CC nodes are in the right partition. Let $d_{\min} = \min\{d, \widetilde{d}\}$ denote the minimum of the distance of the two codes. The hypergraph product $\mathcal{Q}$ is a $[\![n^2 + m^2, k^2 + \widetilde{k}^2, d_{\min}]\!]$ quantum code. The reason why $d_{\min}$ is hamstrung by the distance of the underlying classical codes is shown below in lemma 14.

Lemma 13. *The code $\mathcal{Q}$ defined by the parity check matrices H_X and H_Z is a valid code.*

Proof. To ensure this is a valid code, we need to ensure that stabilizers commute. Therefore we need to verify that the inner product between any pair of X and Z stabilizers is 0, i.e. $\mathrm{H}_X \mathrm{H}_Z^t = 0 \pmod 2$. This inner product follows from the design of the parity check matrices. Indeed, we have

$$\mathrm{H}_X \mathrm{H}_Z^t = \mathrm{H}^t \otimes \mathrm{H} + \mathrm{H}^t \otimes \mathrm{H} = 0 \pmod 2 \ .$$

□

Ensuring that the stabilizers commute thus becomes a trivial task because of the way the code is designed. Constructing such codes is a difficult task because of a fundamental conflict. It is challenging to define quantum error correcting codes that simultaneously maintain the weight of stabilizers while at the same time ensuring that they commute. Demonstrating that good LDPC codes exist in classical coding theory is a much simpler task [89]. There have been several constructions that have attempted to design quantum LDPC codes in the past (for examples, see [90, 91, 92] for constructions based on Cayley graphs). These attempts had shortcomings in terms of achievable code parameters; in particular, these codes have growing distance, but scale poorly as a function of the block size. There also exist codes that are based on hyperbolic geometry [93, 94, 95, 96, 97]. Although hyperbolic codes achieve a linear scaling of k with the block size, they are still fundementally constrained [98].

The hypergraph product was the first class of codes with finite rate and distance growing as the square-root of the block size. We shall refer to the logical operators of the quantum code $\mathcal{Q}$ as the embedded logical operators to distinguish them from the logical operators that we introduce later by creating defects. The embedded logical operators of the code are described as follows.

Lemma 14. ***(Embedded logical operators)***

1. *the X logical operators of $\mathcal{Q}$ are spanned by*

$$\left(\ker\left(\mathrm{H}\right) \otimes \left(\mathbb{F}_2^n / \operatorname{rs}\left(\mathrm{H}\right)\right) | \boldsymbol{0}_{m^2}\right) \cup \left(\boldsymbol{0}_{n^2} \mid \left(\mathbb{F}_2^m / \operatorname{rs}\left(\mathrm{H}^t\right)\right) \otimes \ker\left(\mathrm{H}^t\right)\right)$$

2. *the Z logical operators of $\mathcal{Q}$ are spanned by*

$$\left(\left(\mathbb{F}_2^n / \operatorname{rs}\left(\mathrm{H}\right)\right) \otimes \ker\left(\mathrm{H}\right) | \boldsymbol{0}_{m^2}\right) \cup \left(\boldsymbol{0}_{n^2} \mid \ker\left(\mathrm{H}^t\right) \otimes \left(\mathbb{F}_2^m / \operatorname{rs}\left(\mathrm{H}^t\right)\right)\right)$$

Proof. The style of the proof follows arguments presented in lemma 17 of [18]. We first show that the spaces above are contained in the set of logical operators, and then use counting arguments to show that this must be the entire space of logical operators.

We deal with the X type logical operators and note that the Z logical operators follow using a similar argument. Let α be an X logical operator, i.e.

$$\alpha \in \left(\ker\left(\mathrm{H}\right) \otimes \left(\mathbb{F}_2^n / \operatorname{rs}\left(\mathrm{H}\right)\right) | \mathbf{0}_{m^2}\right) \cup \left(\mathbf{0}_{n^2} \mid \left(\mathbb{F}_2^m / \operatorname{rs}\left(\mathrm{H}^t\right)\right) \otimes \ker\left(\mathrm{H}^t\right)\right) \ .$$

This object clearly commutes with the Z stabilizers.

For the sake of contradiction, assume that α is in fact in the span of the X stabilizers, i.e. that there exists a non-trivial vector $a \in \mathbb{F}_2^{V \times C}$ such that $a\,\mathrm{H}_X = \alpha$. Without loss of generality, let us assume that the VV portion of α is non-trivial and let $\pi(\alpha)$ be the projection of α on to the VV type qubits.

It follows that

$$a(\mathbb{1}_V \otimes \mathrm{H}) = \pi(\alpha) \ . \tag{4.3}$$

For $u, v \in V$, we can index the elements of $\pi(\alpha)$ as $\pi(\alpha)[u, v]$. Furthermore, for fixed $u \in V$, we let $\pi(\alpha)[u, *]$ denote the vector over $\mathbb{F}_2^V$ obtained by fixing the first component of $\pi(\alpha)$.

Similarly, we can index the elements of a as $a[v, c]$ for $v \in V$ and $c \in C$ and let $a[v, *]$ denote the vector over $\mathbb{F}_2^C$. Eq. 4.3 implies that there exists some index $u \in V$ such that

$$b_u\,\mathrm{H} = \beta_u \ ,$$

where $b_u := a[u, *]$ and $\beta_u := \pi(\alpha)[u, *]$. However, this is a contradiction since $\beta_u \in \mathbb{F}_2^V / \operatorname{rs}\left(\mathrm{H}\right)$ and lies outside the row-span of H.

The row rank of H is $n-k$ and so the number of cosets in $\mathbb{F}_2^n/\operatorname{rs}(\mathrm{H})$ is $n-(n-k)=k$. Therefore the number of elements in $\ker(\mathrm{H})\otimes\mathbb{F}_2^n/\operatorname{rs}(\mathrm{H})$ is k^2. Similarly, the number of cosets $\mathbb{F}_2^m/\operatorname{rs}(\mathrm{H}^t)$ is $m-(m-\widetilde{k})=\widetilde{k}$. Therefore the number of elements in $\mathbb{F}_2^m/\operatorname{rs}(\mathrm{H}^t)\otimes\ker(\mathrm{H}^t)$ is $\widetilde{k}^2$. On counting the operators, we see that there are indeed k^2 vectors of VV type and $\widetilde{k}^2$ vectors of CC type, thus adding up to the correct number of logical operators. □

4.2 The small-set-flip algorithm

The `flip` algorithm introduced in chapter 1 is a linear time decoding algorithm for classical codes. Unfortunately the algorithm is not guaranteed to work when naively generalized to the quantum realm. There exist constant weight errors where the algorithm gets stuck. It will be instructive to study this further to motivate the quantum decoding algorithm `small-set-flip`.

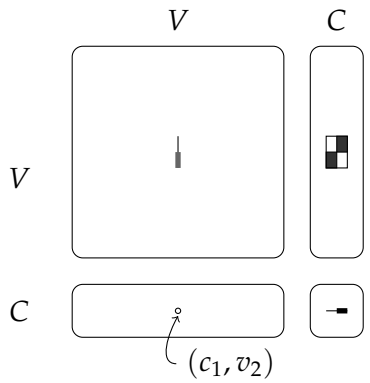


Figure 4.1 Schematic for hypergraph product of two classical codes. We have indicated the neighborhood of the Z stabilizer (c_1, v_2) on $V \times V$ and $C \times C$. Half of the neighborhoods have been marked to indicate the location of the Z error. The X stabilizers that detect this error are in $V \times C$. Stabilizers that are connected to one error are shaded, meaning these will have non-trivial syndromes. The rest will be connected to two errors and thus will have trivial syndromes.

Consider the Z stabilizer (c_1, v_2) as shown in the schematic of the hypergraph product. Its neighborhood is $\Gamma(c_1)\times\{v_2\}$ and $\{c_1\}\times\Gamma(v_2)$. Suppose we had a Z error that was half of each of the neighborhoods. Let $\mathcal{E}_V \in \Gamma(c_1)$ and $\mathcal{E}_C \in \Gamma(v_2)$ be the sets that are afflicted by Z. We let $\mathcal{E}_V^c := \Gamma(c_1)\setminus\mathcal{E}_V$ and $\mathcal{E}_C^c := \Gamma(v_2)\setminus\mathcal{E}_C$ denote the complements of the error sets within the neighborhoods of c_1 and v_2 respectively.

The set of unsatisfied X checks is therefore given by the sets

$$(\mathcal{E}_V \times \mathcal{E}_C^c) \cup (\mathcal{E}_V^c \times \mathcal{E}_C) \, .$$

Consider a variable node $u_1 \in \mathcal{E}_V$. If we find that it is connected to more unsatisfied than satisfied checks, this means that $|\mathcal{E}_C^c| \geq |\mathcal{E}_C|$. Therefore if $|\mathcal{E}_C^c| = |\mathcal{E}_C|$, then the algorithm will get stuck. Likewise if $|\mathcal{E}_V^c| = |\mathcal{E}_V|$, the algorithm will get stuck.

Thus there exists a constant weight error pattern that could confound `flip`. This is expected since `flip` is a local decoder, and it is stuck when it encounters degenerate errors [99]. We recall from eq. 2.4 that two errors E and E' are degenerate when there exists a stabilizer element $S \in \mathcal{S}$ such that

$$EE' = S \, .$$

Of course, the particular error pattern is contrived and is unlikely to occur in practice. However it is a symptom of degeneracy and could potentially derail our decoding algorithm. Degeneracy means that a Z error contained within the support of a Z check can be corrected either using the same error, or by applying Z errors on all the other qubits in the support of the Z check. In the case that the original error covers half of the support of a Z check, these two alternative will be equally likely. The local `flip` algorithm is unable to decide between the two alternatives so it gets stuck.

This motivates a simple modification – rather than flip individual qubits, flip a subset of qubits within the support of a Z stabilizer such that the syndrome is reduced. In this case we can flip $\mathcal{E}_V \times \{v_2\}$ and $\{c_1\} \times \mathcal{E}_C$ and this will fix the error. This algorithm is called small-set-flip.

Let us now formally state the result of Leverrier, Tillich and Zémor [100]. Suppose $\mathcal{G} = (V \cup C, E)$ is (Δ_V, Δ_C)-biregular bipartite graph. In other words, all its variable nodes have degree Δ_V and all its check nodes have degree Δ_C. We denote $|V| = n$ and $|C| = m$ such that $n \geq m$, $\Delta_V \leq \Delta_C$.

One key difference between the Sipser-Spielman requirements for decoding, in addition to the decoder, is that the expansion requirements for the underlying Tanner graphs are stronger. Recall that the `flip` algorithm relied on the Tanner graph being *expander* graphs, i.e. if we considered the neighborhood of a set of variable nodes, then it appears as if its neighborhood is growing very fast. We require that the graphs be *bipartite* expanders, a property that we now define. We stick to common notation that refer to these objects as left- and right-expanders. Before proceeding to definitions, we recall that for a graph

$\mathcal{G} = (V \cup C, E)$, the edges $E = V \times C$. The variable nodes are on the left of this partition, and check nodes on the right. Thus if the variable nodes are expanding, the graph is said to be left-expanding. On the other hand if the check nodes are expanding, the graph is said to be right-expanding.

Definition 15. *Directional expansion can be defined as follows:*

1. (γ_V, δ_V) *left-expanding if* $\forall S \subseteq V$

$$|S| \leq \delta_V n \implies |\Gamma(S)| \geq \gamma_V |S| \,.$$

2. (γ_C, δ_C) *right-expanding if* $\forall T \subseteq C$

$$|T| \leq \delta_C m \implies |\Gamma(T)| \geq \gamma_C |T| \,.$$

3. *If the graph is both left- and right-expanding, then we say it is a* $(\gamma_V, \delta_V, \gamma_C, \delta_C)$*-expander.*

Why is the story here more complicated? We want to port the intuition from the classical algorithm to the quantum algorithm: If a set of qubits is afflicted by an error, we want as many checks as possible to flag this error. In the hypergraph product, both variable and check nodes from the classical Tanner graphs are promoted to qubit type nodes. Therefore unless the check nodes also obey an expansion property, this cannot happen.

Theorem 16. *Let* $G = (V \cup C, E)$ *be a* (Δ_V, Δ_C)*-biregular* $(\gamma_V, \delta_V, \gamma_C, \delta_C)$*-expander. Assume* $\gamma_V \geq 5\Delta_V/6$ *and* $\gamma_C \geq 5\Delta_C/6$*. Letting* Δ_V, Δ_C *be fixed and allowing* n, m *to grow, there exists a decoding algorithm for the associated quantum code* $\mathcal{Q}$ *that runs in time* $O(n^2 + m^2)$ *and corrects errors of weight up to* $O(\min(n, m))$*.*

The algorithm is presented in algorithm 2 below.

Readers interested in perusing the original paper by Leverrier, Tillich and Zémor will find it handy to have the following expander dictionary to translate between the notation we use here and the original. γ_{LTZ} and δ_{LTZ} are what they refer to as γ and δ.

Expander dictionary: $\delta_{LTZ} = 1 - \gamma/\Delta$ and $\gamma_{LTZ} = \delta$.

Building on this result, the authors of [101] were able to show that the hypergraph product codes equipped with the `small-set-flip` algorithm exhibited threshold behavior. In other words, if the probability of physical error was below some threshold value, then the decoder gets better with increasing block size. They first show this for adverserial noise

Algorithm 2 Input: syndrome vector $s \in \mathbb{F}_2^{V \times C}$

$s_0 \leftarrow s$, syndrome at initial time
$e \leftarrow \varnothing$, deduced error
while $|s_i| \neq 0$ **do**
  $s_i = s_{i-1} + \sigma_X(e_i)$
  $e_i =$ **ArgMax**: $(|s_{i-1}| - |s_i|)/|e_i|$
  Subject to:
  • supp $\{e_i\} \subseteq (\Gamma(c) \times v) \cup (c \times \Gamma(v))$, for any $c \in C$, $v \in V$.
  • $|s_i| < |s_{i-1}|$
  if such an e_i does not exist **then**
    output failure
  else
    $e_{ded} \leftarrow e_{ded} \, \Delta \, e_i$,
  end if
end while
Output e_{ded}

models and for errors of weight up to square-root of the block-size. They then show that with high probability, the small-set-flip decoder corrects a constant fraction of random errors in the case of perfect syndrome measurements. Analytical lower bounds for the threshold were also added but these are difficult to obtain, and rather poor for practice.

Author contributions: The following summarizes some results of numerical simulations done together with Antoine Grospellier.

A better way to ascertain the performance of these codes is numerical simulation. The simulation was performed using a simple noise model as shown in [102]. The qubits are assumed to have been afflicted by independent bit and phase flip errors. For simplicity, we assume that both occur with the same probability p. The logical error rate of these codes versus the strength of the noise is shown below in fig. 4.2.

Since this is a CSS code, the X and Z parts could be decoded separately. The quantum code is formed as a product of the classical code with itself and is labeled by n and m, the number of variable and check nodes respectively. The resulting quantum code is a $[\![n^2 + m^2, n^2 - m^2]\!]$ code. In particular, the classical code used to construct the code has degrees 5 and 6, meaning the variable nodes all have degree 5 and the check nodes all have degree 6. The code was chosen to have as low a degree as possible because the complexity of decoding increasing exponentially in the degree of the code. On the other hand, this means that the rate of the code albeit a constant is not very high, and is roughly 0.016.

The toric code is a code closely related to the surface code (and actually came before the

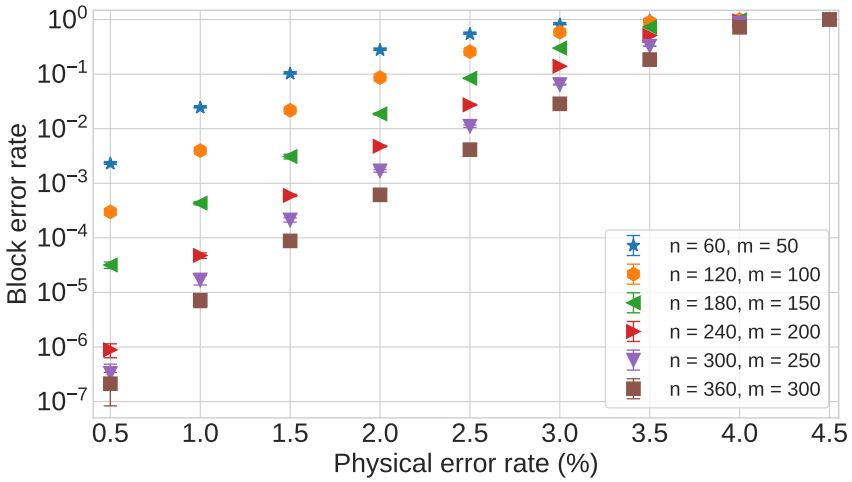


Figure 4.2 Results of simulation of $(5,6)$-codes, i.e. this figure shows the logical error rate versus physical error rate of hypergraph product codes formed as the product of biregular graphs with degrees 5 and 6.

surface code). Comparing the toric code and the hypergraph product code is not straightforward because at the outset it is not obvious which parameters to fix. In [102], the authors compared the logical error rate for the hypergraph product and $k/2$ copies of the toric code for a toric code of the same rate as the hypergraph product code. They showed that by the time the code carried about 500 logical qubits, the hypergraph product code was able to perform better than the surface code.

Recently, Fawzi et al. [103] showed that in the presence of syndrome noise, the `small-set-flip` decoder leaves a number of residual qubit errors on the state after decoding. The number of errors is proportional to the number of syndrome errors. This is reminiscent Spielman's result for classical codes (See chapter 1). These results require even stronger expansion constraints than that required by [100].

How do we construct such graphs? At present, we do not know of any deterministic ways to construct graphs with such large expansion factors. Random constructions of graphs are known to yield graphs with the desired expansion. However, the problem with these constructions is that these are only asymptotic guarantees and may require large block sizes and large degrees before the expansion 'kicks in'. These expansion factors have become increasingly challenging the more we ask of the code. Recall that in the classical case, Sipser and Spielman required an expansion of $\gamma \geq 3/4\Delta_V$. The `small-set-flip` required an expansion of $\gamma \geq 5/6\Delta$ (both for V and C) to show that the decoder can correct errors of weight proportional to the distance. The next iteration by Fawzi et al. [101] required $\gamma \geq 7/8\Delta$ (for both V and C) to guarantee that the `small-set-flip` decoder could correct

a constant fraction of errors on average. Finally, `small-set-flip` requires an expansion factor of $\gamma \geq 15/16\Delta$ (for both V and C) to guarantee that the decoder is fault tolerant. We add that these are requirements to prove theorems; in the real world, simulations indicate that we may be able to get away with less.

What can we expect in the future? Inspired by developments in classical coding theory, we can probably bet that variants of the flip decoder may not be the ultimate decoding algorithm. The classical `flip` algorithm, as well its quantum counterpart, only become effective for very large block lengths. For modern applications such as WiFi, we use variants of iterative decoding algorithms in the classical realm. Recently Pantaleev et al. have demonstrated some remarkable results [104] for belief propagation on small codes. This may point the way to improving decoders in the future.

4.3 Chapter summary

In this chapter, we introduced the hypergraph product code family discovered by Tillich and Zémor. We demonstrated that this is a generalization of the technique illustrated in chapter 3 used to construct the surface code from two repetition codes. We illustrated that the commutation relations between the X and Z stabilizers follow immediately from the definition of the code. We then proceeded to discuss the `flip` algorithm to decode hypergraph product codes and why it fails. We discussed how the `small-set-flip` decoder of Leverrier, Tillich and Zémor overcomes this problem. We then reviewed some recent developments in the decoding of hypergraph product codes. We highlighted the work of Fawzi et al. [101], [103] where it was shown that the `small-set-flip` decoder could be used together with quantum expander codes to achieve fault-tolerant quantum computation. Finally we supplemented this with some numerical results by Grospellier and Krishna [102] to show that these codes have reasonable thresholds.

This is strong evidence that the hypergraph product codes can be used as quantum memories. We are now ready to understand how to use these codes to manipulate encoded information.

Chapter 5

Defects on Hypergraph Product Codes

As explained in chapter 3, defect-based encodings are one way to encode information in the surface code. In this chapter, we describe the first techniques to perform Clifford gates fault tolerantly on hypergraph product codes by generalizing defect-based techniques. Our method is an instance of code deformation, a general framework to perform gates on quantum codes. Continuing in the spirit of the hypergraph product construction, we express defects on the surface code as purely algebraic and graph-theoretic concepts. Importantly, the code remains LDPC over the course of code deformation. Code deformation entails modifying the hypergraph product code sequentially. Although we end at the same code we started with, the interim codes will involve modifications which we shall describe in detail below. Fault tolerance follows because the modifications we make at each step as we transition from one code in the sequence to the next are local (in a graph-theoretic sense). The generalized defects are capable of encoding several logical qubits. If we choose to use a subset of these qubits to encode information, then the rest can be considered as gauge qubits. A gauge qubit is a virtual qubit in which we choose not to encode any information. We discuss constraints on code deformation that keeps the spaces of logical and gauge qubits separate. To conclude, we show that we can achieve a universal gate set on the logical qubits via state injection.

We emphasize at this point that we are not providing a technique to *compile* a Clifford gate of interest. We provide a framework within which it is possible to realize Clifford gates via code deformation. These Clifford gates can then be composed to generate a larger group of transformations. We show that the framework is sufficiently rich to realize all different types of generating gates, but depending on the code, this may or may not encompass the entire space of Clifford operators. We will return to this discussion later.

This approach can be contrasted to Gottesman's work which entails partitioning logical qubits into blocks of LDPC codes. Each block is of 'intermediate' size and a computation with k logical qubits requires blocks whose size scales as $O(k/\operatorname{poly}\log(k))$. Gates are then performed by state injection using ancilla states. This was described in chapter 2. The size of these blocks limits the number of gates that can be implemented at any given time. Furthermore, the savings of LDPC codes become compelling only as we increase the block size. Partitioning the logical qubits into blocks implies that it will take longer for this effect to manifest. In contrast, we propose performing quantum computation on a single block. Our proposal does not limit the number of qubits that can be processed at any given time to a constant. On the other hand, the time required to perform a gate could scale so it is unclear whether these gates will be faster.

The material in this chapter appears in [20].

5.1 Punctures

A puncture is a defect on the hypergraph product created by removing both qubits and stabilizers belonging to some (small) portion of the code. This shall be effected by measuring single-qubit Pauli operators within the interior of the puncture. This is similar to creating a puncture on the surface code [105].

5.1.1 Definition

We begin this section with some notation. Let $S \subseteq V$ denote a connected subset of variable nodes, i.e. for every $u \in S$, there exists at least one check node c such that c is connected to another variable node $v \in S$. $N = \Gamma(S) \subseteq C$ is its neighborhood and $A = \Gamma^{-1}(S)$ is its ancestor as shown in fig. 5.1(a).

$$N = \{c \in C : \exists u \in S \text{ such that } (u,c) \in E\} \qquad A = \{c \in C : \forall u \in \Gamma(c), u \in S\}.$$

For any set $V' \subseteq V$, we let $\mathbb{1}_{V'}$ denote the projector on V' over $\mathbb{F}_2^V$. We also write $\mathrm{H}_{V'} = \mathrm{H}\mathbb{1}_{V'}$ for the restriction of the parity check matrix to V'.

Similarly, let $T \subseteq C$ denote a connected subset of check nodes. $M = \Gamma(T) \subseteq V$ is its neighborhood and $B = \Gamma^{-1}(T)$ is its inverse neighborhood as shown in fig. 5.1(b).

$$M = \{v \in V : \exists c \in T \text{ such that } (v,c) \in E\} \qquad B = \{v \in V : \forall c \in \Gamma(v), c \in T\}.$$

For any set $C' \subseteq C$, we let $\mathbb{1}_{C'}$ denote the projector on C' over $\mathbb{F}_2^C$. We also write $\mathrm{H}_{C'} = \mathbb{1}_{C'}\,\mathrm{H}$ for the restriction of the parity check matrix to C'.

Note that by definition, $A \subseteq N$, and $B \subseteq M$.

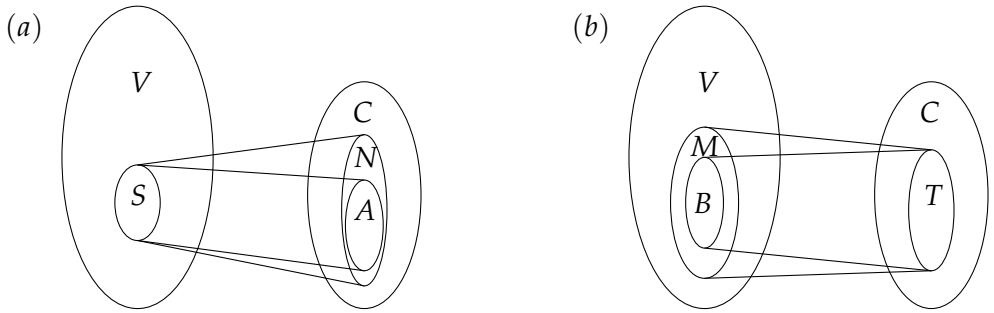


Figure 5.1 Schematic of factor graphs. (a) Denotes the subgraph induced by S. N is its neighborhood and A is its ancestor. (b) Denotes the subgraph induced by T. M is its neighborhood and B is its ancestor.

At this juncture, we make some observations that will be useful later.

$$\Gamma(A) \subseteq S \implies \Gamma(S^c) \subseteq A^c \qquad \Gamma(B) \subseteq T \implies \Gamma(T^c) \subseteq B^c$$
$$A^c = N^c \cup (N \setminus A) \qquad B^c = M^c \cup (M \setminus B)\,.$$

To create a puncture on the quantum code, we will stop measuring certain stabilizers, and modify others when carving out a portion of the interior. The punctures will be classified by how stabilizers are modified.

Definition 17 (Smooth puncture). *Let $S \subseteq V$ and $T \subseteq C$ be connected sets of variable and check nodes. Let N, A and M, B denote induced sets as defined above. A smooth puncture is defined by the stabilizers H'_X and H'_Z where*

$$\mathrm{H}'_X = \left(\mathbb{1}_B \otimes \mathrm{H}_S \,|\, \mathrm{H}_T^t \otimes \mathbb{1}_A\right) \qquad \mathrm{H}'_Z = \left(\mathrm{H}_T \otimes \mathbb{1}_S | \mathbb{1}_T \otimes \mathrm{H}_S^t\right)\,.$$

Note that this is not exactly the graph product of the two subgraphs selected by T and S. This is verified by noting that it is missing elements from $M \times S$. Rather it follows the hypergraph product construction on the interior nodes of both graphs. For simplicity, we abuse notation and refer to this as the graph product $T \times S$.

The defining trait of a smooth puncture is that Z stabilizers are not broken across its boundary. We refer to the schematic in fig. 5.2(a) below. Such stabilizers would have to be of the form (c, v) for some check $c \in C$ and $v \in V$ where either the check node c or the vari-

able node v are in the boundary of T or S respectively. This does not exist by construction – check nodes in T are contained entirely within the puncture, as are variable nodes in S. The internal qubits of a smooth puncture are the nodes $B \times S \cup T \times A$ and will be measured in the Z basis to create the puncture. The qubits on the boundary of a smooth puncture correspond to the sets

$$(M \setminus B) \times S \cup T \times (N \setminus A) \,.$$

The X stabilizers on the boundary of a smooth puncture correspond to the sets

$$(M \setminus B) \times N \cup M \times (N \setminus A) \,.$$

Their support on the interior of the puncture, $B \times S \cup T \times A$, is removed. Therefore X stabilizers on the boundary of a smooth puncture are broken.

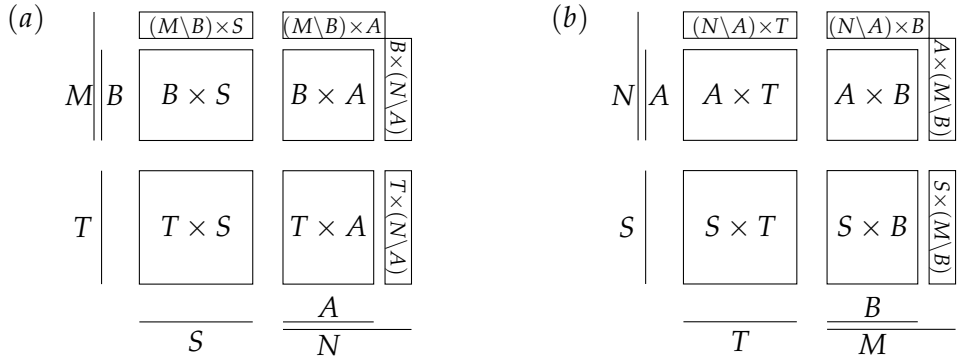


Figure 5.2 Schematic for a puncture on a hypergraph product code. The subgraphs selected by T and S are flattened and placed below and to the left. Their product is represented using four quadrants. The qubits are in the North-West and South-East quadrants. The Z stabilizers are in the South-West quadrant. The X stabilizers are in the North-East quadrant. (a) Smooth puncture defined by T and S. The Z stabilizers $T \times S$ are completely within the puncture and are thus not broken. (b) Rough puncture defined by S and T. The X stabilizers $S \times T$ are completely within the puncture and are thus not broken.

In a similar manner, a rough puncture can be created by interchanging the roles of T and S on the graphs. It is formally defined as follows.

Definition 18 (Rough puncture). *Let $S \subseteq V$ and $T \subseteq C$ be connected sets of variable and check nodes. Let N, A and M, B denote induced sets as defined above. A rough puncture is defined by the stabilizers H'_X and H'_Z where*

$$\mathrm{H}'_X = \left(\mathbb{1}_S \otimes \mathrm{H}_T \,|\, \mathrm{H}_S^t \otimes \mathbb{1}_T\right) \qquad \mathrm{H}'_Z = \left(\mathrm{H}_S \otimes \mathbb{1}_B | \mathbb{1}_A \otimes \mathrm{H}_T^t\right) \,.$$

Abusing notation, this can be thought of as a graph product $S \times T$.

The defining trait of a rough puncture is that X stabilizers are not broken across the boundary. We refer to the schematic in fig. 5.2 (b) for the following discussion. Such stabilizers would have to be of the form (v, c) for some check $c \in C$ and $v \in V$ where either the check node c or the variable node v are in the boundary of S or T respectively. For the same reasons as before, such nodes do not exist. The internal qubits of a rough puncture are the nodes $S \times B \cup A \times T$ and will be measured in the X basis to create the puncture. The qubits on the boundary of a rough puncture correspond to the sets

$$S \times (M \setminus B) \cup (N \setminus A) \times T \, .$$

The Z stabilizers on the boundary of a rough puncture correspond to the sets

$$N \times (M \setminus B) \cup (N \setminus A) \times M \, .$$

Their support on the interior of the puncture, $S \times B \cup A \times T$ is removed. Hence Z stabilizers on the boundary of a rough puncture are broken.

To deform $\mathcal{Q}$, we remove the edges that are contained in a puncture. Algebraically, it is described by $\mathrm{H}_X + \mathrm{H}'_X$ and $\mathrm{H}_Z + \mathrm{H}'_Z$. This code is itself not a hypergraph product code but is clearly LDPC. We are merely puncturing an LDPC code; by removing edges, we cannot increase the weight of checks.

We first show that the code defined this way obeys the desired commutation relations. Before doing so, it is useful to note the following identity:

$$\mathrm{H}_B = \mathrm{H}\mathbb{1}_B = \mathbb{1}_T \mathrm{H}\mathbb{1}_B \qquad \mathrm{H}_A = \mathbb{1}_A \mathrm{H} = \mathbb{1}_A \mathrm{H}\mathbb{1}_S \, . \tag{5.1}$$

This follows from the fact that the neighborhoods of sets B and A are completely contained within the sets T and S by definition.

Lemma 19. *The punctured code forms a valid stabilizer code.*

Proof. Consider a smooth puncture created by two subsets $T \subseteq C$ and $S \subseteq V$. The case of a rough puncture follows similarly. The punctured code has stabilizers

$$\mathrm{H}_X + \mathrm{H}'_X \qquad \mathrm{H}_Z + \mathrm{H}'_Z \, .$$

We already know that $H_X H_Z^t = 0 \pmod 2$. We need to check the other relations.

$$\begin{aligned}
H_X(H'_Z)^t &= (\mathbb{1}_V \otimes H \,|\, H^t \otimes \mathbb{1}_C)\left[(H_T \otimes \mathbb{1}_S | \mathbb{1}_T \otimes H_S^t)\right]^t \\
&= H_T^t \otimes H_S + H_T^t \otimes H_S = 0 \pmod 2 \\
H'_X(H_Z)^t &= (\mathbb{1}_B \otimes H_S \,|\, H_T^t \otimes \mathbb{1}_A)\left[(H \otimes \mathbb{1}_V | \mathbb{1}_C \otimes H^t)\right]^t \\
&= H_B^t \otimes H_S + H_T^t \otimes H_A \\
H'_X(H'_Z)^t &= (\mathbb{1}_B \otimes H_S \,|\, H_T^t \otimes \mathbb{1}_A)\left[(H_T \otimes \mathbb{1}_S | \mathbb{1}_T \otimes H_S^t)\right]^t \\
&= H_B^t \otimes H_S + H_T^t \otimes H_A \ .
\end{aligned}$$

In the last line, we have used the identity in eq. 5.1. Inspecting the last two equations, we find that each term appears twice. Therefore the sum of all the terms in these two equations is 0 mod 2 as desired. □

For convenience, we have summarized this section in table 5.1.

	Smooth puncture	Rough puncture
H'_X	$(\mathbb{1}_B \otimes H_S \,\|\, H_T^t \otimes \mathbb{1}_A)$	$(\mathbb{1}_S \otimes H_T \,\|\, H_S^t \otimes \mathbb{1}_T)$
H'_Z	$(H_T \otimes \mathbb{1}_S \| \mathbb{1}_T \otimes H_S^t)$	$(H_S \otimes \mathbb{1}_B \| \mathbb{1}_A \otimes H_T^t)$
Internal qubits	$(B \times S) \cup (T \times A)$	$(S \times B) \cup (A \times T)$
Boundary qubits	$(M \setminus B) \times S \cup T \times (N \setminus A)$	$S \times (M \setminus B) \cup (N \setminus A) \times T$
Boundary X stabilizers	$(M \setminus B) \times N \cup M \times (N \setminus A)$	$\varnothing$
Boundary Z stabilizers	$\varnothing$	$N \times (M \setminus B) \cup (N \setminus A) \times M$

Table 5.1 Summary of properties of punctures. We assume that $S \subseteq V$ and $T \subseteq C$ are (connected) subsets of variable and check nodes. S induces the sets N and A, its neighborhood and ancestor, and similarly T induces the sets M and B.

5.1.2 Logical Pauli operators for punctures

Mirroring the surface code, punctured hypergraph product codes support two types of logical operators - *loop-type operators* that exist only on the boundary of the puncture and *chain-type operators* that are supported on the boundary of the puncture and also extend

into the rest of the code. For the rest of this section, we let $T \subseteq C$ and $S \subseteq V$ be some connected subsets. The sets $M \subseteq V$, $N \subseteq C$ are the respective neighborhoods, and $B \subseteq V$, $A \subseteq C$ are the respective ancestors. We shall derive the form of the logical operators for a smooth puncture defined as above. The logical operators of a rough puncture will follow by exchanging the roles of S and T.

Before proceeding, we impose certain constraints on how S and T are chosen. These constraints apply to smooth and rough punctures both. The constraints stipulate that certain subcodes associated with S and T are *correctable*, i.e. if these portions of the code were erased then we do not lose any codewords of the underlying code in this process.

Definition 20 (Correctability)**.** *A puncture defined by $T \subseteq C$ and $S \subseteq V$ is* correctable *if and only if:*

1. $\ker(\mathrm{H}_S) = \ker\left(\mathrm{H}_T^t\right) = \varnothing$.
2. $\ker\left(\mathrm{H}_A^t\right) = \ker(\mathrm{H}_B) = \varnothing$.

This definition will help establish that creating a puncture will not affect the embedded logical operators. We begin by considering the embedded Z logical operators.

Lemma 21. *There are no embedded Z logical operators supported within the interior of the puncture.*

Proof. The proof idea is to show that if an embedded Z logical operator were completely contained within the puncture, it violates definition 20, part 1. This will entail projecting down to the level of the classical code until we arrive at a contradiction.

Let $\alpha \in \mathcal{L}_Z$ be a logical Z operator , i.e. as given by lemma 14

$$\alpha \in \left(\mathbb{F}_2^n / \operatorname{rs}(\mathrm{H}) \otimes \ker(\mathrm{H}) \,|\mathbf{0}_{m^2}\right) \cup \left(\mathbf{0}_{n^2} |\ker\left(\mathrm{H}^t\right) \otimes \mathbb{F}_2^m / \operatorname{rs}\left(\mathrm{H}^t\right)\right). \tag{5.2}$$

For the sake of contradiction, let α be supported entirely within the interior of the puncture, i.e. only on $B \times S \cup T \times A$.

Without loss of generality, suppose the VV part of α is non-trivial. Let $\pi(\alpha)$ denote the projection of α onto the VV qubits. Let us index the elements of $\pi(\alpha)$ using variable nodes $u, v \in V$ as $\pi(\alpha)[u, v]$. Furthermore, let $\pi(\alpha)[u, *]$ denote the vector obtained by fixing the first component to u. Since $\pi(\alpha)$ is non-trivial and supported on $B \times S$, there must exist at least one $u \in B$ such that the vector $\beta_u := \pi(\alpha)[u, *]$ is non-trivial.

Furthermore, it also implies that β_u is supported only on S. However, this in turn implies that there exists a non-trivial element in $\ker(\mathrm{H}_S)$, i.e. that $\mathrm{H}_S\,\beta_u = 0$.

This violates definition 20, part 1. Therefore there cannot be any logical Z operators completely contained within the interior. □

Next, we proceed to show that there are no embedded X logical operators within the puncture.

Lemma 22. *There are no embedded logical X operators within the interior of the puncture.*

Proof. The proof idea is to show that if a logical X operator were contained entirely within the puncture, it violates definition 20, part 2.

Let $\alpha \in \mathcal{L}_X$ be a logical X operator, i.e. as given by lemma 14

$$\alpha \in (\ker(\mathrm{H}) \otimes \mathbb{F}_2^n / \operatorname{rs}(\mathrm{H}) \,|\mathbf{0}_{m^2}) \cup (\mathbf{0}_{n^2} | \mathbb{F}_2^m / \operatorname{rs}\left(\mathrm{H}^t\right) \otimes \ker\left(\mathrm{H}^t\right)) \,.$$

For the sake of contradiction, let α be supported entirely within the interior of the puncture, i.e. only on $B \times S \cup T \times A$.

Without loss of generality, let us assume that the VV part of α is non-trivial. Let $\pi(\alpha)$ denote the projection of α onto the VV qubits. Let us index the elements of $\pi(\alpha)$ using variable nodes $u, v \in V$. It follows that $\pi(\alpha)$ is supported entirely on $B \times S$.

Let $\pi(\alpha)[*, v]$ denote the vector obtained by fixing the second component to v. Since $\pi(\alpha)$ is non-trivial, there must exist at least one $v \in S$ such that the vector $\beta_v := \pi(\alpha)[*, v] \in \ker(\mathrm{H})$.

By assumption, since α is supported entirely on the interior, β_v is supported only on B. However this in turn implies that there exists a non-trivial element in $\ker(\mathrm{H}_B)$.

This violates definition 20, part 2. Therefore the puncture cannot contain any logical X operators. □

We will later argue that these conditions can be used together with the cleaning lemma [80] to guarantee that the embedded logical operators of the quantum code are unaffected by the puncture.

We will eventually show that certain operators are orthogonal to the stabilizers $\mathrm{H}_X + \mathrm{H}'_X$ and $\mathrm{H}_Z + \mathrm{H}'_Z$. Under certain conditions, some of these operators will not lie in the span of

the stabilizers themselves. To this end, we now state and prove two lemmas that will be useful for proving this claim.

We assume that the classical codes obey certain independence relations.

Definition 23. *A puncture is defined by $T \subseteq C$ and $S \subseteq V$ obeys* independence relations *if and only if:*

1. $\mathrm{rs}\left(\mathrm{H}_{T^c}\right) \cap \mathrm{rs}\left(\mathrm{H}_T\right) = \mathrm{rs}\left(\mathrm{H}^t_{S^c}\right) \cap \mathrm{rs}\left(\mathrm{H}^t_S\right) = \varnothing$.
2. $\ker\left(\mathrm{H}_{T^c}\right) \cap \ker\left(\mathrm{H}_T\right) = \ker\left(\mathrm{H}^t_{S^c}\right) \cap \ker\left(\mathrm{H}^t_S\right) = \varnothing$.

Remark: we clarify the meaning of condition 2. Of course, if $g \in \ker(\mathrm{H})$, then $g \in \ker\left(\mathrm{H}_{T^c}\right)$ as well as $\ker\left(\mathrm{H}_T\right)$. We disregard these operators – $\ker\left(\mathrm{H}_{T^c}\right)$ and $\ker\left(\mathrm{H}_T\right)$ refer to $\ker\left(\mathrm{H}_{T^c}\right) \setminus \ker(\mathrm{H})$ and $\ker\left(\mathrm{H}_T\right) \setminus \ker(\mathrm{H})$ respectively. Likewise, $\ker\left(\mathrm{H}^t_{S^c}\right)$ and $\ker\left(\mathrm{H}^t_S\right)$ refer to $\ker\left(\mathrm{H}^t_{S^c}\right) \setminus \ker\left(\mathrm{H}^t\right)$ and $\ker\left(\mathrm{H}^t_S\right) \setminus \ker\left(\mathrm{H}^t\right)$ respectively.

The next lemma will be useful in showing that operators in the row-space of H'_Z are not in the span of the Z stabilizers $\mathrm{H}_Z + \mathrm{H}'_Z$, i.e. they are not redundant. These operators will later be used to construct logical operators.

Lemma 24. *The stabilizers $\mathrm{H}_Z + \mathrm{H}'_Z$ outside the puncture are independent of the stabilizers H'_Z within the puncture, i.e.*

$$\mathrm{rs}\left(\mathrm{H}_Z + \mathrm{H}'_Z\right) \cap \mathrm{rs}\left(\mathrm{H}'_Z\right) = \varnothing\ .$$

Proof. Let $\alpha \in \mathrm{rs}\left(\mathrm{H}'_Z\right)$ such that it is supported entirely on $(M \setminus B) \times S \cup T \times (N \setminus A)$. For the sake of contradiction, suppose there exists a vector $a \in \mathbb{F}_2^{C \times V}$ such that

$$a(\mathrm{H}_Z + \mathrm{H}'_Z) = \alpha\ . \tag{5.3}$$

Without loss of generality, assume that the VV portion of α is non-trivial. Let $\pi(\alpha)$ denote the projection of α onto its VV part. For $u, v \in V$, we can index the elements of $\pi(\alpha)$ as $\pi(\alpha)[u, v]$. Specifically, let $\pi(\alpha)[u, S]$ denote the restriction of $\pi(\alpha)$ to elements u, v such that $v \in S$. Note that for any fixed $v \in V$, we have $\pi(\alpha)[*, v] \in \mathrm{rs}\left(\mathrm{H}_T\right)$.

We can then write eq. 5.5 as

$$a(\mathrm{H}_{T^c} \otimes \mathbb{1}_S) = \pi(\alpha)[*, S]\ . \tag{5.4}$$

Let $v \in S$ such that $\pi(\alpha)[*, v]$ is non-trivial. This implies that $\mathrm{rs}\left(\mathrm{H}_{T^c}\right) \cap \mathrm{rs}\left(\mathrm{H}_T\right) \neq \varnothing$, violating definition 23, part 1. Therefore the two sets are independent. □

Lemma 25. *Let* $\alpha \in \mathbb{F}_2^{n^2+m^2}$ *such that* $\operatorname{supp}\{\alpha\} \cap (M \setminus B) \times S \cup T \times (N \setminus A) \neq \varnothing$ *and it lies in one of the two following sets:*

1. $\left(\ker\left(\mathrm{H}_{T^c}\right) \otimes \mathbb{F}_2^S / \operatorname{rs}\left(\mathrm{H}_A\right) | \boldsymbol{0}_{m^2}\right)$;
2. $\left(\boldsymbol{0}_{n^2} | \mathbb{F}_2^T / \operatorname{rs}\left(\mathrm{H}_B^t\right) \otimes \ker\left(\mathrm{H}_{S^c}^t\right)\right)$.

Then α *does not lie in the row-span of* $\mathrm{H}_X + \mathrm{H}'_X$.

Proof. We shall focus on the first object, $\left(\ker\left(\mathrm{H}_{T^c}\right) \otimes \mathbb{F}_2^S / \operatorname{rs}\left(\mathrm{H}_A\right) | \boldsymbol{0}_{m^2}\right)$, and note the other follows identically.

Let $\alpha \in \left(\ker\left(\mathrm{H}_{T^c}\right) \otimes \mathbb{F}_2^S / \operatorname{rs}\left(\mathrm{H}_A\right) | \boldsymbol{0}_{m^2}\right)$ such that it has non-trivial support on $(M \setminus B) \times S$.

For the sake of contradiction, suppose it is in the row-span of $\mathrm{H}_X + \mathrm{H}'_X$. There exists a vector $a \in \mathbb{F}_2^{V \times C}$ such that

$$a(\mathrm{H}_X + \mathrm{H}'_X) = \alpha \,. \tag{5.5}$$

Let $\pi(\alpha)$ denote the projection of α onto its VV part. For $u, v \in V$, we can index the elements of $\pi(\alpha)$ as $\pi(\alpha)[u, v]$. Similarly, we can index a as $a[v, c]$ for $v \in V$ and $c \in C$.

First, consider the VV part of eq. 5.5, expressed as

$$a(\mathbb{1}_V \otimes \mathrm{H} + \mathbb{1}_B \otimes \mathrm{H}_S) = \pi(\alpha) \,. \tag{5.6}$$

Since $\pi(\alpha)$ is non-trivial, there exists at least one $c \in N \setminus A$ such that $a[*, c] \in \ker\left(\mathrm{H}_{T^c}\right)$. This can be seen as follows. The set S is only connected to the set N; indeed N is the neighborhood of S. Furthermore, for all $u \in B^c$, we have $\pi(\alpha)[u, *] \in \mathbb{F}_2^S / \operatorname{rs}\left(\mathrm{H}_A\right)$. Therefore we may assume that $c \notin A$ which implies $c \in N \setminus A$.

This assumption however contradicts the CC part of eq. 5.5, expressed as

$$a(\mathrm{H}^t \otimes \mathbb{1}_C + \mathrm{H}_T^t \otimes \mathbb{1}_A) = 0 \,. \tag{5.7}$$

Indeed, projecting the LHS on to $C \times A^c$, i.e. multiply from the right by $\mathbb{1}_C \otimes \mathbb{1}_{A^c}$, we get

$$a(\mathrm{H}^t \otimes \mathbb{1}_{A^c}) = a(\mathrm{H}_T^t \otimes \mathbb{1}_{A^c}) \neq 0 \,. \tag{5.8}$$

This is because by definition 23, part 2 we have assumed that $\ker\left(\mathrm{H}_T\right) \cap \ker\left(\mathrm{H}_{T^c}\right) = \varnothing$. □

Logical operators: With these conditions, we can study the logical Z operators that emerge by creating a puncture. These operators can be classified in terms of the (classical) codespaces associated with H_A and H_B^t.

Theorem 26. *Let $\widetilde{\mathrm{G}}_B$ and G_A be the generator matrices for the codespaces defined by H_B^t and H_A respectively, i.e. the rows of $\widetilde{\mathrm{G}}_B$ and G_A span $\ker\left(\mathrm{H}_B^t\right)$ and $\ker\left(\mathrm{H}_A\right)$ respectively. The logical Z operators are spanned by*

$$(\widetilde{\mathrm{G}}_B^t \otimes \mathrm{G}_A^t)\,\mathrm{H}_Z' \ .$$

Proof. We begin from first principles. The stabilizers are given by $\mathrm{H}_X + \mathrm{H}_X'$ and the single-qubit operators in the interior of the puncture are described by the matrix $\mathrm{INT} = (\mathbb{1}_B \otimes \mathbb{1}_S | \mathbb{1}_T \otimes \mathbb{1}_A)$. The logical operators are defined as $\ker\left(\mathrm{H}_X + \mathrm{H}_X' + \mathrm{INT}\right) / \mathrm{rs}\left(\mathrm{H}_Z + \mathrm{H}_Z'\right)$.

Part 1: $\ker\left(\mathrm{H}_X + \mathrm{H}_X' + \mathrm{INT}\right)$

Suppose $\alpha \in \mathbb{F}_2^{n^2+m^2}$ such that $\alpha \in \ker\left(\mathrm{H}_X + \mathrm{H}_X' + \mathrm{INT}\right)$. We can assume that α is not supported in the interior, and consider the kernel of $\mathrm{H}_X + \mathrm{H}_X'$ instead of $\mathrm{H}_X + \mathrm{H}_X' + \mathrm{INT}$. We use lemma 21 together with the cleaning lemma [80] to note that the embedded logicals are unaffected by the puncture. Any other Z type operator that is supported in the interior will anti-commute with the single-qubit X measurements used to generate the puncture and therefore will be removed.

The operator α must therefore lie in the kernel of H_X outside the puncture. This contains the embedded logical operators and products of old stabilizers that were not in the interior. We shall only focus on the latter here in order to obtain the new logical operators.

The stabilizers in the interior are spanned by H_Z'. Those operators in $\mathrm{rs}\left(\mathrm{H}_Z'\right)$ but not supported in the interior are thus what we seek. The interior of the puncture corresponds to $B \times S \cup T \times A$. Let $a \in \mathbb{F}_2^{C \times V}$ such that $\mathrm{supp}\,\{a\} \subseteq T \times S$. We want the projection of $a\,\mathrm{H}_Z'$ to vanish in the interior, i.e.

$$a\,\mathrm{H}_Z'\,\mathbb{1}_{B\times S \cup T \times A} = 0 \tag{5.9}$$

$$a(\mathrm{H}_B \otimes \mathbb{1}_S | \mathbb{1}_T \otimes \mathrm{H}_A^t) = 0\,. \tag{5.10}$$

Inspecting the VV and CC parts of this equation separately, we find that we must have

$$a^t \in \ker\left(\mathrm{H}_B^t\right) \otimes \ker\left(\mathrm{H}_A\right)\ . \tag{5.11}$$

Equivalently, the space we desire is spanned by

$$(\widetilde{\mathrm{G}}_B^t \otimes \mathrm{G}_A)\, \mathrm{H}_Z' \ . \tag{5.12}$$

Part 2: $\mathrm{rs}\left(\mathrm{H}_Z + \mathrm{H}_Z'\right)$

We refer to lemma 24 which states that the span of the stabilizers from within the puncture are independent of those outside the puncture. Therefore the space defined by eq. 5.12 is not in the span of the Z stabilizers. □

Logical X operators

We now discuss the logical X operators associated to a smooth puncture.

The next lemma will help show that certain X operators are not in the span of the stabilizer $\mathrm{H}_X + \mathrm{H}_X'$. These operators will then be used to construct logical X operators. These are comprised of codewords of the classical codes that are complementary to the subgraphs chosen by S and T. In other words, they will involve the terms $\ker\left(\mathrm{H}_{T^c}\right)$ and $\ker\left(\mathrm{H}_{S^c}^t\right)$.

In the proof that follows, we shall make certain claims on these spaces. Note that since the neighborhood the set B is contained in the set T, it implies that the neighborhood of T^c is contained within B^c. Similarly, the neighborhood of S^c is contained within A^c. Therefore when studying $\ker\left(\mathrm{H}_{T^c}\right)$ and $\ker\left(\mathrm{H}_{S^c}^t\right)$, we shall assume that their support is contained in B^c and A^c respectively.

These operators can be classified in terms of the (classical) codespaces associated with H_{T^c} and $\mathrm{H}_{S^c}^t$.

Theorem 27. *Let O_Z be the Z operators defined by*

$$O_Z = \left(\mathrm{H}_M \otimes \mathbb{1}_S | \mathbb{1}_T \otimes \mathrm{H}_N^t\right),$$

and let $\Omega_X = \ker\left(\mathrm{H}_Z + O_Z\right)$ denote the X type operators in its kernel.

The logical X operators are described by

$$\left[\left(\ker\left(\mathrm{H}_{T^c}\right) \otimes \left(\mathbb{F}_2^S / \,\mathrm{rs}\left(\mathrm{H}_A\right)\right) | \boldsymbol{0}_{m^2}\right) \cup \left(\boldsymbol{0}_{n^2} | \left(\mathbb{F}_2^T / \,\mathrm{rs}\left(\mathrm{H}_B^t\right)\right) \otimes \ker\left(\mathrm{H}_{S^c}^t\right)\right)\right] / \Omega_X \ . \tag{5.13}$$

Before proceeding to the proof, we make the following observations and highlight important features of this claim. At first glance, the logical operators appear to break into two

types, the VV type logicals defined by $\ker(\mathrm{H}_{T^c})$ and the CC type operators defined by $\ker\left(\mathrm{H}^t_{S^c}\right)$. Thus the logical X operators are defined by the code spaces that are left over after the portions corresponding to T and S have been carved out.

The set $\mathrm{H}_Z + O_Z$ represents Z stabilizers *outside* the puncture. Vectors in the kernel of $\mathrm{H}_Z + O_Z$ are unaffected by the addition of the puncture, and in that sense represent some invariant space. The space Ω_X thus contains X stabilizers and logicals whose support does not overlap with the puncture. To help this object seem less alien, let us return to the surface code and consider an example.

Consider a smooth puncture defined on a surface code with only smooth boundaries. This could define a logical qubit with the X string running from the boundary of the smooth puncture to one of the boundaries of the lattice. Depending on the arrangement, this logical operator could be supported only on CC qubits or only VV qubits as shown in fig. 5.3. However, these two objects are equivalent up to stabilizer. This equivalence is captured by

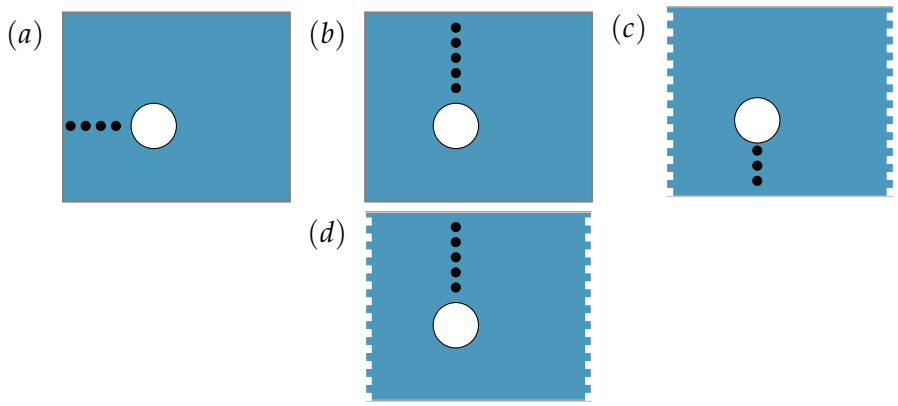


Figure 5.3 Panels (a) and (b) feature a smooth puncture defined on a lattice with only smooth boundaries. The strings of X operators defined only on VV qubits (as in panel (a)) or only on CC qubits (as in panel (b)) are equivalent. Panels (c) and (d) feature a lattice with smooth and rough boundaries, with a smooth punctured carved out from the inside. The logical X shown running from the smooth puncture to the boundary in panels (c) and (d) are equivalent up to an embedded logical X.

Ω_X.

Furthermore, consider a lattice with two smooth and rough boundaries as shown in panels (c) and (d) of fig. 5.3. The two representations of the logical X operator shown running from the smooth puncture to the boundary are equivalent up to an embedded logical X operator, and X stabilizers. This equivalence is also captured by Ω_X.

With these comments, we proceed to the proof of theorem 27.

Proof. We start from first principles. Recall that we defined the matrix $\mathrm{INT} = (\mathbb{1}_B \otimes \mathbb{1}_S | \mathbb{1}_T \otimes \mathbb{1}_A)$. The logicals are defined as

$$\ker\left(\mathrm{H}_Z + \mathrm{H}'_Z\right) / \mathrm{rs}\left(\mathrm{H}_X + \mathrm{H}'_X + \mathrm{INT}\right) .$$

Part 1: $\ker\left(\mathrm{H}_Z + \mathrm{H}'_Z\right)$

We would like to understand the structure of vectors $\alpha \in \ker\left(\mathrm{H}_Z + \mathrm{H}'_Z\right)$. We shall assume that α is not supported on the interior of the puncture $B \times S \cup T \times A$. If α is supported within the interior, it can be removed using the single-qubit operators described by INT. As shown by lemma 22, the embedded logical X operators are unaffected by the puncture.

We shall argue that α ought to have a certain structure using a proxy. Let us define $a \in \mathbb{F}_2^{C \times V}$ as

$$a = \mathrm{H}_Z\, \alpha = \mathrm{H}'_Z\, \alpha \, . \tag{5.14}$$

The only occasion when a is non-trivial is when α is supported on the boundary. If not, α is either a stabilizer or logical belonging to the code that is unaffected by the puncture. We shall mod out by this set, and this will correspond to Ω_X.

The VV and CC portions of eq. 5.14 stipulate that

1. $a \subseteq \mathrm{im}\left(\mathrm{H}_T\, \mathbb{1}_{M \setminus B}\right) \cap \mathrm{im}\left(\mathrm{H}\, \mathbb{1}_{B^c}\right) \otimes \mathbb{F}_2^S$;
2. $a \subseteq \mathbb{F}_2^T \otimes \mathrm{im}\left(\mathrm{H}_S^t\, \mathbb{1}_{N \setminus A}\right) \cap \mathrm{im}\left(\mathrm{H}\, \mathbb{1}_{A^c}\right)$

respectively.

Equivalently, this means that

$$\alpha \in \left(\ker\left(\mathrm{H}_{T^c}\right) \otimes \mathbb{F}_2^S | \mathbf{0}_{m^2}\right) \cup \left(\mathbf{0}_{n^2} | \mathbb{F}_2^T \otimes \ker\left(\mathrm{H}_{S^c}^t\right)\right) . \tag{5.15}$$

Part 2: $\mathrm{rs}\left(\mathrm{H}_X + \mathrm{H}'_X + \mathrm{INT}\right)$

Since the operators we are interested in only lie outside the puncture by assumption, we may ignore INT and need only concern ourselves with $\mathrm{rs}\left(\mathrm{H}_X + \mathrm{H}'_X\right)$. Let $g \in \ker\left(\mathrm{H}_{T^c}\right)$ and $x \in \mathrm{rs}\left(\mathrm{H}_A\right)$ such that $a\, \mathrm{H}_A = x$. Its product $g \otimes x$ is clearly in $\ker\left(\mathrm{H}_{T^c}\right) \otimes \mathrm{rs}\left(\mathrm{H}_A\right)$ and therefore in the kernel of $\mathrm{H}_Z + \mathrm{H}'_Z$. We shall show that this vector lies in the span of the X

stabilizers as well. Indeed, it can be expressed as

$$(g \otimes a)\left(\mathrm{H}_X + \mathrm{H}'_X\right) = g \otimes x\,.$$

In a similar manner, we can show that any vector in $\mathrm{rs}\left(\mathrm{H}_B^t\right) \otimes \ker\left(\mathrm{H}_{S^c}^t\right)$ is in the row span of $\mathrm{H}_X + \mathrm{H}'_X$.

As was already shown in lemma 24, any vector $\alpha \in \mathbb{F}_2^{n^2+m^2}$ that is in the row span of

1. $\left(\ker\left(\mathrm{H}_{T^c}\right) \otimes \mathbb{F}_2^S / \mathrm{rs}\left(\mathrm{H}_A\right) | \mathbf{0}_{m^2}\right)$
2. $\left(\mathbf{0}_{n^2} | \mathbb{F}_2^T / \mathrm{rs}\left(\mathrm{H}_B^t\right) \otimes \ker\left(\mathrm{H}_{S^c}^t\right)\right)$

do not lie in the row span of the stabilizer $\mathrm{H}_X + \mathrm{H}'_X$.

This completes the proof. □

5.2 Wormholes

We have now established how to construct punctures by carving out portions of the hypergraph product code. On the surface code, punctures facilitate CNOT gates on encoded qubits via braiding, but it is limited. This process maps physical (and logical) X operators to X operators and Z operators to Z operators. In other words, it is a CSS-preserving operation. To complete even just the Clifford group, we require operations that can map X operators to Z operators on both the physical and logical levels. In particular, we need ways to perform logical single-qubit Clifford operations. On a 2-dimensional code, twist defects [106, 75, 76, 77] can be used to encode qubits, and also perform single-qubit Clifford gates on these qubits.

Twist defects however rely on symmetries of 2-dimensional codes that do not naturally extend to general LDPC codes. For instance, an error chain on the surface code has two frustrated stabilizers on either end regardless of the length of the chain. LDPC codes however do not possess these properties. For instance, expander codes have the property that the number of frustrated stabilizers grows with the size of the error. For these reasons, we have to look for other ways of generalizing twist defects.

In the previous chapter, we introduced a defect called a wormhole that addresses this issue. Rather than rely on line-like defects, it builds upon and generalizes puncture defects. Since

we already know how to construct punctures on the hypergraph product code, it is natural to extend them to wormholes.

The key idea is to entangle stabilizers along the boundaries of punctures. Doing so yields hybrid stabilizers whose weight does not scale with the size of the puncture. As we shall see, these stabilizers are created by measuring two-qubit Pauli operators. These measurements locally break the CSS nature of the code and serve as a resource to complete the Clifford group.

Let $\mathcal{G} = (V \cup C, E)$ be a bipartite graph corresponding to a classical code $\mathcal{C}$. Consider a hypergraph product of a graph $\mathcal{G}$ with itself. As before, let $S \subseteq V$ and $T \subseteq C$ be connected subsets of variable nodes and check nodes respectively. Furthermore the induced subgraphs do not overlap, i.e. they obey $N \cap T = M \cap S = \varnothing$. These sets must be correctable, i.e., they obey conditions specified in definition 20.

The wormhole is created by entangling the stabilizers along the boundaries of two punctures. This alters the structure of the code along the boundaries, and we must ensure that these enlarged regions remain correctable. Hence, we need to strengthen the notion of correctability to include the neighborhoods that define the punctures.

Definition 28 (Extended correctability)**.** *In addition to obeying definitions 20 and 23, wormholes will also need to obey*

1. $\ker\left(\mathrm{H}_{\overline{S}}\right) = \ker\left(\mathrm{H}^{t}_{\overline{T}}\right) = \varnothing$.
2. $\mathrm{rs}\left(\mathrm{H}_{\overline{T}^c}\right) \cap \mathrm{rs}\left(\mathrm{H}_{tT}\right) = \mathrm{rs}\left(\mathrm{H}_{\overline{S}^c}\right) \cap \mathrm{rs}\left(\mathrm{H}_{\overline{S}}\right) = \varnothing$.
3. $\ker\left(\mathrm{H}_{\overline{T}^c}\right) \cap \ker\left(\mathrm{H}_{tT}\right) = \ker\left(\mathrm{H}_{\overline{S}^c}\right) \cap \ker\left(\mathrm{H}_{\overline{S}}\right) = \varnothing$.

This will be necessary because the stabilizers on the boundary of the puncture will be removed to form hybrid stabilizers. To argue that the logical operators that emerge have certain properties, we shall use the above extended correctability condition. Equivalently these can be thought of as the conditions for a puncture defined using the sets $\overline{S} := M$ and $\overline{T} := N$.

With these constraints established, we can associate a smooth puncture to the product $T \times S$ and a rough puncture to the product $S \times T$. These punctures will be used to construct a wormhole in the following sections.

5.2.1 Measurements and hybrid stabilizers

Within the interior of the punctures, we perform the same measurements as we did to initialize a puncture – single-qubit X measurements within the smooth puncture and single-qubit Z measurements within the rough puncture. We then perform two-qubit measurements along the boundaries of the two punctures. This yields hybrid stabilizers.

For what follows, it will be helpful to use the schematic for the smooth and rough puncture shown in fig. 5.4. Recall that the boundaries of punctures are described as follows:

1. Smooth puncture: $T \times (N \setminus A) \cup (M \setminus B) \times S$.
2. Rough puncture: $(N \setminus A) \times T \cup S \times (M \setminus B)$.

For any VV qubit (u, u') or CC qubit (c, c'), we shall let $P(u, u')$ or $P(c, c')$ denote the single-qubit Pauli operator P on that qubit.

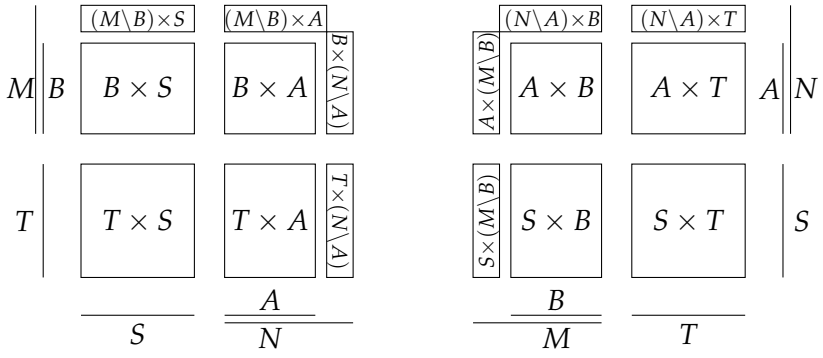


Figure 5.4 Schematic for a wormhole on hypergraph product codes. Smooth puncture on the left and a rough puncture on the right.

The hybrid stabilizers are generated by the following measurements along these boundaries.

1. CC qubits: for every $c \in N \setminus A$, $c' \in T$, measure $X(c, c') \otimes Z(c', c)$; we denote this as

$$(N \setminus A) \times T \leftrightarrow T \times (N \setminus A) \,.$$

2. VV qubits: for every $u \in S$, $u' \in M \setminus B$, measure $X(u, u') \otimes Z(u', u)$; we denote this as

$$S \times (M \setminus B) \leftrightarrow (M \setminus B) \times S \,.$$

These two-qubit measurements do not commute with the stabilizers located on the boundary of the puncture. The next proposition will provide an appropriate choice of hybrid stabilizers that commute with the measurements. Suppose P and Q are some sets of variable and check nodes respectively, sets of the form $P \times Q$ will refer to X stabilizers and $Q \times P$ to Z stabilizers. We shall write $P \times Q \leftrightarrow Q \times P$ to denote the hybrid stabilizer formed by pairing stabilizers in a natural way. In other words, for $p \in P$ and $q \in Q$, we let $P \times Q \leftrightarrow Q \times P$ denote the hybrid stabilizers acting as X on the support of (p, q) and Z on the support of (q, p). As a matter of convention, the operators with X support are always denoted on the left, and those with Z support on the right.

The set of proposed hybrid stabilizers contain stabilizers that are adjacent to the puncture minus those in the interior. In other words, this is the set of stabilizers that live on the boundary of the punctures.

Proposition 29. *Upon performing the measurements listed above, the new hybrid stabilizers are associated with*

$$(M \times N) \setminus (B \times A) \leftrightarrow (N \times M) \setminus (A \times B) \,,$$

where the double-arrow denotes the one-to-one pairing between the two sets as described above. This choice of hybrid stabilizers resolves the commutation relations.

Proof. Since we begin with a puncture, certain stabilizers have already been removed from our code. These correspond to the stabilizers $B \times A$ of X type and $A \times B$ of Z type from the interior of respective punctures. We remove these stabilizers from a larger set corresponding to $M \times N$ and $N \times M$ respectively. With the interior carved out, this leaves only the boundary of the two punctures.

Next, consider two nodes c, v such that $c \in N$ and $v \in M$. The Z stabilizer (c, v) will anticommute with the X measurements if either $c \in N \setminus A$ or $v \in M \setminus B$. By symmetry, any X measurement on the support of the Z stabilizer (c, v) will also act as Z on the support of the X stabilizer (v, c). The two individual stabilizers are frustrated, but this can be resolved by pairing them.

This produces the desired hybrid stabilizers. □

The weight of the hybrid stabilizers is thus independent of the size of the code. For this reason, this construction guarantees that we still have an LDPC code.

Thus when creating a wormhole, we begin as before by carving out certain portions of the code. Then, we perform two-qubit measurements along the boundaries of these punctures. We remove X and Z stabilizers from the code either because they lie within a puncture, or they anti-commute with a two-qubit measurement and are replaced by a hybrid stabilizer.

The code thus has the following X, Z and hybrid (denoted h) stabilizers:

$$\begin{aligned} X: &\quad (V \times C) \setminus [(S \times T) \cup (M \times N)] \\ Z: &\quad (C \times V) \setminus [(T \times S) \cup (N \times M)] \\ h: &\quad (M \times N) \setminus (B \times A) \leftrightarrow (N \times M) \setminus (A \times B) \\ &\quad (N \setminus A) \times T \leftrightarrow T \times (N \setminus A) \\ &\quad S \times (M \setminus B) \leftrightarrow (M \setminus B) \times S \,. \end{aligned}$$

We conclude by reiterating the origin of these objects. Note that there are two punctures that are used to create the wormhole, one smooth and one rough. The set of all X stabilizers corresponds to $V \times C$, but we subtract those stabilizers in these punctures. This corresponds to $S \times T$ from the rough puncture and $M \times N$ from the smooth puncture and the hybrid stabilizers. Similarly, the set of all Z stabilizers corresponds corresponds to $C \times V$, but we subtract the stabilizers $T \times S$ from the smooth puncture and $N \times M$ from the rough puncture and the hybrid stabilizers. The hybrid stabilizers are created using the boundaries of these sets and are stated in proposition 29. The last two lines of hybrid stabilizers correspond to the two-qubit measurements used to generate the wormhole. We remind the reader that in this notation, operators with X support are to the left of the arrow, and those with Z support are to the right of the arrow.

Before presenting the logical operators, we make some remarks. The measurements that we make along the boundary of existing punctures defined by $T \times S$ and $S \times T$ clearly modify the punctures. In effect, the existing stabilizers on the boundary of these stabilizers have also been modified. This modification can also be expressed as follows. Sitting atop the smooth puncture defined by $T \times S$ is a rough puncture defined by $\overline{S} \times \overline{T}$ where $\overline{S} := M$ and $\overline{T} := N$. Likewise the rough puncture $S \times T$ lies beneath another smooth puncture $\overline{T} \times \overline{S}$. Introducing this second set of punctures is merely a matter of convenience, and simplifies the derivation of the logical operators. We shall find that we have two types of logical operators. The first type is associated to the smooth puncture $T \times S$, and the second to the smooth puncture $\overline{T} \times \overline{S}$.

5.2.2 Logical Pauli operators for wormholes

When we create a wormhole from two punctures, there are two ways in which stabilizers are updated. First, there are stabilizers within the puncture that are jettisoned because they anti-commute with the single-qubit measurements. Second, there are stabilizers on the boundary of the puncture that are replaced by hybrid stabilizers. This results in two sets of logical operators for the wormhole as we shall see below. As in the case of punctures, there are two varieties of logical operators: loop-type operators and chain-type operators. These logical operators are inherited from the underlying punctures.

The first set of logical operators can be described as the logical operators corresponding to the punctures $S \times T$ and $T \times S$. Given the symmetry of the construction, there is a one-to-one correspondence between the loop-type logical X operators around the rough puncture $S \times T$ and the loop-type logical Z operators around the smooth puncture $T \times S$.

Lemma 30. *The logical Z operators correspond to loop-type operators around one of the punctures. They come in two sets which are described as follows.*

- ***Type 1:*** *The first set of logical Z operators correspond to the smooth puncture $T \times S$. These operators were defined in theorem 26.*
- ***Type 2:*** *The second set of logical Z operators corresponding to the smooth puncture $N \times M$. These operators can be obtained by using theorem 26 with $\overline{T} := N$ and $\overline{S} := M$.*

Proof. To show that these objects are no longer part of the stabilizer group but commute with the X and Z stabilizers, we point to the proof of theorem 26. The development is similar save for the new stabilizers, the new two-qubit operators that were measured along the boundaries of the two punctures.

Type 1: Note that the measurements surrounding the smooth puncture are of Z type and will therefore commute with the loop-type logical Z operators of type 1. Furthermore, these logical operators are defined only along the boundary of the puncture $T \times S$. A product of the two-qubit stabilizers is necessarily defined on both punctures, as X on the puncture $T \times S$ and Z on the puncture $S \times T$. Therefore this implies that the proposed logical operators of type 1 cannot be in the span of the stabilizers.

Type 2: The logical operators of type 2 are defined on $(\Gamma(N) \setminus S) \times M \cup N \times (\Gamma(M) \setminus T)$. Thus they do not interact with the two-qubit measurements. For the same reason, they cannot be expressed as a product of the two-qubit measurement operators. By definition

28, these objects do not affect the embedded logical operators, and are themselves not in the span of the Z stabilizer.

Logical operators of types 1 and 2 are themselves independent of each other. First, note that they are supported on two disjoint portions of the hypergraph product code. We have already shown in the proof of theorem 26 that under definitions 20, the loop-type logical operators are not products of stabilizers outside the puncture. We have extended these conditions in 28, which in turn guarantee that one set of loop-type operators are independent of the other. □

Before proceeding to the conjugate logical operators, it will be useful to highlight a symmetry of this construction. For logical Z operators, we chose the vector of Z operators around the puncture $T \times S$ for type 1 and $N \times M$ for type 2. Equivalently we could have chosen the vector of X operators around the puncture $S \times T$ for type 1 or $M \times N$ for type 2. The next lemma states that these two choices are equivalent.

Lemma 31. *Every logical loop-type operator for a wormhole has two equivalent representations: an X type loop around one puncture or a Z type loop around the other.*

Proof. We shall deal with each type in turn.

Type 1:

Consider loop-type logical X operators that emerge from the puncture $S \times T$. These logical operators are supported on $(N \setminus A) \times T \cup S \times (M \setminus B)$. Each qubit on this boundary has a unique partner on the other boundary $T \times (N \setminus A) \cup (M \setminus B) \times S$. By symmetry, there is a one-to-one correspondence between the loop-type logical X operators on $S \times T$ and the loop-type logical Z operators on $T \times S$. These can be mapped to one another because of the two-qubit measurements.

Type 2:

Let $\widetilde{\mathrm{G}}_S$, G_T be the matrices whose rows span $\ker\left(\mathrm{H}_S^t\right)$ and $\ker\left(\mathrm{H}_T\right)$ respectively.

Let $g \in \widetilde{\mathrm{G}}_S^t$ and $f \in \mathrm{G}_T^t$ be any two rows of $\widetilde{\mathrm{G}}_S^t$ and G_T^t respectively. By the considerations above and theorem 26, we can define α_Z as a loop-type operator around $T \times S$, where

$$\alpha_Z := (g \otimes f)(\mathrm{H}_N \otimes \mathbb{1}_M | \mathbb{1}_N \otimes \mathrm{H}_M^t) \, .$$

Similarly, α_X can be defined as a loop-type operator around $S \times T$, where

$$\alpha_X := (f \otimes g)(\mathbb{1}_M \otimes \mathrm{H}_N \,|\, \mathrm{H}_M^t \otimes \mathbb{1}_N)$$

is also a logical operator.

To show that these are in the span of the stabilizers, note that the hybrid stabilizers are given by

$$(\mathrm{H}_N \otimes \mathbb{1}_M | \mathbb{1}_N \otimes \mathrm{H}_M^t)_Z + (\mathrm{H}_A \otimes \mathbb{1}_B | \mathbb{1}_A \otimes \mathrm{H}_B^t)_Z \leftrightarrow (\mathbb{1}_M \otimes \mathrm{H}_N \,|\, \mathrm{H}_M^t \otimes \mathbb{1}_N)_X + (\mathbb{1}_B \otimes \mathrm{H}_A \,|\, \mathrm{H}_B^t \otimes \mathbb{1}_A)_X \,.$$

Thus the operator

$$(g \otimes f)(\mathrm{H}_N \otimes \mathbb{1}_M | \mathbb{1}_N \otimes \mathrm{H}_M^t)_Z \leftrightarrow (f \otimes g)(\mathbb{1}_M \otimes \mathrm{H}_N \,|\, \mathrm{H}_M^t \otimes \mathbb{1}_N)_X$$

maps the loop-type logical Z operator to the loop-type logical X operator.

Since this is true for arbitrary f and g, any operator in the space can be mapped between one puncture and the other. □

The logical X operators for the wormhole are constructed from the logical operators of the corresponding punctures. The form of these operators are given in theorem 27.

Lemma 32. *For every loop-type logical Z operator L_Z around $T \times S$ and the unique loop-type logical X operator L_X on $S \times T$ corresponding to L_Z, let the conjugate chain-type operators be Q_X and Q_Z respectively. The product $Q_X Q_Z$ is the conjugate logical operator to the operator L_Z.*

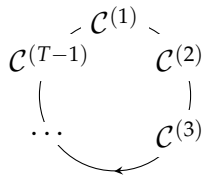
Proof. Following the proof of theorem 27, the logical chain-type operators evidently commute with the X and Z stabilizers and are not spanned by them.

From the symmetry of the construction, any overlap with the two-qubit measurement operators always occurs in pairs if at all. Therefore these operators commute with the two-qubit stabilizers.

Furthermore since the two-qubit measurement operators are only supported on the boundary, it cannot span the logical chain-type operators. Since the chain-type operators anti-commute with the logical loop-type operators, it cannot be expressed as a product of stabilizers alone. □

5.3 Code deformation

Having described how to create defects, we can now proceed to discuss how to use them. Logical transformations will be effected using a technique called code deformation (see for instance [75, 107, 108, 105]). The core idea behind this technique is to perform a sequence of $T-1$ elementary transformations of a code $\mathcal{C} =: \mathcal{C}^{(1)}$, obtaining codes $\mathcal{C}^{(2)}, ..., \mathcal{C}^{(T-1)}, \mathcal{C}^{(T)}$ in the process.



Each elementary transformation is comprised of measurements of Pauli operators. As shown in the schematic, the overall result is to leave the codespace globally unchanged, i.e. $\mathcal{C}^{(1)} = \mathcal{C}^{(T)} = \mathcal{C}$, but the logical operators of the code may, and hopefully will, undergo a non-trivial transformation. Since this transformation maps all Pauli operators to Pauli operators, the resulting operation must be a logical Clifford operation.

We begin by reviewing code deformation to highlight some useful properties. Rather than focus right away on hypergraph product codes, we step back and study code deformation as it applies to general quantum codes. Our intent is to track the transformation of the logical operators. As we shall see below, these steps are organized such that the transformations are 'small' with respect to the logical operators we are interested in. Therefore this process is fault-tolerant.

5.3.1 Non-mixing

For all $t \in \{1, ..., T\}$, the quantum error correcting code $\mathcal{C}^{(t)}$ is defined on n qubits for some fixed n. At step t, $\mathcal{C}^{(t)}$ is the eigenspace of the stabilizer group $\mathcal{S}^{(t)}$.

The logical operators of the code are denoted $\mathcal{L}^{(t)}$. These are the objects that we wish to track as we transform the code. Let $\mathcal{Z} := \mathcal{S}^{(t)} \cap \mathcal{S}^{(t+1)}$ be the operators that are common to both $\mathcal{S}^{(t)}$ and $\mathcal{S}^{(t+1)}$.

The following lemma states that when we transition from $\mathcal{C}^{(t)}$ to $\mathcal{C}^{(t+1)}$, the logical operators that need to be updated are either removed entirely or mapped by multiplying by a stabilizer element.

Lemma 33. *If a logical operator $L \in \mathcal{L}^{(t)}$ anti-commutes with the measurement of $S^{(t+1)} \in \mathcal{S}^{(t+1)} / \mathcal{Z}$, then either*

1. *L is moved to the space of errors; or*
2. *there exists a unique $S' \in \mathcal{S}^{(t)} \setminus \mathcal{Z}$ such that $L \mapsto LS'$.*

Proof. Consider any element $S \in \mathcal{S}^{(t+1)} \setminus \mathcal{Z}$. One of two things can happen to S as we transition from step t to $t+1$:

1. S can commute with all of $\mathcal{S}^{(t)} \setminus \mathcal{Z}$, i.e. $S \in \mathcal{L}^{(t)}$.
2. S can anti-commute with some element $S' \in \mathcal{S}^{(t)} \setminus \mathcal{Z}$. If there are any other elements S_0 that anti-commuted with S, we map $S_0 \to S_0 S'$. We can therefore choose S' as a representative element.

In turn, this leads to two possibilities for the logical operators $L^{(t)}$. Suppose we have an operator $L \in \mathcal{L}^{(t)}$ that anti-commutes with an element $S \in \mathcal{S}^{(t+1)} \setminus \mathcal{Z}$. We then update the logical as follows:

1. If S is an operator in $\mathcal{L}^{(t)}$, then we remove the operator L from the logical operators. It must now be an error.
2. If there exists an element $S' \in \mathcal{S}^{(t)} \setminus \mathcal{Z}$ such that S anti-commutes with S', then $L \mapsto LS'$.

This proves the claim. □

As we just saw, the operators in $\mathcal{L}^{(t)}$ that are in the span of $\mathcal{S}^{(t+1)} \setminus \mathcal{S}^{(t)}$ will be removed from the stabilizer group.

In addition, the sets $\mathcal{S}^{(t)}$ and $\mathcal{L}^{(t)}$ can also exchange operators. The operators in $\mathcal{S}^{(t)} \setminus \mathcal{S}^{(t+1)}$ that commute with all of $\mathcal{S}^{(t+1)}$ will be transformed into logical operators. For instance, this happens when we create a new logical qubit in the surface code by forming a puncture. Recall that stabilizer generators and errors can be partitioned into pairs such that a stabilizer generator S and error E only anti-commute with each other, and commute with all other operators. When a stabilizer S is transformed into a logical operator, the conjugate errors E becomes the unique conjugate error. Note that this construction breaks down if we have an

overcomplete set of generators. For instance this happens when we have a smooth puncture on a lattice with only rough boundaries. The string of X from the smooth boundary of the puncture cannot be terminated on the boundary. Thus creating a smooth puncture on a lattice with only rough boundaries cannot be used to store a qubit (because there are redundant checks).

This analysis proves that logical operators transform linearly as summarized by the following lemma.

Lemma 34. *For $t \in \{1, ..., T\}$, let $\mathcal{L}^{(t)}$ denote the set of logical operators of the code $\mathcal{C}^{(t)}$. Let $\mathbb{L}^{(t)} \in \mathbb{F}_2^{k \times n}$ be the generator matrix for this space and $\mathbb{S}^{(t)} \in \mathbb{F}_2^{(n-k) \times n}$ be the matrix whose rows span the stabilizers. There exists a matrix $Q^{(t)}$ such that we can write the logical operators $\mathbb{L}^{(t)}$ as*

$$\mathbb{L}^{(t+1)} = Q^{(t)} \begin{pmatrix} \mathbb{L}^{(t)} \\ \mathbb{S}^{(t)} \end{pmatrix} \tag{5.16}$$

As we proceed with code deformation, we will encounter problems unique to codes that carry several logical qubits. We define below the notions of non-mixing and small transformations as guidelines for studying such transformations.

First, there may potentially be several ways of updating the logical operators. This is because there is no preferred basis for us to express the logical operators in the intermediary steps. Equivalently, the matrix $Q^{(t)}$ is not unique as there could be several different ways of expressing the logical operators over the course of code deformation. However, the global transformation $Q = Q^{(T)} Q^{(T-1)} \dots Q^{(2)} Q^{(1)}$ generated by the entire sequence of code deformation is unique if we choose the same logical operator basis for $\mathcal{C}^{(1)} = \mathcal{C}^{(T)}$.

For $t \in \{1, ..., T\}$, let $\mathcal{L}^{(t)}$ denote the set of logical operators of the code $\mathcal{C}^{(t)}$. Let

$$\langle L_j^{(t)} \rangle_j := \langle L_g^{(t)} \rangle_g \times \langle L_b^{(t)} \rangle_b$$

be a partition of the set of logical operators $\mathcal{L}^{(t)}$ into good and bad operators. These sets are defined such that the operators in the set g all have weight above some threshold, whereas those in b have weight below this threshold. Furthermore, assume that $\langle L_g^{(t)} \rangle$ contains $k' < k$ independent operators. The set of qubits defined by $\{L_b^{(t)}\}_b$ shall be considered as gauge qubits [109]. As mentioned in the introduction to this chapter, a gauge qubit is a logical qubit in which we choose not to store information. Thus some of the associated logical operators can instead be cast as gauge operators. We wish to avoid the space of gauge qubits interacting with our logical qubits and to this end, define the non-mixing condition.

Definition 35 (Non-mixing). *We say that code deformation is* non-mixing *with respect to the partition if there exists a direct-sum decomposition*

$$Q^{(t)} = Q_g^{(t)} \oplus Q_b^{(t)} ,$$

where matrices $Q_g^{(t)}$, $Q_b^{(t)}$ only act on the spaces $\langle L_g^{(t)} \rangle_g$, $\langle L_b^{(t)} \rangle_b$ respectively. Each elementary step in code deformation is small with respect to $\mathcal{Q}_g^{(t)}$ if $\mathcal{Q}_g^{(t)}$ is rank k' over the good partition.

By guaranteeing that an operation is non-mixing with respect to this partition, we can show that the gauge qubits do not affect the logical qubits. The constraint on the rank will guarantee that none of the good logical operators are mapped to either the stabilizers or gauge operators over the course of code deformation.

The non-mixing condition is important because we cannot guarantee that the number of logical operators will remain a constant over the course of code deformation. In general, hypergraph product codes need not be translation invariant like the toric code; even if we maintain a puncture of a fixed radius, the number of logical operators created by this puncture could change as it moves. If we move a puncture by enlarging it and then shrinking it, this could also change the number of logical operators supported by the puncture. However the two conditions on the high-weight operators regulate their transformation.

To illustrate, we consider encoding logical qubits on the surface in a slightly unusual way. Consider the pair of punctures on the surface code shown in fig. 5.5. This pair shall be

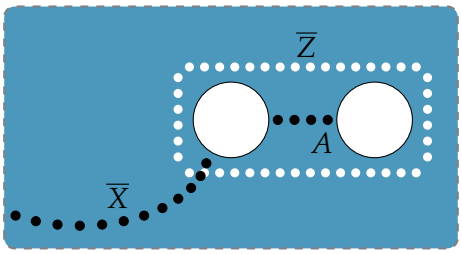


Figure 5.5 A pair of punctures used to encode a single logical qubit on the surface code.

treated as a single entity that encodes a logical qubit. The logical Z operator is the loop encircling the pair, denoted $\overline{Z}$. The logical X operator is a string of Xs running from the boundary of a puncture to the boundary of the lattice, denoted $\overline{X}$. The operator A running between punctures is a low-weight string of Xs and this logical operator is a potential liability. We therefore treat it as a gauge qubit. When encoding information, we only store logical information using the qubit defined by $\overline{Z}$ and $\overline{X}$. So long as we do not braid using A or drag another defect between these two punctures, this troublesome chain will not cause

a problem. In this way the operation will be non-mixing because the operator A will never be entangled with other qubits of interest.

5.3.2 Code deformation on the hypergraph product code

We now return to hypergraph product codes, and in this section shall discuss how to perform Clifford gates with the help of an ancilla. Recall lemma 11 which we restate here for convenience.

Lemma 36. *Let A and B be distinct, non-trivial single-qubit Pauli operators. Let S and T be two Pauli operators, not necessarily distinct. The two-qubit measurements $A_1 S_a$ and $B_1 T_a$, together with all single-qubit Pauli measurements on qubit a are sufficient to generate the single-qubit Clifford group on qubit* 1.

First, we point out that regardless of whether the qubit 1 is an embedded logical qubit, or merely another wormhole, the ancilla qubit(s) shall be encoded in a wormhole. Second, we will require these wormholes to encode Y resource states. For reasons that will become clear shortly, we will find it difficult to perform single-qubit Y measurements on the logical level. If however we are provided ancilla qubits that are prepared in the Y state, we may use these objects catalytically to perform a Y measurement. This is necessary, at least as per lemma 36, to complete the Clifford group.

In addition to these single-qubit gates, we also require entangling gates between multiple logical qubits. The circuit shown in fig. 5.6 shows how this too can be accomplished on the logical level with the help of an ancilla and Pauli measurements. This completes the

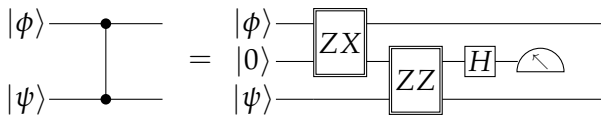


Figure 5.6 A measurement-based circuit to perform controlled-Z. We introduce an ancilla prepared in the $|0\rangle$ state. The double-boxes indicate a non-destructive projective measurement. The labels PQ on these measurements indicate that the projection is performed along the $+1$ and -1 eigenstates of the 2-qubit Pauli operator PQ. Finally, we perform a Hadamard and destructively measure the ancilla qubit in the computational basis.

requirements to perform Clifford gates. In the next subsection, we shall study how exactly to perform these measurements.

5.3.3 Measurements and traceability

The measurements of Pauli operators described above will have to be performed on the logical operator and the ancilla qubit encoded in a wormhole. In addition, we use a regular puncture based qubit to perform the measurement. This puncture shall be referred to as a needle.

We are interested in logical operators of the needle that can be measured fault tolerantly. In turn these operators will be used to measure the logical Pauli operators of a wormhole or even an embedded logical qubit. For instance, suppose we have a smooth puncture that has high weight X and Z operators. The loop-type operator can be measured by shrinking the size of the puncture while simultaneously maintaining the size of the chain-type conjugate logical operator. This would of course make the logical qubit susceptible to logical Z error. However this will not affect the measurement outcome so long as the logical X operator remains high weight. Similarly, the chain-type logical X operator can be measured fault tolerantly by shrinking its size while maintaining the size of the loop-type Z logical operator. Unfortunately, the logical Y operator of a puncture cannot be measured fault tolerantly as this would require that we minimize both the size of the chain-type operator as well as that of the loop-type operator. The logical qubit would then become unprotected, and the measurement outcome error prone. The impossibility to fault-tolerantly measure Y operators will cause some problems as we shall see below.

Let w denote a logical qubit, or sets of logical qubits whose state we wish to measure. This could refer to a set of logicals on the wormhole, or an embedded logical, or some combination thereof. Let P_p refer to a logical operator of the puncture that is fault tolerantly measurable. A logical operator Q_w on system w is said to be *traceable* if there is a unitary operation U implementable by code deformation such that

$$U^\dagger P_p U = Q_w P_p \ .$$

Such operations will be used to measure traceable operators Q_w in order to effect measurements of logical operators in lemma 36. The operator Q_w shall be measured using the standard ancilla-assisted way: we shall prepare the ancilla in an eigenstate of P_p, applying the unitary U and then measure the operator P_p.

If Q_w is either X or Z, then we need to find an operator P_p and an operation U such that

$$U^\dagger P_p U \to Q_w P_p \ .$$

On the other hand, if $Q_w = Y$, then the situation is complicated for reasons we will discuss shortly. In such a case, we shall assume that we are given access to an another system b prepared in a Y state. The aim is to perform the operation

$$U^\dagger P_p U \to Y_w Y_b P_p \ ,$$

which would not have been possible without the system b. If we now use this operation to perform a $Y_w Y_b$ measurement, then we can do so without affecting the state of system b. In this way, the system b serves as a catalyst and can be used for the next Y measurement on w. In the next subsection, we shall discuss how to obtain such a resource states to serve as catalysts.

We remark that this completes the requirements to perform Clifford gates on the system w. We now make some comments about traceability.

We begin by noting that the advantage of the wormhole in this process is that it permits both the X and Z type logical operators associated to a defect to be traced. We provided an example in chapter 3; whether or not a logical operator is traceable on a specific code is a code-dependent question. The other advantage of a wormhole is that it permits the use of a Y resource state whose role is catalytic in the implementation of CSS breaking operations. Without the wormhole, the Y resource states would be consumed by CSS breaking operations and we would therefore require a constant supply of these states.

In the case of the surface code, the new wormhole defects that we have introduced make it possible to trace both the X and Z type logicals associated to a wormhole (see chapter 3). Suppose that P_p above is the logical X operator of a smooth puncture. As it traces the support of a logical $Y = iXZ$ operator, it will encounter the location when the X and Z logical operators cross. As shown in fig. 5.7, the trailing chain-type X operator is mapped to the logical Y operator and this has the effect of breaking the original protocol. So instead of mapping X_p to $Y_w X_p$ as desired, we are mapping it to $Y_w Y_p$. Completing the protocol would require measuring Y_p, but this cannot be done because it is not fault tolerantly measurable.

Not all is lost however; we may find that *products* of logical Y operators are traceable. As an example, this was demonstrated in the case of the surface code with wormhole defects, so the framework we describe is sufficiently rich to enable the complete Clifford group in principle.

We would like to highlight that we are not providing a constructive approach to compiling specific logical operators. Compiling the operation required to trace operators not only depends on the code, but also on the representation of the logical we are interested in per-

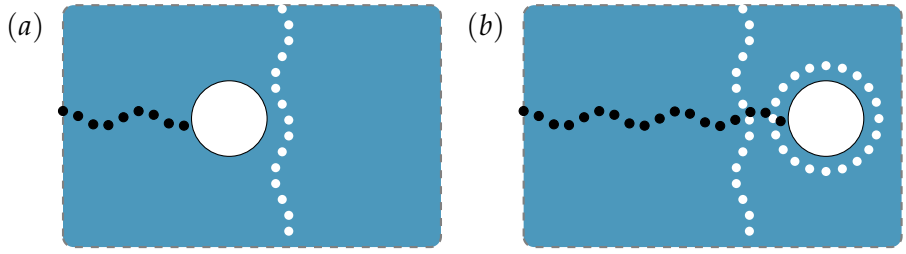


Figure 5.7 A puncture crossing its own path. The puncture first leaves a trail of Zs, and passes through a wormhole. Upon crossing Z, the logical X is mapped to XZ. This operator is no longer guaranteed to be needle-measurable.

forming. The process described merely provides a framework within which to search for such an operation. For instance, given a specific code, we could search for non-trivial Clifford operators that we can perform using brute force. Once a set of Clifford operations has been found, these can be used as a basis to compose Clifford gates of interest using standard compiling tools.

We consider an example, perhaps perverse, to illustrate that simply having a wormhole and a puncture is insufficient to generate all Clifford gates. Suppose we have two copies of the toric code that are disconnected from each other. If we were to initialize a wormhole and a needle on one of the two codes, then clearly this is insufficient to perform all gates fault tolerantly on all logical qubits. This is because there is clearly no way for the needle to move to the second code. This suggests that in general, the ability to perform all gates may be closely tied to graph connectivity. We will return to this idea in section 5.3.5.

5.3.4 Resource states

Clifford gates by themselves only generate a finite group [88]. Furthermore, the techniques that we have discussed above only map Pauli operators to Pauli operators. Therefore they can at best generate logical Clifford operations.

As is well known, it is sufficient to add any gate that is not already in the Clifford group to achieve a universal gate set. One such non-Clifford gate is the T gate, defined as $T = e^{i\pi Z/8}$. We assume that we have access to physical T gates and wish to construct a logical T gate. The technique presented here will mean that this logical T gate is not inherently fault tolerant. In turn the T gate will be noisy and therefore need to be distilled [110].

By its definition, the logical $T = e^{i\pi Z/8}$ gate can be executed on the support of the logical Z operator. Suppose we have a logical qubit encoded in a smooth puncture prepared in

the $|+\rangle$ state. The logical Z operator is a loop-type operator that is supported only on the boundary of the puncture. We can inject the T gate on to the code by performing an explicit circuit on this boundary. One might hope that the stabilizers and logicals on the boundary obey some symmetry such as tri-orthogonality [53]. In that case, the circuit to inject the T gate could be made fault tolerant as we would merely require transversal physical T gates in order to implement the logical T gate. However we do not assume that the puncture obeys these symmetries, and thus the circuit to perform the T is not fault tolerant. Thus we would want to minimize the size of the circuit to minimize the number of faulty locations. To this end, we may shrink the puncture i.e. reduce the size of the boundary that supports the logical qubit. Upon performing the T circuit, we increase the size of the puncture again to make it resistant to logical Z type errors. While doing so of course, the logical qubit is still subject to logical Z errors and is thus subject to dephasing errors. The end result is a noisy version of the T state defined as $|A\rangle = T|+\rangle$.

Once we have several such logical qubits carrying potentially noisy T states, we can perform state distillation on the hypergraph product code. This process was described in chapter 2. State distillation is a technique that uses several noisy T states and produces fewer, but higher fidelity copies of the T state. These higher fidelity copies can then be used in the computation if they are sufficiently reliable. State distillation only requires Clifford gates and Pauli measurements, and these are operations we already have the ingredients to perform.

To perform a logical T on an embedded logical qubit, we can use the T gate and a single-qubit teleportation circuit [45].

In addition to using resource states to inject the T gate, we also require resource states that are prepared in the Y basis as was discussed in the previous section. However since these resource states are used catalytically, they can be prepared once before the beginning of the computation. To prepare these states, we follow a procedure similar to the preparation of a T state. We shall perform the circuit required to prepare a puncture qubit in a logical Y state. We note that the circuit to prepare the Y state will have to be performed on the support of both the X and Z logical operators. To minimize the size of this circuit, we therefore reduce the size of both the loop-type and the chain-type logical operators. This circuit will likely not be fault tolerant; after performing the circuit we will have to increase the size of the loop-type and chain-type logical operators. During this period the logical qubit supported on the puncture may be subject to depolarizing noise.

Once we have prepared several such logical qubits, we will have access to several noisy Y resource states. To purify theme, we will have to perform distillation. Of course, there may

be more optimal ways to prepare these states, but this is sufficient to generate the desired resource states.

5.3.5 Point-like punctures

Before we conclude, we consider a scheme that involves the movement of point-like punctures. This deviates slightly from the framework that we have discussed above, and since the punctures are point-like, the setup is not fault tolerant. However, we feel that it still may help understand movement in these codes.

When discussing the surface code, traceability is relatively simple. We can move a rough puncture around a smooth puncture and thereby perform a non-trivial Clifford operation. Whether or not such an operation is possible is dictated by topology; a rough puncture can trace any closed loop of Z operators. The situation is not so simple in the case of hypergraph product codes.

In this subsection, we discuss when logical operators are traceable using a point-like puncture to serve as the needle. Of course, using a point-like puncture is not fault tolerant as the loop-type logical operators are low-weight and therefore error prone. This discussion will however shed light on when something non-trivial is possible. It also illustrates that we can generalize braiding to graph-theoretic concepts. This could also be of independent interest, for example in condensed matter physics.

We shall show that in the case of point-like punctures, traceability can be cast as walks on a graph. Thus whether or not a logical operator is traceable boils down to verifying whether a certain path on a graph exists. This shows that it may be possible to efficiently verify when an operator is traceable.

Consider a point-like puncture, i.e. one created by removing a single stabilizer generator. For the sake of illustration, this puncture is smooth. Code deformation entails that there exist a series of steps such that the point-like puncture corresponds to $T_j \times S_j$ for $j = 1, ..., N$.

Let $T_j = \{c_j\}, S_j = \{v_j\}$ be singleton sets and let $T_j \times S_j$ be the associated point-like puncture. The logical Z operator α that emerges is then just the support of the Z stabilizer (c_j, v_j) i.e. $\alpha_j := \Gamma(c_j) \times v_j \cup c_j \times \Gamma(v_j)$. Let the conjugate logical operator be β_j. Let us study how these objects transform as we transition from step j to step $j + 1$.

Moving a single step: Suppose we consider moving this puncture by changing $T_j \to T_{j+1} = \{c_{j+1}\}$, where c_j and c_{j+1} share a bit u_j in their common neighborhood as shown in fig. 5.8.

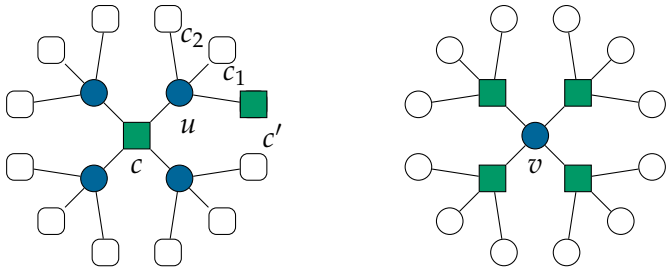


Figure 5.8 A point-like puncture centered at c, v. The check c' is connected to c via the variable node u.

In the growth phase, we would measure X on the qubit (u, v_j). This anti-commutes with all the Z stabilizers that are incident to (u, v_j), i.e. (c_1, v_j), (c_2, v_j) and (c_{j+1}, v_j). The logical Z operator α_j (corresponding to (c_j, v_j)) also anti-commutes with this operation. These objects are updated per the stabilizer update rule. We multiply the operators (c_1, v_j), (c_2, v_j) and α_j by (c_{j+1}, v_j). The stabilizer (c_{j+1}, v_j) is itself removed from the stabilizer group.

In the contraction phase, we measure the stabilizer (c_j, v_j), returning it to the stabilizer group. This anti-commutes with the measurement $X(u, v_j)$. This also anti-commutes with the logical operator β_j. To resolve this anti-commutation relation, we map $\beta_j \to \beta_{j+1} := \beta_j X(u, v)$. We then discard $X(u, v)$ from the stabilizer group.

Thus the logical operator β_j has grown by a single qubit. However, we have not yet returned to the code space. The logical operator β_{j+1} still anti-commutes with the operators (c_1, v_j) and (c_2, v_j). These objects have still not been returned to the stabilizer group. We highlight this matter because it is this issue that does not allow us to fit the movement of a point-like puncture into the framework described in the previous sections.

There are two concerns associated with these frustrated stabilizers:

1. Will they return to the stabilizer group?
2. Will the weight of these objects grow or remain upper bounded by some constant?

First, since the path we traverse corresponds to a logical, it must commute with all the stabilizers. Thus at some point during the course of the point-like puncture moving, it will commute with each stabilizer that it frustrated. Exactly when these operators will be returned to the stabilizer group will depend on the path being traversed.

Secondly, the weight of these stabilizers is always guaranteed to be upper bounded by a constant. In fact, these frustrated operators are always pairwise products of the puncture

at step j and themselves. For instance, consider the example above where we transitioned by one step, from j to $j+1$. In the very next step, suppose the next point to be removed corresponds to the stabilizer (c_{j+2}, v_j). The stabilizer (c_{j+1}, v_j) is returned to the stabilizer group. Therefore the frustrated stabilizers (c_1, v_j) and (c_2, v_j) are now multiplied by (c_{j+2}, v_j). This will continue until we measure another qubit in the support of (c_1, v_j) and (c_2, v_j), at which point they will return to the stabilizer group.

For a point-like puncture, this analysis shows that the logical can grow one qubit at a time. With this insight, we can cast the problem of whether or not a logical is traceable as a graph problem. In this problem, we first consider a logical operator, say Q, that has no Y operators in its support. We use this operator to define a graph $\mathcal{G}_Q$ as follows. The stabilizers that are adjacent to the qubits become the vertices of $\mathcal{G}_Q$ and the qubits in the support of Q become the edges of $\mathcal{G}_Q$. If there exists a sequence of stabilizers that we can puncture, each connected by a single qubit then this path becomes traceable.

In particular, this can be cast as Eulerian cycle [111]. Eulerian cycle is an efficient algorithm that can be stated as follows:

Algorithm 3 Eulerian cycle

1: **Input:** Graph $\mathcal{G} = (V, E)$.
2: **Output:** A path on the graph such that every edge is traversed exactly once if it exists.

It is well known that an Eulerian cycle exists only when the degree of each vertex in the graph is even. Thus given a graph with n vertices, the existence of an Eulerian cycle can be verified efficiently. This example shows that braiding may generalize to a purely graph-theoretic concept. Moreover, there may exist an efficient algorithm to answer when a logical is traceable.

5.4 Chapter summary

In this chapter, we began by understanding how to generalize puncture-based defects on the surface code to general hypergraph product codes. A puncture can be expressed as the graph product of two subgraphs that comprise the hypergraph product code. There are two types of punctures - rough and smooth - that can be classified according to their boundary. Smooth punctures do not break Z stabilizers across the boundary, whereas rough punctures do not break X stabilizers across the boundary. Creating a puncture only removes edges from the graph, and therefore the quantum code remains LDPC.

We then proceeded to discuss the structure of the logical operators of the code. Wormholes help break the CSS-nature of the code. In turn this is used as a resource to map X operators to Z operators during code deformation. We derived the explicit form of the loop-type and chain-type logical operators that emerge when we create a puncture. These were subject to correctability constraints 20. Having introduced puncture-based defects, we went on to discuss how to generalize wormhole defects. These defects build on the idea of puncture-based defects and are created by performing two-qubit measurements along the boundary. It was shown that these defects could also be expressed algebraically, i.e. the stabilizers could be expressed in terms of elements of the graph product code. Wormhole defects result in hybrid stabilizers that emerge as a product of one X and one Z stabilizer. This is independent of the size of the code. Therefore the code remains LDPC even after the creation of a wormhole defect.

Finally, we discussed code deformation on the hypergraph product codes. The objective is to gradually transform the code such that local errors are mapped to local errors. In this way, the process is fault tolerant. We began by noting that defects could carry several logical qubits. To ensure that the resulting process did not introduce logical errors on qubits of interest, we introduced the non-mixing condition. This condition stipulated that logical qubits and gauge qubits on the puncture would not be allowed to mix.

We then discussed the ingredients needed to perform all Clifford gates using wormhole defects. This discussion highlighted the need for resource states to perform Clifford gates. Namely, we need Y type resource states in order to perform CSS-breaking operations. Fortunately, these states are not consumed, but rather are used catalytically. Finally, to perform a universal gate set, we require the T gate and we discussed techniques to perform state injection. Together this completes the requirements for a universal gate set. We wrapped up this chapter by studying point-like punctures. It was shown that we could understand when a logical operator was traceable using an efficient algorithm called Eulerian cycle. This showed that braiding can be cast as a graph-theoretic concept.

Conclusion

It is still unclear what architecture we ought to use in the long term to construct quantum circuits. It is therefore imperative to explore all the possibilities, and understand the trade-offs involved in choosing one quantum error correcting code over another. In this thesis, we have focused on hypergraph product codes, a class of quantum Low-Density Parity-Check (LDPC) codes. Specifically, we provided a general framework to implement Clifford gates on this class of codes.

This framework is based on code deformation, and generalizes defect based encoding from topological codes. Wormholes are created using two-body measurements along the boundaries of punctures. These defects are a unified representation of both puncture and twist type defects. The stabilizers within the interior of the mouths of the wormholes have been removed from the code much like in the case of punctures. By entangling the boundaries of these punctures, we see interesting physics when we consider the movement of anyons on the surface of the lattice. Wormholes are capable of encoding a logical qubit and we can perform all gates in the Clifford group using topologically non-trivial operations.

These defects ensure that the code remains LDPC at each step of code deformation. In contrast to a previous scheme suggested by Gottesman, these operations are defined on a single block. The generalized punctures that we obtain are capable of encoding several logical qubits. We discussed a framework that is rich enough to permit all Clifford gates on encoded qubits. Whether a particular code permits these gates is a code dependent question. Finally, we discussed the movement of point-like charges on these graphs. These defects serve to illustrate that something non-trivial can be accomplished on hypergraph product codes. Furthermore they demonstrate how braiding can be generalized to a purely graph-theoretic notion.

Of course, this is merely a proof of concept, and there is a lot of work to be done in the future. In no particular order, we discuss some issues that need to be addressed; this is by no means a complete list. Using point-like punctures helps us understand that traceability in certain

instances is connected to Eulerian cycle. As punctures become larger however, it is unclear how this algorithm will generalize. Furthermore, we would like efficient algorithms that can verify whether a given representation of a logical is traceable.

On a practical note, it would also be interesting to understand experimental implementations. The difficulty in implementing ideas discussed in this thesis concerns establishing long-range connectivity with high fidelity. This is not an easy task and depends on the architecture in question. However exploring this avenue is imperative as local architectures have severe limitations. We could first consider architectures that can support a limited amount of non-locality. This is possibly simpler than demanding all-to-all connectivity. There have been promising developments in this direction on superconducting qubit architectures [82, 83, 84]. This research could point the way towards implementations of wormhole defects. Next we could consider general quantum LDPC codes which require a high degree of non-locality. For NV centers and ion traps, this appears to be possible at least in theory. Indeed proposals such as that laid out by Nickerson et al. [81] indicate that we may be able to share entanglement between two arbitrary points on a grid of qubits. This proposal builds on a scheme for entangling photons due to Barrett and Kok [112]. The drawback with this approach appears to be the difficulty in photon detection. According to estimates by [14], our efficiency for photon detection needs to improve by several orders of magnitude before such a scheme becomes practical. Addressing these difficulties will help understand whether or not it is practical for us to implement quantum LDPC codes.

We believe that addressing these questions will be intimately connected to the specific code we wish to use. It is of course important to consider specific classes of codes to understand which Clifford gates we can generate using this process. At this juncture however, this seems premature as there is as yet no consensus as to which hypergraph product codes offer the best performance. We would like to consider codes that have a small block size, but can still protect logical qubits up to some desired degree of accuracy. As the theory progresses, this will likely be informed by decoding algorithms, but perhaps the ability to perform Clifford gates could also factor into this choice. The punctures may exhibit symmetries that do not require state distillation to prepare resource states. Since these codes are no longer local, it is unclear what sorts of gates can be implemented transversally, and which cannot.

Addressing these questions will help establish the role of hypergraph product codes in quantum computation.

Bibliography

[1] Ladd, T. D., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C., and O'Brien, J. L. *Nature* **464**(7285), 45 (2010).

[2] Kitaev, A. Y. *Russian Mathematical Surveys* **52**(6), 1191–1249 (1997).

[3] Aharonov, D. and Ben-Or, M. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 176–188. ACM, (1997).

[4] Knill, E., Laflamme, R., and Zurek, W. H. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 454, 365–384. The Royal Society, (1998).

[5] Aliferis, P., Gottesman, D., and Preskill, J. *Quantum Information & Computation* **6**(2), 97–165 (2006).

[6] Gottesman, D. *Quantum Information & Computation* **14**(15-16), 1338–1372 (2014).

[7] Raussendorf, R., Harrington, J., and Goyal, K. *Annals of physics* **321**(9), 2242–2270 (2006).

[8] Raussendorf, R. and Harrington, J. *Physical Review Letters* **98**(19), 190504 (2007).

[9] Raussendorf, R., Harrington, J., and Goyal, K. *New Journal of Physics* **9**(6), 199 (2007).

[10] Kitaev, A. Y. *Annals of Physics* **303**(1), 2–30 (2003).

[11] Bravyi, S. B. and Kitaev, A. Y. *arXiv preprint quant-ph/9811052* (1998).

[12] Fowler, A. G., Mariantoni, M., Martinis, J. M., and Cleland, A. N. *Physical Review A* **86**(3), 032324 (2012).

[13] Barends, R., Kelly, J., Megrant, A., Veitia, A., Sank, D., Jeffrey, E., White, T. C., Mutus, J., Fowler, A. G., Campbell, B., et al. *Nature* **508**(7497), 500 (2014).

[14] O'Gorman, J. and Campbell, E. T. *Physical Review A* **95**(3), 032338 (2017).

[15] Bravyi, S., Poulin, D., and Terhal, B. *Physical Review Letters* **104**(5), 050503 (2010).

[16] Bravyi, S. and König, R. *Physical Review Letters* **110**(17), 170503 (2013).

[17] Campbell, E. T., Terhal, B. M., and Vuillot, C. *Nature* **549**(7671), 172 (2017).

[18] Tillich, J.-P. and Zémor, G. *IEEE Transactions on Information Theory* **60**(2), 1193–1202 (2014).

[19] Krishna, A. and Poulin, D. *arXiv arXiv:1909.07419* (2019).

[20] Krishna, A. and Poulin, D. *arXiv preprint arXiv:1909.07424* (2019).

[21] Hamming, R. W. *The Bell system technical journal* **29**(2), 147–160 (1950).

[22] Shannon, C. E. *Bell system technical journal* **27**(3), 379–423 (1948).

[23] Wikipedia.

[24] Sipser, M. and Spielman, D. A. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, 566–576. IEEE, (1994).

[25] Richardson, T. and Urbanke, R. *Modern coding theory*. Cambridge university press, (2008).

[26] Berlekamp, E., McEliece, R., and Van Tilborg, H. *IEEE Transactions on Information Theory* **24**(3), 384–386 (1978).

[27] Mitchell, D. G., Lentmaier, M., and Costello, D. J. *IEEE Transactions on Information Theory* **61**(9), 4866–4889 (2015).

[28] Arikan, E. *IEEE Transactions on Information Theory* **55**(7), 3051–3073 (2009).

[29] Şaşoğlu, E. et al. *Foundations and Trends® in Communications and Information Theory* **8**(4), 259–381 (2012).

[30] Hui, D., Sandberg, S., Blankenship, Y., Andersson, M., and Grosjean, L. *ieee vehicular technology magazine* **13**(4), 60–69 (2018).

[31] Sudan, M. *Essential coding theory*. (2017).

[32] Spielman, D. A. *IEEE Transactions on Information Theory* **42**(6), 1723–1731 (1996).

[33] Gottesman, D. *arXiv preprint quant-ph/9705052* (1997).

[34] Knill, E. and Laflamme, R. *Physical Review A* **55**(2), 900 (1997).

[35] Calderbank, A. R. and Shor, P. W. *Physical Review A* **54**(2), 1098 (1996).

[36] Steane, A. *Proceedings of the Royal Society A* **452**(1954), 2551–2577 (1996).

[37] Aaronson, S. and Gottesman, D. *Physical Review A* **70**(5), 052328 (2004).

[38] Gottesman, D. *arXiv preprint quant-ph/9807006* .

[39] Ross, N. and Selinger, P. *Quant. Info. and Comp.* **16**, 901 (2016).

[40] Eastin, B. and Knill, E. *Physical Review Letters* **102**(11), 110502 (2009).

[41] Zeng, B., Cross, A., and Chuang, I. L. *IEEE Transactions on Information Theory* **57**(9), 6272–6284 (2011).

[42] Paetznick, A. and Reichardt, B. *Physical Rev. Lett.* **111**, 090505 (2013).

[43] Jochym-O'Connor, T. and Laflamme, R. *Physical Rev. Lett.* **112**, 010505 (2014).

[44] Pastawski, F. and Yoshida, B. *Physical Review A* **91**(1), 012305 (2015).

[45] Zhou, X., Leung, D. W., and Chuang, I. L. *Physical Review A* **62**(5), 052316 (2000).

[46] Gottesman, D. and Chuang, I. L. *Nature* **402**(6760), 390 (1999).

[47] Bravyi, S. and Kitaev, A. *Physical Rev. A* **71**, 022316 (2005).

[48] Bravyi, S. and Haah, J. *Physical Rev. A* **86**, 052329 (2012).

[49] Meier, A. M., Eastin, B., and Knill, E. *Quant. Info. and Comp.* **13**, 0195 (2013).

[50] Haah, J., Hastings, M. B., Poulin, D., and Wecker, D. *Quantum* **1**, 31 (2017).

[51] Haah, J., Hastings, M. B., Poulin, D., and Wecker, D. *Quantum Inf. Comput.* **18**, 114 (2018).

[52] Haah, J. and Hastings, M. *Physical Rev. Lett.* **120**, 050504 (2018).

[53] Bravyi, S. and Haah, J. *Physical Review A* **86**(5), 052329 (2012).

[54] Krishna, A. and Tillich, J.-P. *arXiv preprint arXiv:1811.03112* (2018).

[55] Bardet, M., Dragoi, V., Otmani, A., and Tillich, J.-P. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, 230–234. IEEE, (2016).

[56] Hastings, M. B. and Haah, J. *Physical Review Letters* **120**(5), 050504 (2018).

[57] Krishna, A. and Tillich, J.-P. *Physical Review Letters* **123**(7), 070507 (2019).

[58] Randriambololona, H. In *Algorithmic arithmetic, geometry, and coding theory*, volume 637, 3–78. AMS (2015).

[59] Cui, S. X., Gottesman, D., and Krishna, A. *Physical Review A* **95**(1), 012329 (2017).

[60] Anderson, J. T., Duclos-Cianci, G., and Poulin, D. *Physical Rev. Lett.* **113**, 080501 (2014).

[61] Bombín, H. *New Journal of Physics* **18**(4), 043038 apr (2016).

[62] Bravyi, S., Suchara, M., and Vargo, A. *Physical Review A* **90**(3), 032326 (2014).

[63] Tuckett, D. K., Bartlett, S. D., and Flammia, S. T. *Physical Review Letters* **120**(5), 050505 (2018).

[64] Darmawan, A. S. and Poulin, D. *Physical Review E* **97**(5), 051302 (2018).

[65] Darmawan, A. S. and Poulin, D. *Physical Review Letters* **119**(4), 040502 (2017).

[66] Duclos-Cianci, G. and Poulin, D. *Physical Review Letters* **104**(5), 050504 (2010).

[67] Fowler, A. G., Whiteside, A. C., and Hollenberg, L. C. *Physical Review Letters* **108**(18), 180501 (2012).

[68] Fowler, A. G. *arXiv preprint arXiv:1310.0863* (2013).

[69] Delfosse, N. and Nickerson, N. H. *arXiv preprint arXiv:1709.06218* (2017).

[70] Horsman, C., Fowler, A. G., Devitt, S., and Van Meter, R. *New Journal of Physics* **14**(12), 123011 (2012).

[71] Landahl, A. J. and Ryan-Anderson, C. *arXiv preprint arXiv:1407.5103* (2014).

[72] Litinski, D. *arXiv preprint arXiv:1808.02892* (2018).

[73] Litinski, D. and Oppen, F. v. *Quantum* **2** (2018).

[74] Bombin, H. and Martin-Delgado, M. A. *Physical Review Letters* **97**(18), 180501 (2006).

[75] Bombin, H. *New Journal of Physics* **13**(4), 043005 (2011).

[76] Yoder, T. J. and Kim, I. H. *Quantum* **1**, 2 (2017).

[77] Brown, B. J., Laubscher, K., Kesselring, M. S., and Wootton, J. R. *Physical Review X* **7**(2), 021029 (2017).

[78] Zheng, H., Dua, A., and Jiang, L. *Physical Review B* **92**(24), 245139 (2015).

[79] Kovalev, A. A. and Pryadko, L. P. *Physical Review A* **87**(2), 020304 (2013).

[80] Bravyi, S. and Terhal, B. *New Journal of Physics* **11**(4), 043029 (2009).

[81] Nickerson, N. H., Li, Y., and Benjamin, S. C. *Nature communications* **4**, 1756 (2013).

[82] Campagne-Ibarcq, P., Zalys-Geller, E., Narla, A., Shankar, S., Reinhold, P., Burkhart, L., Axline, C., Pfaff, W., Frunzio, L., Schoelkopf, R., et al. *Physical Review Letters* **120**(20), 200501 (2018).

[83] Axline, C. J., Burkhart, L. D., Pfaff, W., Zhang, M., Chou, K., Campagne-Ibarcq, P., Reinhold, P., Frunzio, L., Girvin, S., Jiang, L., et al. *Nature Physics* , 1 (2018).

[84] Kurpiers, P., Magnard, P., Walter, T., Royer, B., Pechal, M., Heinsoo, J., Salathé, Y., Akin, A., Storz, S., Besse, J.-C., et al. *Nature* **558**(7709), 264 (2018).

[85] Kovalev, A. A. and Pryadko, L. P. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, 348–352. IEEE, (2012).

[86] Bravyi, S. and Hastings, M. B. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 273–282. ACM, (2014).

[87] Jochym-O'Connor, T. *Quantum* **3**, Art–No (2019).

[88] Nielsen, M. A. and Chuang, I. (2002).

[89] Gallagher, R. (1963).

[90] MacKay, D. J., Mitchison, G., and McFadden, P. L. *IEEE Transactions on Information Theory* **50**(10), 2315–2330 (2004).

[91] MacKay, D. J., Shokrollahi, A., Stegle, O., and Mitchison, G. *http://www. inference. phy. cam. ac. uk/mackay/QECC. html* (2007).

[92] Couvreur, A., Delfosse, N., and Zémor, G. *IEEE Transactions on Information Theory* **59**(9), 6087–6098 (2013).

[93] Breuckmann, N. P. and Terhal, B. M. *IEEE Transactions on Information Theory* **62**(6), 3731–3744 (2016).

[94] Breuckmann, N. P., Vuillot, C., Campbell, E., Krishna, A., and Terhal, B. M. *Quantum Science and Technology* **2**(3) (2017).

[95] Conrad, J., Chamberland, C., Breuckmann, N. P., and Terhal, B. M. *Phil. Trans. R. Soc. A* **376**(2123), 20170323 (2018).

[96] Londe, V. and Leverrier, A. *arXiv preprint arXiv:1712.08578* (2017).

[97] Hastings, M. B. *arXiv preprint arXiv:1312.2546* (2013).

[98] Delfosse, N. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, 917–921. IEEE, (2013).

[99] Poulin, D. and Chung, Y. *arXiv preprint arXiv:0801.1241* (2008).

[100] Leverrier, A., Tillich, J.-P., and Zémor, G. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, 810–824. IEEE, (2015).

[101] Fawzi, O., Grospellier, A., and Leverrier, A. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 521–534. ACM, (2018).

[102] Grospellier, A. and Krishna, A. *arXiv preprint arXiv:1810.03681* (2018).

[103] Fawzi, O., Grospellier, A., and Leverrier, A. *arXiv preprint arXiv:1808.03821* (2018).

[104] Panteleev, P. and Kalachev, G. *arXiv preprint arXiv:1904.02703* (2019).

[105] Vuillot, C., Lao, L., Criger, B., Almudever, C. G., Bertels, K., and Terhal, B. *New Journal of Physics* (2019).

[106] Bombin, H. *Physical Review Letters* **105**(3), 030403 (2010).

[107] Anderson, J. T., Duclos-Cianci, G., and Poulin, D. *Physical Review Letters* **113**(8), 080501 (2014).

[108] Bombin, H. and Martin-Delgado, M. A. *Journal of Physics A: Mathematical and Theoretical* **42**(9), 095302 (2009).

[109] Poulin, D. *Physical Review Letters* **95**(23), 230504 (2005).

[110] Bravyi, S. and Kitaev, A. *Physical Review A* **71**(2), 022316 (2005).

[111] Moore, C. and Mertens, S. *The nature of computation*. OUP Oxford, (2011).

[112] Barrett, S. D. and Kok, P. *Physical Review A* **71**(6), 060310 (2005).