Error correction for quantum computing at scale

Sam J. Griffiths

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Physics and Astronomy

UCL

April 2025

I, Samuel Jacob Griffiths, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Acknowledgements

Firstly, I give enormous thanks to my supervisor, Prof. Dan Browne, whose expert knowledge, steady guidance and reassuring mentorship has been truly invaluable for this work and my time at UCL throughout. I also acknowledge the Engineering and Physical Sciences Research Council (EPSRC) and Riverlane (particularly, James Cruise and Neil Gillespie), who made this work possible.

I thank my colleagues in Prof. Browne's research group and the wider UCL community for fostering a formative academic experience. In particular, I thank Dr Asmae Benhemou for her contributions to our collaborative work and her ever-present counsel. I also thank my newfound colleagues at OQC for their support during the final stages of this work.

I thank my family and friends for their irreplaceable ability to reinvigorate my motivation, focus and sanity. Finally, I thank Eleanor, for supporting me through thick and thin, for raising me when low, and for believing when I doubted.

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

1. **1. For a research manuscript that has already been published** (if not yet published, please skip to section 2)**:**

   (a) **What is the title of the manuscript?** Union-find quantum decoding without union-find

   (b) **Please include a link to or doi for the work:** 10.1103/PhysRevResearch.6.013154

   (c) **Where was the work published?** Physical Review Research

   (d) **Who published the work?** APS

   (e) **When was the work published?** 9th February 2024

   (f) **List the manuscript's authors in the order they appear on the publication:** Sam J. Griffiths, Dan E. Browne

   (g) **Was the work peer reviewed?** Yes

   (h) **Have you retained the copyright?** Some author rights retained as per the APS Transfer of Copyright Agreement

   (i) **Was an earlier form of the manuscript uploaded to a preprint server (e.g. medRxiv)? If 'Yes', please give a link or doi** Yes: 10.48550/arXiv.2306.09767

   If 'No', please seek permission from the relevant publisher and check the box next to the below statement:

   ☐ *I acknowledge permission of the publisher named under 1d to include in this thesis portions of the publication named as included in 1c.*

2. **For a research manuscript prepared for publication but that has not yet been published** (if already published, please skip to section 3)**:**

   (a) **What is the current title of the manuscript?**

   (b) **Has the manuscript been uploaded to a preprint server e.g. medRxiv?**
   If 'Yes', please please give a link or doi:

   (c) **Where is the work intended to be published?**

   (d) **List the manuscript's authors in the intended authorship order:**

   (e) **Stage of publication:**

3. **For multi-authored work, please give a statement of contribution covering all authors** (if single-author, please skip to section 4)**:** The work was my own, with guidance and ideas from my supervisor Dan Browne

4. **In which chapter(s) of your thesis can this material be found?** These results are primarily presented and discussed in Chapter 3

e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/senior author unless this is not appropriate, e.g. if the paper was a single-author work)**:**

Candidate:
Date: 9th April 2025

Supervisor/Senior Author signature (where appropriate)**:**
Date: 19th April 2025

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

1. **1. For a research manuscript that has already been published** (if not yet published, please skip to section 2)**:**

   (a) **What is the title of the manuscript?**

   (b) **Please include a link to or doi for the work:**

   (c) **Where was the work published?**

   (d) **Who published the work?**

   (e) **When was the work published?**

   (f) **List the manuscript's authors in the order they appear on the publication:**

   (g) **Was the work peer reviewed?**

   (h) **Have you retained the copyright?**

   (i) **Was an earlier form of the manuscript uploaded to a preprint server (e.g. medRxiv)? If 'Yes', please give a link or doi**

   If 'No', please seek permission from the relevant publisher and check the box next to the below statement:

   ☐ *I acknowledge permission of the publisher named under 1d to include in this thesis portions of the publication named as included in 1c.*

2. **For a research manuscript prepared for publication but that has not yet been published** (if already published, please skip to section 3)**:**

(a) **What is the current title of the manuscript?** Online Gaussian elimination for quantum LDPC decoding

(b) **Has the manuscript been uploaded to a preprint server e.g. medRxiv?**

If 'Yes', please please give a link or doi:

Yes: 10.48550/arXiv.2504.05080

(c) **Where is the work intended to be published?**

(d) **List the manuscript's authors in the intended authorship order:** Sam J. Griffiths, Asmae Benhemou, Dan E. Browne

(e) **Stage of publication:** Prepublished on arXiv

3. **For multi-authored work, please give a statement of contribution covering all authors** (if single-author, please skip to section 4)**:** The techniques and results presented were my own work, with help for quantum code constructions used in simulations from Asmae Benhemou, and with guidance and ideas from our supervisor Dan Browne

4. **In which chapter(s) of your thesis can this material be found?** These results are primarily presented and discussed in Chapter 4

e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/senior author unless this is not appropriate, e.g. if the paper was a single-author work)**:**

Candidate:

Date: 9th April 2025

Supervisor/Senior Author signature (where appropriate)**:**

Date: 19th April 2025

Abstract

Quantum error correction (QEC) is considered essential for the development of scalable, fault-tolerant quantum computers in the medium- to long-term. By encoding quantum information across many physical qubits, carefully-designed measurements can be taken to gain limited knowledge as to errors which may have occurred in the system, known as the error syndrome, without causing quantum decoherence. In surface codes, qubits are arranged in a two-dimensional topological space, such as a torus or plane. In quantum low-density parity-check (qLDPC) codes, this is generalised to effectively arbitrary arrangements of qubits and syndrome checks.

This work studies decoders: the algorithms responsible for inferring error-correcting decisions from syndromes. In particular, the union–find decoder is studied in terms of its performance at scale. A simulation of a union–find decoding architecture is used to identify computational bottlenecks and propose improvements. In literature, it is assumed that the decoder runs in time scaling quadratically in the number of qubits under a naive implementation, but near-linearly when including two well-known optimisations. A key result is that, under independent and phenomenological noise models on surface codes, this complexity is strictly linear regardless of these optimisations. A supporting analytical argument is presented using percolation theory.

Generalisation of the decoding problem to qLDPC codes is also studied, with approaches broadly relying on Gaussian elimination to find appropriate solutions. A strategy is proposed which uses a novel online variant of the Gaussian elimination algorithm to solve this linear system incrementally on growing local clusters, with accompanying complexity analysis and empirical data demonstrating a reduction in runtime. An investigation into the use of metachecks is also presented, inspired by single-shot decoding, with implications for how such qLDPC decoders could be further improved.

Impact Statement

In a world increasingly dominated by digital information, computation and communication, few emergent technologies bring such stark potential and bold promises as the quantum computer. By finely leveraging the nuances of quantum theory on physical systems, such devices can – in theory – solve a wide range of problems deemed intractable on classical computers, in terms of the time required to do so. Perhaps the most well-known example is Shor's algorithm for factorising integers on a quantum computer, which provides a superpolynomial speedup over what is currently the best-known classical algorithm for the same task [1]. Shor's publication of this result in 1994 triggered widespread attention to the field; for example, many contemporary cryptography schemes like RSA relied on the assumption that factorising integers is computationally intractable at scale, sparking a scramble to develop 'post-quantum' alternatives [2].

Beyond cryptography, the emergence of quantum computing is widely considered to hold significant potential across fields with extremely high socioeconomic impact such as drug discovery [3], artificial intelligence [4], climate modelling [5], materials science including battery design [6], and more. This had led to increasingly intense economic strategy focus; indeed, a recent report by Oxford Economics modelled that adoption of quantum technologies could see UK economic productivity increase by 7% by 2045 [7]. Correspondingly, UK governments have outlined strategies for quantum adoption [8] and, most recently, announced an additional £121 million investment in quantum technologies [9].

Before achieving such lofty goals, however, many serious challenges in the field have yet to be resolved. By their very nature, quantum systems are extremely volatile, susceptible to environmental noise and difficult to control accurately [10]. It does not suffice to rely on developments in the physical engineering of quantum systems: realistically, just how error correction has enabled deployment of reliable classical systems, quantum error correction (QEC) is widely considered essential for the feasibility of fault-tolerant quantum computers at useful scale.

This project presents a review of contemporary techniques in QEC and investigates ways of improving decoders, an essential algorithmic component of QEC pipelines which forms a key bottleneck in the classical control hardware stack required for practical implementations. Some techniques are proposed and analysed which can improve the overhead of these algorithms by reducing their runtime or memory consumption.

Inside academia, the results contained in this work provide additional context and theoretical framework which can be used for developing decoders and reasoning with the behaviour of such algorithms on a wide range of error-correcting codes. The proposed improvements to existing decoders can also directly aid both simulation and practical implementation of these QEC schemes. Results from this work have been published or prepublished [11, 12] and presented at the APS March Meeting 2023 in Las Vegas [13]. At the time of writing, these results have been cited by a number of subsequent works [14–21] and most notably used directly in the development of other novel decoders [22, 23].

Outside academia, this work contributes directly to efforts in deploying fault-tolerant quantum computing at scale, hopefully helping to enable significant socioeconomic advances in the aforementioned fields, in line with UK scientific and economic strategy.

Contents

Chapter 1

Introduction

Quantum computing is a wide-ranging and increasingly popular field studying the theory and implementation of quantum computers and related quantum technology [10]. Quantum computers are devices which leverage the distinct nonclassical effects of quantum mechanics on physical systems to implement new models of computation which can, in theory, fundamentally outperform classical computers. For example, let us take what is currently the most popular model of quantum computing: the quantum bit, or *qubit*. This is directly analogous to the classical bit, a variable which can only take one of two states: one or zero, i.e. on or off. A qubit, similarly, is a two-level quantum system, which means it can exist not just in one of these two states, but as a *superposition*: a linear combination of these basis states forming a probability density function, which only 'collapses' into one of these states when measured. The superposition of states gives rise to distinct interference effects, a property known as *coherence*, which separates this domain from merely probabilistic computing. Alongside *entanglement* – the ability of qubits to exhibit strong correlations which cannot be explained classically – qubits provide a fundamentally more powerful model of computation and communication than can exist under classical mechanics.

Perhaps the most well-known example of this is Shor's algorithm for factorising integers [1]. By delegating the task of discrete Fourier transform to a specialised quantum algorithm, Shor's algorithm provides a superpolynomial improvement in time complexity over what is currently the best-known classical algorithm for factorisation, the general number field sieve [10]. This result brought serious ramifications on contemporary techniques in cryptography and spawned the field of post-quantum cryptography [2]. Other examples of quantum algorithms which provide variously significant speedups over classical alternatives are Grover's search algorithm for function inversion [24], hybrid quantum–classical optimisation algorithms such as QAOA [25] and VQE [26], and quantum phase estimation for general eigenstate calculation [27].

In theory, any two-level quantum system can be used as a qubit. In practice, the most popular physical implementations currently include superconducting qubits [28], trapped-ion qubits [29], photonic qubits [30] and neutral-atom qubits [31]. Regardless of the choice of implementation, a defining characteristic of these quantum systems is that, by their very nature, they are extremely susceptible to environmental noise and difficult to control accurately [10]. For this reason, quantum error correction (QEC) is considered essential for the development of scalable, fault-tolerant quantum computers in the medium to long term.

As in classical error correction, a key concept in QEC is the introduction of redundancy: using a greater number of physical qubits to encode the information of fewer logical qubits, in order to ease the detection and correction of errors of physical errors. A wide range of approaches in this vein have been proposed. Kitaev's *surface codes* are one of the most well-studied schemes, in which qubits are embedded on a regular lattice in a two-dimensional topological space, such as a torus or plane [32]. More recently, *quantum low-*

density parity-check (qLDPC) codes have also seen increased attention, as a generalised family which includes surface codes, yet also includes more arbitrary schemes which can more readily approach theoretically-optimum performance [33, 34].

A key feature across these codes is that carefully-chosen quantum measurements can indicate the presence of stochastic errors – information known as the *error syndrome* – without disturbing the precious-yet-fragile quantum logical state used for computation [10]. As part of this scheme, algorithms which take observed syndromes and decode them into suitable corrections are required. These decoders are classical in nature and form a critical component of the control stack for quantum computers at scale. Many approaches to decoding have been studied and proposed, ranging from statistical inference [35] through to graph theory (e.g. matching) [36], machine learning [37] and beyond. Whilst minimising the overall failure rate, these schemes must also be as efficient as possible in order to avoid becoming an unacceptable bottleneck in the stack. A further compounding factor is noise in the syndrome measurements themselves, which demands particular attention to tackle and further reduces the degree of fault tolerance obtainable [36, 38].

The study and development of QEC, including decoders, is thus of utmost importance to quantum computing as a field. Experimental work continues to yield promising results and justify the approaches discussed, such as a recent surface code demonstration with 101 qubits by Google [39]. Accordingly, this project studies the state-of-the-art of modern QEC, with attention to both surface and qLDPC codes, and investigates the impact of decoder algorithms and ways in which they can be improved in order to enable fault tolerance at scale.

In Chapter 2, a review of contemporary theory and techniques in QEC is presented, ranging from foundational linear coding theory through to recent advances in efficient decoding of qLDPC codes.

In Chapter 3, I present a study of the popular *union–find* decoder for surface codes [40] and propose a variety of potential improvements to the algorithm and its practical implementation. In particular, I present analytical and empirical arguments that the algorithm has lower time complexity than previously assumed, even when simplifying implementation by omitting certain optimisations which are identified as redundant.

In Chapter 4, I present a study of generalisations of the decoding problem to qLDPC codes. Gaussian elimination, an old and well-known algorithm for solving systems of linear equations, generally forms a distinct bottleneck in decoding qLDPC codes; possibilities for both improving and avoiding Gaussian elimination are discussed. In particular, I propose an online variant of the Gaussian elimination algorithm, which can reduce the runtime of decoders which perform elimination multiple times on strictly-growing subproblems, such as the generalised union–find decoder [41]. An *online algorithm* here means an algorithm which can commence work even before the whole problem input is known by iteratively building upon a solution [42]. Both a formal complexity analysis and empirical data are presented to support this proposal. Finally, I present the use of *metachecks* generated from linear coding theory as a method of avoiding Gaussian elimination which may hold some promise, with further thought required to implementation on specific code families.

Chapter 2

Review of quantum error correction

2.1 Information theory

2.1.1 Repetition codes

In classical computing, a bit may take one of two values: 0 and 1. The simplest classical noise model is the *binary symmetric channel*, in which a bit is flipped with probability p and left unchanged with probability $1 - p$. To protect against bit-flip errors, a fundamental approach is to employ redundancy, copying the information such that discrepancies are more likely to be detected and, hopefully, corrected. In this way, logical bits are encoded into a subspace of a greater number of physical bits, known as the *codespace* [43].

Take a simple repetition code of three bits, i.e. the $(n, 1)$-Hamming code where $n = 3$ [44]. A logical 0 is encoded as three physical 0 bits and, likewise, a logical 1 as three physical 1 bits. The codespace is therefore composed of two words: 000 and 111. If any other word is received – for example, 001 or

110 – then a bit-flip error must have occurred on at least one of the physical bits. The only situation in which an error is undetectable is if an error occurs on every physical bit, thus flipping the logical bit.

One method of correcting the error – that is, returning the received word back into the codespace such that it matches the intended word, rather than the unintended one – is as simple as taking a majority vote. For example, 001 would be decoded to a logical 0, 101 to a logical 1 etc. This could be facilitated by performing two parity checks: one comparing the first and second bits for inequality, and another, likewise, for the second and third bits. The outcomes of these checks form the *error syndrome*: any positive result indicates the presence of an error, but also the location of the positive result(s) indicates which bit needs to be flipped in order to return the word to the codespace.

This majority-vote decoder on the $n = 3$ code succeeds if only one error occurs (or, trivially, no errors) but fails if two or three occur. Therefore, for a physical error rate p, the logical error rate is

$$p_L = p^3 + 3p^2(1-p) \tag{2.1}$$

which is less than p as long as $p < 1/2$. It is worth noting that if $p > 1/2$, the majority-vote decision can simply be inverted. Therefore, $p = 1/2$ is the worst-case scenario which maximises the logical error rate, which is intuitive as $p = 1/2$ is the state with maximum information entropy.

The number of errors which occur follows a binomial distribution. In general, for n physical bits, the scheme fails if the number of errors k is greater than $\lfloor n/2 \rfloor$. Therefore, the logical error rate takes the form of an

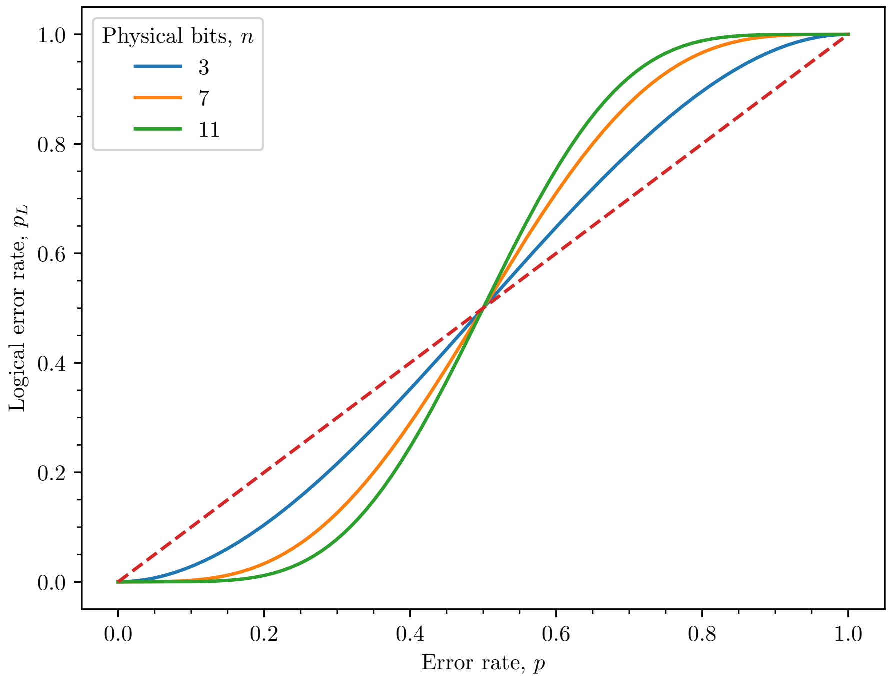


Figure 2.1: Logical error rate p_L of the classical repetition code as in equation 2.2, showing a threshold at $p = 1/2$ below which p_L is suppressed as n increases. The dashed line marks the case with no error correction scheme, $p_L = p$. Above the threshold, the fault-tolerant regime is left but note that – in this simple classical case – the result may simply be inverted.

inverted cumulative binomial distribution:

$$p_L = 1 - \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{k} p^k (1-p)^{n-k} \ . \tag{2.2}$$

Figure 2.1 shows this logical error rate of the classical repetition code for increasing values of n, illustrating that the logical error rate can be suppressed arbitrarily close to zero by increasing n as long as the physical error rate is below the threshold of $p = 1/2$. The ability to achieve an arbitrarily low logical error rate given the appropriate resources is known as the *noisy-channel coding theorem*, a result with significant and promising implications and one which is commonly regarded as critical to the establishment of information theory as a discipline [45, 46].

The repetition code is amongst the simplest and most inefficient error correction codes in practice, but it suffices here as a classical illustration of the use of redundancy and the emergence of a threshold to achieve fault-tolerance.

2.1.2 Linear codes

In general, an $[n, k]$ code encodes k logical bits using n bits in total. We denote the *message* of logical bits as the bitstring $\mathbf{u} = u_1 u_2 \dots u_k$, which is encoded into a *codeword* $\mathbf{x} = x_1 x_2 \dots x_n$, where generally $n > k$ to introduce redundancy [43]. In a linear code, codewords are formed by linear combinations of message bits, such that every linear combination of codewords is itself a codeword.

A *generator matrix* G defines this encoding, such that $\mathbf{u}G = \mathbf{x}$. In the example of a $[5, 1]$ repetition code, the possible messages are $\mathbf{u} \in \{0, 1\}$ and codewords are $\mathbf{x} \in \{00000, 11111\}$. Therefore, the generator matrix of the code is

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{2.3}$$

such that

$$0 \cdot G = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.4}$$

$$1 \cdot G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix} \ . \tag{2.5}$$

To detect errors, one or more parity checks (i.e. sum modulo 2) are performed on the bits. This is represented by a parity-check matrix H, where each row corresponds to a parity check on a subset of the bits.

For example, the parity-check matrix representing an $n =$ repetition code is

$$H = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \ , \tag{2.6}$$

which is equivalent to testing if pairs of neighbouring bits differ in value. The outcomes of these parity checks can be represented as a syndrome vector σ, sometimes more clearly denoted $\sigma(\mathbf{x})$, such that

$$H\mathbf{x}^\top = \sigma \ . \tag{2.7}$$

$\mathbf{x}$ is in the codespace if and only if $\sigma(\mathbf{x}) = H\mathbf{x}^\top = \mathbf{0}$. If, in this example,

$$H \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \ , \tag{2.8}$$

then we know $\mathbf{x}$ has left the codespace because $\sigma(\mathbf{x}) \neq \mathbf{0}$.

If a code C comprises a set of codewords, then the *dual code* $C^\perp$ comprises the set of codewords which are orthogonal to every codeword in C, i.e.

$$C^{\perp} = \{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{c} \rangle = 0 \quad \forall \mathbf{c} \in C\} \ . \tag{2.9}$$

The parity-check matrix of C is equal to the generator matrix of $C^{\perp}$ and vice versa. Through this reasoning, it can be shown that if a parity-check matrix is an $(n-k) \times n$ matrix of the form

$$H = [A \mid I_{n-k}] \ , \tag{2.10}$$

then the corresponding generator matrix is the $k \times n$ matrix

$$G = [I_k \mid -A^{\top}] \ , \tag{2.11}$$

noting that $-A = A$ in $\mathbb{Z}_2$.

The *Hamming distance* between two bitstrings is the number of positions in which their value differs. The *distance* d of a code is the minimum Hamming distance between any two codewords. For a linear code, this is equivalent to the minimum weight of a nonzero codeword.

In the previous example of a $[5, 1]$ repetition code, by inspection, $d = 5$. Therefore, a minimum of 5 individual errors are needed to transform one codeword into another, i.e. constitute an undetectable error overall. We denote this example a $[5, 1, 5]$ code.

In summary, given an intended codeword $\mathbf{x}$ subject to an error vector $\mathbf{e}$, the error-correction problem is equivalent to solving the equation $H(\mathbf{x}+\mathbf{e})^{\top} = \sigma$; that is, decoding a syndrome σ into the most likely error explaining it, returning to the codespace in a way which matches the intended message $\mathbf{u}$ with as high probability as possible.

2.1.3 Quantum information

The fundamental concepts of error correction transfer elegantly to the domain of quantum computing, despite some compounding factors which serve to complicate the problem. A quantum bit, or *qubit*, exists as a superposition (that is, a linear combination) of the two basis states $|0\rangle$ and $|1\rangle$, forming a two-level quantum system [10]. These basis states, known as the *computational basis*, are defined as

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.12}$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} , \tag{2.13}$$

such that a qubit $|\psi\rangle$ takes the form

$$|\psi\rangle = a|0\rangle + b|1\rangle \tag{2.14}$$

$$= \begin{pmatrix} a \\ b \end{pmatrix} , \tag{2.15}$$

where a and b are complex numbers such that $|a|^2$ and $|b|^2$ give the probability of measuring the states $|0\rangle$ and $|1\rangle$, respectively. The state is thus constrained by a normalisation condition:

$$|a|^2 + |b|^2 = \langle\psi|\psi\rangle = 1 \ . \tag{2.16}$$

This qubit state is a vector in two-dimensional Hilbert space, which can be represented as a three-dimensional Bloch sphere, as shown in Figure 2.2. By convention, the $|0\rangle$ and $|1\rangle$ basis states are set as positive and negative along the $\mathbf{z}$ axis, respectively. All pure states (i.e. 'true', non-noisy superpositions)

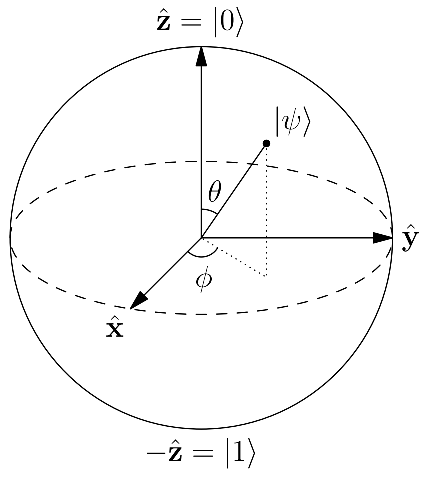


Figure 2.2: Bloch sphere representation of a qubit. Image from [47].

lie on the surface of the sphere; for example, the equal superposition state

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.17}$$

points directly in the positive $\mathbf{x}$ axis. Any operation which takes a qubit from one pure state to another can thus be interpreted as a three-dimensional rotation.

Therefore, the first issue encountered is that unlike the classical case, wherein only bit-flip errors need to be considered, qubits can be subjected to a continuous spectrum of errors. Take the following operator definitions:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.18}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.19}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} . \tag{2.20}$$

These, together with the identity matrix, form the set of Pauli matrices $P = \{I, X, Y, Z\}$, also sometimes denoted $P = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$. The Pauli X

operator (an $\mathbf{x}$-axis rotation by π) is equivalent to a bit flip:

$$X|0\rangle = |1\rangle \tag{2.21}$$

$$X|1\rangle = |0\rangle \ . \tag{2.22}$$

However, other errors have no classical equivalent. For example, the Pauli Z operator (a $\mathbf{z}$-axis rotation by π) performs a phase flip:

$$Z|0\rangle = |0\rangle \tag{2.23}$$

$$Z|1\rangle = -|1\rangle \tag{2.24}$$

which leaves the $|0\rangle$ state unchanged and introduces a relative phase of -1 to the $|1\rangle$ state.

Even if just considering the correction of only bit-flip errors, translating directly from the classical repetition code, another issue which immediately arises is *decoherence*: the act of measuring the quantum state causes it to collapse, thus destroying the quantum information we wish to hold. Simply attempting to copy the state and measure the copy will not suffice, as a copy of the state will remain entangled with the original, with copying into a separable (i.e. non-entangled) state proven impossible universally by the *no-cloning theorem* [48].

2.1.4 Quantum error correction (QEC)

Similar to the classical case, a [[3, 1]] repetition code[1] using three physical qubits can be defined:

$$|0_L\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle = |000\rangle \tag{2.25}$$

$$|1_L\rangle = |1\rangle \otimes |1\rangle \otimes |1\rangle = |111\rangle \ , \tag{2.26}$$

such that any logical state takes the form

$$|\psi_L\rangle = |\psi\psi\psi\rangle \tag{2.27}$$

$$= a|000\rangle + b|111\rangle \ , \tag{2.28}$$

such as

$$|+_L\rangle = |{+}{+}{+}\rangle \tag{2.29}$$

$$= \frac{|000\rangle + |111\rangle}{\sqrt{2}} \ . \tag{2.30}$$

Rather than considering the problem as one of measuring the three physical qubits to take a majority vote, two parity checks can be taken – as in the classical case – to compare the equality of the first and second qubits and, likewise, the second and third. Measuring the observable ZZI (that is, $Z \otimes Z \otimes I$) yields a result of $+1$ if the first and second qubits are equal in the computational basis and -1 if they are not. The observable IZZ achieves the same for the second and third qubits.

If the syndrome measurements are both $+1$, then the qubits must be in the form $a|000\rangle + b|111\rangle$, so no bit-flip errors have occurred. If the ZZI measurement is -1, the qubits must be in the form $a|100\rangle + b|011\rangle$, so a

[1]Note the use of double brackets to distinguish a quantum code from a classical code.

majority vote would involve flipping the first qubit. By only measuring correlations between physical qubits, no information is gained about the logical qubit state a and b, so the original quantum information is preserved. Therefore, this code can protect against single-qubit bit-flip errors. The analysis of this scheme for now remains the same as in the classical case shown in Figure 2.1, with a threshold of $p = 1/2$ above which the outcome may simply be inverted.

Single-qubit phase-flip errors can similarly be protected against by the same approach, instead encoding the logical qubit as

$$|0_L\rangle = |{+}{+}{+}\rangle \tag{2.31}$$

$$|1_L\rangle = |{-}{-}{-}\rangle \ , \tag{2.32}$$

or, equivalently, instead obtaining an error syndrome by measuring the observables XXI and IXX. The *Shor code* combines these approaches by nesting the encodings, i.e. encoding the logical qubit using nine physical qubits:

$$\begin{aligned} |0_L\rangle &= |{+}{+}{+}_L\rangle && (2.33)\\ &= \left(\frac{|000\rangle + |111\rangle}{\sqrt{2}}\right)\left(\frac{|000\rangle + |111\rangle}{\sqrt{2}}\right)\left(\frac{|000\rangle + |111\rangle}{\sqrt{2}}\right) && (2.34)\\ &= \frac{(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)}{2\sqrt{2}} \ , && (2.35) \end{aligned}$$

which can therefore protect against both bit-flip and phase-flip errors affecting a single physical qubit [49]. This is more powerful than it may initially seem: first, note that any single-qubit Pauli error E (as a three-dimensional rotation on the Bloch sphere) may be decomposed in the form

$$E = e_0 I + e_1 X + e_2 Z + e_3 Y \ . \tag{2.36}$$

By using the identity

$$ZX = -XZ = iY \ , \tag{2.37}$$

the operator E can instead be decomposed in the form

$$E = e_0 I + e_1 X + e_2 Z + e_3 XZ \ . \tag{2.38}$$

Therefore, correcting both bit-flip and phase-flip errors on a qubit is equivalent to correcting *any arbitrary error* on a qubit. Additionally, X and Z parity checks are strictly independent in many popular codes[2]; for this reason, many QEC schemes need only discuss the correction of either X or Z errors, as it can be assumed that the process may be conjugated and repeated for the other, thus correcting any arbitrary error.

2.1.5 Stabilisers and generators

An operator A is said to *stabilise* a state $|\psi\rangle$ if it leaves the state unchanged:

$$A|\psi\rangle = |\psi\rangle \ . \tag{2.39}$$

In other words, $|\psi\rangle$ is an eigenstate of A with eigenvalue $+1$.

Single-qubit bit-flip detection using the encoding for $|\psi_L\rangle$ given in equation 2.28 is performed with any two of the three operators ZZI, ZIZ and IZZ. These operators all stabilise $|\psi_L\rangle$: they are physical error configurations which do not actually alter the logical state. Another (trivial) stabiliser of this state is the identity operator III. Together, they form a stabiliser group S:

$$S = \{III, ZZI, ZIZ, IZZ\} \ . \tag{2.40}$$

[2] *Calderbank–Shor–Steane (CSS) codes* are a broad group of codes with this condition, to which the surface codes, colour codes and 3D toric code discussed below all belong [50].

Any two operators A and B *commute* if $AB = BA$, which can be quantified by the commutator

$$[A, B] = AB - BA = 0 \ . \tag{2.41}$$

A and B instead *anticommute* if $AB = -BA$. Note that all of the elements in S commute with one another, thus it is an abelian group.

V_S is the subspace of vectors which are stabilised by all of the operators in S. For example, the subspace stabilised by ZZI is formed by all linear combinations of the spanning set $\{|000\rangle, |001\rangle, |110\rangle, |111\rangle\}$. In other words, ZZI stabilises any state where the first and second qubits are the same. The subspaces stabilised by ZIZ and IZZ are similarly formed by the spanning sets $\{|000\rangle, |010\rangle, |101\rangle, |111\rangle\}$ and $\{|000\rangle, |100\rangle, |011\rangle, |111\rangle\}$, respectively.

Trivially, the subspace stabilised by III is the set of all states. Therefore, V_S is the subspace formed by the intersection of all four spanning sets, which is merely $\{|000\rangle, |111\rangle\}$. This is the *computational subspace* as already defined in equation 2.28. If any of the operators in S do not stabilise a state $|\psi_L\rangle$, then clearly $|\psi_L\rangle \notin V_S$, that is, some error has taken the logical state outside of the computational subspace.

In general, if E_j is a unitary[3] error affecting the state $|\psi_L\rangle$ with a stabiliser S_j, the resultant state is $E_j|\psi_L\rangle$. Measuring S_j results in $+1$ if E_j commutes with S_j (i.e. E_j is also a stabiliser):

$$\begin{aligned}
\langle S_j\rangle &= \langle\psi_L|E_j^\dagger S_j E_j|\psi_L\rangle && (2.42)\\
&= \langle\psi_L|E_j^\dagger E_j S_j|\psi_L\rangle && (2.43)\\
&= \langle\psi_L|S_j|\psi_L\rangle && (2.44)\\
&= \langle\psi_L|\psi_L\rangle && (2.45)\\
&= 1 \ , && (2.46)
\end{aligned}$$

[3]A is unitary if $A^\dagger = A^{-1}$, where $A^\dagger$ is the conjugate transpose.

but results in -1 if E_j anticommutes with S_j:

$$\begin{aligned}
\langle S_j \rangle &= \langle \psi_L | E_j^\dagger S_j E_j | \psi_L \rangle && (2.47)\\
&= \langle \psi_L | (-E_j^\dagger E_j S_j) | \psi_L \rangle && (2.48)\\
&= \langle \psi_L | (-S_j) | \psi_L \rangle && (2.49)\\
&= -\langle \psi_L | \psi_L \rangle && (2.50)\\
&= -1\ . && (2.51)
\end{aligned}$$

Stabiliser measurements can therefore be used as an error syndrome, with the error being detected if it anticommutes with at least one stabiliser of the logical state.

The stabiliser group S can be represented more compactly by a *generating set*, which is a set of operations from which all elements in S can be formed via multiplication. For example, a possible set of generators for S here is $\langle ZZI, IZZ \rangle$ because the other elements of S may be formed as

$$\begin{aligned}
ZIZ &= (ZZI)(IZZ) && (2.52)\\
III &= (ZZI)^2 = (IZZ)^2\ . && (2.53)
\end{aligned}$$

Note that these two generators correspond to the syndrome measurements defined in Chapter 2.1.4. The above statement may be strengthened such that an error is detected if it anticommutes with at least one stabiliser generator.

In general, an $[[n, k]]$ stabiliser code has $m = n - k$ stabiliser generators, yielding 2^{n-k} stabilisers in S. Any error inflicted on n qubits belongs to the

n-qubit Pauli group, generated as

$$G_1 = \{\pm A, \pm iA \quad \forall A \in P\} = \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\} \tag{2.54}$$

$$G_n = \{\pm A, \pm iA \mid A = A_1 \otimes \cdots \otimes A_n \quad \forall (A_1, \ldots, A_n) \in P^n\} \tag{2.55}$$

where P^n is the n-ary Cartesian power[4] of the set of Pauli matrices. The number of elements in the n-qubit Pauli group is thus 4^{n+1}.

The set of all undetectable errors is therefore the set of errors which commute with every stabiliser in S. This is the *centraliser* Z of S in G_n, defined as

$$Z_{G_n}(S) = \{A \in G_n \mid AB = BA \quad \forall B \in S\}\ , \tag{2.56}$$

and so the set of logical operators is the set of undetectable errors which are not themselves stabilisers, i.e. $Z_{G_n}(S) \setminus S$.

2.2 Surface codes

2.2.1 Toric code

In the previous section, we saw how the stabiliser formalism can be used as a powerful and broad framework to begin constructing quantum error-correcting codes. One of the most popular contemporary categories of stabiliser code is *topological codes*, in which qubits and stabiliser measurements lie on the edges, vertices or faces of a tessellation on a D-dimensional space [36, 38, 51]. The most well-known of these are the *surface codes*, which are codes on 2-dimensional grids based upon Kitaev's *toric code* [32].

In the toric code, as the name suggests, qubits are arranged as a lattice

[4](the set of all n-tuples of elements in a set)

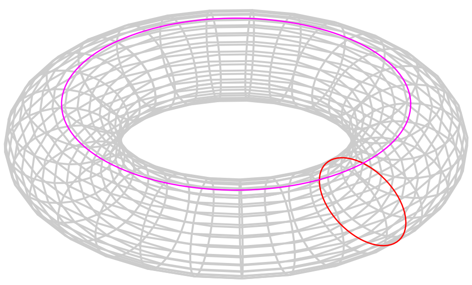


Figure 2.3: Torus demonstrating two degrees of freedom as nontrivial cycles. Image generated by code adapted from [52].

on the surface of a torus. The topology of the torus leads to periodic boundary conditions in both dimensions of the 2D surface. Cycles on the surface can be organised into equivalence classes, of which the torus has four. The first class contains all trivial cycles, that is, those which form a boundary and can be contracted to a single point. The other three classes contain the nontrivial cycles in the two degrees of freedom, that is, those which are noncontractible, as shown in Figure 2.3, and their combination. This topology may be exploited to construct a stabiliser code.

Qubits are positioned on the edges of the lattice such that, in a lattice of length L, there are $2L^2$ qubits. A *plaquette* (or *face*) operator is defined as the combination of four Z operators on neighbouring qubits, forming a trivial cycle on the primal lattice (Figure 2.4). Similarly, a *star* (or *vertex*) operator is the combination of four X operators forming a trivial cycle on the dual lattice [53].[5] The 'dual lattice' here means the lattice formed by perpendicular bisection of all edges in the original, or 'primal', lattice [54]. Note that the assignment of Z and X to the two types of operator is arbitrary and largely by convention. Formally, the X star and Z plaquette operators

[5]Hereinafter, the terms 'plaquette' and 'face' will be used interchangeably, but 'star' will be preferred over 'vertex' to reduce ambiguity when discussing graph topology.

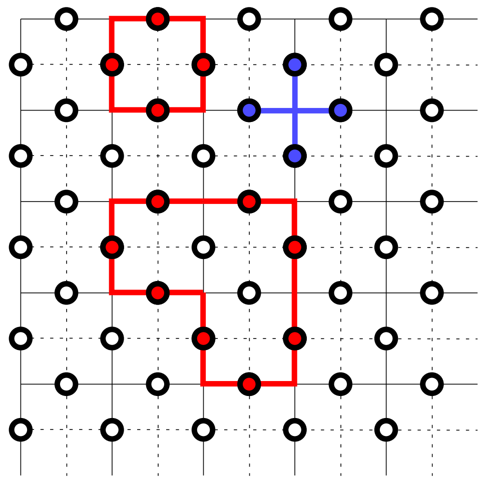


Figure 2.4: Toric code with $L = 5$. Physical qubits are circles on the edges of the lattice. The dashed lines show the dual lattice. Plaquettes of Z operators in red and stars of X operators in blue are stabilisers, as they form trivial cycles on the primal and dual lattices, respectively.

can be defined using group homomorphisms as

$$S_v^X = \prod_{e \in \partial v} X_e \ , \ S_f^Z = \prod_{e \in \partial f} Z_e \ , \tag{2.57}$$

where $e \in \partial f$ is shorthand for $e \in \partial_2 f$ (i.e. the set of edges on the boundary of a face f) and $e \in \partial v$ is shorthand for $e \mid v \in \partial_1 e$ (i.e. the set of edges with which a vertex v is associated) [55].

The L^2 plaquettes and L^2 stars are stabiliser generators – commuting with themselves and each other – so any product of them are also stabilisers, forming any trivial cycle on the lattice. Note that an individual plaquette is identical to the product of all other plaquettes, such that there are in fact L^2-1 independent plaquettes, with the same holding true of stars. Therefore, there are $m = 2(L^2 - 1)$ independent stabiliser generators. With $n = 2L^2$

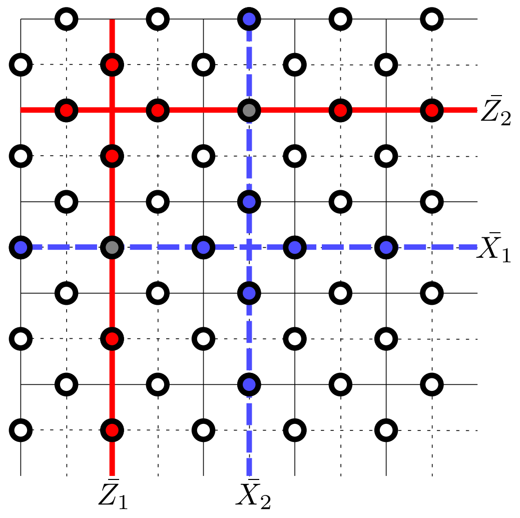


Figure 2.5: Logical operators Z and X for two logical qubits are nontrivial cycles on the primal and dual lattices, respectively.

physical qubits, the number of logical encoded qubits is

$$k = n - m = 2\ . \tag{2.58}$$

Logical Z operators are defined as nontrivial cycles on the primal lattice and logical X operators as nontrivial cycles on the dual lattice, as shown in Figure 2.5. This gives the toric code a distance d exactly equal to L, making it a $[[2L^2, 2, L]]$ code. These operators commute with the stabilisers but are not themselves stabilisers, as they cannot be formed by multiplication of the generators. Therefore, they act nontrivially on the logical qubits within the codespace. The operators are defined such that $\bar{Z}_1$ and $\bar{Z}_2$ commute, as do $\bar{X}_1$ and $\bar{X}_2$, but $\bar{Z}_1$ and $\bar{X}_1$ anticommute, as do $\bar{Z}_2$ and $\bar{X}_2$.

The above arises from the observation that two X/Z operators on the lattice commute if the number of physical qubits they both act on is even. We have seen that error detection can be performed by measuring the stabiliser generators to obtain an error syndrome. Here, measuring the X stars

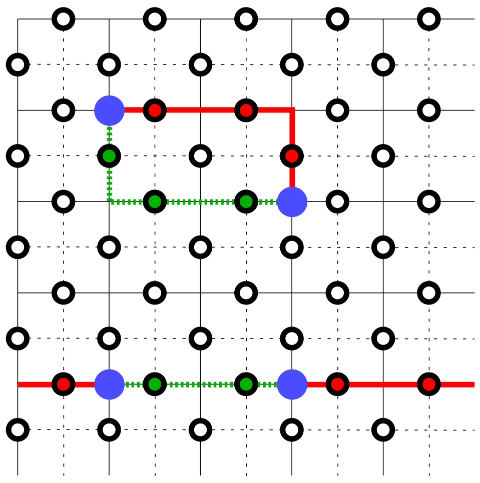


Figure 2.6: A string of errors is detected by a -1 syndrome measurement at each end (blue circles). The top error is transformed into a stabiliser by applying a correction (green) in the same equivalence class, but the bottom error is erroneously transformed into a logical operator by applying a correction in a different class.

gives $+1$ if an even number of Z operators are present at a star and -1 if an odd number are present. A single error on an edge therefore has a syndrome of -1 measurements at its two neighbouring vertices. These ‘excited’ parity-check measurements constituting the syndrome are sometimes referred to as *defects*; for consistency, this terminology will be used hereinafter. Neighbouring edge errors will form a string, with a defect at both ends, as shown in Figure 2.6.

Error correction will involve forming and applying a correction operator using the observed syndrome. It is not necessary to deduce the exact error responsible for the syndrome, but rather any operator consistent with the syndrome in the same topological equivalence class, which will transform the error into a stabiliser. If the correction operator is in a different equivalence class, however, the correction fails, transforming the error into a logical operator.

2.2.2 Planar code

It is impractical to consider physically arranging qubits in a nontrivial topology such as a torus, especially when considering desired interactions between logical qubits. It is more convenient to consider qubits arranged on a more trivial plane – thankfully, the periodic boundary conditions of the toric code (whilst mathematically elegant) are not mandatory[6], leading to *planar codes* [36, 38].

In the previous section, only data qubits were discussed. In practice, the lattice would also contain ancilla qubits, used to perform the four-way syndrome measurements. We can represent this more practical view of the lattice by alternating data qubits with both X and Z ancilla qubits, as shown in Figure 2.7.

The periodic boundary conditions of the toric code are replaced by hard boundaries of X ancillae in one dimension and Z ancillae in the other. There are now $d(d-1)$ independent plaquettes, and the same number of independent stars. Therefore, there are $m = 2d(d-1)$ independent stabiliser generators. As there are $n = d^2 + (d-1)^2$ data qubits, the number of logical encoded qubits is

$$k = n - m = 1 \ , \tag{2.59}$$

thus the planar code encodes only one logical qubit as opposed to the toric code's two, making it a $[[L^2 + (L-1)^2, 1, L]]$ code. The logical operator $\bar{X}$ is defined as a string of X operators connecting the X boundaries, and similarly for Z.

As mentioned, correction schemes on suitable codes can tackle X and

[6]This is because checks constituting the periodic boundaries are not linearly-independent rows in the check matrix H (see Chapters 2.3.2 and 4.3).

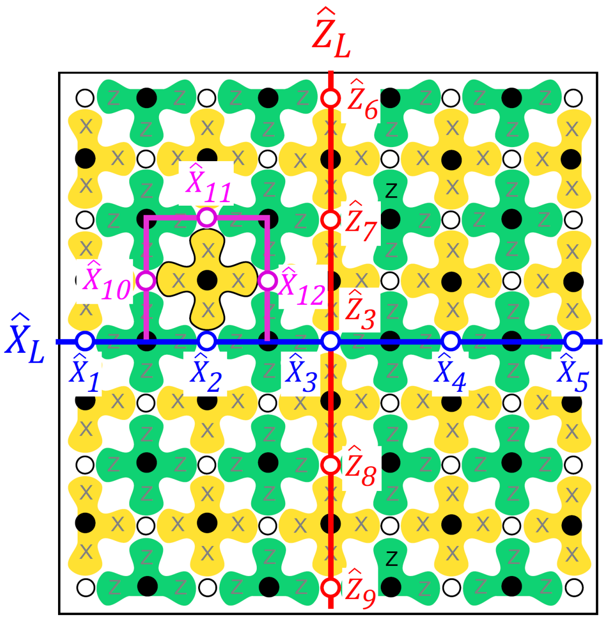


Figure 2.7: Planar code ($d = 5$) with alternating data (white) and X and Z ancilla (black) qubits. Image from [38].

Z errors independently, so we need only describe schemes for one of these two regimes. Considering just the Z ancillae, for example, to detect and correct X errors, the lattice will now appear uneven, with open (a.k.a. rough) boundaries at the left and right and closed (a.k.a. smooth) boundaries at the top and bottom (in terms of Figure 2.7, at least – recall that the configuration of X and Z ancillae is ultimately arbitrary). An X logical operator is one which connects the open boundaries, meaning a Z logical operator connects the closed boundaries or, equivalently, the open boundaries on the dual lattice [56]. Unlike the toric code, an odd number of defects can now appear (in the case of an error occurring on the open boundary). This must be addressed by decoding schemes when moving from toric to planar codes.

2.3 Measurement error

2.3.1 Difference syndrome

The schemes so far intend to tackle stochastic Pauli errors occurring in data qubits; the most straightforward approach to modelling this is an independent model, with an error inflicted on each qubit with probability p. Unfortunately, these schemes are derailed by errors in the syndrome measurements themselves. Consider instead a *phenomenological* error model, in which syndrome measurements are flipped with probability q. A decoder becomes far more likely to fail due to defects appearing and disappearing in a manner no longer representative of the Pauli data errors [36, 38].

To combat measurement errors, syndrome measurements can be per-

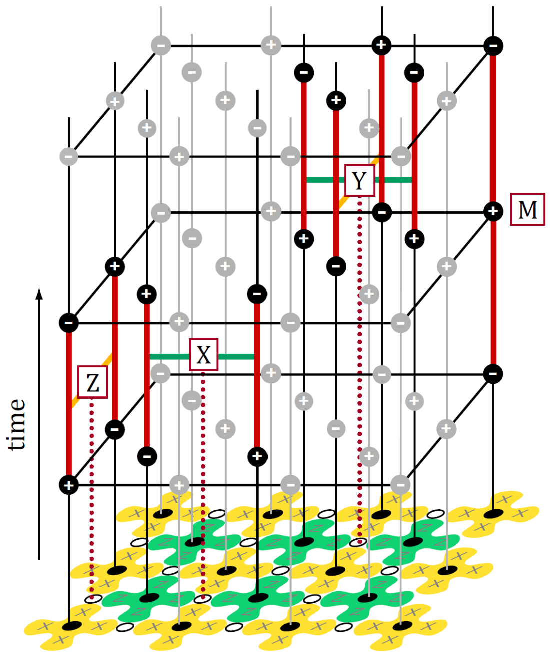


Figure 2.8: Planar code extended to 3D by an additional time dimension. Changes in syndrome measurements are represented as timelike errors in the vertical axis. Image from [38].

formed repeatedly over time, extending the lattice to three dimensions by an additional time axis (Figure 2.8). Changes in a syndrome measurement from one measurement round to the next are represented as 'timelike' errors on the vertical edges between ancillae. The overall syndrome can be converted to a difference syndrome, such that a defect at time t now indicates a change from its state at time $t-1$ [57].

The various approaches for decoding 2D surface codes which will be discussed below scale naturally to incorporate an additional dimension in the decoding graph, making the inclusion of measurement errors surprisingly elegant to tackle in many cases. The 3D correction operator calculated by these decoders will therefore include both horizontal (spacelike) corrections and vertical (timelike) corrections. Of course, only the spacelike corrections hold physical meaning after the final round of measurements, so a 2D correction operator is inferred from this result by simply extracting the spacelike corrections from across the time slices. A step-by-step illustration of this approach via computational simulation can be seen in Figure 3.1.

2.3.2 Single-shot codes

In the difference-syndrome approach discussed above, repeated rounds of measurements are taken and the entire syndrome history is used by the decoder. In contrast, for suitable codes, *single-shot* decoding uses only the current noisy syndrome round at each decoding stage [58]. One approach to determining suitability is *confinement.* In short, a code is *linearly confined* if the weights of syndromes increase no less than linearly with the weights of errors which cause them.

More generally, first define the *reduced weight* $|x|^{\text{red}}$ of an error x as the minimum weight of all errors which share the same syndrome as x. A code is (t,f)-confined if, for all x with $|x|^{\text{red}} < t$,

$$f(|\sigma(x)|) \geq |x|^{\text{red}} , \tag{2.60}$$

where f is some increasing function with $f(0) = 0$. Linear confinement is sufficient for a code to be single-shot under stochastic noise, and more general confinement can show codes to be single-shot under adversarial noise (errors chosen as if by an intentional adversary) [59]. These properties ensure that data and measurement errors remain bounded over repeated rounds and will not become uncorrectable.

The 2D toric code is *not* confined – a string of errors will yield two defects no matter the length of the string – and thus it is not an example of a single-shot code [60].

Generally, there are two approaches to decoding a single-shot code. Firstly, single-stage decoding applies a decoder to the noisy syndrome and relies on the single-shot property to bound the accumulation of residual error. In contrast, two-stage decoding first attempts to correct a noisy syndrome before then proceeding to decode the error. This instead relies on redundancies of parity checks (i.e. linear dependencies between rows of H) to yield *metachecks*. In other words, the syndrome itself is treated as a codespace,

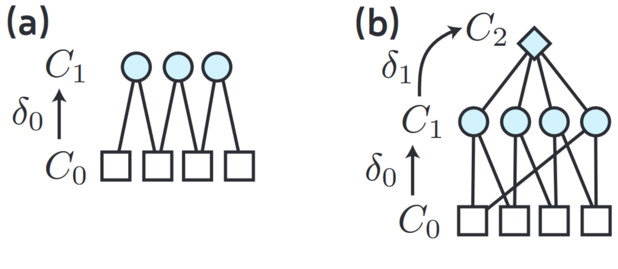


Figure 2.9: (a) the 4-bit classical repetition code; (b) the 4-bit classical repetition code with an additional check and corresponding metachecks. C_0 are bits, C_1 are checks and C_2 are metachecks. Image from [61].

where invalid syndromes might themselves be detected with parity checks [59]. Figure 2.9 shows Tanner graphs illustrating a simple example of this layout (for an introduction to Tanner graphs, see Chapter 2.6.1). Note that not all single-shot codes necessarily have metachecks; that is, their parity-check matrices can be full-rank [60].

2.4 Surface-code decoding

2.4.1 Optimal decoder

As mentioned, a critical part of the error-correction process is the decoding problem: the task of inferring the most likely intended codeword given an error syndrome σ. Classically, this is usually a matter of determining the most likely error; for quantum surface codes, it suffices to find any topologically-equivalent error as a correction operator which results in a stabiliser of the logical state.

Formally, for a set of edges E and vertices V, the decoding problem is one of selecting a correction operator $\mathcal{C} \subset E$ consistent with a syndrome (i.e. a set of defects) $\sigma \subset V$ in the same topological equivalence class as the data errors $E_Z \subset E$, i.e. selecting $\mathcal{C}$ as an approximation of E_Z up to a stabiliser. Therefore, the optimal strategy is to calculate the probabilities of all possible candidates for E_Z consistent with σ and thus determine the most likely equivalence class.

In Chapter 2.1.1, we saw the emergence of a *code threshold*: a critical value for the data error rate p below which the logical error rate p_L is suppressed asymptotically by increasing the code distance d. We denote the code threshold p_c, which becomes a singularly useful metric for quantifying the fault-tolerance of a given correction scheme, with direct indication of the scalability of a scheme in practice.

decoder	worst-case complexity	threshold, p_c
optimal	$O(2^n)$	~0.1100
MWPM	$O(n^3)$	~0.1031
HDRG	$O(n^2)$	~0.0840
union–find	$O(n\alpha(n))$	~0.099

Table 2.1: Comparison of some 2D decoder algorithms, in terms of code threshold under independent noise and complexity in the number of physical qubits. Note that the complexity for union–find is amortised and $\alpha(n)$ is extremely slow-growing.

The threshold for the optimal decoder on the toric code with an independent noise model has been shown to be $p_c \approx 0.1100$. This can be derived as a root of $R = 1 - 2H(p)$, where R is the asymptotic rate of error-free transmission, dropping to zero at the threshold [36, 50].[7] The problem also has a dual in statistical mechanics and has been mapped onto the random-bond Ising model (RBIM) in which the code threshold emerges as a phase transition. For example, this technique is utilised in [63] to obtain a result of $p_c = 0.1094 \pm 0.0002$, consistent with the value obtained using R.

The space of possible errors scales exponentially in the number of physical qubits n. At scale, this measurement–decoding cycle must be repeated after every logical operation, or at least on the same order of magnitude [64]; it is no surprise, then, that the use of an exponential-time decoder is generally considered infeasible in practice. The development of decoders which strike a balance between computational efficiency and achieving a threshold as close as possible to the optimal is thus a popular open problem.

2.4.2 MWPM decoder

An approximation of determining the most likely equivalence class is instead determining the single most likely error consistent with the syndrome σ.

[7] $H(p) = -p\log_2 p - (1-p)\log_2(1-p)$ is the *binary entropy function*, or *Shannon entropy*. Similarly, the hashing bound (a generalisation of the Shannon entropy) gives the correct threshold for the toric code under depolarising noise [62].

Conceptually, this is a relatively straightforward task, as it is equivalent to the *minimum-weight perfect matching* (MWPM) of all defects. In the independent error model of uniform p for all qubits, the syndrome graph is unweighted, such that the matching graph comprises the Manhattan distance between all defects, that is, the L^1 distance, equal to the sum of the horizontal and vertical distances.

The MWPM of the matching graph is the connection of defects into pairs such that the total weight of the connection is minimised. This is intuitively equivalent to the single most likely error consistent with the syndrome. This can itself be considered a good approximation of the most likely equivalence class, as the threshold for the MWPM decoder has been shown to be only slightly less than that of the optimal decoder, with a result of $p_c = 0.1031 \pm 0.0001$ obtained through the RBIM technique [65]. A result of $p_c = 0.0293 \pm 0.0002$ is also given for three-dimensional MWPM, that is, the phenomenological error model with measurement error rate $q = p$. We therefore expect phenomenological error thresholds to be approximately one order of magnitude smaller than independent error thresholds.

When transitioning from the toric to the planar code, recall that an odd number of defects may be present, due to lone defects occurring which should be matched to an open boundary. This is incorporated by adding to the matching graph the distance from each defect to its nearest open boundary [66]. An additional caveat arises that these 'virtual' boundary nodes must also participate in the matching, which would force all defects to match to the open boundary. To prevent this, the virtual boundary nodes are connected together with zero-weight edges, such that unused boundary nodes may be 'paired-off' for zero cost (Figure 2.10).

Minimum-weight perfect matching can be performed in time polynomial in the number of vertices. The blossom algorithm introduced by Edmonds

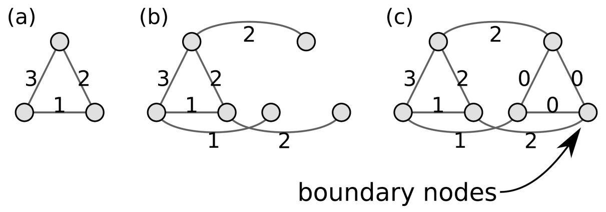


Figure 2.10: Matching graph for a planar code. All defects (a) are connected to each other and their nearest open boundaries, weighted by Manhattan distance (b), with the boundary nodes interconnected with zero weight (c). Image from [66].

in 1965 finds the MWPM in $O(|E||V|^2)$ time [67]. As the matching graph consists of the distances between all defects, it is a perfect graph, such that $|E| = |V|(|V| - 1)/2$. Therefore, the time complexity of Edmonds' blossom algorithm becomes $O(V^4)$. There have been many improvements of the blossom algorithm over time, improving complexity to $O(V^3)$. Currently, a overall frontrunner in performance is Blossom V, introduced by Kolmogorov in [68].

In general, then, a MWPM decoder runs in time polynomial in the number of physical qubits. The MWPM decoder is generally considered the foundational approach from which multiple variations have been proposed and against which decoders are frequently compared. A local variant of MWPM given in [57, 69] considers only the matching graph formed from the m nearest neighbours of each defect – providing a notable speedup over exact matching – with a linear time complexity in m and a threshold which converges upon that of exact matching for approximately $m \geq 16$ in the phenomenological model.

Local matching is justifiable due to the likelihood with which defects are located close together compared to the overall size of the lattice, which by definition occurs for sub-threshold error rates on arbitrarily large-scale systems (see Chapter 3.4.1). This insight can be used by alternative approaches based in clustering, such as the *hard-decision renormalisation group* (HDRG) decoder [70]. The HDRG decoder iteratively searches the neighbourhood of each defect formed by increasing Manhattan distance and trivially matches with the first defect discovered within this neighbourhood (hence *hard-decision*) [71]. This forms a greedy approximation of minimum-weight matching which runs in quadratic time, with a correspondingly reduced threshold of $p_c = 0.0840 \pm 0.0001$ [72].

2.4.3 Union–find decoder

A leading alternative decoder scheme is the *union–find* (UF) decoder [40]. The primary motivation behind the UF decoder is the concept of an *erasure error*: any error occurring in quantum hardware wherein a qubit is taken out of the computational subspace, but which also has the mitigating quality of a known position. For example, a qubit which uses the $|0\rangle$ and $|1\rangle$ states of some physical system may leak into the unused $|2\rangle$ state, which is detectable by the projective measurement $|0\rangle\langle 0| + |1\rangle\langle 1|$ without decohering the computational subspace.

When erasure is detected on a physical qubit, it can be returned to the computational subspace, e.g. with a maximally-mixed (i.e. depolarised) state or a basis state, with syndrome extraction then proceeding as normal. An erasure is therefore equivalent to a random Pauli error at a known location; an appropriate decoder can use this positional information accompanying the syndrome for more efficient and accurate error correction [73].

The UF decoder exploits this by reinterpreting the syndrome of Pauli errors with a covering set of erasure errors; intuitively, this is beneficial as it subjectively reduces the uncertainty in the location of errors. Correcting erasure errors can be performed in linear time [73] and 'converting' Pauli errors to erasure errors can be performed in near-linear time, forming a two-

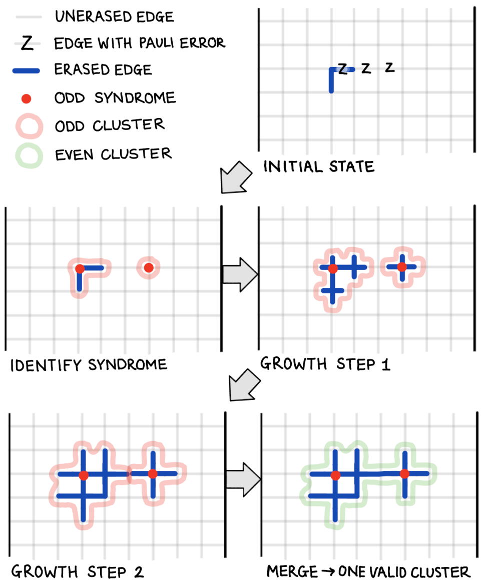


Figure 2.11: Syndrome validation in the union–find decoder. Note that whilst both a syndrome and initial erasure are provided in this example, the decoder also works without an initial erasure. Image from [40].

stage, near-linear-time decoder overall.

The first stage, referred to as *syndrome validation*, is performed by iteratively growing clusters from each defect, not dissimilarly to the clustering approaches discussed above. Given an observed syndrome $\sigma \subset V$ and an initial observed erasure $\varepsilon_0 \subset E$ (i.e. the set of qubits with a detected erasure error), a cluster is created for each defect, which is grown out by a half-edge each iteration (such that two adjacent defects are connected by one edge after a single iteration). Whenever clusters meet, they are merged together in a *disjoint-set*, or *union–find*, data structure. Only clusters which support an odd number of defects continue to grow, such that validation finishes as soon as all clusters are even. All grown edges are added to the modified erasure ε (Figure 2.11).

Next, a correction operator must be found within ε. In order to ensure optimal complexity by limiting suboptimal choices, cycles are removed from

ε; in other words, a spanning forest F_ε is found upon ε [73]. This can trivially be performed in linear time via depth-first search [74], Kruskal's algorithm [75] etc. The *peeling decoder* from [73] can then be applied to F_ε to obtain a correction operator $\mathcal{C}$. A *leaf* is defined as an edge $\{u, v\}$ connected to F_ε only by vertex v, such that vertex u is defined as a *pendant.* Until F_ε is empty, a leaf $e = \{u, v\}$ is removed from F_ε and, if $u \in \sigma$, e is added to $\mathcal{C}$, u is removed from σ and v is flipped in σ (that is, v is removed from σ if present and added if not).

The peeling decoder requires only minor alteration when moving to the planar code. Previously, the spanning forest F_ε is chosen arbitrarily. In a planar code, if any of the edges in ε lie on an open boundary, the corresponding spanning tree must be seeded in – that is, grown from – that open edge, with peeling then performed in the reverse direction. This ensures the open edge is not merely treated as a leaf and is included in the final correction operator.

The amortised time complexity[8] of a single operation in the union–find data structure is $O(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function, which is so extremely slow-growing that it is effectively constant (see Chapter 2.5.1). In the worst case, $n-1$ merges are performed; therefore, syndrome validation runs in $O(n\alpha(n))$. Both construction of a spanning forest and the peeling decoder are linear-time algorithms, so the overall complexity of the UF decoder is $O(n\alpha(n))$, i.e. almost linear-time. This is a significant improvement over the polynomial-time decoders discussed above, yet the threshold for the two-dimensional UF decoder under independent noise is as high as 0.099. Similarly, the three-dimensional phenomenological threshold is as high as 0.026, compared to 0.0293 for MWPM given above. This is intuitive when interpreting the UF decoder graphically as finding a good local approxima-

[8]*Amortised* complexity refers to the limit of the complexity of a single operation over arbitrarily many operations [76].

tion of MWPM [77].

A weighted variant of the UF decoder has similarly been proposed, yielding a threshold of 0.0062 under circuit-level depolarising noise, compared to 0.0072 for weighted MWPM [78].

2.5 Implementing union–find

2.5.1 Disjoint-set data structure

The basis of the syndrome validation stage of the union–find decoder is the ability to track clusters merging together as disjoint (i.e. non-intersecting) sets of vertices. This can be efficiently implemented using a *disjoint-set data structure*, also known as a *union–find data structure*, so named after its two operations: *union* and *find*. The find operation must determine which set a given element belongs to, whilst the union operation must merge the sets of two given elements together [79].

A common method of implementing this structure is as a forest of elements, with each element pointing to its parent to form trees (Figure 2.12). Each set is identified by a representative element which is the root of the tree;

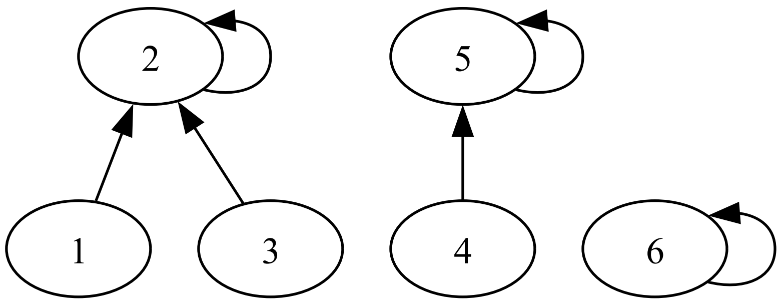


Figure 2.12: Example of a disjoint-set data structure. Elements 1, 2 and 3 form a set (with 2 as the root, or representative element), as do 4 and 5 (with 5 as the root), and finally 6 is the root of its own set.

the element forming the root is arbitrary and depends on the exact method by which and order in which sets are merged. Root elements are their own parent: the structure is initialised with all elements pointing to themselves.

Given an element, the find operation traverses up the tree until reaching an element which is its own parent, reporting this as the root of the set. Given two elements, the union operation sets the parent of one of the tree's roots to be the other's root, thus merging the two sets into one.

Naively, the trees can grow with arbitrary height; in the worst case, trees degenerate into lists, such that the worst-case complexity of traversing them is $O(n)$. Therefore, performing m operations takes $O(mn)$ time. This would give the union–find decoder a complexity of $O(n^2)$, as the number of operations in the decoder varies with the number of defects, which is of order np.

Two optimisations can be used in order to improve the complexity of the operations. Firstly, one can ensure that when merging two trees, the smaller tree is always merged into the larger tree; that is, ensuring the root of the larger tree becomes the new root of the union. This is known as *union-by-size* when defining the size of a set as the number of nodes in the tree (*rank* may be used as an alternative metric, which is an upper bound on the height of a node). When using union-by-size, a single operation may still take $O(n)$ time if the two trees are of the same size. However, performing union-by-size bounds the height of trees to order $O(\log n)$ instead of $O(n)$, thus the time taken by m operations is now bounded by $O(m \log n)$ [80]. This would give the union–find decoder an improved complexity of $O(n \log n)$.

Secondly, the find operation can be altered to change the structure of the trees as they are traversed. *Path compression* changes the parent pointer of every node traversed to point directly to the root after it is found, such that performing the find operation in future on any of those nodes again

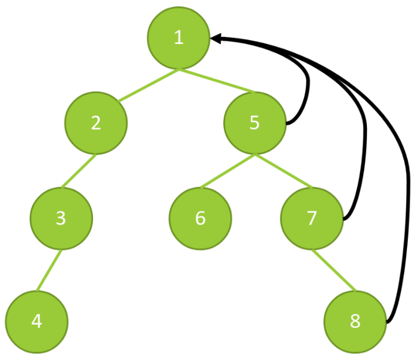


Figure 2.13: Example of path compression. If all nodes form a tree as shown and the find operation is invoked on node 8, all nodes encountered when traversing to the root (node 1) have their parent set to the root ex post facto, such that nodes 5, 7 and 8 all become direct descendants of the root, optimising any future operations involving these nodes.

(assuming no union operations are performed since) becomes constant-time (Figure 2.13). When combining this technique with union-by-size, the time taken by m operations becomes $O(m\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. The derivation of this emergent term in the complexity is nontrivial and beyond the scope of this review [79]. It suffices to note that this function is extremely slow-growing:

$$\alpha(61) = 3 \tag{2.61}$$

$$\alpha(2^{2^{2^{16}}} - 3) = 4 \tag{2.62}$$

$$\alpha(2^{2^{2^{2^{16}}}} - 3) = 5\ . \tag{2.63}$$

In the computational regime, then, $\alpha(n)$ is a local constant which is effectively global. This gives the union–find decoder an optimised complexity of $O(n\alpha(n))$, which is effectively linear-time.

2.5.2 AFS architecture

In reality, any viable error correction scheme must be implemented on physical microarchitecture with efficient and scalable design. Das et al. present such a design in [81], implementing the union–find decoder discussed in Chapter 2.4.3, named the *AFS* ('accurate, fast and scalable') decoder. The primary design consists of three modules, each consisting of the data structures required to implement the three stages of the decoder, as shown in Figure 2.14.

The first module, the graph generator, is responsible for the syndrome validation stage. In this module, the spanning tree memory (STM) stores syndrome measurements in a grid data structure, with two edge cells between each vertex cell, implementing the half-edge growth. The zero data register (ZDR) stores a bit for each row of the STM signifying whether they have any non-zero bits, speeding up traversal of the STM by allowing empty rows to be skipped. The root and size tables manage the union–find data structure, storing for each vertex the parent and, if it is a root, the size of the set tree. The traversal registers note the elements encountered in a find operation such

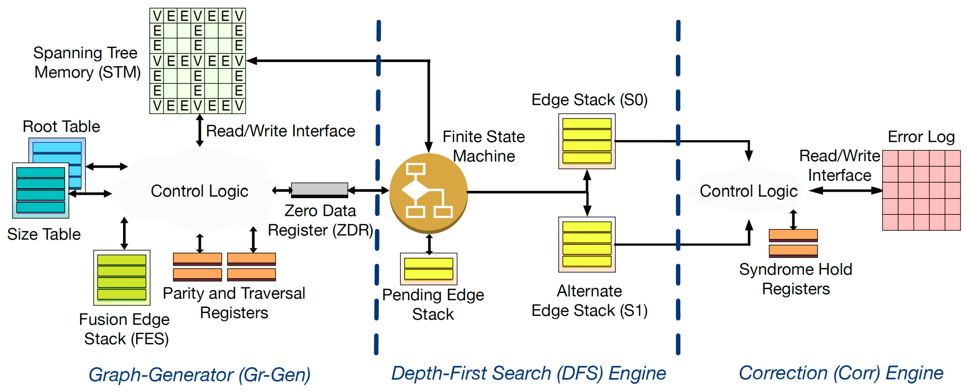


Figure 2.14: Three-stage pipeline of the AFS microarchitecture. Image from [81].

that their parent pointers can be set to the root once found, implementing path compression. The parity registers store the parity of each cluster (that is, whether the number of defects supported is odd or even) in order that only odd clusters are grown. The fusion edge stack (FES) stores any grown edges which would connect to another cluster, delaying writing them to the STM until all clusters have grown in that iteration. This prevents ‘double growth’, in which the second cluster would include the first in its new boundary and would thus grow out again from all edges in the first cluster.

The second module, the depth-first search (DFS) engine, is responsible for performing depth-first search on a fully-grown cluster in order to remove cycles, i.e. find a spanning tree. Depth-first search is easily codified by a stack data structure (‘last-in, first-out’, or *LIFO*). As edges are discovered by the depth-first search, they are added to the pending edge stack. When edges are expanded, they are placed on an edge stack, forming a list of edges in the spanning forest in the order in which they were expanded. The architecture allocates two edge stacks (S0 and S1), such that the third module can work on a spanning tree stored in one stack whilst the DFS engine can work on the next cluster, building a spanning tree in the other stack.

The third module, the correction engine, is responsible for implementing the peeling algorithm to generate the final correction operator for each cluster. The correction engine updates the error log, which persists from the previous error correction cycle, with the new correction bits. The syndrome hold registers are used to store when vertices are removed from or flipped in σ as part of the peeling algorithm.

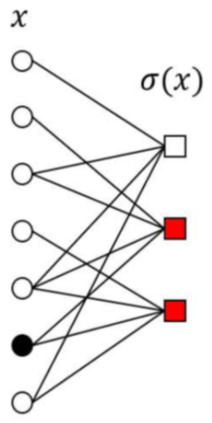
Minimisation of memory consumption is a key design goal in the architecture, with attention given to how the size of the largest components (such as the STM and edge stacks) varies with code distance d. The proposal demonstrates how memory consumption can be lower than that of a MWPM implementation, especially as d increases. To parallelise the solution in practice, multiple instances of all three modules can be used. As much more time is spent working in the graph generator than the other two modules, the proposal includes how the architecture can be scaled more intelligently by multiplexing fewer DFS and correction engines with more graph generators, at varying ratios.

The proposal concludes by briefly considering some different data compression strategies for the syndrome data in order to alleviate bandwidth demands. The most straightforward approach, a sparse representation, starts with an indicator bit which is one if there are any ones in the data, followed by the indices of all ones. Another approach, dynamic zero compression (DZC), splits the data into equal partitions of width m; the data starts with one indicator bit for each block (which is zero if its corresponding block is non-zero) followed by all of the non-zero data blocks in order. Finally, geometry-based compression extends DZC to higher-dimensional partitions to align more closely with the lattice-based structure of the correction code. It is reasoned that sparse representation is preferable when the error rate is very low, due to the vanishingly small number of nonzero bits present; in a noisier regime, the other approaches become more effective.

2.6 LDPC codes

2.6.1 qLDPC codes and Tanner graphs

Recall the general definition of a linear code, defined by a parity-check matrix H. Classically, a *low-density parity-check* (LDPC) code was defined as one with a small number $j \geq 3$ of checks associated with each bit, and a small number $w > j$ of bits associated with each check [33]. In modern QEC usage, a quantum LDPC (qLDPC) code can often be defined to include any code



$$H\mathbf{x} = \sigma$$

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

Figure 2.15: On the left, a Tanner graph of the $[[7, 1, 3]]$ Steane code, with qubits on the lefthand nodes and checks on the righthand nodes (image from [41]). On the right, an equation showing the parity-check matrix represented by the Tanner graph (as well as the highlighted error mechanism and syndrome).

with a sufficiently sparse H: most sources do indeed consider surface codes as LDPC because the above values j and w remain constant as the size of the code increases [82, 83], but the term is commonly used to indicate codes which in contrast are not necessarily topological or graphlike [84].

A characteristic feature of topological codes is that checks are heavily local; that is, checks are generally performed between spatially-neighbouring qubits. In the general case, this is no longer true, although various LDPC codes can exhibit parametric notions of locality. Significantly, in surface codes, qubits (on edges) are associated with exactly two checks (on vertices), meaning that the decoding graph is strictly graphlike.[9] On arbitrary codes, however, qubits can be associated with any number of checks, forming hypergraphs.

A hypergraph is a mathematical generalisation of a graph, in which hyperedges can be associated with any number of vertices, rather than exactly two. One approach to a graphlike representation of a hypergraph is a bipar-

[9](ignoring open boundary conditions in planar constructions)

tite graph, with one set of nodes representing vertices and the other representing hyperedges. In the context of error correction, this is often referred to as a *Tanner graph* [85]. Figure 2.15 shows a simple example of a code represented as a Tanner graph.

2.6.2 3D toric code

Our first example of a code with a hypergraphlike decoding structure is the generalisation of the 2D toric code, introduced in Chapter 2.2.1, to higher-dimensionality topologies; specifically, we can extend to a 3D toric code by adding an additional dimension with the same periodic boundary conditions [86, 87]. Qubits are similarly embedded on the edges of this cubic lattice, with the X and Z stabiliser checks defined as star and plaquette operators, respectively, in the same manner as the 2D toric code (Equation 2.57).

Although these operators use the same fundamental definition as the 2D case, it is important to note how their properties differ on this new topology. The star operators become weight-6 (rather than weight-4 as in the 2D code) as each vertex now has exactly 6 incident edges; the decoding graph on star operators is still graphlike, however, because each edge is still incident to exactly 2 vertices. In contrast, the plaquette operators, now also known as *face operators*, remain weight-4; however, the decoding graph on face operators is now hypergraphlike, because each edge is now incident to 4 edges.

As there are now three topological degrees of freedom, the 3D toric code with periodic boundary conditions intuitively encodes $k = 3$ logical qubits, versus the 2D toric code's $k = 2$. However, the differing structures of the stabiliser checks as discussed above lead to an asymmetrical error-correcting capacity: the minimum logical-Z weight is still L, but the minimum logical-X weight is now L^2. Therefore, the code distance is variant with the type of

Pauli noise, making it a $[[3L^3, 3, d_X = L^2, d_Z = L]]$ code [87].

Finally, note that the 3D toric code is not to be confused with the difference-syndrome approach discussed in Chapter 2.3.1, which instead extends a 2D surface code to a 3D decoding graph by adding a timelike dimension.

2.6.3 Colour codes

A 2D *colour code* is a topological code in which qubits are embedded on the vertices of a 2D lattice satisfying two properties: first, that each vertex has three incident edges; second, that the faces formed by the lattice are 3-colourable, i.e. they can be assigned one of three 'colours' (conventionally 'red', 'green' and 'blue') such that no two neighbouring faces share a colour [55, 89].

Formally, contrasting with the operators defined for the surface codes in Equation 2.57, the stabiliser generators for the colour codes are defined as

$$S_f^X = \prod_{v \in \partial f} X_v \ , \ S_f^Z = \prod_{v \in \partial f} Z_v \ . \tag{2.64}$$

Note that, unlike the surface codes, these generators are homogeneous – they

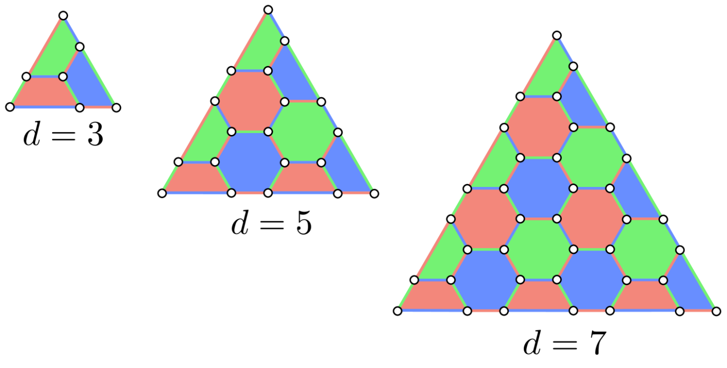


Figure 2.16: 6.6.6 colour code with triangular boundaries. Image from [88].

are defined as the same (face) operators, such that the parity-check matrices defining the X and Z stabiliser checks are identical.

Few 2D lattices satisfy the above conditions: arguably the simplest is the 6.6.6 (a.k.a. honeycomb/hexagonal) tiling, as shown in Figure 2.16. The code is most commonly seen in its planar version with triangular boundaries, but can equally be considered with periodic boundary conditions.

Finally, to see how such a code translates to a hypergraphlike decoding problem, note that the 6.6.6 colour code for $d = 3$ is equal to the $[[7, 1, 3]]$ Steane code shown in Figure 2.15.

2.7 LDPC-code decoding

2.7.1 Belief propagation

Belief propagation (BP) is an algorithm for calculating marginal distributions on graphical probability models (such as Markov random fields or, indeed, Tanner graphs) based on an observation [35]. It is a message-passing algorithm in which each node provides its neighbours with a function able to calculate beliefs based on the information provided by its own neighbours. For a graphical model of n random variables forming a tree, BP provides an exact solution after a single pass in time $O(n)$. On general graphs (i.e. containing cycles), then 'loopy' BP can provide an approximate solution after multiple iterations, although in some cases it may only converge on a local optimum or fail to converge at all. Generally, loopy BP tends to perform well for approximate solutions outside of adversarial models, especially as exact marginal calculation is NP-hard in the general case [90].

BP can therefore be used as a decoder for LDPC codes, by approximating the marginal error probabilities for each bit given an observed syndrome. In each iteration, the marginal probability for each bit $P(x_i = 1)$ is updated as a soft decision, such that the hard decision for each bit is calculated as

$$x_i = \begin{cases} 1, & P(x_i) \geq 0.5 \\ 0, & P(x_i) < 0.5\ . \end{cases} \tag{2.65}$$

If the hard decision on the error is consistent with the observed syndrome, then the decoder has converged and can terminate, else the iteration continues [91].

Although Tanner graphs are not trees in the general case, the BP decoder nonetheless tends to perform very well on classical LDPC codes, due to the fact that sparse bipartite graphs are locally tree-like [92, 93]. However, BP is less applicable to quantum LDPC codes due to degeneracy; the existence of multiple optimal solutions means that the decoder can frequently fail to converge [91].

2.7.2 Generalised union–find

An alternative approach to decoding LDPC codes is to attempt to generalise the union–find decoder, as introduced in Chapter 2.4.3. As in the original version for surface codes, the generalised approach is to start with the error syndrome σ and iteratively add neighbouring nodes until a grown erasure forest $\mathcal{E}$ covers a valid correction operator [41]. The key difference is that clusters are now grown on a hypergraph, i.e. a bipartite Tanner graph, instead of a graphlike structure with local edges.

Firstly, given a set of vertices $A \subseteq V$, the *interior* of the set $\mathrm{Int}(A)$ is defined as only the elements of A which are not part of the boundary, where the boundary constitutes all elements which have neighbours outside of A. The erasure $\mathcal{E}$ is initialised to σ. The neighbours of $\mathcal{E}$ are iteratively added to $\mathcal{E}$ until at least one valid correction operator is contained within $\mathrm{Int}(\mathcal{E})$.

Here, 'valid' simply means an error mechanism consistent with the syndrome. Only the interior is considered because any qubits on the boundary of $\mathcal{E}$ by definition must act on checks outside of $\mathcal{E}$ and therefore cannot be part of a valid correction.

This generalisation as given in [41] is more akin to a meta-algorithm which relies on two black-box functions: syndrome validation and solution generation. The former tells us when to stop growing clusters, i.e. when a valid correction operator exists. In the original UF algorithm, this is simply when clusters support an even number of qubits, but this condition does not extend to the hypergraph case. The latter finds a valid correction operator within a fully-grown cluster, which is equivalent to the peeling algorithm, although this likewise does not trivially extend to the hypergraph case.

The approach used for these functions in [41] is to use Gaussian elimination to directly solve the decoding equation $H\mathbf{x} = \sigma$ in cubic time. Applied independently, this could be considered the most general decoder for linear codes, with the clear downside that a naive implementation effectively returns an arbitrary valid correction operator, yielding extremely poor decoding accuracy. The generalised UF decoder, therefore, acts as a way to optimise this 'Gaussian decoder' by considering a reduced system of checks and qubits – specifically, the minimum set of nodes containing a solution. This guides Gaussian elimination to approach a minimum-weight solution.

Each growth step, as nodes are added, rows and columns are added to a reduced parity-check equation representing $\mathcal{E}$ and Gaussian elimination is performed, from which the presence of a solution becomes clear. If a solution exists, then growth terminates and a solution can be easily generated from the results of Gaussian elimination. This process is explained in more detail, including its strengths and weaknesses, in Chapter 4.2.1.

A key weakness of the generalised UF decoder is that clusters can grow extremely quickly on an arbitrarily densely-connected Tanner graph. However, it is shown in [41] that the decoder performs well for certain classes of LDPC code. The *covering radius* of a syndrome σ is defined as the number of growth steps needed to cover a valid correction operator, denoted $\rho_{\text{cov}}(\sigma)$. It was shown that if the covering radius is upper-bounded logarithmically in error weight, then the decoder successfully corrects all errors below a certain weight. Take a code with distance d whose Tanner graph has maximum degree δ. Formally, if there exist constants w and C such that

$$\rho_{\text{cov}}(\sigma) \leq C \log |\mathbf{x}| \quad \forall |\mathbf{x}| < w \ , \tag{2.66}$$

then the generalised UF decoder successfully corrects errors with weight

$$|\mathbf{x}| < \min(w, Ad^{\alpha}) \ , \tag{2.67}$$

where

$$\alpha = \frac{1}{1 + C \log \delta} \ , \ A = \left(\frac{1}{2\delta^2}\right)^{\alpha} \ . \tag{2.68}$$

Let us hereinafter refer to codes which satisfy this condition as 'well-behaved' codes; such codes will be explored in a complexity analysis of the generalised UF decoder in Chapter 4.2.3. Some broad examples of codes proved in [41] to be well-behaved by this definition are hyperbolic codes [94], locally-testable codes [95] (e.g. expander codes [96]) and D-dimensional toric codes for $D \geq 3$.

2.7.3 BP–OSD

In Chapter 2.7.1, belief propagation (BP) was introduced as a linear-time decoder popular for classical LDPC codes, but insufficient for quantum LDPC codes due to degeneracy. In Chapter 2.7.2, we saw how generalised union–find (UF) relies upon Gaussian elimination to solve the decoding equation

$H\mathbf{x} = \sigma$. This system of equations is, in general, *underdetermined*: there exist multiple valid solutions for the error $\mathbf{x}$ given a syndrome σ. Rather than return any arbitrary correction, generalised UF attempts to approximate a minimum-weight solution by performing Gaussian elimination on a neighbourhood no larger than the covering radius.

An alternative approach is to perform Gaussian elimination on the global system, but to use the results of an unconverged BP to inform the choice of free variables, rather than setting them arbitrarily. This is equivalent to reordering the columns (i.e. bits) in descending order of their soft-decision marginal probabilities before performing Gaussian elimination on a full-rank submatrix. This technique is known as *ordered statistics decoding* [97], forming the BP–OSD decoder [91, 98].

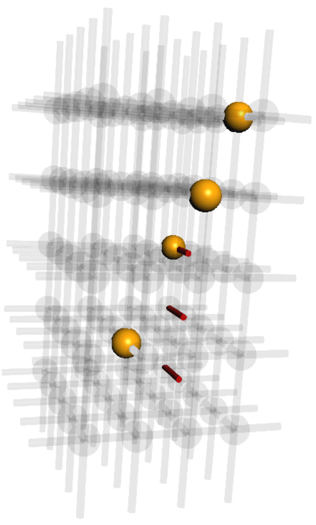
Chapter 3

Surface-code decoding at scale

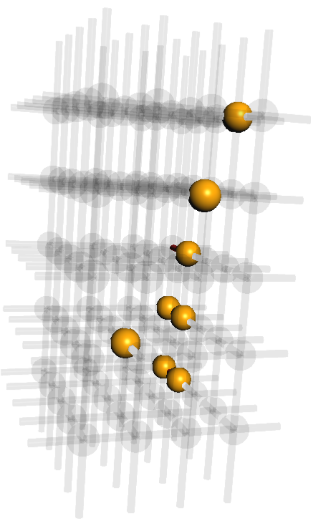
3.1 Motivation and methods

In Chapter 2.4.3, the union–find (UF) decoder was introduced as an accurate and fast decoder for surface codes. It remains one of the most popular choices in this regime as it forms a good local approximation of minimum-weight perfect matching (MWPM) with significantly lower time complexity. In addition, it performs particularly well under the erasure noise channel. Experimental attention is increasingly prevalent on engineering qubits where erasure is the dominant noise, due to the relative ease of detection and correction. Indeed, erasure dominance is one of the key advantages of implementations such as photonic qubits [30], with recent work in developing dual-rail techniques to attain this same attribute with superconducting qubits [99]. For these reasons, the UF decoder seems a rewarding approach to focus further study upon.

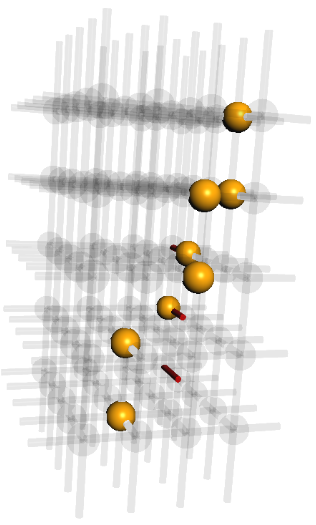
Due to the frequency with which decoding rounds must be performed, decoder time complexity forms a key bottleneck on the performance of an error-corrected quantum computer at scale, especially with a difference-syndrome approach or similar [64]. Space complexity (i.e. memory consumption) is also



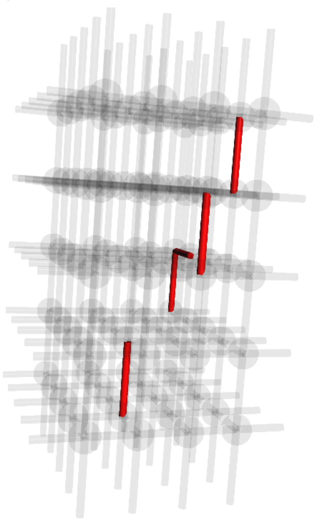
(a) A data error is inflicted on each round forward in time with probability p. Each ancilla is also inflicted with a measurement error with probability q.



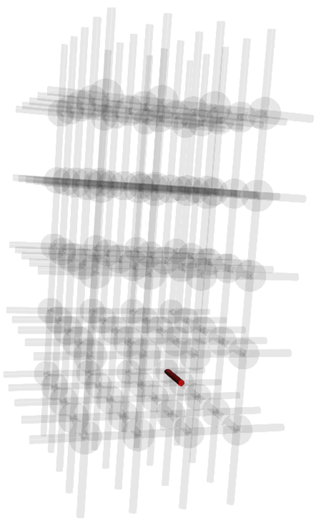
(b) Syndrome measurements are performed as usual via edge parity, flipped if a measurement error was inflicted.



(c) The syndrome is converted to a difference syndrome, wherein a defect at t indicates a change from $t-1$.



(d) MWPM is performed on the difference syndrome.



(e) The correction operator is collapsed (sum modulo two). Note that this successfully corrects the final round in (a).

Figure 3.1: Example difference-syndrome correction on a 2D planar code, where $p = q = 0.01$ and the final measurement round is at the bottom.

a concern with embedded implementations, especially in cryogenic systems [81], as is the overall simplicity of implementation. Therefore, it is critical to identify and solve fine-grained performance decisions.

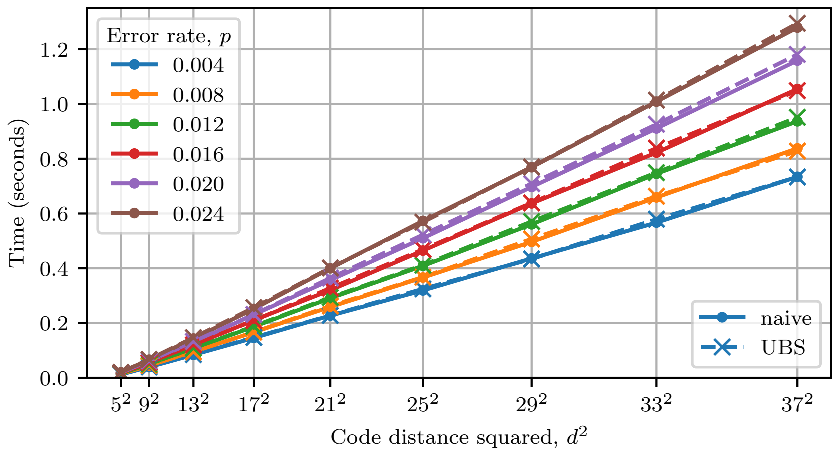
In order to inspect the behaviour of decoding algorithms on surface codes, I developed an exhaustive simulation library in C++. This included custom implementations of 2D toric and planar lattices (Chapters 2.2.1 and 2.2.2), the MWPM decoder using the LEMON library[1] and the union–find decoder based on the AFS architecture (Chapter 2.5.2). I also implemented a simulation of difference-syndrome decoding with a phenomenological model (Chapter 2.3.1), using the Easy3D library[2] for 3D lattice visualisation, as seen in Figure 3.1. In Section 3.2, I present brief notes on some novel decisions I made in the implementation detail of the decoder.

With this simulation library, I initially performed basic timing experiments where the real time taken by the AFS simulation to decode 5000 random instances was measured (Figure 3.2). The naive approach and union-by-size (UBS) generally performed similarly, whereas adding path compression caused the time taken to increase, especially prevalent at greater p and d. This is a surprising result, as complexity analysis in the literature (as introduced earlier in Section 2.5.1) suggests that both UBS and path compression should act to improve the amortised time complexity.

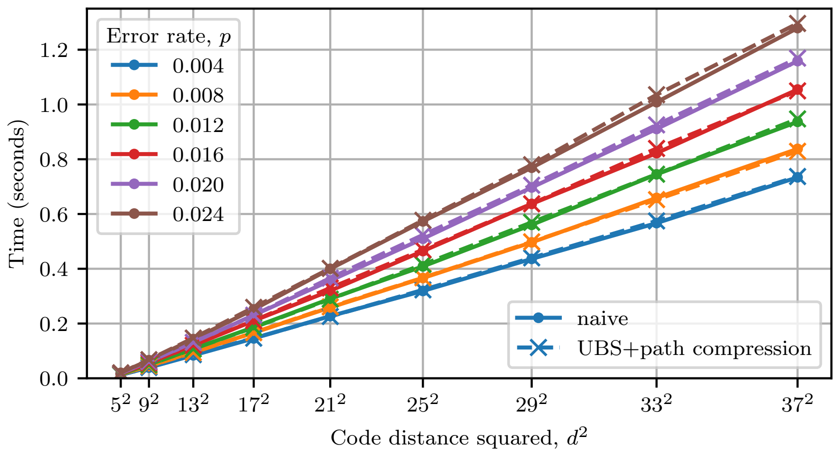
To explain this observation, in Section 3.3.1 I first present the results of a deeper investigation into the complexity of the disjoint-set data structure itself, identifying a mode of operation in which UBS and path compression worsen runtime, which I denote the *unsaturated regime*. In Section 3.3.2, I present more rigorous empirical data confirming that adding UBS and/or path compression to the union–find decoder strictly increases the number of root and size table accesses. These concepts are then directly linked together

[1]`https://lemon.cs.elte.hu`

[2]`https://github.com/LiangliangNan/Easy3D`



(a)



(b)

Figure 3.2: Wall-clock time taken by the AFS simulation to decode 5000 random instances. Each point is the minimum time from 10 identical runs. The plots compare the time taken by a naive approach with (a) UBS and (b) both UBS and path compression.

by *percolation theory*, which I introduce briefly in Section 3.4.1 and then use to prove that the clusters formed by the decoder for $d \to \infty$ are strictly sparse, and thus operate in the unsaturated regime, in Section 3.4.2.

I implemented the simulation for the percolation analysis in Section 3.4.2 in Python using the NetworkX library.[3]

Finally, Section 3.5 offers concluding remarks, such as the potential practical benefits of forgoing these optimisations, and points to some ways in which these published results have been cited and used [14–23].

3.2 AFS implementation and improvements

First, let us consider the AFS architecture [81] as an example implementation of the UF decoder and briefly discuss some alternative decisions – or even improvements – used for the simulations in the remainder of this work.

Whilst the AFS architecture describes the practice of recalculating cluster boundaries in each iteration of growth during syndrome validation, my simulations instead store and update lists of boundary sites, with the benefit that this data becomes readily available for analysis (as used in Chapter 3.4).

In the AFS architecture, a *fusion edge stack* (FES) is used to prevent double-growth by delaying the writing of edges between newly-connected clusters to the STM. As we are now storing boundary lists, I similarly define and use a *new edge stack* (NES) to store and delay the writing of every newly-grown edge, to prevent a similar issue. I also define a *confinement register* (one bit per vertex) to disable the growth of clusters which meet the open boundary.

A notable benefit of this approach is that the NES, by definition, contains exactly the new boundary list for a grown cluster. Therefore, the boundary

[3]`https://networkx.org`

list and NES can operate as a double buffer, wherein the NES becomes the new boundary list. Swapping the two lists and clearing the new NES (now containing the old boundary data) ready for reuse can both be performed in $O(1)$ (constant) time.

The main drawback of this approach is that boundary lists must be concatenated when clusters are to be merged. Using arrays, this has time complexity linear in the size of the shorter list. Using linked lists, this can be improved to constant time, but this is unlikely to be worthwhile overall due to cache inefficiency and roughly doubling memory usage.

3.3 Observations

3.3.1 Saturation regimes

Firstly, let us consider the complexity analysis in practice of the disjoint-set data structure (as introduced in Chapter 2.5.1) independently of application to surface-code decoding. Figure 3.3 demonstrates the amortised time complexity of operations on a standalone disjoint-set data structure via a Monte Carlo simulation, constituting m merge operations between two random elements in a forest of size n. By dividing the observed number of root table accesses by m, we expect the plots to collapse onto a single trend, representing the amortised complexity irrespective of m.

For $n \ll m$, a linear growth in the number of root table accesses is observed with n under naive implementation. This changes to logarithmic with union-by-size (UBS) and near-constant with both UBS and path compression. Specifically, in the latter case, the number of accesses converges upon a constant[4] of exactly 8, because asymptotically effectively all elements be-

[4]This constant is technically local but effectively global, due to the extremely slow-growing nature of the $\alpha(n)$ term as discussed in Chapter 2.5.1.

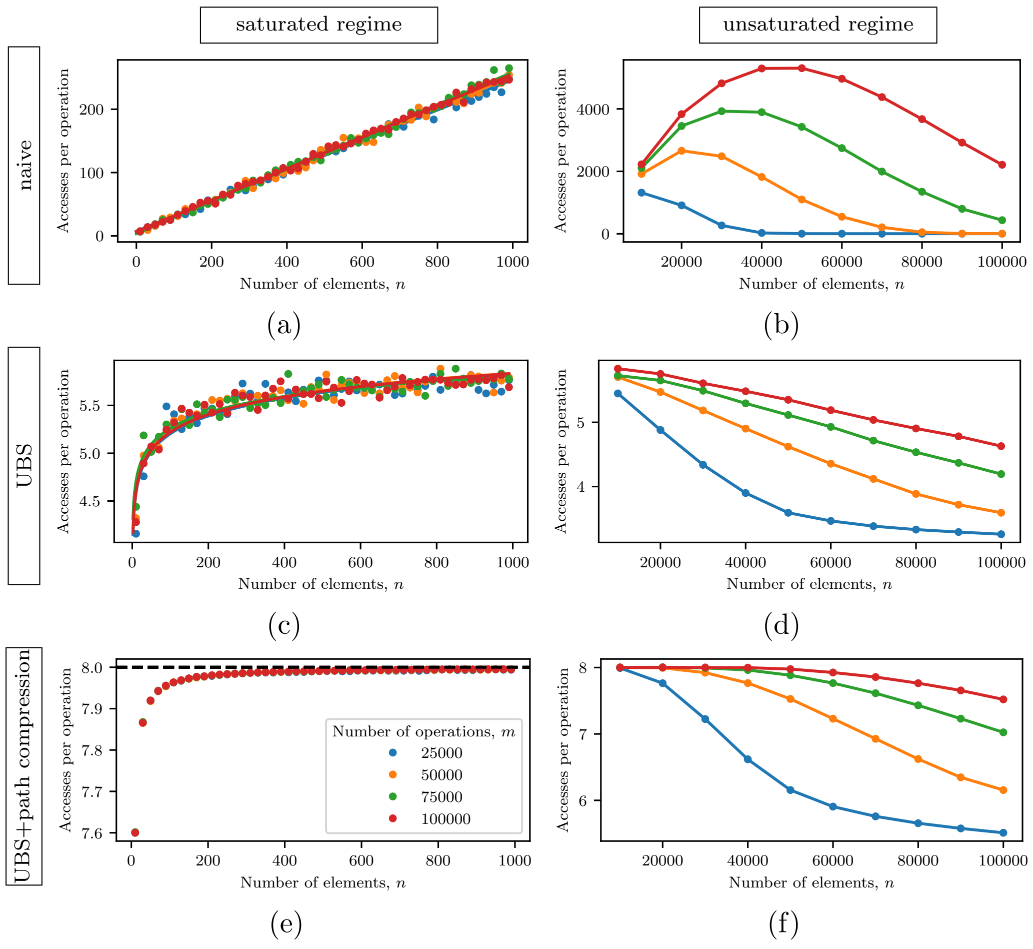


Figure 3.3: Monte Carlo simulation showing the amortised time complexity of operations on a disjoint-set data structure. This data is from stochastically simulating workload on a standalone data structure (i.e. this data is not from surface-code decoding). Each point is the number of root table accesses on a forest of size n taken by m random merge operations, divided by m. (a) shows a naive approach, (c) shows UBS and (e) shows both UBS and path compression, demonstrating linear, logarithmic and near-constant scaling, respectively. (b), (d) and (f) extend these plots to higher n, showing a change of the scaling in the unsaturated regime. (b) and (d) converge down to 3, whilst (f) converges down to 5.

come a direct descendant of the same root, yielding exactly 4 accesses per find operation.[5] Therefore, I denote this mode of operation the *saturated regime*, because the data structure is dominated by a single set, i.e. a single tree spanning the entire forest.

With $n \ll m$, the emergent behaviour accurately reflects the theoretical amortised complexity because the number of operations m is effectively infinite. However, if $n \not\ll m$, trees do not have opportunity to form at or near their bounded height, such that the complexity is seen to degenerate – the heights of the trees are now bounded by m, not n. As the trees of characteristic height do not dominate the forest, I denote this mode of operation the *unsaturated regime*. In this regime (as $n \to \infty$) every random merge will be between two isolated elements, which are sparsely distributed throughout the forest and previously untouched by merges. Therefore, the amortised complexity converges down to the minimum number of accesses required by this situation (5 if using path compression, 3 if not).

3.3.2 Redundancy of optimisations

Next, let us return to the context of the union–find decoder for surface codes. For a clearer view on the operation of the decoder than the wall-clock times in Figure 3.2, I performed simulations capable of counting the number of root table (and size table) accesses. Figure 3.4 shows the scale factor in the number of root table accesses when adding UBS and path compression; that is, the number of accesses divided by the corresponding number from the naive implementation. The number of accesses decreases marginally when

[5]In each find operation, the parent of the descendant is queried (which is the root), then the parent of the root is queried (which is its own parent), then both are rewritten due to path compression, hence 4 accesses. This occurs for both merge arguments, hence 8 accesses. A final 9th merging access does *not* occur asymptotically as all elements will have converged into one global set.

adding UBS, although this then requires the maintaining and accessing of a size table in conjunction. In the worst case, adding path compression (or *path splitting*[6]) exactly doubles the number of accesses, decreasing only marginally below this at scale.

As a more specific example, Figure 3.5 shows histograms of the number of table accesses in a simulation with $d = 49$ and $p = 0.08$. A naive implementation required 4818.79 root table accesses on average (mean over 10^6 instances). With UBS, this was reduced to 4489.41, but also required 3722.97 size table accesses, thus requiring 3393.59 more accesses in total. When adding path compression, the number of size table accesses remained the same, but the number of root table accesses increased to 7841.13, thus requiring 6745.31 more accesses than the naive implementation in total.

It is apparent that both UBS and path compression not only fail to provide an improvement in decoding time, but in fact tend to worsen it at scale. In the case of UBS, marginal reduction in root table accesses is dwarfed by the additional overhead of maintaining and accessing a size table. In the case of path compression, root table accesses are strictly increased, with the additional overhead of tree traversal registers. Even if path splitting is used in place of compression – saving the overhead of registers and a second pass – the increase in the number of root table reads is identical.

The reasoning for this is twofold. First, recall our definition of the data structure's *unsaturated regime*, in which the time complexity degenerates if the sets are bound by their population rate, not by the size of the system, with these sets sparsely distributed throughout the system. This logic is found verbatim in *percolation theory*, which states that if the population rate of clusters on a lattice is below some threshold, the clusters are sparse with

[6] *Splitting* is an efficient alternative strategy in which each node's parent pointer is set to its grandparent as they are discovered, i.e. in a single pass, but is asymptotically equivalent to compression [79].

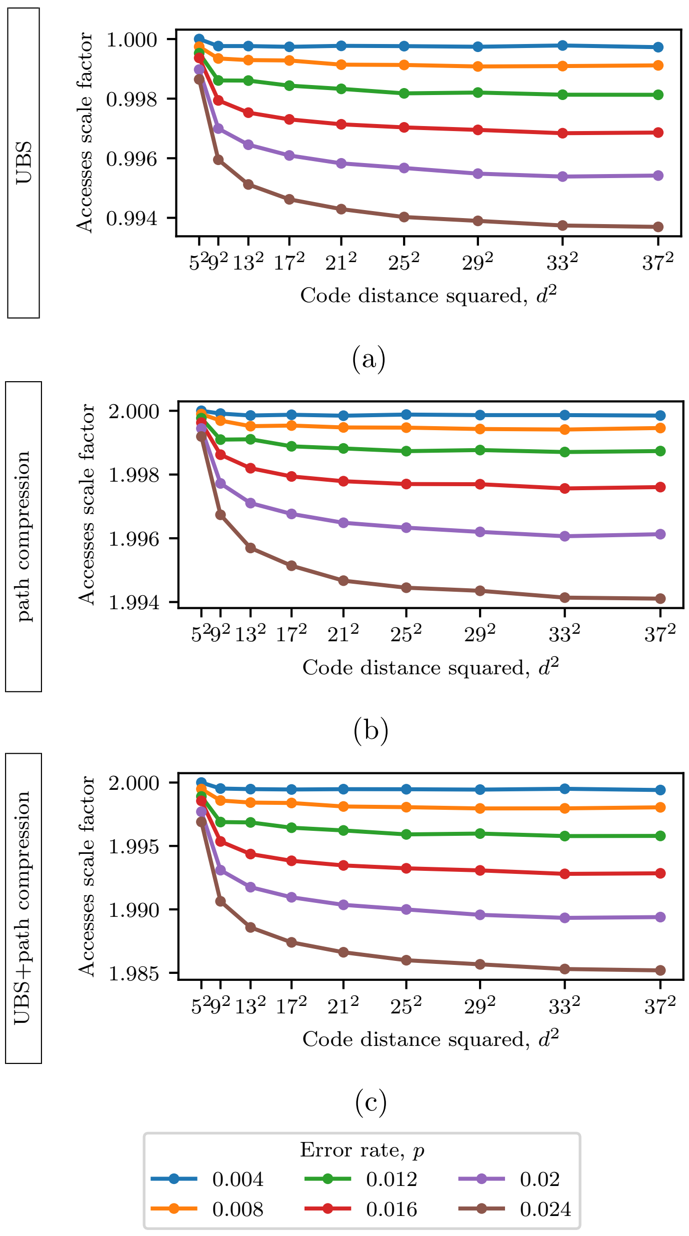
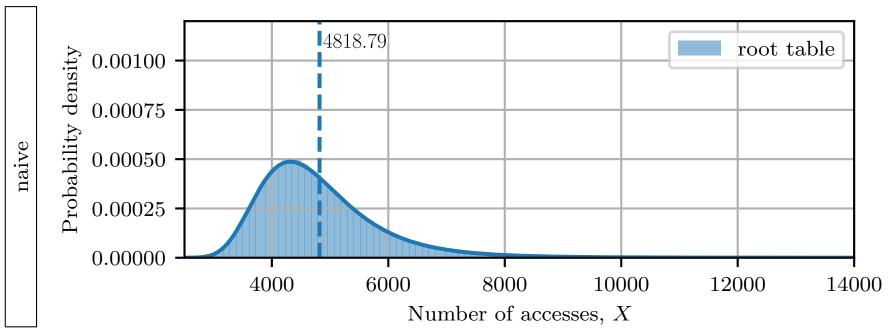
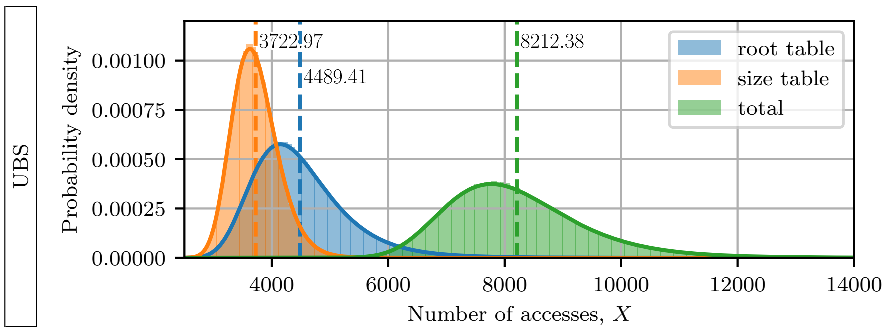


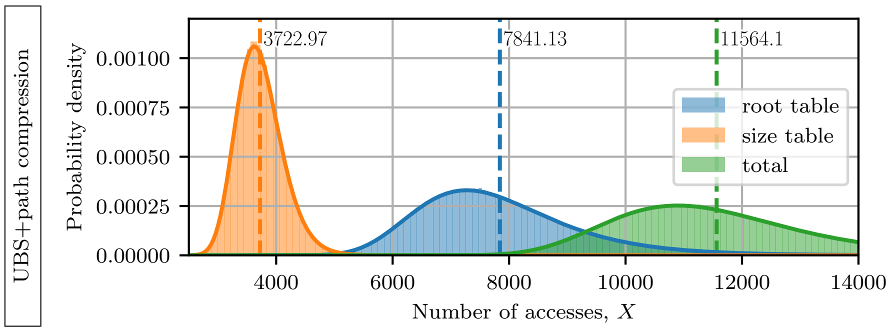
Figure 3.4: Monte Carlo simulation showing the scale factor in the number of root table accesses when applying (a) UBS, (b) path compression and (c) both UBS and path compression to the graph generator during surface-code decoding. Each point is the total number of reads over 10^5 runs, divided by the corresponding number from the naive implementation.



(a)



(b)



(c)

Figure 3.5: Histograms showing the number of root table and size table accesses required in each of 10^6 shots by the graph generator during surface-code decoding, with $d = 49$ and $p = 0.08$. (a) shows a naive approach, (b) shows UBS and (c) shows both UBS and path compression. The mean of each dataset is marked, highlighting how the mean total number of accesses is increased by both UBS and path compression. It transpires that an *alpha distribution* fits the number of accesses X particularly well, as shown by the bold contour lines, which is defined as $1/X$ being both normally distributed and nonnegative [100, p. 173]. This could possibly model hyperbolically-decaying boundary effects on a binomial distribution in the sparse regime.

sizes bound solely by population, not by the size of the lattice [101]. In other words, percolation theory can be applied to find a threshold for p below which the clusters formed by the union–find decoder are sparsely distributed and thus, for $d \to \infty$, the number of accesses in each merge operation converges down to its minimum (Figure 3.3). Percolation theory is introduced in more depth in Chapter 3.4.1, then used to develop this analytical model proving that the data structure indeed operates in the unsaturated regime in Chapter 3.4.2.

In proving that the data structure operates in the unsaturated regime, it is guaranteed that both UBS and path compression only worsen runtime at scale, because they require strictly more total table accesses on sparse sets, with their benefits only emerging on sets involved in multiple merge operations (hence why they benefit only the *amortised* complexity).

Second, the complexity analysis in Chapter 2.5.1 assumes the use of merge operations between two elements in the forest selected randomly and uniformly. In the context of the decoder, however, merge arguments are not uniformly distributed and are instead much more likely to contain at least one cluster root. This is due to the fact that the decoder iterates over odd and unconfined clusters to grow and therefore begins growth with preexisting knowledge of the root. In the simplest yet ubiquitous case, a cluster merges with an unpopulated vertex – with both arguments roots of their respective sets, no traversal at all is required. This nonuniform operation acts to naturally limit the tree heights formed without requiring explicit path compression.

3.4 Analytics

3.4.1 Percolation theory

In order to prove that the data structure acts in the unsaturated regime, we can turn to a mathematical framework known as *percolation theory.* This framework studies the addition of edges to sets of vertices and how clusters subsequently emerge [101].

Specifically, when considering a population rate p on some regular lattice (that is, the independent and uniform probability that a bond on the lattice is occupied/present), there emerges a distinct threshold p_c above which there tends to exist a single cluster which connects opposing boundaries of the lattice, i.e. a cluster which *percolates* the lattice. The existence of this state transition is a key phenomenon and this framework is used in a wide range of models such as solid-state mechanics (e.g. dielectric breakdown [102]), forest fires [103] and porous media (e.g. coal [104] and ground coffee [105]).

In this section, let us briefly familiarise ourselves with percolation theory from first principles in the literature and derive some expressions to clearly argue the expected behaviour of clusters on generalised lattices (including the square lattices relevant for Kitaev surface codes).

Consider a regular square lattice of vertices, where edges – or *bonds* – can be independently populated with a uniform probability p (Figure 3.6). The percolation rate – that is, the probability that randomly-populated edges connect two opposite boundaries of the lattice – naturally increases with p. Indeed, if plotting the percolation rate against p, a sigmoid curve will be observed. The sigmoid will steepen as the length of the lattice L increases, such that it converges on a step function for $L \to \infty$ (Figure 3.7). This state transition looks exactly like that which we saw previously in the modelling of logical error rates (Figure 2.1) which is no coincidence: this distinct threshold

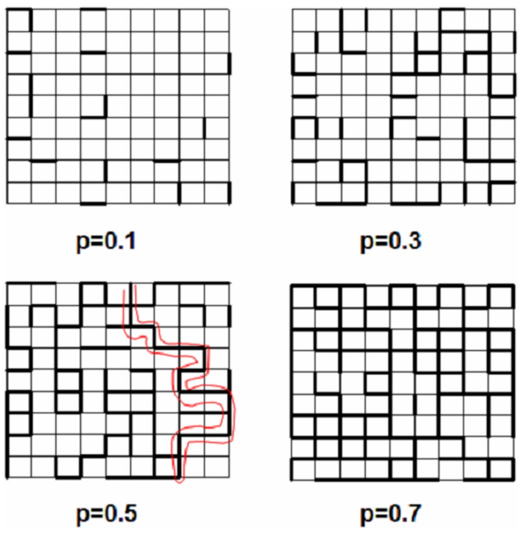


Figure 3.6: Illustration of bond percolation for varying population rates on a 2D square lattice. Asymptotically, clusters spanning opposite lattice boundaries emerge above the threshold, i.e. for $p > 0.5$. Image from [106].

which emerges in percolation theory is a direct analogue to the code threshold in error correction, especially when considering topological codes.

In one dimension, the percolation threshold $p_c = 1$ exactly, as every single bond must be populated in order to percolate in an infinitely long linear chain. Accordingly, exact solutions for the statistics of clusters in one dimension are known [101, p. 19].

In two or more dimensions, few exact solutions are known. It is possible but nontrivial to prove that $p_c = 0.5$ exactly on the 2D square lattice [107], as shown empirically in Figure 3.7. In general, thresholds for arbitrary structures tend to be only approximable and/or empirical.

Finally, it is important to understand the scaling of the average number of clusters and average cluster size in the sub-threshold vs super-threshold regimes. Take a hypercubic lattice with D dimensions (i.e. a square lattice for $D = 2$, a cubic lattice for $D = 3$ etc.) and length L. The size s of a cluster is the number of sites belonging to it. Above the threshold, for

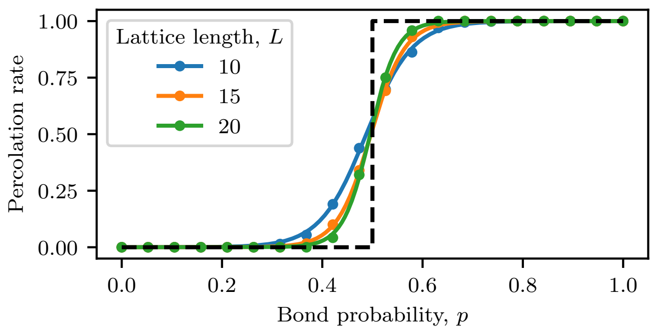


Figure 3.7: Monte Carlo simulation of bond percolation on a 2D square lattice, demonstrating logistic fits converging to a step function at $p_c = 0.5$ for $L \to \infty$ (dashed line). Each point is obtained from 500 samples.

$L \to \infty$, there is exactly one cluster, which percolates the lattice; therefore, the size of this cluster is bound directly by the number of lattice sites L^D. More generally, below the threshold, the *normalised cluster number* n_s is the expected number of clusters of s-size *per lattice site* [101, p. 20].

A general postulate for the scaling of the normalised cluster number, for a fixed rate p, is

$$n_s \propto e^{-s} \ , \tag{3.1}$$

i.e. clusters of increasingly large size are exponentially less likely to occur [101, p. 35]. As this represents the number of s-clusters per site, let us express the expected total number of s-clusters across the lattice as

$$N_s = L^D n_s \ , \tag{3.2}$$

and therefore the expected total number of clusters overall is the sum across all possible cluster sizes:

$$\bar{N} \approx \sum_{s=1}^{L^D} L^D e^{-s} \tag{3.3}$$

$$\approx L^D(e^{-1} + e^{-2} + \cdots + e^{-L^D}) \ . \tag{3.4}$$

So, we have shown that the mean total number of clusters sub-threshold scales linearly with L^D (more precisely, the scaling quickly converges upon linear for $L^D \to \infty$).

A correlation function $g(r)$ gives the probability that sites separated by a distance of r belong to the same cluster, yielding a *correlation length* ξ, which is an average distance between two sites within a cluster. The correlation length is related to p:

$$\xi \propto |p - p_c|^{-\nu} \ , \tag{3.5}$$

where ν is a critical exponent constant with lattice dimensionality[7], showing that ξ diverges as $p \to p_c$ [101, pp. 59–60]. When $p \ll p_c$, $L \gg \xi$, therefore the mean cluster size S scales according to

$$\begin{aligned} S &\propto \xi^{\frac{\gamma}{\nu}} && (3.6)\\ &\propto |p - p_c|^{-\gamma} \ , && (3.7) \end{aligned}$$

where γ is another critical exponent[8] [101, p. 37] [108, p. 30]. Therefore, the mean cluster size scales only with p.

At the threshold, however, $L \not\gg \xi$, so instead

$$S \propto L^{\frac{\gamma}{\nu}} \ , \tag{3.8}$$

such that the mean cluster size instead scales with L [108, p. 30] (i.e. the single percolating cluster is bound definitionally by the size of the lattice, as previously stated).

[7]For $D = 2$, $\nu = 4/3$ exactly, and for $D = 3$, $\nu \approx 0.88$ [101, Table 2]. It is strongly suggested that the values of critical exponents depend only on the dimensionality and not on any specific lattice structure beyond that [101, p. 51].

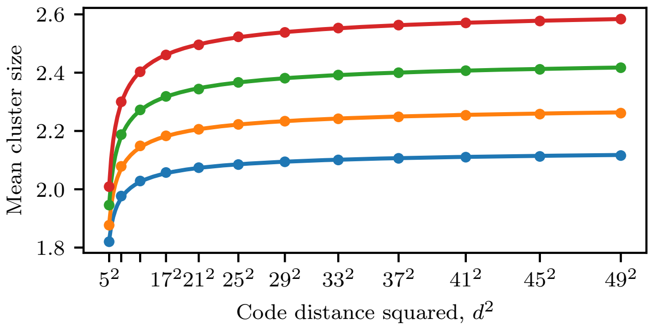
[8]For $D = 2$, $\gamma = 43/18$ exactly, and for $D = 3$, $\gamma \approx 1.80$ [101, Table 2].

3.4.2 Erasure percolation

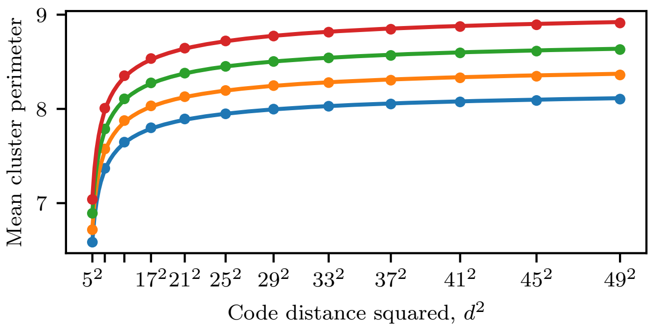
Consider the erasure clusters formed by syndrome validation. Boundary effects at small code distance d limit cluster sizes, preventing a truly constant relation with d. The prevalence of clusters incident to the boundary scales as $1/d$ (via a trivial perimeter-over-area argument), tending to zero as d increases. Thus, the mean cluster size follows a plot of $A - B/d$, where A is dependent on the error rate p (Figure 3.8). Let us also define the *perimeter* of clusters as the length of the list of boundary sites – this follows similar scaling to the size. The mean total number of clusters, however, scales linearly with d^2 (or d^3 in 3D) assuming a uniform error distribution. This patently agrees with the reasoning from percolation theory for sub-threshold behaviour presented in the previous section.

The time taken in each find operation depends on the height of the trees formed in the forest. In the sparse regime, the tree heights are – as with cluster size – invariant with d barring hyperbolically-decaying boundary effects.

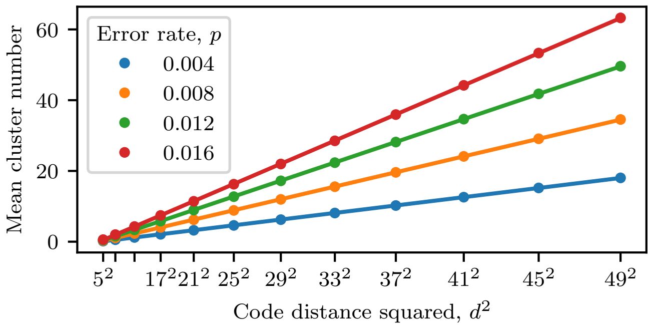
If it can be shown that the erasure clusters exist in a sparse regime (as is implied by the trends in Figure 3.8), then it is clear that the data structure operates in an unsaturated regime, with average cluster sizes depending only on p and invariant with d. Clusters are grown from defects, which have a nontrivial population distribution arising from p. *Syndrome percolation* is defined in [72] as the existence a path of neighbouring defects between opposite boundaries of the lattice in the initial syndrome. However, we wish to more specifically consider the size of erasure trees formed at the conclusion of syndrome validation. Let us instead define *erasure percolation* as the existence of an erasure tree spanning between opposite boundaries of the lattice. This is a more relaxed definition than syndrome percolation, as it can arise, for example, from fewer defects positioned equidistantly across a dimension, requiring multiple growth iterations.



(a)

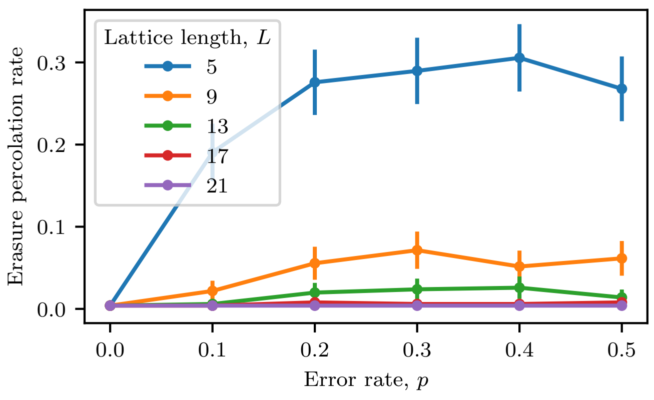


(b)

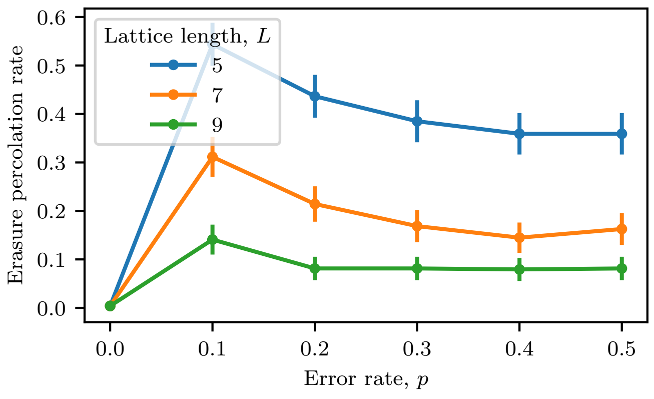


(c)

Figure 3.8: Sub-threshold, the mean cluster size (a) and perimeter (b) is bound solely by p, when accounting for hyperbolically-decaying boundary effects. The mean number of clusters (c) scales linearly with d^2. Each point is obtained from 10^5 runs.



(a)



(b)

Figure 3.9: Erasure percolation on the planar code with 2D independent (a) and 3D phenomenological (b) error models ($q = p$). Instead of sigmoids showing a threshold value, percolation rates strictly tend to zero for $L \to \infty$. Each point is obtained from 500 samples, with error bars using the Wilson score with two standard deviations (∼95% confidence interval) [109].

Therefore, not only is there abstraction from the distribution of qubit errors to the distribution of defects in the syndrome, but to the distribution of all vertices included in grown clusters. In order to model the erasure percolation threshold at higher dimensions without requiring full implementation of syndrome validation for $D > 2$, I instead find a minimum-weight perfect matching on the initial syndrome graph and estimate the erasure trees which would be formed by including all vertices within distance $\lfloor w/2 \rfloor$ from each defect in a matching, where w is the weight of the matching. In other words, as clusters grow uniformly in all directions, we assume that clusters resemble hyperspheres meeting at the middle with radius $\lfloor w/2 \rfloor$.

If an erasure percolation threshold exists and is greater than the code threshold, then it follows that the decoder will always operate the data structure in the unsaturated regime (for relevant usage, as there is little motivation to consider behaviour above the code threshold). Figure 3.9 shows the results of a Monte Carlo simulation of syndrome validation on the 2D planar code with both 2D independent and 3D phenomenological error models.[9] One might expect sigmoid plots converging on a step function at $p = p_c$ for $L \to \infty$, as in Figure 3.7.

Instead, we see that the erasure percolation rate converges to zero for all p for $L \to \infty$. This indicates that an erasure percolation threshold does not exist at all for any mode of operation of the union–find decoder; that is, the decoder at scale will asymptotically never experience syndromes yielding erasure percolation. Intuitively, this is due to the ability of defects to merge across periodic boundary conditions in the toric code (or, equivalently, to merge to the boundary in the planar code); for example, two defects separated by an axis-aligned error string of weight $w > L/2$ will merge across the

[9] Percolation is detected by adding shared neighbouring nodes to all opposite boundary nodes and using Dijkstra's algorithm to determine if an uninterrupted path exists between them.

boundary, such that erasure percolation will only occur if $w = L/2$ exactly.

As the average size of erasure clusters will never be bound by the size of the lattice (ignoring boundary effects on small lattices), the disjoint-set data structure operates strictly in the unsaturated regime, confirming that both union-by-size and path compression are asymptotically irrelevant (and indeed counterproductive).

3.5 Conclusion

In Chapter 2.5.1, we saw how existing literature suggests that the worst-case time complexity of a naive implementation of the union–find decoder (i.e. forgoing union-by-size and path compression) would be $O(n^2)$, as opposed to $O(n\alpha(n))$ when including the optimisations. However, the numerical data which I presented in this chapter does not support this. The derivation of that scaling assumes that the complexity of each find operation is $O(n)$ in the naive case, which is true only if the size of clusters varies with the lattice size. These results demonstrate that the nature of the decoder algorithm leads to a strictly sparse (i.e. never-percolating) regime of erasure clusters. Therefore, the number of clusters may be linear in n, but their average size depends only on p, instead yielding an overall complexity of just $O(n)$. I demonstrated this absence of a percolation threshold in both 2D independent and 3D phenomenological models for the planar surface code. In other words, the probability of the worst-case complexity scenario (of erasure clusters percolating) is suppressed to zero for $d \to \infty$.

Therefore, one can suggest that both union-by-size and path compression may be comfortably omitted from implementations of the union–find decoder without paying a penalty, and indeed memory usage and runtime are actually improved. In the example of the AFS architecture in [81], this would involve omitting the size tables and traversal registers, as well as their associated logic. The size table is stated to be the single most memory-consuming component (e.g. 54.9 KB out of a total 133 KB for $d = 25$), so forgoing union-by-size would yield significant gains in a memory-critical (e.g. cryogenic) environment.

Finally, I suggested an alternative implementation of syndrome validation based upon the AFS architecture, which stores boundary lists (instead of recalculating them in each iteration) whilst minimising overhead via double-buffering with a *new edge stack* (NES). Depending on the exact method with which it is contrasted, this may save computational time over the course of decoding at the slight expense of storage and merging.

At the time of writing this thesis, these results have been cited and used to justify forgoing the aforementioned optimisations in implementations of the UF decoder [14], to quote the more accurate $O(n)$ time complexity of the UF decoder [15], and to reason with cluster statistics in order to develop novel approaches such as the *breadth-first graph traversal union–find decoder* [22] and the *local clustering decoder* [23]. These results have also been cited by a number of works to more generally help justify the relevance and performance of the UF decoder and its properties [16–20] and as an example of the application of percolation theory (from the perspective of nanotechnological structures and effects) to quantum technology [21].

Chapter 4

LDPC-code decoding at scale

4.1 Motivation and methods

In Chapter 2.6, quantum low-density parity-check (qLDPC) codes were introduced as a wide-ranging class of codes (including, but not limited to, surface codes) satisfying a sparse structure of stabiliser generators. Such generalised codes constitute an increasingly popular area of research. Classically, it is well-understood that LDPC codes can approach optimum performance under many noise channels [110], yet are computationally efficient due to their sparsity: their locally-tree-like Tanner graphs make the linear-time belief propagation (BP) decoder perform very effectively [92, 93]. This motivation extends to qLDPC codes: it has been proven that 'good' qLDPC codes exist, by which we mean codes with logical qubits k and distance d scaling no slower than linearly with n [34].

For qLDPC codes, the BP decoder has failure modes caused by solution degeneracy, so it is not as effective alone as with classical LDPC codes. Some of the leading alternative approaches to decoding for qLDPC codes were introduced in Chapter 2.7. BP–OSD uses cubic-time Gaussian elimination to solve the decoding equation, informed by the posterior probabilities given by

BP. Alternatively, the union–find (UF) decoder can be generalised to Tanner graphs, repeatedly performing Gaussian elimination on an increasingly large neighbourhood of the decoding graph.

In generalised UF, the key problems arise from the two black-box subroutines: syndrome validation and solution generation. Both of these are computationally-easy problems in the surface code case, with the former consisting of simple parity checks and the latter consisting of the peeling algorithm. Neither of these generalise readily to LDPC codes, in which the decoding graph becomes a hypergraph.

To understand why, consider a hypothetical generalised peeling decoder. In the graphlike case, peeling is commenced from *leaves*, i.e. edges connected to the erasure forest through only one endpoint, and vertices are flipped in and out of membership of σ, effectively moving defects together to annihilate [73]. In the hypergraph case, however, it cannot be assumed that leaves are unambiguously defined and that parities trivially annihilate in this way. Instead, consider attempting to directly find an optimal solution on the erasure without peeling. Whilst minimum graph matching can be performed in polynomial time (see Chapter 2.4.2), minimum hypergraph matching is an NP-hard problem which, regardless, no longer guarantees a valid solution, as is the more general problem of minimum set cover [111]. We are left with the option of a greedy approach, although a major problem is the presence of invalid choices, i.e. selecting qubits which do not appear in any valid solution. Backtracking would allow the decoder to undo invalid choices, although this is equivalent to an informed depth-first search on a combinatorial space which also runs in exponential time [74]. Thus, Gaussian elimination appears to be necessary for computational tractability in the general case.

Two separate approaches to improving a generalised UF decoder which I explored are presented hereinafter: improving Gaussian elimination and avoiding it entirely. In Chapter 4.2, I present a novel online variant of Gaussian elimination, with a complexity analysis showing improved runtime in the general case and simulations demonstrating its performance on various qLDPC codes. I constructed these simulations in C++ using the Eigen library[1] for optimised matrix manipulation, with custom $\mathbb{Z}_2$ arithmetic (i.e. XOR row, dot-product and matrix-product operations). In Chapter 4.3, I present notes on techniques for avoiding Gaussian elimination via inspiration from single-shot codes.

4.2 Improving Gaussian elimination

4.2.1 Gaussian elimination

As stated in Equation 2.7, the decoding problem ultimately deconstructs to solving a system of linear equations of the form

$$A\mathbf{x} = \mathbf{b} \ , \tag{4.1}$$

where A is the $m \times n$ coefficient matrix representing m equations and n variables. Gaussian elimination is an algorithm which takes a matrix into row echelon form (REF), which is defined as a matrix in which the first nonzero entry of each row, known as the *pivot*, is to the right of the pivots of all rows above. Rows containing only zeroes are relegated to the bottom of the matrix.

The Gaussian elimination algorithm takes a matrix into REF by repeatedly applying three elementary row operations:

1. Swapping two rows;

[1]`https://eigen.tuxfamily.org`

2. Multiplying a row by a constant;

3. Subtracting a multiple of a row from another row.

These operations, and thus the algorithm itself, can be defined for fields beyond real numbers. Conveniently for our context, the $\mathbb{Z}_2$ field (i.e. binary with addition/subtraction modulo 2) is arguably the simplest possible variant of Gaussian elimination. Subtraction modulo 2 is merely the XOR operation and the only possible nonzero constant is 1, simplifying Operation 3 and removing Operation 2 entirely. Algorithm 1 lists pseudocode for an explicit implementation of Gaussian elimination in $\mathbb{Z}_2$.

The row echelon form reveals useful information about the matrix and its dimensionality. For example, the *rank* of a matrix – defined as the number of linearly independent rows (or, equivalently, columns) – is trivially the number of nonzero rows in the REF.

To solve a system of the form in Equation 4.1, we first define the *augmented* matrix $A|\mathbf{b}$ as the matrix A with the vector $\mathbf{b}$ appended as an additional column. As per the Rouché–Capelli theorem, the system is *inconsistent*, i.e. has no solutions, if $\text{rank}(A|\mathbf{b}) > \text{rank}(A)$, but it is *consistent*, i.e. has one or more solutions, if $\text{rank}(A|\mathbf{b}) = \text{rank}(A)$ [112]. Both of these ranks can be calculated by performing Gaussian elimination on $A|\mathbf{b}$.

If the system is consistent, then a solution(s) can be obtained by *back-substitution*: the lowermost equation in REF contains only one unknown and so is trivially solvable, which is then substituted into the equation above it such that it too contains only one unknown, and so on. If $\text{rank}(A|\mathbf{b}) = \text{rank}(A) = r$ and $n = r$, then the system is *determined* and there exists a unique solution, but if $n > r$, then it is *underdetermined* and there exist infinitely many solutions generated by $n - r$ free variables (or, in the case of $\mathbb{Z}_2$, a finite yet exponentially large number of solutions).

Algorithm 1 Gaussian elimination in $\mathbb{Z}_2$

```
Input: $m \times n$ matrix $A$
Output: $m \times n$ matrix $A$ in row echelon form
 1: $r \leftarrow 0,\ c \leftarrow 0$
 2: while $r < m$ and $c < n$ do
 3:     $i \leftarrow r$                              ▷ Find next row with a one in column $c$
 4:     while $i < m$ and $A[i, c] = 0$ do
 5:         $i \leftarrow i + 1$
 6:     end while
 7:     if $i = m$ then               ▷ If column had only zeroes left, move on to next column
 8:         $c \leftarrow c + 1$
 9:     else
10:         Swap $A$.row[$i$] and $A$.row[$r$]
11:         $i \leftarrow r + 1$         ▷ Eliminate ones beneath the pivot in this column by XORing rows
12:         while $i < m$ do
13:             if $A[i, c] = 1$ then
14:                 $A$.row[$i$] $\leftarrow$ $A$.row[$i$] $\oplus$ $A$.row[$r$]
15:             end if
16:             $i \leftarrow i + 1$
17:         end while
18:         $r \leftarrow r + 1,\ c \leftarrow c + 1$                   ▷ Move on to next row and column
19:     end if
20: end while
```

Algorithm 2 LUP decomposition in $\mathbb{Z}_2$

```
Input: $m \times n$ matrix $A$
Output: Matrices $L, U, P$ satisfying $PA = LU$, where $L$ and $P$ are $m \times m$, and $U$ is $m \times n$
    in row echelon form
 1: $L \leftarrow 0_m$
 2: $U \leftarrow A$
 3: $P \leftarrow I_m$
 4: $r \leftarrow 0,\ c \leftarrow 0$
 5: while $r < m$ and $c < n$ do
 6:     $i \leftarrow r$                    ▷ Find next row with a one in column $c$
 7:     while $i < m$ and $U[i, c] = 0$ do
 8:         $i \leftarrow i + 1$
 9:     end while
10:     if $i = m$ then           ▷ If column had only zeroes left, move on to next column
11:         $c \leftarrow c + 1$
12:     else
13:         Swap $L$.row[$i$] and $L$.row[$r$]
14:         Swap $U$.row[$i$] and $U$.row[$r$]
15:         Swap $P$.row[$i$] and $P$.row[$r$]
16:         $i \leftarrow r + 1$   ▷ Eliminate ones beneath the pivot in this column by XORing rows
17:         while $i < m$ do
18:             if $U[i, c] = 1$ then
19:                 $U$.row[$i$] $\leftarrow$ $U$.row[$i$] $\oplus$ $U$.row[$r$]
20:                 $L[i, r] \leftarrow 1$
21:             end if
22:             $i \leftarrow i + 1$
23:         end while
24:         $r \leftarrow r + 1,\ c \leftarrow c + 1$          ▷ Move on to next row and column
25:     end if
26: end while
27: $L \leftarrow L + I_m$                            ▷ Set leading diagonal to ones[a]
```

[a] $PA = LU$ is satisfied when L is unit-triangular, i.e. has ones along its leading diagonal. For online update, this would require repeatedly subtracting/adding I at the start/end of every iteration. This is merely a mathematical constraint rather than storing meaningful information, therefore it is most efficient to skip this entirely for the online variant.

4.2.2 Online Gaussian elimination

Recall that the generalised union–find decoder as given in [41] delegates two black-box subroutines: syndrome validation and solution generation. To understand this problem clearly, let us first define H' as the 'reduced' parity-check matrix H, filtered to contain only rows and columns representing the checks and variables, respectively, currently contained in $\mathrm{Int}(\mathcal{E})$. Gaussian elimination solves this problem in the general case: one or more solutions exist, and thus clusters can stop growing, when $\mathrm{rank}(H'|\sigma) = \mathrm{rank}(H') = r$, yielding a solution generator with $n - r$ free variables, as we can expect degeneracy (i.e. underdetermination) in the general case.[2]

The decoder is initialised with $\mathcal{E} = \sigma$ and neighbouring nodes are iteratively added until a solution(s) exists. By definition, $\rho_{\mathrm{cov}}(\sigma)$ growth steps are required, meaning Gaussian elimination is performed this many times on a matrix $H'|\sigma$ of strictly increasing size.

Instead, I present an online variant of Gaussian elimination which removes redundant computational work between growth steps. An *online algorithm* is one in which a valid solution is maintained as new input is obtained over time, i.e. the final problem data is not required in whole to commence work [42].

Let H_0 be an augmented matrix in REF from a previous growth step and H_1 be the same matrix with additional rows and columns appended. The new data must be brought up-to-date with decisions made in previous growth steps. Decisions made by the rounds of Gaussian elimination are recorded by maintaining an LUP decomposition. This is equivalent to Gaussian elimination, except the elementary row operations are explicitly represented by

[2]Recall that the rank of each matrix block is calculated simply as the number of nonzero rows in the block after Gaussian elimination.

a matrix factorisation of the form

$$PA = LU \ , \tag{4.2}$$

where A is the original matrix and U is the matrix in row echelon form (upper-triangular). Swapped rows are recorded by the permutation matrix P and row subtractions are recorded by the lower-triangular matrix L. Algorithm 2 shows how this factorisation is equivalent to performing Gaussian elimination (Algorithm 1) whilst recording decisions.

Firstly, by maintaining the matrix factors P and L, previous row operations can be performed on new rows and columns when they are added to the system. Secondly, by definition of the decoder, H_0 represents an inconsistent system, suggesting the existence of 'missing' pivots (i.e. zeroes) from the leading diagonal of U. These positions are candidates for pivots to be found within the newly-added rows. Commencing from the first missing pivot position, Gaussian elimination is performed, except that only the newly-added rows need to be searched through and, by extension, subtracted from. This is justified as H_0 is already in REF and is thus upper-triangular, such that only zeroes can be present beneath the leading diagonal in the old rows. In this way, the LUP decomposition is updated in each growth step to return U into REF given the new rows and columns. Algorithm 3 lists pseudocode for this online LUP decomposition update performed in each growth step.

Algorithm 3 Online LUP decomposition update in $\mathbb{Z}_2$

```
Input: Matrices $L, U, P$ where all have $r_{\text{new}}$ new rows and $U$ has $c_{\text{new}}$ new columns
Output: Updated matrices $L, U, P$ satisfying $PA = LU$, with $U$ returned to row echelon form
 1: $U$[old rows, new cols] $\leftarrow PU$[old rows, new cols]    ▷ Swap new cols in old rows from old $P$[a]
 2: $r \leftarrow 1$    ▷ XOR new cols according to $L$
 3: while $r < r_{\text{old}}$ do
 4:     $c \leftarrow 0$
 5:     while $c < r$ do
 6:         if $L[r, c] = 1$ then
 7:             $U[r, \text{new cols}] \leftarrow U[r, \text{new cols}] \oplus U[c, \text{new cols}]$
 8:         end if
 9:         $c \leftarrow c + 1$
10:     end while
11:     $r \leftarrow r + 1$
12: end while
13: $r, c \leftarrow 0$    ▷ Recommence Gaussian elimination procedure
14: while $r < m$ and $c < n$ do
15:     $i \leftarrow r$    ▷ Find next row with a one in column $c$
16:     if $U[i, c] = 0$ then    ▷ (skip to new rows if still in old columns)
17:         if $c < c_{\text{old}}$ then
18:             $i \leftarrow \max(r_{\text{old}}, r + 1)$
19:         else
20:             $i \leftarrow r + 1$
21:         end if
22:         while $i < m$ and $U[i, c] = 0$ do
23:             $i \leftarrow i + 1$
24:         end while
25:     end if
26:     if $i = m$ then    ▷ If column had only zeroes left, move on to next column
27:         $c \leftarrow c + 1$
28:     else
29:         Swap $L$.row[$i$] and $L$.row[$r$]
30:         Swap $U$.row[$i$] and $U$.row[$r$]
31:         Swap $P$.row[$i$] and $P$.row[$r$]
32:         if $c < c_{\text{old}}$ then    ▷ Eliminate ones beneath the pivot in this column by XORing rows
33:             $i \leftarrow \max(r_{\text{old}}, r + 1)$    ▷ (skip to new rows if still in old columns)
34:         else
35:             $i \leftarrow r + 1$
36:         end if
37:         while $i < m$ do
38:             if $U[i, c] = 1$ then
39:                 $U$.row[$i$] $\leftarrow U$.row[$i$] $\oplus U$.row[$r$]
40:                 $L[i, r] \leftarrow 1$
41:             end if
42:             $i \leftarrow i + 1$
43:         end while
44:         $r \leftarrow r + 1,\ c \leftarrow c + 1$    ▷ Move on to next row and column
45:     end if
46: end while
```

[a]Generalised slicing notation where $U[a, b]$ is the intersection of rows and columns given by the tuples a and b.

4.2.3 Complexity analysis

For a square $n \times n$ matrix (as arises with an exactly-determined system of equations) the time complexity of Gaussian elimination is $O(n^3)$. More generally, for an $m \times n$ (i.e. rectangular) matrix, time complexity is $O(mn \min(m, n))$, a.k.a. *big-times-small-squared* [113].

It is straightforward to see how the cubic complexity arises. For each of the n pivots, the row is subtracted from $O(n)$ other rows, which each contain n elements. No order of complexity is added by obtaining an LUP decomposition, as this amounts to merely logging the operations which have been performed (Algorithm 2). Finally, back-substitution is trivially $O(n^2)$.

In contrast, the online LUP update (Algorithm 3), for each of the $O(n)$ outstanding pivot positions, needs only search through and subtract from the newly-added rows. Therefore, it is asymptotically equivalent to performing a single LUP decomposition on the final-sized system.

Once one or more solutions exist, the erasure $\mathcal{E}$ stops growing and the final reduced parity-check matrix H' has dimensions $r \times c$ where $r + c = |\mathcal{E}|$. The number of XOR operations required by Gaussian elimination on H' – and thus by the online decoder – is $O(|\mathcal{E}|^3)$. In the worst case, $|\mathcal{E}| = n$, taking n to be the total of both qubits and checks in the code to simplify analysis. This gives the online decoder a worst-case complexity of $O(n^3)$.

This can be contrasted with the original offline description of the decoder in [41]. Gaussian elimination is performed anew for each of the $\rho_{\text{cov}}(\sigma)$ growth steps (denoted ρ for concision), in each of which the size of the erasure is increased by a factor of δ, where δ is the maximum degree of the Tanner graph. In other words, the size of the erasure after the ith iteration $|\mathcal{E}|_i \approx \delta|\mathcal{E}|_{i-1}$. Reversing this logic, with a final erasure size n, the penultimate erasure size is approximately n/δ. The number of operations required in the worst case can thus be approximated by the following sum (with a symbolic expansion on the right-hand side):

$$\sum_{i=0}^{\rho} \left(\frac{n}{\delta^i}\right)^3 = \frac{n^3\delta^{-3\rho}(\delta^{3\rho+3}-1)}{\delta^3-1} \ , \tag{4.3}$$

from which it follows that the offline decoder also has a worst-case complexity of $O(n^3)$. However, it is apparent that the online decoder has significantly reduced overhead; indeed, the number of operations skipped by the online variant is therefore approximately

$$\sum_{i=0}^{\rho-1} \left(\frac{n}{\delta^i}\right)^3 = \frac{n^3\delta^{3-3\rho}(\delta^{3\rho}-1)}{\delta^3-1} \ , \tag{4.4}$$

which is itself $O(n^3)$.

This analysis can be refined in the case of a 'well-behaved' code, defined in Chapter 2.7.2 as the property identified in [41], where $\rho \leq C \log |\mathbf{x}|$ for all $|\mathbf{x}| < w$ for some constants C, w. In this case, the decoder corrects all errors where $|\mathbf{x}| < \min(w, Ad^\alpha)$, where A and α are constants which depend on the degree of the Tanner graph (see Equation 2.68), d is the code distance, and the erasure formed is bounded by

$$|\mathcal{E}| \leq \delta^2 |\mathbf{x}| \cdot \delta^\rho \tag{4.5}$$

$$\leq \delta^2 |\mathbf{x}|^{1+C\log\delta} \ . \tag{4.6}$$

The number of operations performed by the online decoder is approximately

$$|\mathcal{E}|^3 \leq \delta^6 |\mathbf{x}|^{3+3C\log\delta} \ , \tag{4.7}$$

and thus the number performed by the offline decoder is approximately

$$(\delta^6|\mathbf{x}|^3)(1 + |\mathbf{x}|^3 + \cdots + |\mathbf{x}|^{3C\log\delta}) \ , \tag{4.8}$$

where the upper bound for $|\mathbf{x}|$ varies with code-dependent properties. In this well-behaved regime, it is bounded in terms of the code distance d by Ad^{α}, defined as

$$Ad^{\alpha} = \left(\frac{d}{2\delta^2}\right)^{\frac{1}{1+C\log\delta}} . \tag{4.9}$$

This suggests an upper bound for the size of the erasure formed:

$$|\mathcal{E}| \leq \delta^2 \cdot \frac{d}{2\delta^2} = \frac{d}{2} , \tag{4.10}$$

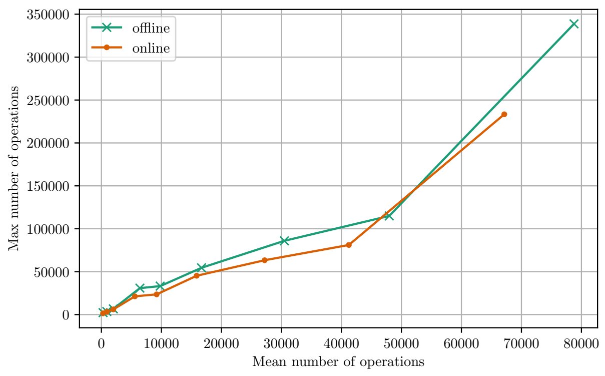
which implies that the online decoder has a complexity of $O(d^3)$ and that the number of operations skipped in contrast to the offline decoder is approximately

$$\begin{aligned} (\delta^2|\mathbf{x}|^{1+C\log\delta-1})^3 &= \left(\frac{d}{2|\mathbf{x}|}\right)^3 \qquad (4.11)\\ &= \left(\frac{d}{2Ad^{\alpha}}\right)^3 . \qquad (4.12) \end{aligned}$$

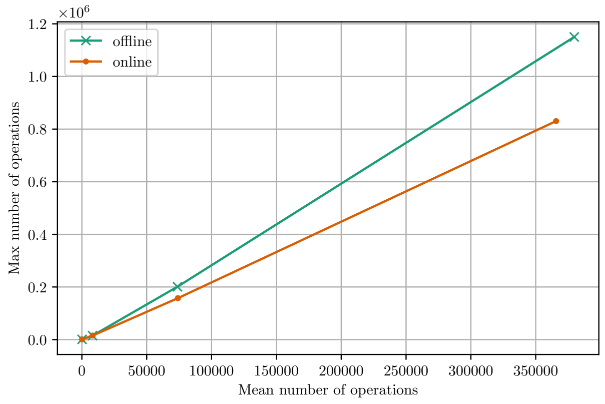
4.2.4 Simulation

The complexity analysis above appears to show a well-defined speed-up for the online decoder versus the offline decoder. However, this relies on certain assumptions as to the exact behaviour of Gaussian elimination on the decoding instances. To illustrate this, I performed Monte Carlo simulations on three different code constructions: the 2D toric code, 3D toric code and 2D 6.6.6 colour code with periodic boundary conditions. Note that it was shown in [41] that the toric codes are well-behaved as per the definition above, but this does not trivially extend to the colour code despite the structural similarity of these code families.

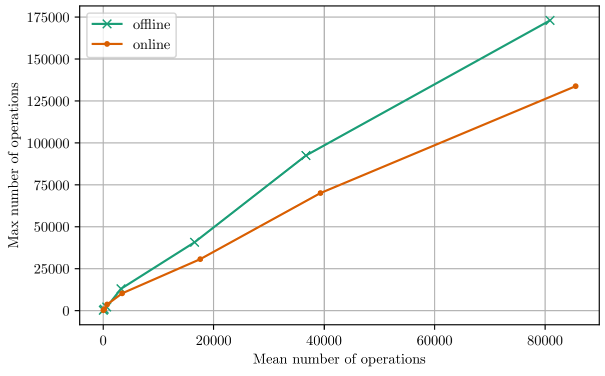
Figure 4.1 shows the mean and maximum number of XOR operations performed by both online and offline decoders on code instances of increasing



(a) 2D toric code with $L = (7, 9, \ldots, 23)$.

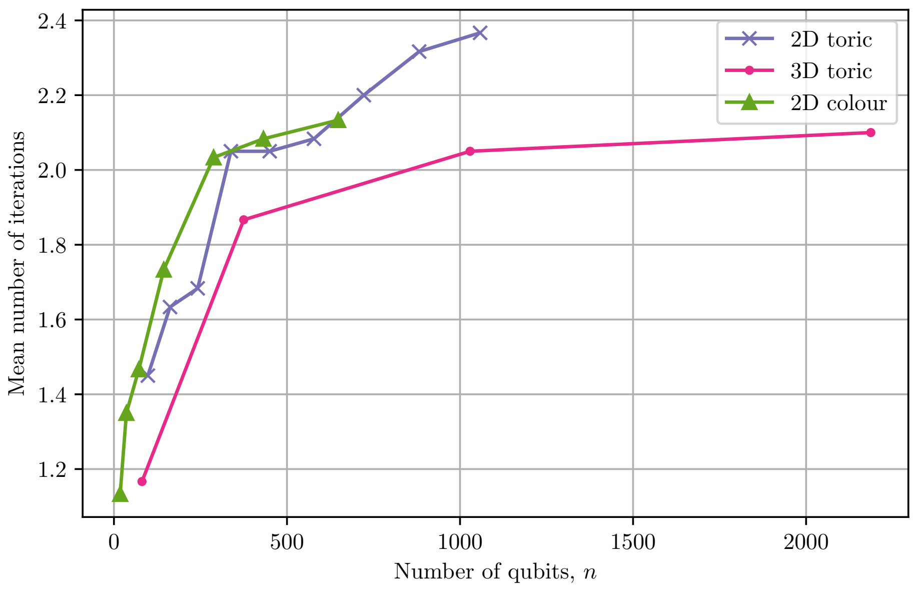


(b) 3D toric code with $L = (3, 5, \ldots, 11)$.

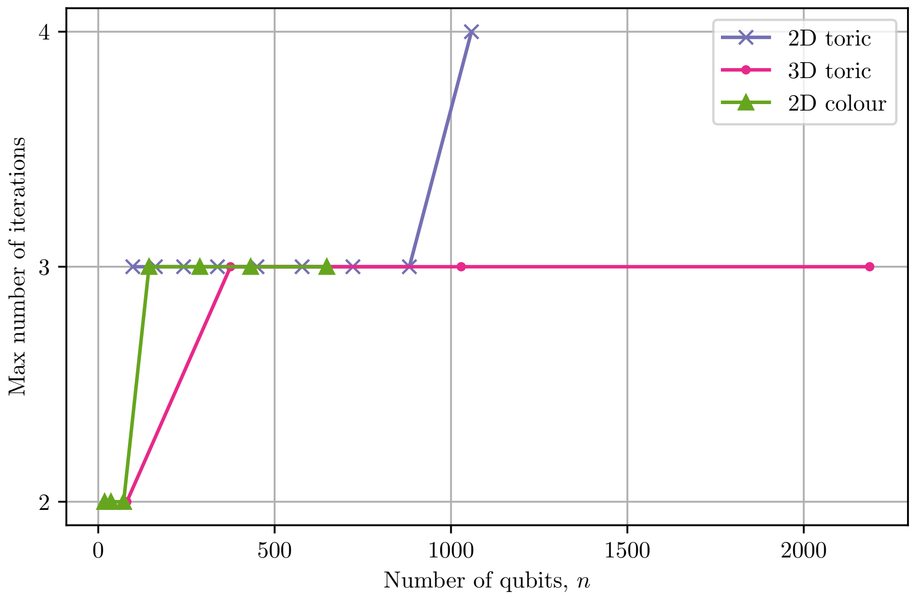


(c) 2D colour code with $n = (18, 36, 72, 144, 288, 432, 648)$.

Figure 4.1: Parametric plots showing the maximum and mean number of operations performed by offline and online decoders for increasing system size. Data for three different codes are shown: the 2D toric code (a), the 3D toric code (b), and the 2D 6.6.6 colour code (c), all generated using $p = 0.05$ and 60 shots per point.



(a)



(b)

Figure 4.2: Plots showing the mean (a) and maximum (b) number of iterations performed by the same decoding instances as in Figure 4.1. The numbers of qubits for the 2D and 3D toric codes are obtained via $2L^2$ and $3L^3$, respectively. These statistics apply to both offline and online versions as that implementation detail does not affect the covering radius.

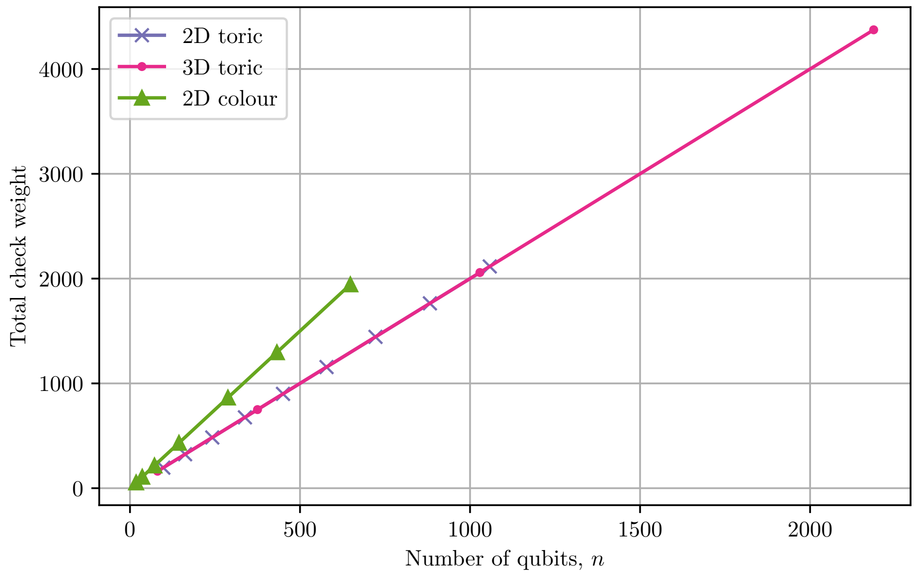


Figure 4.3: Total parity-check matrix weight for the three codes studied above.

size. Specifically, decoding of X stabilizer measurements is demonstrated, under independent noise with $p = 0.05$ with 60 shots per data point. Firstly, it is clear that across all three codes, the max number of operations is reduced by the online variant for increasing n, i.e. the worst-case complexity is invariably improved. This is broadly consistent with the complexity analysis above, which suggests a polynomially-scaling reduction in the number of operations.

Secondly, the behaviour of the mean number of operations, i.e. the average-case complexity, is more nuanced. The mean number of operations also clearly improves for the 2D toric code, although this improvement is more slight for the 3D toric code – meanwhile, the online decoder actually performs worse on this metric for the 2D colour code. To understand why, the covering radius was empirically determined for these same instances by recording the number of growth iterations, as shown in Figure 4.2. By the nature of the online update, it is intuitive that a greater improvement should correlate with a higher number of iterations; that is, we expect the improvement to be starker on codes with a higher covering radius. The 2D toric code demonstrates the highest covering radius for increasing n, so it comes as no surprise that it should see the greatest improvement in average-case complexity. Whilst the complexity analysis suggests that the online variant should perform strictly faster regardless of covering radius, the reality is that retaining information between growth iterations may lead to suboptimal choices in pivot selection. The Gaussian elimination procedure has fewer rows (and thus pivots) to choose from at each stage on the smaller systems of earlier cluster growth cycles, such that these decisions are based on less information of the final system when compared to the equivalent final offline procedure.[3] This can subtly increase the number of operations required at later growth cycles when filling missing pivots; the effect is dwarfed by the asymptotic improvement for higher covering radii, but can be significant for lower covering radii.

Finally, however, we see that the mean covering radius for the 2D colour code begins to converge somewhere between that of the other two codes, and yet it displays the worst performance for the online decoder. This can be attributed to the fact that, despite the number of iterations averaging between those of the toric codes, the mean number of operations is higher compared to the toric codes for similar values of n. This is demonstrated more clearly in Figure 4.3, which shows the total weight of the parity-check matrices (i.e. the total number of ones) for each code with increasing n. The 2D colour code outpaces both toric codes, demanding a greater number of operations for a similar covering radius; we can thus expect any suboptimal choices made by the online decoder to have a greater impact on its performance. This,

[3]More literally, the online procedure may skip over all-zero columns in search of a pivot candidate on earlier cycles, despite the fact that new rows from later cycles may have had a one in that column.

however, is promising, as it suggests that this technique performs better for sparser parity-check matrices, which more closely represent the LDPC codes it is intended for.

4.3 Avoiding Gaussian elimination

4.3.1 Metacheck validation

One of the key problems solved by Gaussian elimination is syndrome validation: knowing when a solution exists and thus when to stop growing clusters. In the surface code case, due to parities on a graphlike structure, a solution exists when a cluster supports an odd number of defects. In the LDPC case, it appears that no such trivial arguments can be made in the general case.

In Chapter 2.3.2, single-shot codes were introduced, an approach which can involve the use of metachecks to detect noisy syndrome measurements. In this section, I investigate the possibility of similarly using metachecks to determine when a syndrome becomes valid in order to stop growing clusters.

An $[n, k]$ linear code C has generator matrix G_C and parity-check matrix H_C. Firstly, note that a generator matrix can be trivially formed by listing all codewords as rows (although this representation may be reducible due to linearly-dependent rows). Let us define the corresponding *syndrome code* S as the one which 'encodes' the n physical bits into syndrome bits. The generator matrix of this code, G_S, is equal to the transpose of H_C. Then, recall from Chapter 2.1.2 that the parity-check matrix of a code is equal to the generator matrix of its dual code and vice versa. Thus, in order to find the parity-check matrix of the syndrome code, H_S, we must find the dual of the syndrome code. H_S defines the metachecks which can be used to determine if a syndrome is valid.

For a worked example, take the $[5, 1]$ repetition code with periodic boundary conditions:

$$H_C = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} . \tag{4.13}$$

The corresponding syndrome code S encodes 5 physical bits into 5 syndrome bits. The codewords in S are obtained by:

$$G_S = H_C^\top = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} . \tag{4.14}$$

Recall the standard forms of the matrices H and G given in Equations 2.10 and 2.11; H_S can be obtained by determining the submatrix A, which is equivalent to taking G_S into reduced row echelon form via Gauss–Jordan elimination:

$$(G_S)_{\mathrm{RREF}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} , \tag{4.15}$$

and then taking only nonzero rows as the $k \times n$ (here 4×5) generator matrix in standard form:

$$(G_S)_{\mathrm{std}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} = [I_k \mid -A^\top] . \tag{4.16}$$

So, finally, the parity-check matrix of the syndrome code is the $(n-k) \times n$ matrix formed as

$$H_S = [A \mid I_{n-k}] = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \end{pmatrix} . \tag{4.17}$$

By inspection, this is the single parity check which determines if the syndrome has even weight, as expected.

4.3.2 Practicality

We have seen a procedure for generating metachecks capable of syndrome validation from general linear code definitions. Ideally, these metachecks could be implemented with the purpose of stopping cluster growth once the observed syndrome becomes valid on the currently-supported code H', avoiding the need for Gaussian elimination for rank inspection.

Crucially, a number of caveats arise. Firstly, metachecks are generated at each growth step for the reduced system H', with the intent of avoiding Gaussian elimination; however, we have seen that Gauss–Jordan elimination is required for generating metachecks. This is intuitive as the problem is not only equivalent, but in fact more general: generating metachecks provides a system for checking *any* syndrome, not just the augmented observed syndrome σ.

Instead, we might hope that we could find the corresponding syndrome code S once on the global code, which could be filtered in a similar fashion as clusters are grown during decoding. However, metachecks might only emerge when considering a local cluster and may not be present in the global syndrome code. In the extreme case, every combination of checks might be a valid syndrome for the global code, i.e. there are no global metachecks at all.

For example, take the same $[5, 1]$ repetition code as in Equation 4.13 but with open boundary conditions, such that the parity-check matrix is now

$$H_C = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} . \tag{4.18}$$

Following the same procedure as above (that is, performing Gauss–Jordan elimination on the transposed matrix) gives the following generator matrix for the syndrome code in reduced row echelon form:

$$(G_S)_{\text{RREF}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} . \tag{4.19}$$

Note how the rightmost column of ones no longer appears, due to the fact that there were no longer any linearly-dependent rows in H_C; that is, H_C is full-rank. This means that no metachecks exist: the parity-check matrix of the syndrome code H_S is empty. This is consistent with the fact that all odd syndrome vectors are valid with open boundary conditions, not just even ones.

It is clear, then, that Gauss–Jordan elimination must be performed on the subcode formed by each instance of a grown cluster in order to obtain the correct metacheck(s) for syndrome validation. In order to gain benefit from this approach, it could be effective to cache the metachecks of clusters, or otherwise precompute common configurations as in a lookup-table decoder. This would intuitively be more effective in a sparse error regime where small error configurations are likely to repeatedly occur. In particular, it would be effective in codes with a high level of translational invariance, maximising the reusability potential of precomputed/cached metachecks; an example of such codes, generally, would be topological codes.

4.4 Conclusion

In this chapter, some open challenges surrounding the use of qLDPC codes at scale have been presented, with primary focus on the difficulty of decoding such generalised quantum codes efficiently. Reliance on Gaussian elimination to solve a parity-check equation was identified as a key bottleneck in qLDPC decoding, with the generalised union–find decoder performing Gaussian elimination repeatedly upon clusters grown iteratively on Tanner graphs. Two different research directions were therefore identified: improving Gaussian elimination, and avoiding it entirely.

On the former, Chapter 4.2 introduced an online variant of the Gaussian elimination subroutine used in qLDPC decoding. It was shown that – in theory – one can expect a polynomially-increasing reduction in the number of operations required compared to a standard offline implementation via complexity analysis inspired by the framework in [41]. While this asymptotic improvement competes with the negative effects of making suboptimal choices in pivot selection, it was shown that the online variant still outperforms offline in codes with a sparser parity-check matrix (i.e. LDPC codes) and higher covering radius.

During completion of this project, we became aware of recently-published work by Hillmann et al. in [114], where a technique related to online Gaussian elimination is discussed. In their *localised statistics decoding* (LSD), clusters initialised from a syndrome are grown by one variable node at a time, based on the next-most-likely error mode informed by marginal probabilities from an initial application of belief propagation. This BP–LSD decoder can therefore be interpreted as a weighted variant of the generalised union–find decoder with explicit parallelisation of cluster growth. An LUP decomposition is similarly used to retain row-operation decisions between iterations. Whilst the BP–LSD decoder uses this online strategy in this context, the exact algorithm, complexity analysis and empirical reasoning presented in this chapter hold novelty in the broad application of this strategy to qLDPC decoding and similar sparse linear problems.

As for avoiding Gaussian elimination, Chapter 4.3 presented notes on defining a *syndrome code* to solve the problem of syndrome validation, inspired by single-shot codes. The possibility of observed syndromes being invalid on local subcode clusters necessarily implies the existence of metachecks given by the local syndrome code; however, generating these metachecks is itself reliant on Gauss–Jordan elimination. It was reasoned that it may be effective to cache or precompute metachecks in a lookup-table-style approach, especially on codes with a high level of translational invariance, such as topological codes.

Chapter 5

Summary

In this work, a review of modern theory and techniques in quantum error correction (QEC) has been presented, a field increasingly deemed critical for the development of practical quantum computing at scale. Decoding algorithms have been identified as key bottlenecks in most major current QEC protocols. For surface codes, the union–find (UF) decoder was studied, with a variety of potential improvements identified and proposed. In particular, a novel complexity analysis was presented – via the application of percolation theory on grown erasure clusters – which suggests that the UF decoder has time complexity strictly linear in the number of qubits n. This means that the decoder implementation can be simplified considerably by forgoing union-by-size and path compression, leading to a reduction of both runtime and memory consumption in practice.

Future work could investigate how this reasoning affects the implementation and development of related decoders, especially distributed variants; indeed, these results have already been cited in subsequent works to justify forgoing the aforementioned optimisations [14], to quote an accurate complexity analysis of the UF decoder [15], to reason with the scaling of erasure clusters in order to develop novel decoders [22, 23], to justify the UF decoder and its properties more generally [16–20] and as an example of a nanotechnological theory applied to quantum technology [21].

For quantum low-density parity-check (qLDPC) codes, a generalisation of the UF decoder was studied. By identifying Gaussian elimination as a bottleneck in generalised decoders, methods to either improve or avoid Gaussian elimination were investigated. In particular, a novel online variant of the Gaussian elimination algorithm was presented, which can reduce the runtime of decoders which apply this algorithm repeatedly, such as the generalised UF decoder. As the overhead of decoding qLDPC codes remains significantly greater than decoding surface codes, this result will hopefully enable more efficient future iterations of qLDPC decoders. The use of metachecks was also discussed: while the theoretical benefit of using them to avoid Gaussian elimination is apparent, fundamental costs involved in their calculation make it unclear how best to apply them effectively. Further work could investigate precomputation approaches, especially for topological qLDPC codes.

Bibliography

[1] P. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, Nov. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[2] D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds., *Post-Quantum Cryptography*. Berlin, Heidelberg: Springer, 2009, ISBN: 978-3-540-88701-0. DOI: 10.1007/978-3-540-88702-7.

[3] Y. Cao, J. Romero, and A. Aspuru-Guzik, “Potential of quantum computing for drug discovery,” *IBM Journal of Research and Development*, vol. 62, no. 6, 6:1–6:20, Nov. 2018, ISSN: 0018-8646. DOI: 10.1147/JRD.2018.2888987.

[4] V. Dunjko and H. J. Briegel, “Machine learning & artificial intelligence in the quantum domain: A review of recent progress,” *Reports on Progress in Physics*, vol. 81, no. 7, p. 074 001, Jun. 2018, ISSN: 0034-4885. DOI: 10.1088/1361-6633/aab406.

[5] M. Singh, C. Dhara, A. Kumar, S. S. Gill, and S. Uhlig, “Quantum artificial intelligence for the science of climate change,” in *Artificial Intelligence, Machine Learning and Blockchain in Quantum Satellite, Drone and Network*, CRC Press, 2022, ISBN: 978-1-003-25035-7.

[6] A. Ho, J. McClean, and S. P. Ong, “The promise and challenges of quantum computing for energy storage,” *Joule*, vol. 2, no. 5, pp. 810–813, May 16, 2018, ISSN: 2542-4785, 2542-4351. DOI: 10.1016/j.joule.2018.04.021.

[7] “Ensuring that the UK can capture the benefits of quantum computing,” Oxford Economics, Feb. 6, 2025.

[8] “National quantum strategy missions,” Department for Science, Innovation and Technology. (Dec. 14, 2023), [Online]. Available: https://www.gov.uk/government/publications/national-quantum-strategy/national-quantum-strategy-missions.

[9] “£121 million boost for quantum technology set to tackle fraud, prevent money laundering and drive growth,” Department for Science, Innovation and Technology. (Apr. 14, 2025), [Online]. Available: https://www.gov.uk/government/news/121-million-boost-for-quantum-technology-set-to-tackle-fraud-prevent-money-laundering-and-drive-growth.

[10] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th Anniversary Edition. Cambridge University Press, Dec. 9, 2010, ISBN: 978-1-107-00217-3. DOI: 10.1017/CBO9780511976667.

[11] S. J. Griffiths and D. E. Browne, “Union-find quantum decoding without union-find,” *Physical Review Research*, vol. 6, no. 1, p. 013154, Feb. 9, 2024. DOI: 10.1103/PhysRevResearch.6.013154. arXiv: 2306.09767 [quant-ph].

[12] S. J. Griffiths, A. Benhemou, and D. E. Browne. “Online Gaussian elimination for quantum LDPC decoding.” arXiv: 2504.05080 [quant-ph]. (Apr. 7, 2025), [Online]. Available: http://arxiv.org/abs/2504.05080, pre-published.

[13] S. J. Griffiths, “Union–find quantum decoding without union–find,” presented at the APS March Meeting (Las Vegas, Nevada), Mar. 7, 2023.

[14] T. Chan and S. C. Benjamin, “Actis: A strictly local union–find decoder,” *Quantum*, vol. 7, p. 1183, Nov. 14, 2023. DOI: `10.22331/q-2023-11-14-1183`.

[15] T.-H. Lin and C.-Y. Lai, “Union-intersection union-find for decoding depolarizing errors in topological codes,” *IEEE Journal on Selected Areas in Information Theory*, vol. 6, pp. 163–175, 2025, ISSN: 2641-8770. DOI: `10.1109/JSAIT.2025.3581810`.

[16] M. Fu, C. Tian, Z. Fan, and H. Ma, “Resource-efficient decoding of topological color codes via neural-guided union-find optimization,” *Applied Sciences*, vol. 15, no. 16, p. 8937, Jan. 2025, ISSN: 2076-3417. DOI: `10.3390/app15168937`.

[17] A. Benhemou, K. Sahay, L. Lao, and B. J. Brown, “Minimising surface-code failures using a color-code decoder,” *Quantum*, vol. 9, p. 1632, Feb. 17, 2025. DOI: `10.22331/q-2025-02-17-1632`.

[18] A. Benhemou, “Progress in quantum error correction and simulation, a many-body physics approach,” Doctoral, UCL (University College London), Jun. 28, 2025, 244 pp.

[19] T. Chan. “Snowflake: A distributed streaming decoder.” arXiv: `2406.01701 [quant-ph]`. (Dec. 15, 2024), [Online]. Available: `http://arxiv.org/abs/2406.01701`, pre-published.

[20] A. Ghosh, A. Chatterjee, and S. Ghosh. “Design automation in quantum error correction.” arXiv: `2507.12253 [quant-ph]`. (Jul. 16, 2025), [Online]. Available: `http://arxiv.org/abs/2507.12253`, pre-published.

[21] S. N. Domgueu, J. V. Nguepnang, C. M. Ekengoue, and A. K. Jiotsa, "Investigation of polaron properties in semiconductors quantum dot under the influence of an oscillating electric radiation," *Brazilian Journal of Physics*, vol. 55, no. 3, p. 104, Mar. 11, 2025, ISSN: 1678-4448. DOI: 10.1007/s13538-025-01733-w.

[22] M. C. Löbl, S. X. Chen, S. Paesani, and A. S. Sørensen. "Breadth-first graph traversal union-find decoder." arXiv: 2407.15988 [quant-ph]. (Jul. 22, 2024), [Online]. Available: http://arxiv.org/abs/2407.15988, pre-published.

[23] A. B. Ziad *et al.* "Local clustering decoder: A fast and adaptive hardware decoder for the surface code." arXiv: 2411.10343 [quant-ph]. (Nov. 15, 2024), [Online]. Available: http://arxiv.org/abs/2411.10343, pre-published.

[24] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96, New York, NY, USA: Association for Computing Machinery, Jul. 1, 1996, pp. 212–219, ISBN: 978-0-89791-785-8. DOI: 10.1145/237814.237866.

[25] E. Farhi, J. Goldstone, and S. Gutmann. "A quantum approximate optimization algorithm." arXiv: 1411.4028 [quant-ph]. (Nov. 14, 2014), [Online]. Available: http://arxiv.org/abs/1411.4028, pre-published.

[26] A. Peruzzo *et al.*, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, p. 4213, Jul. 23, 2014, ISSN: 2041-1723. DOI: 10.1038/ncomms5213. PMID: 25055053.

[27] A. Kitaev, "Quantum measurements and the Abelian stabilizer problem," Electronic Colloquium on Computational Complexity (ECCC), TR96-003, Jan. 17, 1996. arXiv: `quant-ph/9511026`.

[28] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, p. 021 318, Jun. 17, 2019, ISSN: 1931-9401. DOI: `10.1063/1.5089550`.

[29] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, p. 021 314, May 29, 2019, ISSN: 1931-9401. DOI: `10.1063/1.5088164`.

[30] P. Kok, W. J. Munro, K. Nemoto, T. C. Ralph, J. P. Dowling, and G. J. Milburn, "Linear optical quantum computing with photonic qubits," *Reviews of Modern Physics*, vol. 79, no. 1, pp. 135–174, Jan. 24, 2007. DOI: `10.1103/RevModPhys.79.135`.

[31] M. Saffman, T. G. Walker, and K. Mølmer, "Quantum information with Rydberg atoms," *Reviews of Modern Physics*, vol. 82, no. 3, pp. 2313–2363, Aug. 18, 2010. DOI: `10.1103/RevModPhys.82.2313`.

[32] A. Y. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, no. 1, pp. 2–30, Jan. 2003, ISSN: 00034916. DOI: `10.1016/S0003-4916(02)00018-0`. arXiv: `quant-ph/9707021`.

[33] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962, ISSN: 2168-2712. DOI: `10.1109/TIT.1962.1057683`.

[34] P. Panteleev and G. Kalachev, "Asymptotically good quantum and locally testable classical LDPC codes," in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC

2022, New York, NY, USA: Association for Computing Machinery, Jun. 10, 2022, pp. 375–388, ISBN: 978-1-4503-9264-8. DOI: 10.1145/3519935.3520017.

[35] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," in *Probabilistic and Causal Inference: The Works of Judea Pearl*, 1st ed., vol. 36, New York, NY, USA: Association for Computing Machinery, Mar. 4, 2022, pp. 129–138, ISBN: 978-1-4503-9586-1. DOI: 10.1145/3501714.3501727.

[36] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, Sep. 2002, ISSN: 0022-2488, 1089-7658. DOI: 10.1063/1.1499754. arXiv: quant-ph/0110143.

[37] R. Sweke, M. S. Kesselring, E. P. L. van Nieuwenburg, and J. Eisert, "Reinforcement learning decoders for fault-tolerant quantum computation," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 025 005, Dec. 2020, ISSN: 2632-2153. DOI: 10.1088/2632-2153/abc609.

[38] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, no. 3, p. 032 324, Sep. 2012, ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.86.032324. arXiv: 1208.0928.

[39] Google Quantum AI and Collaborators, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, no. 8052, pp. 920–926, 2025, ISSN: 0028-0836. DOI: 10.1038/s41586-024-08449-y. PMID: 39653125.

[40] N. Delfosse and N. H. Nickerson, "Almost-linear time decoding algorithm for topological codes," *Quantum*, vol. 5, p. 595, Dec. 2021. DOI: 10.22331/q-2021-12-02-595.

[41] N. Delfosse, V. Londe, and M. E. Beverland, "Toward a union-find decoder for quantum LDPC codes," *IEEE Transactions on Information Theory*, vol. 68, no. 5, pp. 3187–3199, May 2022, ISSN: 1557-9654. DOI: 10.1109/TIT.2022.3143452.

[42] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, Feb. 2005, ISBN: 978-0-521-61946-2.

[43] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Elsevier, 1977, ISBN: 978-0-444-85010-2.

[44] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, Apr. 1950, ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[45] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948, ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[46] S. Verdu, "Fifty years of Shannon theory," *IEEE Transactions on Information Theory*, vol. 44, no. 6, p. 22, 1998.

[47] Glosser.ca. "Bloch Sphere.svg." (Dec. 19, 2012), [Online]. Available: https://commons.wikimedia.org/wiki/File:Bloch_Sphere.svg. (This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported licence.)

[48] J. L. Park, "The concept of transition in quantum mechanics," *Foundations of Physics*, vol. 1, no. 1, pp. 23–33, Mar. 1970, ISSN: 1572-9516. DOI: 10.1007/BF00708652.

[49] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Physical Review A*, vol. 52, no. 4, R2493–R2496, Oct. 1995. DOI: 10.1103/PhysRevA.52.R2493.

[50] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Physical Review A*, vol. 54, no. 2, pp. 1098–1105, Aug. 1996, ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.54.1098. arXiv: quant-ph/9512032.

[51] V. V. Albert and P. Faist. "Topological code," The Error Correction Zoo. (2022), [Online]. Available: https://errorcorrectionzoo.org/c/topological.

[52] Krishnavedala. "Torus cycles.svg." (Apr. 14, 2014), [Online]. Available: https://commons.wikimedia.org/wiki/File:Torus_cycles.svg. (This file is made available under the Creative Commons CC0 1.0 Universal Public Domain Dedication.)

[53] J. Bausch, T. S. Cubitt, A. Lucia, D. Perez-Garcia, and M. M. Wolf, "Size-driven quantum phase transitions," *Proceedings of the National Academy of Sciences*, vol. 115, no. 1, pp. 19–23, Jan. 2, 2018. DOI: 10.1073/pnas.1705042114.

[54] R. K. Pathria and P. D. Beale, "Phase transitions: Exact (or almost exact) results for various models," in *Statistical Mechanics (Fourth Edition)*, R. K. Pathria and P. D. Beale, Eds., Academic Press, Jan. 1, 2022, pp. 487–554, ISBN: 978-0-08-102692-2. DOI: 10.1016/B978-0-08-102692-2.00022-3.

[55] H. Bombín, "Topological codes," in *Quantum Error Correction*, D. A. Lidar and T. A. Brun, Eds., Cambridge: Cambridge University Press, Sep. 2013, pp. 455–481, ISBN: 978-0-521-89787-7. DOI: 10.1017/CBO9781139034807. arXiv: 1311.0277 [quant-ph].

[56] N. Delfosse, P. Iyer, and D. Poulin. "Generalized surface codes and packing of logical qubits." arXiv: `1606.07116 [quant-ph]`. (Jun. 22, 2016), [Online]. Available: `https://arxiv.org/abs/1606.07116`, pre-published.

[57] O. Higgott and N. P. Breuckmann, "Subsystem codes with high thresholds by gauge fixing and reduced qubit overhead," *Physical Review X*, vol. 11, no. 3, p. 031 039, Aug. 2021, ISSN: 2160-3308. DOI: `10.1103/PhysRevX.11.031039`. arXiv: `2010.09626`.

[58] H. Bombín, "Single-shot fault-tolerant quantum error correction," *Physical Review X*, vol. 5, no. 3, p. 031 043, Sep. 2015. DOI: `10.1103/PhysRevX.5.031043`.

[59] A. O. Quintavalle, M. Vasmer, J. Roffe, and E. T. Campbell, "Single-shot error correction of three-dimensional homological product codes," *PRX Quantum*, vol. 2, no. 2, p. 020 340, Jun. 2021. DOI: `10.1103/PRXQuantum.2.020340`.

[60] O. Higgott and N. P. Breuckmann, "Improved single-shot decoding of higher dimensional hypergraph product codes," *PRX Quantum*, vol. 4, no. 2, p. 020 332, May 2023, ISSN: 2691-3399. DOI: `10.1103/PRXQuantum.4.020332`. arXiv: `2206.03122 [quant-ph]`.

[61] E. T. Campbell, "A theory of single-shot error correction for adversarial noise," *Quantum Science and Technology*, vol. 4, no. 2, p. 025 006, Feb. 2019, ISSN: 2058-9565. DOI: `10.1088/2058-9565/aafc8f`.

[62] J. Roffe, L. Z. Cohen, A. O. Quintavalle, D. Chandra, and E. T. Campbell, "Bias-tailored quantum LDPC codes," *Quantum*, vol. 7, p. 1005, May 15, 2023. DOI: `10.22331/q-2023-05-15-1005`.

[63] A. Honecker, M. Picco, and P. Pujol, "Nishimori point in the 2D +/- J random-bond Ising model," *Physical Review Letters*, vol. 87, no. 4, p. 047 201, Jul. 2001, ISSN: 0031-9007, 1079-7114. DOI: `10.1103/PhysRevLett.87.047201`. arXiv: `cond-mat/0010143`.

[64] Y. S. Weinstein, "Quantum error correction during 50 gates," *Physical Review A*, vol. 89, no. 2, p. 020 301, Feb. 5, 2014. DOI: `10.1103/PhysRevA.89.020301`.

[65] C. Wang, J. Harrington, and J. Preskill, "Confinement-Higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory," *Annals of Physics*, vol. 303, no. 1, pp. 31–58, Jan. 2003, ISSN: 0003-4916. DOI: `10.1016/S0003-4916(02)00019-2`.

[66] D. S. Wang, A. G. Fowler, A. M. Stephens, and L. C. L. Hollenberg, "Threshold error rates for the toric and planar codes," *Quantum Information & Computation*, vol. 10, no. 5, pp. 456–469, May 1, 2010. DOI: `10.26421/QIC10.5-6-6`. arXiv: `0905.0531 [quant-ph]`.

[67] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, Jan. 1965, ISSN: 0008-414X, 1496-4279. DOI: `10.4153/CJM-1965-045-4`.

[68] V. Kolmogorov, "Blossom V: A new implementation of a minimum cost perfect matching algorithm," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 43–67, Jul. 2009, ISSN: 1867-2957. DOI: `10.1007/s12532-009-0002-8`.

[69] O. Higgott, "PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, 16:1–16:16, Jun. 30, 2022. DOI: `10.1145/3505637`.

[70] S. Bravyi and J. Haah, "Analytic and numerical demonstration of quantum self-correction in the 3D cubic code," *Physical Review Letters*, vol. 111, no. 20, p. 200 501, Nov. 2013, ISSN: 0031-9007, 1079-7114. DOI: `10.1103/PhysRevLett.111.200501`. arXiv: `1112.3252`.

[71] J. Wootton, "A simple decoder for topological codes," *Entropy*, vol. 17, no. 4, pp. 1946–1957, Apr. 2015, ISSN: 1099-4300. DOI: `10.3390/e17041946`.

[72] H. Anwar, B. J. Brown, E. T. Campbell, and D. E. Browne, "Fast decoders for qudit topological codes," *New Journal of Physics*, vol. 16, no. 6, p. 063 038, Jun. 2014, ISSN: 1367-2630. DOI: `10.1088/1367-2630/16/6/063038`.

[73] N. Delfosse and G. Zémor, "Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel," *Physical Review Research*, vol. 2, no. 3, p. 033 042, Jul. 9, 2020. DOI: `10.1103/PhysRevResearch.2.033042`.

[74] R. E. Korf, "Artificial intelligence search algorithms," in *Algorithms and Theory of Computation Handbook: Special Topics and Techniques*, 2nd ed., Chapman & Hall/CRC, Jan. 1, 2010, p. 22, ISBN: 978-1-58488-820-8.

[75] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956, ISSN: 0002-9939, 1088-6826. DOI: `10.1090/S0002-9939-1956-0078686-7`.

[76] R. E. Tarjan, "Amortized computational complexity," *SIAM Journal on Algebraic Discrete Methods*, vol. 6, no. 2, pp. 306–318, Apr. 1985, ISSN: 0196-5212. DOI: `10.1137/0606031`.

[77] Y. Wu, N. Liyanage, and L. Zhong. “An interpretation of union-find decoder on weighted graphs.” arXiv: 2211.03288 [quant-ph]. (Nov. 6, 2022), [Online]. Available: https://arxiv.org/abs/2211.03288, pre-published.

[78] S. Huang, M. Newman, and K. R. Brown, “Fault-tolerant weighted union-find decoding on the toric code,” *Physical Review A*, vol. 102, no. 1, p. 012 419, Jul. 2020, ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.102.012419. arXiv: 2004.04693.

[79] R. E. Tarjan and J. van Leeuwen, “Worst-case analysis of set union algorithms,” *Journal of the ACM*, vol. 31, no. 2, pp. 245–281, Mar. 1984, ISSN: 0004-5411. DOI: 10.1145/62.2160.

[80] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, Jul. 2009, ISBN: 978-0-262-53305-8.

[81] P. Das *et al.*, “AFS: Accurate, fast, and scalable error-decoding for fault-tolerant quantum computers,” in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Apr. 2022, pp. 259–273. DOI: 10.1109/HPCA53966.2022.00027.

[82] N. P. Breuckmann and J. N. Eberhardt, “Quantum low-density parity-check codes,” *PRX Quantum*, vol. 2, no. 4, p. 040 101, Oct. 11, 2021. DOI: 10.1103/PRXQuantum.2.040101.

[83] L. Golowich and V. Guruswami, “Decoding quasi-cyclic quantum LDPC codes,” in *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, Oct. 2024, pp. 344–368. DOI: 10.1109/FOCS61266.2024.00029.

[84] D. Poulin and Y. Chung, “On the iterative decoding of sparse quantum codes,” *Quantum Information & Computation*, vol. 8, no. 10,

pp. 987–1000, Nov. 1, 2008, ISSN: 1533-7146. DOI: 10.26421/QIC8.10-8. arXiv: 0801.1241 [quant-ph].

[85] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981, ISSN: 1557-9654. DOI: 10.1109/TIT.1981.1056404.

[86] C. Castelnovo and C. Chamon, "Topological order in a three-dimensional toric code at finite temperature," *Physical Review B*, vol. 78, no. 15, p. 155 120, Oct. 21, 2008. DOI: 10.1103/PhysRevB.78.155120.

[87] A. Kulkarni and P. K. Sarvepalli, "Decoding the three-dimensional toric codes and welded codes on cubic lattices," *Physical Review A*, vol. 100, no. 1, p. 012 311, Jul. 8, 2019. DOI: 10.1103/PhysRevA.100.012311.

[88] Y. Takada and K. Fujii, "Improving threshold for fault-tolerant color-code quantum computing by flagged weight optimization," *PRX Quantum*, vol. 5, no. 3, p. 030 352, Sep. 17, 2024. DOI: 10.1103/PRXQuantum.5.030352.

[89] H. Bombín and M. A. Martin-Delgado, "Topological quantum distillation," *Physical Review Letters*, vol. 97, no. 18, p. 180 501, Oct. 30, 2006. DOI: 10.1103/PhysRevLett.97.180501.

[90] G. F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *Artificial Intelligence*, vol. 42, no. 2, pp. 393–405, Mar. 1990, ISSN: 0004-3702. DOI: 10.1016/0004-3702(90)90060-D.

[91] J. Roffe, D. R. White, S. Burton, and E. T. Campbell, "Decoding across the quantum LDPC code landscape," *Physical Review Research*, vol. 2, no. 4, p. 043 423, Dec. 2020, ISSN: 2643-1564. DOI: 10.1103/PhysRevResearch.2.043423. arXiv: 2005.07016 [quant-ph].

[92] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001, ISSN: 1557-9654. DOI: 10.1109/18.910572.

[93] M. Mézard and A. Montanari, *Information, Physics, and Computation.* Oxford University Press, Jan. 2009, ISBN: 978-0-19-154719-5.

[94] L. Guth and A. Lubotzky, "Quantum error correcting codes and 4-dimensional arithmetic hyperbolic manifolds," *Journal of Mathematical Physics*, vol. 55, no. 8, p. 082 202, Aug. 6, 2014, ISSN: 0022-2488. DOI: 10.1063/1.4891487.

[95] D. Aharonov and L. Eldar, "Quantum locally testable codes," *SIAM Journal on Computing*, vol. 44, no. 5, pp. 1230–1262, Jan. 2015, ISSN: 0097-5397. DOI: 10.1137/140975498.

[96] A. Leverrier, J.-P. Tillich, and G. Zémor, "Quantum expander codes," in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, Oct. 2015, pp. 810–824. DOI: 10.1109/FOCS.2015.55.

[97] M. Fossorier and S. Lin, "Soft decision decoding of linear block codes based on ordered statistics for the Rayleigh fading channel with coherent detection," *IEEE Transactions on Communications*, vol. 45, no. 1, pp. 12–14, Jan. 1997, ISSN: 1558-0857. DOI: 10.1109/26.554278.

[98] P. Panteleev and G. Kalachev, "Degenerate quantum LDPC codes with good finite length performance," *Quantum*, vol. 5, p. 585, Nov. 2021. DOI: 10.22331/q-2021-11-22-585.

[99] J. Wills, M. T. Haque, and B. Vlastakis. "Error-detected coherence metrology of a dual-rail encoded fixed-frequency multimode superconducting qubit." arXiv: 2506.15420 [quant-ph]. (Jun. 18, 2025), [Online]. Available: http://arxiv.org/abs/2506.15420, pre-published.

[100] N. L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions*, 2nd ed. Wiley, Oct. 1994, vol. 1, 784 pp., ISBN: 978-0-471-58495-7.

[101] D. Stauffer and A. Aharony, *Introduction to Percolation Theory*, Revised 2nd ed. London: Taylor & Francis, Jul. 18, 1994, 181 pp., ISBN: 0-7484-0253-5. DOI: `10.1201/9781315274386`. (Cited page numbers correspond to the print version matching the ISBN, not the eBook matching the DOI.)

[102] R. Degraeve *et al.*, "New insights in the relation between electron trap generation and the statistical properties of oxide breakdown," *IEEE Transactions on Electron Devices*, vol. 45, no. 4, pp. 904–911, Apr. 1998, ISSN: 1557-9646. DOI: `10.1109/16.662800`.

[103] A. Duane, M. D. Miranda, and L. Brotons, "Forest connectivity percolation thresholds for fire spread under different weather conditions," *Forest Ecology and Management*, vol. 498, p. 119 558, Oct. 2021, ISSN: 0378-1127. DOI: `10.1016/j.foreco.2021.119558`.

[104] D. W. van Krevelen, "Development of coal research — a review," *Fuel*, vol. 61, no. 9, pp. 786–790, Sep. 1982, ISSN: 0016-2361. DOI: `10.1016/0016-2361(82)90304-0`.

[105] A. Fasano, F. Talamucci, and M. Petracco, "The espresso coffee problem," in *Complex Flows in Industrial Processes*, A. Fasano, Ed., Boston, MA: Birkhäuser, 2000, pp. 241–280, ISBN: 978-1-4612-1348-2. DOI: `10.1007/978-1-4612-1348-2_8`.

[106] B. Suki, "The major transitions of life from a network perspective," *Frontiers in Physiology*, vol. 3, p. 94, Apr. 2012, ISSN: 1664-042X. DOI: `10.3389/fphys.2012.00094`.

[107] H. Kesten, “The critical probability of bond percolation on the square lattice equals 1/2,” *Communications in Mathematical Physics*, vol. 74, no. 1, pp. 41–59, Feb. 1980, ISSN: 1432-0916. DOI: `10.1007/BF01197577`.

[108] K. Christensen. “Percolation theory,” Imperial College London. (Oct. 9, 2002), [Online]. Available: `https://www.physics.rutgers.edu/~morozov/677_f2016/Physics_677_2016_files/student_papers/percol_notes.pdf`.

[109] R. G. Newcombe, “Two-sided confidence intervals for the single proportion: Comparison of seven methods,” *Statistics in Medicine*, vol. 17, no. 8, pp. 857–872, 1998, ISSN: 1097-0258. DOI: `10.1002/(SICI)1097-0258(19980430)17:8<857::AID-SIM777>3.0.CO;2-E`.

[110] T. Richardson, M. Shokrollahi, and R. Urbanke, “Design of capacity-approaching irregular low-density parity-check codes,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001, ISSN: 1557-9654. DOI: `10.1109/18.910578`.

[111] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms* (Algorithms and Combinatorics). Berlin, Heidelberg: Springer, 2018, vol. 21, ISBN: 978-3-662-56038-9. DOI: `10.1007/978-3-662-56039-6`.

[112] I. R. Shafarevich and A. O. Remizov, *Linear Algebra and Geometry.* Springer Science & Business Media, Aug. 2012, ISBN: 978-3-642-30994-6.

[113] S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares.* Cambridge University Press, Jun. 2018, ISBN: 978-1-316-51896-0.

[114] T. Hillmann, L. Berent, A. O. Quintavalle, J. Eisert, R. Wille, and J. Roffe. “Localized statistics decoding: A parallel decoding algorithm for quantum low-density parity-check codes.” arXiv: 2406.18655 [quant-ph]. (Jun. 26, 2024), [Online]. Available: https://arxiv.org/abs/2406.18655, pre-published.