QUANTUM ERROR CORRECTION: FROM THE BLACKBOARD TO THE CLOUD

Habilitationsschrift

vorgelegt der Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

James R. Wootton

aus Ely, Vereinigtes Königreich

Basel, 2023

Acknowledgements

I have received a great deal of support over the years, professionally and personally, without which the work presented here would not have been possible.

For the former, I would particularly like to thank Dr Jiannis K. Pachos for his support during and after my PhD, and Prof Daniel Loss for his support during the long time within his group (even when I decided to spend a year making computer games). In addition, I am grateful to all those within both groups, both during my time within them as well as those who I have met and worked with thereafter.

In addition I would like to thank IBM Quantum and the Qiskit community. Without them, there would have been no cloud quantum computing revolution, and my activities over the past six years would have been very different. In particular I would like to thank Dr Walter Riess, who has given indispensable support and guidance during my time at IBM.

I would also like to thank my family and friends. In particular my wife, Mansura, since it is only through her efforts in all other spheres of life that I am able to succeed at work. And also my children, Freda and Lana, for ensuring that I do not get lost in the theoretical details of quantum error correction.

Of course, I also need to thank my parents, Ena and Robin Wootton, for all the support given to me throughout my life. And in that spirit, this thesis is dedicated to the memory of my mother.

Summary

It has been several decades since the idea of quantum error correction was first proposed. In that time, many different approaches have been developed and studied. Until the last decade, this work was almost exclusively restricted to theory, with a great gulf between the worlds of theory and experiment. For example, a theorist calculating a threshold during this era might regard a code with 1000 qubits as relatively small. But at the same time, an experiment with just 4 qubits could be seen as revolutionary.

This disconnect between theory and experiment was not useful for the development of quantum computing. Quantum error correction is the point at which quantum hardware and software meet, and so will inevitably need to be a compromise between the two. Such a compromise can only be reached if theoretical efforts focus on the challenges and constraints of realistic hardware, and experimental efforts strive to realize the features that fault-tolerance cannot do without.

The need for such a compromise has become widely recognized over the past decade. In part, this has been due to the availability of quantum hardware through cloud services, which has allowed theorists to perform experiments as easily as they would previously run simulations. This has brought about a new era in which theory and experiment can be treated as software and hardware, and therefore more naturally intermingle.

In this thesis we explore some of the possibilities of this new era. This is done in three parts. The first two parts contain two collections of papers, each of which serve as worked examples of how cloud-quantum computing systems can help advance proof-of-principle experiments for quantum error correction. Each of these is preceded by an overview of these papers, and the story they tell as a combined whole. The third part contains a summary of other areas in which cloud-quantum computing has made an impact on the work of the author.

Contents

Part I

Matching Codes

Chapter 1

Introduction

The introduction of Kitaev's honeycomb lattice model [1] was highly influential in both the fields of condensed matter and quantum computation. Even the 'Odds and Ends' section of this work, a set of thoughts laid out as bullet points, formed the basis of many follow up papers. This includes the introduction of twists in surface codes [2].

Though these early twists took inspiration from the honeycomb lattice model, their implementation in standard surface codes was not so elegant. It was for this reason that 'matching codes' were developed [3], now known also as XYZ^2 codes [4]. These codes formalize the Abelian phase of the honeycomb lattice model in the language of stabilizer codes, and allow the idea behind them to be expanded in that framework. This included showing how twists may be implemented in such codes, retaining the elegance of the implementation in the honeycomb lattice model as unpaired Majorana modes.

A result of this elegance was relative ease of implementation. Taking this to the extreme, it was found that just five qubits would suffice to demonstrate a logical gate through the exchange of twists, corresponding to the braiding of Majoranas. Though the experiment was proposed in the original paper [3], reproduced here as Chapter 2, access to five qubit systems was severely limited at the time. It was therefore doubtful that such an experiment would be implemented.

This situation changed completely in 2016 when IBM launched its 'Quantum Experience' (now the IBM Quantum Composer and Lab [5]). This offered 5 qubits with public access, allowing circuits to be designed and run via a web interface. During lunch one day, it was realized that this service was sufficient to implement the Majorana braiding experiment. By the end of the working day, the experiment had been run! The paper [6], reproduced here as Chapter 3, was then soon on the arXiv. Putting 5 qubits on the cloud had started a revolution in the transfer of ideas from theory to experiment.

Over the coming years, IBM Quantum's hardware would progress beyond 5 qubits. At first, a square lattice architecture was pursued. However, this was later replaced with a roadmap based on a so-called 'heavy hexagonal' coupling map: qubits residing on the vertices and edges of a hexagonal lattice, with entangling gates possibly only between neighbouring vertex/edge pairs [7].

Such a coupling map is well-suited to approaches based on the honeycomb lattice model, since it allows the easy measurement of the two-body observables associated with each link. The only missing piece is measurement of the six-body plaquette observables, which correspond to a product of the link operators around each plaquette. For this reason, a means to infer measurement of the plaquettes via measurements of the links was devised in [8], reproduced here as Chapter 4.

Given this measurement scheme, as well as compatible hardware, a new set of experiments became possible. By repeatedly measuring the plaquettes, the values for subsequent measurements of the same plaquette could be compared. Since these are stabilizers of the code, and commute with all the (directly measured) link operators, their values should not change. Any changes present are then signatures of errors. An experiment measuring the plaquettes would then serve two purposes: as a proof-of-principle that such high weight observables can be measured in IBM Quantum hardware, and as a way of probing the amount of error that they detect. This experiment was also performed and reported in [6], and in Chapter 4.

At around the same time as the above work, Hastings and Haah proposed their Floquet codes [9]. These were also based on the honeycomb lattice model, and also allowed the plaquette measurements to be inferred from measurements of the link operators. A similar idea was present in the Floquet Color code of Kesselring, Brown and coauthors [10]. This is similarly based on notions of link operators whose measurements can be combined into measurements of plaquette operators. Though they don't derive from the honeycomb lattice model, they are nevertheless well-suited to the heavy hexagonal architecture. The same form of experiments can then also be performed for the Floquet codes and Floquet Color code. This was performed in [11], and is reproduced in Chapter 5, which serves as a direct parallel to Chapter 4.

These experiments allow these codes to be directly compared and contrasted, at least at the level of how much error is detected by their stabilizers in current hardware. In all three cases, the results for the plaquette stabilizers are quite consistent. They show results that would be consistent with a simple error model with noise strength $p = 1.5\%$ for the best devices tested, with the worst devices more consistent with $p = 6\%$. As a benchmark of the hardware, this shows that the devices are approaching the regime in which fault-tolerance could be achieved, but still have some way to go.

The results for each device can also be compared to its quantum volume [12]. In each case, no obvious correlation between success in these experiments and the achieved quantum volume can be seen. This motivates tailored benchmarks for fault-tolerance, as will also be discussed in Part II.

Chapter 2

A family of stabilizer codes for $D(\mathbb{Z}_2)$ anyons and Majorana modes

Abstract

We study and generalize the class of qubit topological stabilizer codes that arise in the Abelian phase of the honeycomb lattice model. The resulting family of codes, which we call 'matching codes' realize the same anyon model as the surface codes, and so may be similarly used in proposals for quantum computation. We show that these codes are particularly well suited to engineering twist defects that behave as Majorana modes. A proof of principle system that demonstrates the braiding properties of the Majoranas is discussed that requires only three qubits.

Publication Information

Originally published in *Journal of Physics A: Mathematical and Theoretical*, Volume 48, Number 21 (2015) [3].

Note that the references of this chapter are self-contained, and not included in the overall references.

INTRODUCTION

Quantum error correction is an important aspect of fault-tolerant quantum computation, and many quantum error correcting codes have been proposed to serve this purpose. Recently, a great deal of interest has been focussed on so-called topological codes, such as the surface codes [1] and quantum double models [2]. These support exotic quasiparticles known as anyons, which can be used to implement fault-tolerant quantum gates on encoded information [3, 4].

The anyon model supported by a given code is classified as either Abelian or non-Abelian. This distinction determines a great deal about how the anyons behave, and how they may be used. However this is not always so clear cut, since methods can be applied to Abelian codes that give them properties that resemble non-Abelian ones [5, 6]. The most prominent method is engineering so-called twists in the code [7]. These twists are related to symmetries of the model, and so are symmetry defects [8].

A well-known model that is capable of supporting both Abelian and non-Abelian phases is the honeycomb lattice model [9]. This is not strictly an error correcting code, but is instead an interacting spin Hamiltonian for which anyons emerge as excitations. However, it is known that Abelian codes can be derived from the Abelian phase of this model. It is such codes that we study and generalize here. We show that these codes are well-suited to engineering twists that behave as Majorana modes. In fact they are so well-suited that we need not explicitly use the concept of twists, and can reinterpret the codes in terms of Majoranas.

MATCHING CODES

The Abelian phase of the honeycomb lattice model is typically studied by a perturbative analysis, with the surface code emerging in the low energy subspace [9]. However, a topological code also emerges when the full spectrum is taken into account. It is this approach that we will take here.

Though the codes we consider are inspired by the interacting spin Hamiltonian of the honeycomb lattice model, we will not study them in this context. No Hamiltonian will be considered to energetically suppress errors. Instead will use the standard framework of quantum error correction, in which errors are detected and suppressed by continuous syndrome measurements only. This frees us from some of the restrictions required for realistic Hamiltonians, and allows us to generalize to a large family of codes. We will refer to these, which all take the form of qubit stabilizer codes, as ‘matching codes’.

For the purpose of presentation, it is advantageous to start with the general form of these codes. The specific cases relevant to the honeycomb lattice model will then be discussed later.

Matching codes are defined on trivalent lattices with a qubit on each vertex. For simplicity we will restrict to lattices with periodic boundary conditions. Each edge (or link) of the lattice is labelled x, y or z such that no links of the same type are adjacent. Examples are shown in Fig. 1. For each link, l, of type $\alpha \in \{x, y, z\}$ one defines a link operator $K_l = \sigma_j^\alpha \sigma_k^\alpha$ that acts on the two adjacent qubits. These form a basic set of operators used to construct the stabilizer generators of the codes.

Stabilizer Generators

Many different types of stabilizer code can be defined on spin lattices. To restrict to a certain class, we consider only stabilizer generators that are products of link operators. Since this still leaves considerable freedom, we will also restrict to products for which the links involved form a path.

A path is a subset of the links of the lattice such that no more than two vertices are incident upon an odd number of path edges. If there are no such vertices, we call the path a loop. If there are two then the path is a string with these vertices as endpoints. We will denote a path with endpoints j and k as $P(j, k)$.

With the lattices we consider, the simplest form of loop are those that encircle each plaquette. For a plaquette p, we then define the following plaquette operator using this loop

$$W_p \sim \prod_{l \in p} K_l. \tag{1}$$

This product will yield a tensor product of Pauli operators with a phase of ± 1 or $\pm i$. The $\sim$ signifies that only the former is taken to be W_p, with the phase either ignored or chosen according to convenience. For the honeycomb lattice, shown in Fig. 1(a), the plaquette operators are

$$W_p = \sigma_1^x \sigma_2^y \sigma_3^z \sigma_4^x \sigma_5^y \sigma_6^z. \tag{2}$$

Each plaquette operator commutes with all link operators and products thereof. The plaquette operators therefore mutually commute. Also, note that any loop operator can be expressed as product of the plaquette operators.

We will consider codes for which the plaquette operators are included within the stabilizer. For a lattice of N vertices (and therefore qubits) there are $N/2$ plaquettes. At least a further $N/2$ are required in order to uniquely specify a stabilizer state.

To define the additional stabilizers we consider a matching of the vertices, M. This is simply a set of disjoint pairs of vertices. For each pair (j, k) a path $P(j, k)$ is chosen such that the vertices j and k form the endpoints. The following stabilizers are then defined

$$S_{j,k} \sim \prod_{l \in P(j,k)} K_l. \tag{3}$$

Again the $\sim$ denotes that the phase is ignored and only the resulting tensor product of Pauli operators is used for the $S_{j,k}$.

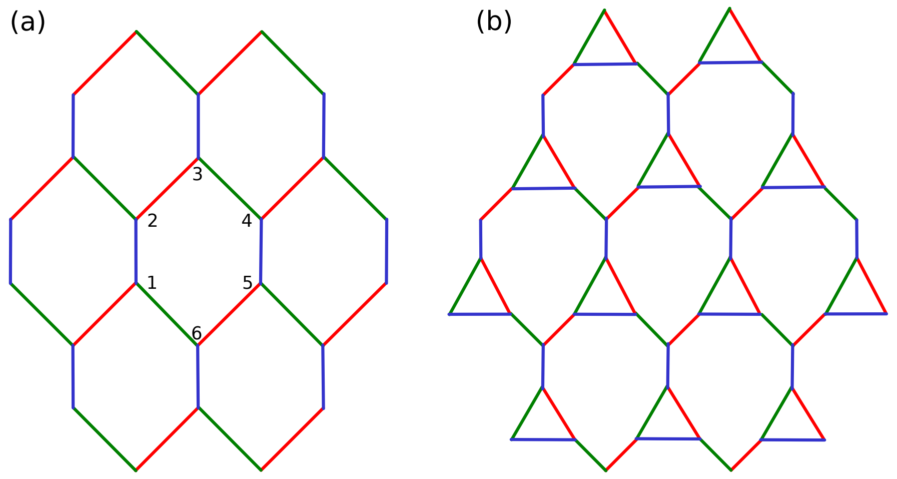


FIG. 1. We consider the two lattices shown, each wrapped around a torus. (a) x links are right leaning diagonal lines, y links are left leaning and z links are vertical. These are shown in red, green and blue, respectively. Numbering of qubits around a plaquette is shown. (b) Modified honeycomb lattice. Vertices on one sublattice are replaced by triangles. The labeling of the new edges is uniquely defined by the condition that no two links of the same type are adjacent.

The string stabilizers anticommute with all link operators incident upon one (but not both) of their endpoints, and commute otherwise. The same is true for products of link operators along a string: they anticommute only if they share a single endpoint and commute otherwise. Since the pairs used to define the stabilizers are non-overlapping, they will share no endpoints with each other. They therefore mutually commute with each other and the plaquette operators.

Clearly any pairing of N vertices will yield $N/2$ pairs, and so there will be $N/2$ such stabilizers. Using both the loop stabilizers W_p and string stabilizers $S_{j,k}$ we have a set of N stabilizer generators.

Properties of the Stabilizer

With a set of stabilizer generators defined, we can now determine the properties of the resulting stabilizer code. Given all the chosen strings $P(j,k)$ we can determine n_l, the number of strings each link, l, is included in. We refer to links as 'even' or 'odd' depending on whether their n_l is even or odd. As such, clearly

$$\prod_{(j,k)\in M} S_{(j,k)} \sim \prod_{l\in \text{odd}} K_l \tag{4}$$

since any even power of the link operators yields the identity.

For each vertex j we can also consider the quantity $\nu_j = \sum_{l\in j} n_l$, where the sum is over the three links incident upon j. Since all vertices are the endpoint of a single string, the quantity ν_j will be odd for all j. As such, an odd number of odd links must be incident upon each vertex. Since all vertices are trivalent, this implies that there will also be an even number of even links incident upon each vertex.

Due to this property, the set of even links corresponds to a path (or set of paths) that form a loop (or loops). The plaquettes may then be bicoloured according whether or not they are enclosed by the set of loops. These colours are referred to as 'black' and 'white', and the corresponding sets of plaquettes are denoted b and w.

Two plaquettes are neighbouring if their boundaries share at least one link. If any of these is an odd link, the two plaquettes will clearly be enclosed by the same loop of even links. They will therefore be of the same colour. Similarly, two neighbouring plaquettes that do not share any odd links will be separated by a loop of even links, and so will be different colours.

Given this bicolouring, we can find similar relations to Eq. (4) for the plaquettes

$$\prod_{p\in b} W_p \sim \prod_{p\in w} W_p \sim \prod_{l\in even} K_l.$$

It is straightforward to see that $\prod_l K_l \sim \mathbb{1}$, and so the product over even links above can be substituted with a product over odd ones. Combining with Eq. (4) we then find the following relation between the three types of stabilizer

$$\prod_{p\in b} W_p \sim \prod_{p\in w} W_p \sim \prod_{(j,k)\in M} S_{(j,k)}. \tag{5}$$

As such, only $N-2$ of the N stabilizer generators are independent. The stabilizers may therefore be used to define a stabilizer code that can store two logical qubits. However, this is not the approach we will take.

Note that the choice of the paths $P(j,k)$ used to define the stabilizer generators does not alter the stabilizer as a whole. The product of an $S_{j,k}$ defined using a path $P(j,k)$ and an $S'_{j,k}$ defined using a path $P'(j,k)$ will be the product of the W_p enclosed by the resulting loop. These alternate $S'_{j,k}$ can therefore equally be regarded as stabilizers. The paths are only required in order to define the bicolouration of the lattice.

Examples of Matching Codes

A simple subclass of matching codes are those for which only nearest neighbour pairing are allowed, and the path used is always the single link between the vertices. The case of z-links used as string stabilizers on the honeycomb lattice is shown in in Fig. 2 (a). Such models are well known from the Abelian phase of the honeycomb lattice model [9]. They are known to support the $D(\mathbb{Z}_2)$ anyon

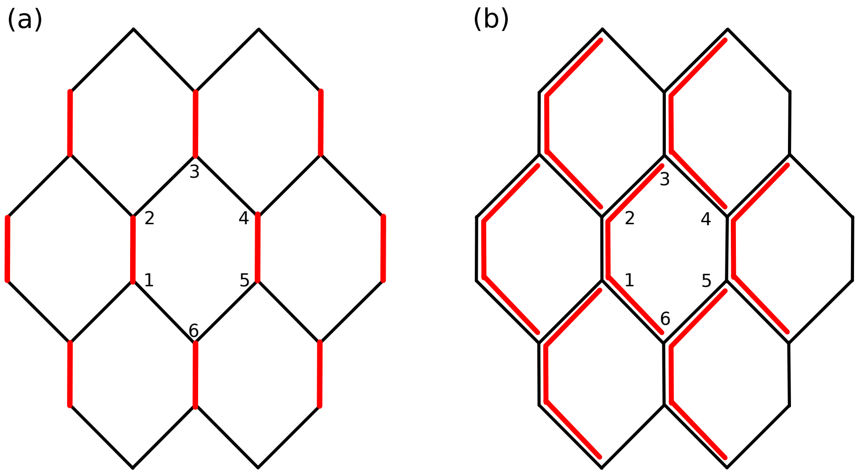


FIG. 2. Thick red lines on the hexagonal lattice denote paths on which stabilizers are defined. (a) String stabilizers are defined on z-links. (b) String stabilizers correspond to the left half of each plaquette.

model [2], and map to the standard square lattice toric code when violations of string stabilizers are prohibited [9]. All other such models behave similarly, though they map to toric codes on different lattices in general.

Another example of a matching code is one for which the top and bottom qubits of each plaquette are paired, with the path chosen to be the three links on the left side of the plaquette. This is shown in Fig. 2 (b). In this case, each string stabilizer is associated with a plaquette, and so can be interpreted as a second plaquette operator

$$S_p = \sigma_6^y \sigma_1^x \sigma_2^y \sigma_3^x. \tag{6}$$

Rather than considering the usual W_p operators, we can use the following operators as stabilizer generators

$$S'_p = S_p W_p = \sigma_3^y \sigma_4^x \sigma_5^y \sigma_6^x. \tag{7}$$

This effectively splits each hexagonal plaquette into two square plaquettes, each with its own plaquette operator [5]. The resulting code corresponds to the Wen plaquette model [10], which is equivalent by local unitaries to the standard square lattice surface code [11].

Planar Boundary Conditions

Periodic boundary conditions are assumed in the above for the simplicity of translational invariance. Practically, however, planar boundary conditions are more appropriate [1]. These are also possible for matching codes.

As an example, consider a Wen plaquette model with planar boundary conditions. On the boundary it is difficult to apply the above interpretation of the model as a matching code. However, in the bulk the interpretation can be easily applied. The matching, M, for the vertices of the bulk is well defined. The specific matching of the Wen plaquette model is not required for consistency of the bulk and boundary, and so any M may be used. We are therefore free to choose any matching code for the bulk, with an assurance that this will be consistent with the standard boundary.

ANYON MODEL

The stabilizer states of the matching codes can be interpreted in terms of anyonic quasiparticles. An anyon is said to reside on a plaquette p for a state $|\,\psi\rangle$ if $W_p\,|\,\psi\rangle = -\,|\,\psi\rangle$, and similarly on a string $P(j,k)$ if $S_{j,k}\,|\,\psi\rangle = -\,|\,\psi\rangle$. An eigenvalue of $+1$ is interpreted as anyonic vacuum in both cases.

In the examples of matching codes above, the anyon model corresponds to $D(\mathbb{Z}_2)$. This is true of all matching codes, as we will now show. To do this we consider stabilizer states, which correspond to states of definite anyon configurations. The effect of applying Pauli rotations to the stabilizer states is to map between them, corresponding to the creation, transport and annihilation of anyons. Using these operators we can therefore determine the braiding and fusion properties of the anyon model.

Anyons on strings

First let us consider the application of link operators. For a link $l = (j, j')$ the link operator K_l will commute with all stabilizers if (j, j') is a pairing of M. Otherwise it will anticommute with the two string stabilizers $S_{j,k}$ and $S_{j',k'}$.

Applying the link operator to a state with definite eigenvalues for these stabilizers will have the effect of flipping the eigenvalue from $+1$ to -1, or vice-versa. The corresponds to the creation (annihilation) of an anyon pair if the two strings initially held vacuum (anyons). If only one string initially held an anyon, the effect is to move it to the other.

Consider a graph for which all the pairs of vertices in M are combined to form single vertices, with each new vertex inheriting the edges from both its predecessors. The anyons created by link operators can be thought of as residing on the vertices of this graph. The anyons can be moved between pairs of vertices connected by an edge. Since the original lattice had no disjoint subgraphs, this property is inherited by the new lattice. It is therefore possible for an anyon to move from any string stabilizer to any other. As such, the same species of anyon must live on all string stabilizers. Let us use ϵ to denote this anyon type. Since it is created in pairs, it is clear that it obeys the fusion rule $\epsilon \times \epsilon = 1$.

Consider three vertices, i, j and k, where both i and k are neighbours of j. We choose this triplet such that each is the endpoint of a unique string operator, which is always possible in general. For j the other endpoint is at vertex j', which has a neighbour j''. Let us start with a state in which both the strings of i and k hold an ϵ, whereas that of j does not. Applying the link operator $K_{i,j}$ will move the ϵ on the string of i to that of j. Applying $K_{j',j''}$ then moves it again to the string of j''. Applying $K_{j,k}$ moves the ϵ on the string of k to that of j, and $K_{i,j}$ moves this on to the string of i. Finally, $K_{j',j''}$ followed by $K_{j,k}$ moves the ϵ on the string of j'' to that of k. The final effect is an exchange of the two ϵ anyons. The total operator used to realize this is

$$K_{j,k}K_{j'j''}K_{i,j}K_{j,k}K_{j',j''}K_{i,j} = K_{j,k}K_{i,j}K_{j,k}K_{i,j} = -\mathbb{1}.$$

This demonstrates that exchange leads the wave function to acquire a phase of -1. The ϵ anyons are therefore fermions.

Anyons on plaquettes

To determine the anyon types living on plaquettes, consider a link l shared by two neighbouring plaquettes p and p'. We use $\alpha \in \{x, y, z\}$ to denote its type and j to denote either of the vertices that it is incident upon. The application of σ_j^α to the qubit on vertex j will anticommute with the plaquette operators of p and p' and no others. It therefore corresponds to the creation of a pair of plaquette anyons. However, we must consider also the additional effects on the string stabilizers.

If l is an odd link, it follows that the other two links incident upon j will either be both odd or both even. The support of the product of all string stabilizers on j will therefore be $\mathbb{1}$ or σ^α, respectively. In either case this product will commute with the applied operation σ_j^α. It therefore results in the creation of an even number of ϵ anyons.

If l is an even link, it follows that exactly one of the other two links incident upon j will be odd. This results in a product of all string stabilizers that anticommutes with σ_j^α. The application of this rotation therefore creates an odd number of ϵ anyons.

From these properties we see that creating a pair of anyons on neighbouring plaquettes of the same colour (i.e., separated by an odd link) will result also in the creation of an even number of ϵ anyons. These may then be removed by the application of link operators. An operation that creates a pair of plaquette anyons same coloured neighbouring plaquettes with no additional ϵ anyons will therefore always exist. This clearly holds also for non-neighbouring plaquettes of the same colour. All anyons on same coloured plaquettes therefore belong to the same anyon type.

For different coloured plaquettes (separated by an even link) there will always be an odd number of ϵ anyons created. The application of link operators can reduce these to a single ϵ, but this cannot be removed. Anyons that live on different coloured plaquettes must therefore belong to different species, which differ up to fusion with an ϵ. We use e to denote the anyons on white plaquettes and m for those on black. These properties can be summarized in the fusion rules

$$e \times e = m \times m = \epsilon \times \epsilon = 1, \ \ e \times m = \epsilon. \tag{8}$$

These rules completely generate the fusion rules of the model. These are the fusion rules of the $D(\mathbb{Z}_2)$ anyon model. This has three non-trivial anyon types. One is a fermion, which we have already shown to be ϵ. The two others have bosonic exchange properties with respect to themselves, and semionic with respect to each other.

MAJORANA MODE INTERPRETATION

Studies of the honeycomb lattice model often make use of a mapping of the problem from qubits to Majorana modes [9]. Specifically, the qubit operators are mapped to Majorana operators by associating four Majorana modes to each vertex. Three of these modes for each vertex are absorbed into a lattice gauge theory. With the remaining Majorana mode c_j at each vertex j, the link operators can be expressed,

$$K_l = (ic_j c_k)\, u_{jk} \tag{9}$$

Here the u_{jk} term comes from the lattice gauge theory. The $ic_j c_k$ term is the parity operator for the Dirac mode that is composed of the two Majorana modes c_j and c_k.

Due to the property $c_j^2 = 1$ for Majorana modes, the string stabilizers may be expressed

$$S_{j,k} \sim (ic_j c_k) \prod_{l \in P(j,k)} u_{jk}. \tag{10}$$

This is the parity operator for the Dirac mode that consists of the Majoranas at the endpoints of the string $P(j,k)$, again with a factor from the gauge theory. The plaquette operators consist only of gauge theory terms with no Majoranas.

This form for the stabilizers agrees exactly with what we know about their anyonic occupations. The string stabilizers are parity operators for a Dirac mode, since their eigenvalue ± 1 signals the presence or absence of a fermion.

The path dependent factor comes from the fact that these Dirac fermions can decay into an e and an m. The occupation of their Dirac modes will therefore depend on the bicolouring of the lattice, which depends on the paths taken by the string stabilizers.

If we consider only states for which there is always vacuum on the plaquettes, the path dependence of the string stabilizers is effectively removed. The system can then simply be interpreted as one for which a single Majorana mode is pinned to each vertex. The string stabilizers simply correspond to an arbitrarily chosen pairing of the Majorana modes, and are defined as the parity operator for the Dirac mode of the pair.

The stabilizer states are those for which there is a definite pairing of all the Majorana modes, and each of the corresponding Dirac modes have a definite occupation. For these states, the Majorana interpretation is nothing more than a mathematical curiosity. However, as we shall see in the next section, it is possible to use these Majoranas to store quantum information, and to braid them in order to process this information. Their full nature as non-Abelian anyons can therefore be realized. As such the matching codes, which would normally be interpreted as Abelian models, may also be interpreted and used as non-Abelian anyon models.

Note that this interpretation is related to the concept of 'twists', which behave as Majorana modes [7, 9]. These have been considered in many cases including surface codes [7, 8, 12] and the honeycomb lattice model [13]. These can be used for quantum computation, and have better resource usage than other surface code based approaches [14].

The study of twists usually considers Majorana modes as being associated with lattice dislocations. The Majoranas are created and annihilated by deforming the lattice on which the code is defined. The Majorana modes of the matching codes, on the other hand, do not require such a complex interpretation. They are always present and do not require lattice deformations. This makes the matching codes a useful framework in which to study these defects.

BRAIDING OF MAJORANA MODES

Let us consider only the anyonic vacuum state (i.e. the stabilizer space) for all stabilizers. In order to add some degeneracy into the stabilizer space, we remove some of the link stabilizer operators from the stabilizer. The corresponding Majoranas are then unpaired. Parity operators can be defined using pairings of these. Different pairings correspond to different bases in which the corresponding stabilizer space states can be measured. We refer to these unpaired Majoranas as computational Majoranas, and the rest as background pairs.

In order to achieve some degree of quantum computation with the computational Majoranas, we must be able to braid them with each other. Since all Majoranas are pinned to vertices, we cannot move them freely. However, we can hope to perform exchange operations that result in their effective movement around the lattice. This can be done using so-called 'anyonic state teleportation' [15].

The simplest possible operation is to exchange a computational Majorana, c_j, with a neighbouring background pair. Two examples of this are shown for the honeycomb lattice in Fig. 3.

The process shown in Fig. 3 (a) exchanges the computational Majorana on vertex i with the background pair in the link (j, k), resulting in the computational Majorana moving to vertex k. To do this we measure the link operator $K_{i,j}$, which will result in a random outcome. The resulting state will be an eigenstate of $K_{i,j}$, and will no longer be stabilized by $S_{j,k}$. The set of pairs,

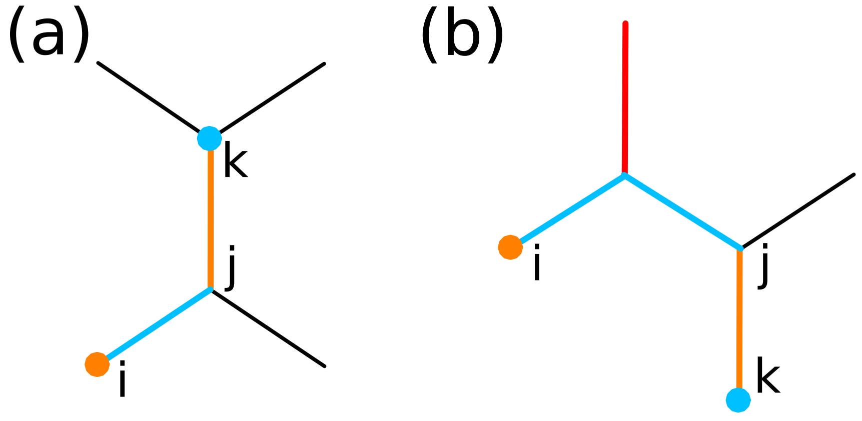


FIG. 3. Two processes by which a computational Majorana on vertex i is moved to vertex k. The path between j and k (orange) corresponds to a stabilizer of the initial state in both cases. The path between i and j (blue) corresponds to a stabilizer of the final state. In (b) the additional vertical line (red) is a stabilizer for both the initial and final states.

M, is therefore altered by replacing the pair (j,k) with (i,j). If the outcome of the measurement is $+1$, the resulting state holds vacuum for the new stabilizer $S_{i,j} = K_{i,j}$ as well as all previous stabilizers. This process corresponds to the background pair on (j,k) moving to (i,j), and the computational Majorana on i moving to the other side of this background pair. All other Majoranas remain stationary. For the outcome -1 the process has the same effect, except that an ϵ is fused with both the moved background pair and computational Majorana. To undo the effect of this, the link operator $K_{j,k}$ is applied. The state will then be the same as if the outcome was $+1$.

For a bipartite lattice, the above process will only allow computational Majoranas to move around the same sublattice. In order for them to move between sublattices, exchanges such as in Fig. 3 (b) must be applied. The only difference between this and the above is that the measured operator $S_{i,j}$ is a product of two link operators.

The exchange of a Majorana with a background pair is a trivial operation, since the net sector of a such a pair is vacuum. Nevertheless the operation would be expected to yield a global phase that depends on the chirality of the exchange. However it is evident that this phase, and the distinction between clockwise and anti clockwise exchanges around a background pair, does not arise in the process described above. As such the braid statistics realized by the Majoranas are projective braid statistics, which are the same as the full braid statistics up to global phases [16].

With the ability move the computational Majoranas around, we can consider the effects of their braiding. Since this involves moving background pairs out of the way, different paths taken by Majoranas between two points will result in different configurations of the background pairs. In order to avoid any ambiguity this may cause when assessing the effects of the exchange, we will consider only braid operations for which the initial and final states have the same background configuration.

For concreteness we will use the modified honeycomb lattice. The only string stabilizers that will be considered are those for the pairing of nearest neighbours, and so are defined on a single link. The set of pairs, M, then becomes the set of links for which there are stabilizers. We use S_l to refer to the stabilizer for the links $l \in M$. This lattice is not bipartite, and so the Majoranas may be moved between any pair of vertices by measuring single link operators only.

One possible choice of links for M is simply all those connected by z-links, and so each z-link operator becomes a stabilizer. Using this as a background, we will define a code for which computational Majoranas are placed on a 1D line.

Consider a 1D row of vertical z-links for which every dth z-link is chosen to be removed from the stabilizer. We call these 'flagged' links. For the jth flagged link from the left, we use j to refer to the lower vertex. For every odd j a path $P(j, j+1)$ is then found, such that the jth and $j+1$th flagged links the are first and last links of the path, and the path has endpoints on the vertices j and $j+1$. Furthermore, the links of the paths should alternate between links that are part of the original pairing (i.e. z-links) and those which are not. The set of pairings M is then modified by removing all z-links along these paths, and adding all other links along them. This results in a code for which computational Majoranas are located on all vertices j. An example of this for $d = 1$, and so all links along the row removed, is shown in shown in Fig. 4(a).

For each neighbouring pair of computational Majoranas $(j, j+1)$ we require a path in order to define the corresponding parity operator. Rather than used the same paths used to generate the code above, we use an alternative one which we denote P_j. Such a path is shown in 4(a). These paths are chosen such that they alternate between links that are within M and those that are not, except at a mid-point at which two links not in M will be connected. The parity operator is then defined,

$$\pi_{j,j+1} = i \prod_{l \in P_j} K_j. \tag{11}$$

Here the product is taken sequentially along the path. The j end corresponds to the right-most factors and the $j+1$ end to the left-most.

There will always be an even number of links along the path, both for the specific example of 4(a) and those for which the distance between computational Majoranas is increased. This means the parity operators have support on an odd number of qubits, and so the factor of i is required to ensure that their eigenvalues are ± 1. It also means that

$$\pi_{j,j+1} = -\pi_{j+1,j}, \tag{12}$$

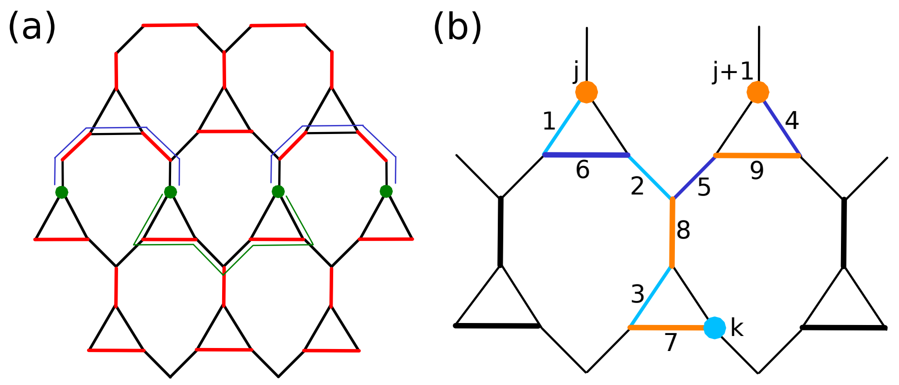


FIG. 4. Stabilizers on the modified honeycomb lattice that lead to a line of unpaired Majoranas. (a) Thick red links correspond to pairings within M. The paths $P(j, j+1)$ along which the Majoranas are created pass above the Majorana line. These are shown with thin blue lines. The path P_j along which parity operators are defined goes below the Majorana line. This is shown with a thin green line. (b) Numbering of links used for the exchange operation. Those in bold are part of M. Different coloured links are used at different points in the exchange process. Links 1, 2 and 3 are coloured light blue, 4, 5 and 6 are dark blue, 7, 8 and 9 are orange.

where the latter is the same product of link operators but in reserved order. This reflects the $ic_j c_{j+1} = -ic_{j+1} c_j$ behaviour expected for the parity operators when the order of the Majorana operators is reversed. The commutation relations for parity operators with each other are also what we expect: those for disjoint pairs of Majoranas will commute, whereas those that share a single Majorana will anticommute.

We explicitly consider the exchange of the two Majoranas shown in green in Fig. 4(b), using $|\,\psi\rangle$ to denote the initial state. To perform a anticlockwise exchange we first move the Majorana at j away from its initial position. Specifically, we move it diagonally downwards and towards the right to the vertex k. This is done by sequential applications of the anyonic teleportation procedure on the light blue edges. The resulting state is

$$|\,\psi'\rangle \sim \prod_{l \in b} (\mathbb{1} + K_l)\,|\,\psi\rangle\,. \tag{13}$$

Here b denotes the set of light blue links. The factors $(\mathbb{1} + K_l)$ are projectors onto subspaces stabilized by the link operators for these edges. The product is again taken sequentially as the Majorana is moved.

Next we move the Majorana at $j+1$ to j. This is done by moving it diagonally downwards and to the left until it intersects with the previous path. It is then moved up that path until it reaches j. The teleportation for this case is achieved using the dark blue link operators. The resulting state is

$$| \psi'' \rangle \sim \prod_{l \in B} (\mathbb{1} + K_l) \, | \psi' \rangle \tag{14}$$

where B denotes the set of dark blue links. Finally the Majorana at k is moved to $j+1$ using the orange links. The final state is

$$U \, | \psi \rangle \sim \prod_{l \in O} (\mathbb{1} + K_l) \prod_{l \in B} (\mathbb{1} + K_l) \prod_{l \in b} (\mathbb{1} + K_l) \, | \psi' \rangle \tag{15}$$

Now we must determine the effective unitary U that is implemented by this exchange.

Note that the initial and final states are stabilized by the same set of link operators, M, which corresponds to the z-links. Also note that $O \subset M$. The above may therefore be rewritten

$$U \, | \psi \rangle \sim \prod_{l \in M} (\mathbb{1} + K_l) \prod_{l \in B} (\mathbb{1} + K_l) \prod_{l \in b} (\mathbb{1} + K_l) \prod_{l \in M} (\mathbb{1} + K_l) \, | \psi' \rangle \, .$$

The factor $\prod_{l \in B} (\mathbb{1} + K_l) \prod_{l \in b} (\mathbb{1} + K_l)$ will yield a sum of products of the blue link operators. Some of the terms in this will anticommute with the link operators of M, whereas others will commute. The effects of the former will be removed by the $\prod_{l \in M} (\mathbb{1} + K_l)$ factors. The latter consists of only two terms: the identity, and the product of the blue links along the path P_j,

$$\prod_{l \in b \cup P_j} K_l \prod_{l \in B \cup P_j} K_l = - \prod_{l \in B \cup P_j} K_l \prod_{l \in b \cup P_j} K_l . \tag{16}$$

The order of the product over b has the links at the j end of P_j to the left. The B product has the $j+1$ end to the left. However, since both factors act on an even number of qubits, their order can be reversed without effect. The r.h.s. of the above can therefore be written as a product sequentially along P_j.

This term includes all of the factors of $\pi_{j,j+1}$ that are not in M. There will always be an even number of such factors, due to the symmetry of the path. As such, this term is related to the parity operator by

$$\prod_{l \in B \cup P_j} K_l \prod_{l \in b \cup P_j} K_l = -i \pi_{j,k} \prod_{l \in M \cup P_j} K_l . \tag{17}$$

The resulting unitary evolution of the exchange can then be written

$$U \, | \psi \rangle = (\mathbb{1} + i \pi_{j,j+1}) \, | \psi \rangle \ \therefore \ U = \frac{\mathbb{1} + i \pi_{j,j+1}}{\sqrt{2}} . \tag{18}$$

Apart from a global phase of $e^{i\pi/8}$, this is exactly the expected result for the exchange of two Majoranas [3, 9].

For the clockwise exchange of j and $j+1$, the mirror image of the above process is applied. Since the ordering of factors in each product is reversed, this will yield the effect

$$U^\dagger = \frac{1}{\sqrt{2}} (\mathbb{1} - i \pi_{j,j+1}) . \tag{19}$$

Again, this is exactly the expected result from Majorana braiding, up to a global phase.

In order to fuse a pair of computational Majoranas, we simply need to determine the occupation of their net Dirac mode. This is done by adding their parity operator into the stabilizer and measuring it. For practical reasons, the pair should be moved close together in order for this operator to be measurable.

Minimal Demonstration of Braiding

The simplest example of an exchange of two computational Majoranas is shown in Fig. 5. The system is composed of six qubits, corresponding to three adjacent z-links in the honeycomb lattice and the x and y links that connect them. The only stabilizer is $S = K_C$. The link operators $\pi_A = K_A$ and $\pi_B = K_B$ are taken to be parity operators for their respective pairs. The fusion space of the computational anyons is four-dimensional. A basis $\{|\, k_A, k_B\rangle\}$ can be defined using the eigenstates of the parity operators, labelled by the eigenvalues $k = \pm 1$

A clockwise exchange of the computational Majorana at 1 with that at 2 can be achieved by first measuring K_D, then K_E, and then finally the stabilizer $S = K_C$. In a proof of principle experiment, we may simply post-select the results for each of these gives the outcome $+1$. Single and double exchanges have the following effect on the basis states for the computational Majoranas

$$\begin{aligned} R :\, |\, k_A, k_B\rangle &\rightarrow \frac{1}{\sqrt{2}} (|\, k_A, k_B\rangle + i\, |\, {-k_A}, -k_B\rangle) \\ R^2 :\, |\, k_A, k_B\rangle &\rightarrow |\, {-k_A}, -k_B\rangle \end{aligned} \tag{20}$$

By preparing and then measuring these basis states, the effects of the braiding may then be shown.

This system may be made simpler still by noting that the unnumbered qubits contribute trivially. The z-link operators may then be truncated onto the numbered qubits: $S = \sigma_2^z$, $\pi_A = \sigma_1^z$ and $\pi_B = \sigma_2^z$. The basis states for the computational Majoranas are then simply the z basis states, and measurement of the stabilizer is a single qubit z measurement. The whole process then corresponds to preparation of z basis states for three qubits, followed by two entangling two qubit measurements, followed by z basis measurements. Using this process, the basic principle behind the non-Abelian braiding of the computational Majoranas may be demonstrated.

Adiabatic Realization of Braiding

In the above it is always assumed that the code is used as part of a circuit model quantum computation. As such, it is done without implementing the Hamiltonian associated with the stabilizer code [17]. However, one could also

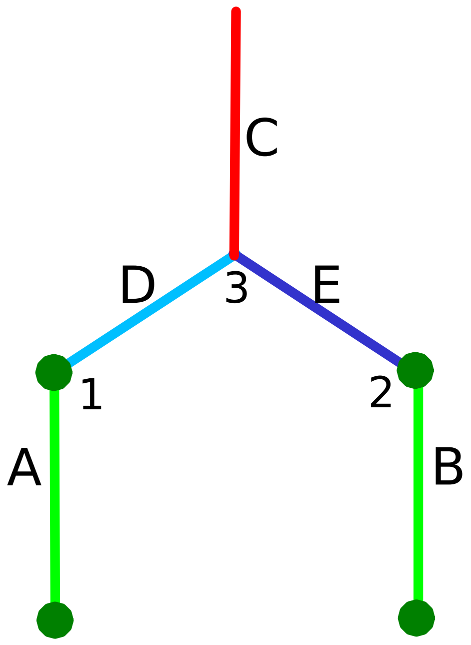


FIG. 5. Two pairs of computational Majoranas, with parity operators given by links A and B. Link C denotes a stabilizer.

consider the case in which this Hamiltonian is indeed used to enhance the error suppression of the code.

The Hamiltonian corresponding to a stabilizer code with stabilizer generators S_j is typically

$$H_S = -\sum_j S_j. \tag{21}$$

This results in a degenerate ground state space that corresponds to the stabilizer space of the code, and energetically penalizes states with non-trivial syndrome. For the matching codes, this Hamiltonian would need to consist of all plaquette and string stabilizers, which are many-body interactions in general. However, a more simple form arises in the case that all matchings correspond to nearest neigbour pairs, and all so all string stabilizers are link operators. In this case we can consider the Hamiltonian

$$H = -J\sum_{l\in M} K_l - h\sum_{l\notin M} K_l, \;\; h \ll J \tag{22}$$

This Hamiltonian consists only of nearest neighbour two-body interactions between qubits. These correspond to link operators, with a higher coupling for the links within the matching than those not. When the effects of the latter as a perturbation on the former are studied, the many-body plaquette interactions emerge [9]. The Hamiltonian will then have the same ground state and as H_S above, with a finite gap [9]. There will be some differences in the excitation spectrum, but these have only local effect. This Hamiltonian corresponds to the Abelian phase of the honeycomb lattice model, from which the basic matching codes are derived.

To create unpaired Majorana modes, implement the Hamiltonian for the relevant set of links, M. This was also studied previously in [13]. To move the unpaired modes, one makes the corresponding alterations to the set M as described in previous sections. The only difference is that this change in M is no longer implemented on the physical system by measuring link operators. Instead it is done by slowly changing the Hamiltonian. To remove a link l from M, and add another link l' in its place, the coupling from M can be slowly tuned from J to h. That for l' can then be raised from h to J. If this is done slowly enough to obey the adiabatic theorem, no additional fermions will arise during the process as they do in the circuit model case.

By this means, unpaired Majorana modes can be engineered, transported and braided using the Abelian phase of the honeycomb lattice model. This is something usually associated with the non-Abelian phase, for which the Majorana modes are pinned to vortex excitations. Using this method, holonomic quantum computation can be implemented [6, 18, 19] using the Majorana modes.

Note that the Majorana modes that emerge from a Hamiltonian in this way are similar to those that arise in nanowires [20]. However protection in the nanowire case is based on fermionic-parity conservation, whereas matching codes have fully topological protection. This has similarly been achieved by a different (but also surface code based) approach in [21].

COLOR CODES FROM MATCHING CODES

Matching codes realize the $D(\mathbb{Z}_2)$ anyon model. However, they can also be used to construct further codes which realize more complex anyon models, through a process of embedding one code inside another. Here we will consider this for the specific example of the celebrated color code [22].

For any stabilizer code we can consider the basis formed by stabilizer states. In this basis we can associate a two-level quantum system with each stabilizer generator. The basis states for each of these correspond to its $+1$ and -1 eigenspaces.

Typically, when considering a stabilizer code, we require all these two-level systems to be in a definite state: the $+1$ eigenspace of the corresponding stabilizer. However, we could instead consider only imposing this restriction on some stabilizers. The others could then be used as if they were qubits themselves to define another stabilizer code. In this way we can embed codes within other codes.

This method is in some ways similar to concatenation. However, the purpose

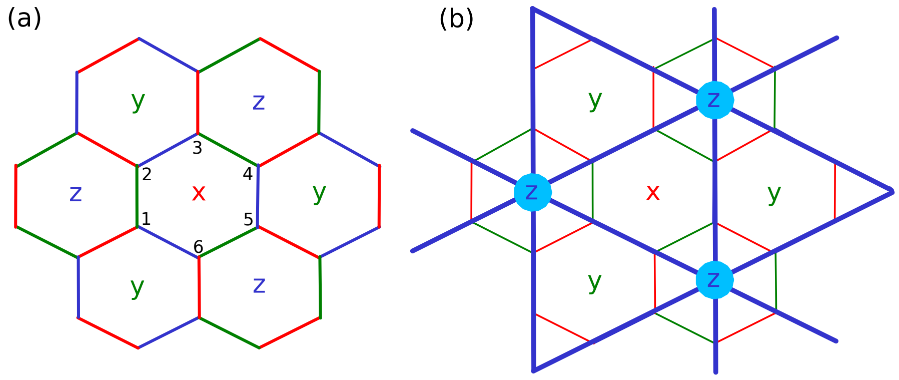


FIG. 6. (a) Hexagonal lattice with plaquettes tricoloured x, y and z. Links are tricoloured correspondingly, with x and y links alternating around a z plaquette, etc. Again red, blue and green denote x, y and z, respectively. (b) The hexagonal lattice with a triangular lattice superimposed on top. The z-links in the former correspond to edges in the latter, x and y plaquettes in the former are plaquettes in the latter, and z plaquettes in the former are vertices in the latter.

is very different. Concatenation combines many low distance codes to create a code of higher distance. For the embedding considered here we start with a single code, which could have arbitrarily high distance, such as surface or matching codes. The purpose of the embedding is simply to generate a new code, which may have more favourable properties than the original.

For a specific example, let us again consider a honeycomb lattice. Rather than use the same tricolouration of links as before, we will instead use that of Fig. 6 (a). For this we define the matching code in which all z-links are used as string stabilizers.

With this tricolouration of links we can also consider a tricolouration of plaquettes, as shown in Fig. 6 (a). The plaquette operators then take the form

$$W_{p_\alpha} = \sigma_1^\alpha \sigma_2^\alpha \sigma_3^\alpha \sigma_4^\alpha \sigma_5^\alpha \sigma_6^\alpha. \tag{23}$$

where p_α is an α-plaquette for $\alpha \in \{x, y, z\}$.

We will use the plaquette stabilizer operators as generators in our embedded code, but not those of the z-links. As such each z-link can be thought of as a two-level quantum system in its own right, with basis states spanned by the vacuum and ϵ occupancies of the link. We will refer to these as ‘link qubits’.

These links qubits can be interpreted as sitting on the edges of an triangular lattice, as shown in Fig. 6 (b). The plaquettes of the original hexagonal lattice of qubits correspond to either plaquettes or vertices of the triangular lattice of link qubits. Specifically the x- and y-plaquettes of the honeycomb lattice are plaquettes of the triangular lattice and the z-plaquettes are vertices.

Let us now define a surface code on the triangular lattice of link qubits, such that we have a surface code embedded in a matching code. When defined on a lattice with qubits on edges, a surface code has stabilizers

$$B_p = \prod_{j \in v} \sigma^x_j, \;\; A_v = \prod_{j \in p} \sigma^z_j. \tag{24}$$

For the lattice of links we must determine what link qubit operators will be used in place of these. This means finding substitutes of σ^z and σ^x.

Let us associate the σ^z basis with the ϵ occupancy basis, and so use the string operators K_j as a substitute for σ^z on the link qubits. The B_p stabilizers of the surface code defined on link qubits then correspond to operators

$$B_{p_\alpha} = \sigma^z_1 \sigma^z_2 \sigma^z_3 \sigma^z_4 \sigma^z_5 \sigma^z_6 \tag{25}$$

on the qubits on the honeycomb lattice for $\alpha \in \{x, y\}$.

The substitute for the σ^x operators must clearly anticommute with K_j, our substitute for σ^z. It will therefore have the effect of changing the ϵ occupancy of the link qubit on which it acts. The A_v operator will then do this for all links around the (triangular lattice) vertex on which it acts. One way to achieve this is to use the product of all x-links around the vertex. The A_v stabilizers of the surface code then correspond to operators

$$A_{p_z} = \sigma^x_1 \sigma^x_2 \sigma^x_3 \sigma^x_4 \sigma^x_5 \sigma^x_6 \tag{26}$$

on the qubits of the corresponding p_z plaquette in the honeycomb lattice.

Given the above construction, each plaquette of the honeycomb lattice will have two corresponding stabilizer generators. One is the W_{p_α} of the underlying matching code and the other is the A_{p_α} or B_{p_α} of the embedded surface code. Each are isotropic tensor products of Pauli operators on the six qubits around the hexagonal plaquette, and each correspond to a different Pauli operator. These two Paulis are not the same for every plaquette.

To simplify the description of the code, let us first introduce a third stabilizer operator for each plaquette. This simply corresponds to a product of the other two. Each plaquette then has three stabilizer operators: $(\sigma^x)^{\otimes 6}$, $(\sigma^y)^{\otimes 6}$ and $(\sigma^z)^{\otimes 6}$. Clearly these can also be generated by an alternate pair of stabilizer generators for each plaquette

$$S^\alpha_p = \sigma^\alpha_1 \sigma^\alpha_2 \sigma^\alpha_3 \sigma^\alpha_4 \sigma^\alpha_5 \sigma^\alpha_6, \;\; \alpha \in \{x, y\}. \tag{27}$$

With these the two generators are the same for every plaquette. These are precisely the stabilizer generators of the well-known color code [22].

It is known that the anyon model of the color code is $D(Z_2 \times Z_2)$, which is equivalent to two independent copies of $D(\mathbb{Z}_2)$ [23]. However, note that these do not simply correspond to the indepedent anyon models contributed by the matching code and surface code. Instead these two anyon models are combined and reorganized by the embedding procedure. For example, in the original matching code the anyons that live on x and y plaquettes are of the same type, and can be moved from one position to the other. After the embedding, however, the plaquette anyons in these positions will correspond to different types. Any attempt to move an anyon from one position to the other will result in the creation of anyons within the surface code.

It is also important to note that many properties of the color code are not be the same as other codes that realize the $D(\mathbb{Z}_2 \times \mathbb{Z}_2)$ anyon model, such as two independent surface codes. One important difference is the wealth of transversal gates possible for color codes [22]. The technique of embedding codes within others may therefore yield other new codes with advantageous properties.

CONCLUSIONS

Codes that realize the $D(\mathbb{Z}_2)$ anyon model are well known, as is the notion that non-Abelian Majorana modes (also known as Ising anyons) may be engineered inside these Abelian codes. However, the means by which the Majoranas are introduced can often be both theoretically and practically inelegant. Often deformations of the lattice are required, making it seem that the Majoranas are a foreign notion introduced to the codes by hand. Also, moving the Majoranas around the code leaves a visible trail along the path they have taken. This obscures their topological nature as non-Abelian Ising anyons, for which paths should be unphysical.

Here we introduce a family of codes in which the Majorana modes occur in a more natural way. The codes can be interpreted as having the Majoranas present at all times, pinned to the vertices of the lattice, without the need for deformations to introduce them. They can be moved around using anyonic state teleportation, allowing their non-Abelian braiding to become evident. The Majoranas will still leave visible trails behind when moved. However, this is simply an artifact of the way braiding is performed by teleportation. The same would occur for any realization of Ising anyons when braiding is performed using this method.

The matching codes should hopefully make it easier to treat these Majorana modes, both theoretically and perhaps even in the lab. For the latter, these codes allow a proof of principle experiment for Majorana braiding to be done with just three qubits and only one and two qubit operations.

Beyond the Majorana interpretation, these codes could also serve another practical use. The potential that $D(\mathbb{Z}_2)$ codes have for quantum computation is well recognized, though most focus is on the square lattice planar variant of the surface code. However, it is known that this does not provide the best protection against every error model [24–26]. The matching codes therefore provide a new family of codes to consider for the optimization of error correction against physical error models.

ACKNOWLEDGMENTS

The author would like to thank Benjamin Brown, Tobias Fuhrer and Daniel Loss for discussions that were crucial in shaping this project, and also Konstantinos Meichanetzidis for pointing out a reference. This work was supported by the Swiss National Science Foundation and NCCR QSIT.

[1] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, J. Math. Phys. **43**, 4452 (2002).
[2] A. Y. Kitaev, Ann. Phys. **303**, 2 (2003).
[3] J. K. Pachos, *Introduction to topological quantum computation* (Cambridge University Press, Cambridge, UK, 2012).
[4] J. R. Wootton, Journal of Modern Optics **59**, 1717 (2012).
[5] J. R. Wootton, V. Lahtinen, Z. Wang, and J. K. Pachos, Phys. Rev. B **78**, 161102 (2008).
[6] J. R. Wootton, V. Lahtinen, B. Doucot, and J. K. Pachos, Annals of Physics **326**, 2307 (2011), ISSN 0003-4916.
[7] H. Bombin, Phys. Rev. Lett. **105**, 030403 (2010).
[8] Y.-Z. You, C.-M. Jian, and X.-G. Wen, Phys. Rev. B **87**, 045106 (2013).
[9] A. Kitaev, Ann. Phy. **321**, 2 (2006).
[10] X.-G. Wen, Phys. Rev. Lett. **90**, 016803 (2003).
[11] B. J. Brown, W. Son, C. V. Kraus, R. Fazio, and V. Vedral, New Journal of Physics **13**, 065010 (2011).
[12] B. J. Brown, S. D. Bartlett, A. C. Doherty, and S. D. Barrett, Phys. Rev. Lett. **111**, 220402 (2013).
[13] O. Petrova, P. Mellado, and O. Tchernyshyov, Phys. Rev. B **90**, 134404 (2014).
[14] M. B. Hastings and A. Geller (2015), arXiv:1408.3379.
[15] P. Bonderson, M. Freedman, and C. Nayak, Phys. Rev. Lett. **101**, 010501 (2008).
[16] M. Barkeshli, C.-M. Jian, and X.-L. Qi, Phys. Rev. B **87**, 045130 (2013).
[17] B. Brown, D. Loss, J. K. Pachos, C. N. Self, and J. R. Wootton (2014), arXiv:1411.6643.
[18] C. Cesare, A. J. Landahl, D. Bacon, S. T. Flammia, and A. Neels (2014), arXiv:1406.2690.
[19] Y.-C. Zheng and T. A. Brun (2014), arXiv:1411.4248.
[20] A. Kitaev (2001), arXiv:cond-mat/0010440.
[21] B. M. Terhal, F. Hassler, and D. P. DiVincenzo, Phys. Rev. Lett. **108**, 260504 (2012).
[22] H. Bombin and M. A. Martin-Delgado, Phys. Rev. Lett. **97**, 180501 (2006).
[23] H. Bombin, G. Duclos-Cianci, and D. Poulin, New Journal of Physics **14**, 073048 (2012).
[24] B. Röthlisberger, J. R. Wootton, R. M. Heath, J. K. Pachos, and D. Loss, Phys. Rev. A **85**, 022313 (2012).
[25] A. Al-Shimary, J. R. Wootton, and J. K. Pachos, New Journal of Physics **15**, 025027 (2013).
[26] A. Kay, Phys. Rev. A **89**, 032328 (2014).

Chapter 3

Demonstrating non-Abelian braiding of surface code defects in a five qubit experiment

Abstract

Currently, the mainstream approach to quantum computing is through surface codes. One way to store and manipulate quantum information with these to create defects in the codes which can be moved and used as if they were particles. Specifically, they simulate the behaviour of exotic particles known as Majoranas, which are a kind of non-Abelian anyon. By exchanging these particles, important gates for quantum computation can be implemented. Here we investigate the simplest possible exchange operation for two surface code Majoranas. This is found to act non-trivially on only five qubits. The system is then truncated to these five qubits, so that the exchange process can be run on the IBM 5Q processor. The results demonstrate the expected effect of the exchange. This paper has been written in a style that should hopefully be accessible to both professional and amateur scientists.

Publication Information

Originally published in *Quantum Science and Technology*, Volume 2, Number 1(2017) [6].

Note that the references of this chapter are self-contained, and not included in the overall references.

INTRODUCTION

Surface codes are the most well known starting point for fault-tolerant quantum computation [1]. One way to store and manipulate information in these to engineer certain kinds of defects [2, 3]. These can be moved around and manipulated in much the same way as particles. However, their restriction to the 2D structure of the surface codes allows them to exhibit some of the exotic behaviour possible for particles in 2D universes.

The important features of these particles are their their effects when fused and braided. These refer to the processes of combining particles and of exchanging the positions of particles, respectively. It has been predicted that the fusion rules for these defects identical to so-called Majorana modes or Ising anyons. Their braiding forms a projective representation of the braid group, but is otherwise also identical to that of Majoranas. We will therefore refer to them simply as Majoranas from henceforth.

In this paper we present an experiment performed on a small patch of surface code. The patch is nevertheless large enough for these defects to be introduced and even for pair of them to exchange positions. We implement this and show that the effects of the process are consistent with the braiding of Majoranas, as predicted.

ANYONS

There are many types of particle in the universe, but their properties when exchanged place them into just two categories: bosons and fermions. Neither of which are very complex. This is because a loop around a point in three spatial dimensions can be continuously deformed to a loop that is not around a point. One can simply pick the loop up and place it elsewhere. As such, all topological properties of one particle moving in a loop around another must be trivial. There are only two ways to achieve this: bosonic and fermionic exchange behaviour.

In a two dimensional universe, this no longer holds. There is no longer an 'up' to move the loop through, and so it is stuck around the point. The braiding of particles may therefore have non-trivial properties. This would allow almost any type of exchange. Such particles are therefore known collectively as *any*ons [4]. For a simple introduction, see [5].

Though we do not live in a two-dimensional universe, we can make two dimensional physical systems. This allows us to find phases of matter in which anyons arise as quasi-particles or other localized features of the system than can be moved and otherwise manipulated in the same manner as particles [4, 6].

MAJORANAS

The most interesting families of anyons are the non-Abelian anyons. One example is the Ising anyon model. This holds two types of particle: one known as a Majorana, and the other a fermion denote by ψ [4].

The fermion type ψ is its own antiparticle. Combining two of these will always result in annihilation.

The Majoranas are their own antiparticle, so a pair of them can be created from vacuum. When combined, these would annihilate back to vacuum. However, it is also possible to obtain a pair of Majoranas from the decay of a ψ. These would recreate the fermion if brought back together.

These two hypothetical pairs of Majoranas are completely indistinguishable. Their memory of whether to annihilate or form a fermion is not stored in any locally accessible feature. Instead it is stored non-locally through quantum entanglement in the underlying physical system.

This non-locality makes these particles an attractive proposition for storing quantum information in a quantum computer. By associating a pair that annihilate with a bit value 0, and a pair that form a ψ with bit value 1, they can be used to store a bit (or qubit). By keeping the Majoranas well separated, it would take a large and concerted effort for errors to affect the bit (by moving the particles together to read out the value, for instance). As such, the information will have an inherent robustness against errors, as long as error correction is performed to detect the errors before they build up [7–9].

Now consider two pairs of Majoranas created from vacuum. If one Majorana from one pair is combined with one from the other, there is no shared history to determine the result. The pair will therefore randomly choose to either annihilate or become a ψ.

Since everything initially came from vacuum, to vacuum they must return. Combining the remaining pair of Majoranas must then always yield the same result as for the first pair, such than any ψ formed by one will be joined by its antiparticle from the other. It is this process, and the resulting correlation, that is the key to the experiment we will perform.

The exchange of Majoranas, which is simply swapping the positions of two of the anyons, also allows for many interesting effects. For one thing, and as we will show experimentally, it can be used to move particles to be combined with strangers, and so exhibit the effect described above as well as many more complicated versions.

Also, suppose we take again the two pairs of Majoranas created from vacuum. One Majorana from one pair is moved around a Majorana from the other. Combining the two pairs will now result in both forming a ψ, and hence flipping the bits that they are storing. This is certainly a non-trivial effect that bosons and fermions could only dream of. It demonstrates the power of non-Abelian anyons to not only store information, but to process it too.

Note that the fault-tolerance described above applies only when the Majoranas can be kept well separated, and when error correction is performed. The small nature of our experimental set-up prevents the possibility of either. As will be seen later, this means that our results will not be free of noise of the effects of noise.

For another approach towards exchanging Majoranas, as well as a summary of all other experimental progress towards Majoranas in a variety of physical systems, see [10, 11] and references therein.

QUBITS

Qubits are the quantum analogue of bits. Just as a bit can be either 0 or 1, a qubit can be either $|\,0\rangle$ or $|\,1\rangle$. But it can also be any quantum superposition of the two, such as,

$$|\,\psi\rangle = a\,|\,0\rangle + b\,|\,1\rangle\,,$$

for any arbitrary complex numbers a and b [12]. For a simple introduction, see [13].

The states $|\,0\rangle$ and $|\,1\rangle$ are referred to as the Z basis of the qubit. If we measure to see whether the qubit is $|\,0\rangle$ or $|\,1\rangle$, this is called a Z measurement.

Two particular examples of possible superpositions are the states $|\,+\rangle$ and $|\,-\rangle$, which are known as the X basis states,

$$|\,\pm\rangle = \frac{1}{\sqrt{2}}\,|\,0\rangle + \frac{\pm 1}{\sqrt{2}}\,|\,1\rangle\,.$$

These are orthogonal states: and as different from each other as $|\,0\rangle$ is from $|\,1\rangle$. This means we can also measure whether a qubit is $|\,+\rangle$ or $|\,-\rangle$, known as an X measurement. Similarly we may also define so-called Y basis states, and a Y measurement.

The manipulation of qubits can be understood using the circuit model. In this work we will be considering the same circuits as used in the IBM Quantum Experience, and so refer to that for an introduction [14].

The most notable operation applied in these circuits is the CNOT. This acts on two qubits, with one called the 'control' and the other the 'target'. The effect is to add the Z basis value of the control to that of the target. However, since the target is restricted to the two states 0 and 1, only the parity of the sum is retained. The target is therefore left in the state 0 if the sum was even, and 1 if odd. If the target was initially in the $|\,0\rangle$ state, the effect is to copy the Z basis state of the control to the target. The CNOT gate is a reversible equivalent of the XOR gate used in electronic circuits.

In the IBM 5Q processor there are five qubits, labelled Q_0 to Q_5. The only CNOTs that can be applied are those with Q_2 as the target. In order to allow more flexibility in circuit design, we can implement a CNOT with Q_2 as control by applying the Hadamard gate, H, on both control and target before and after the standard CNOT is applied.

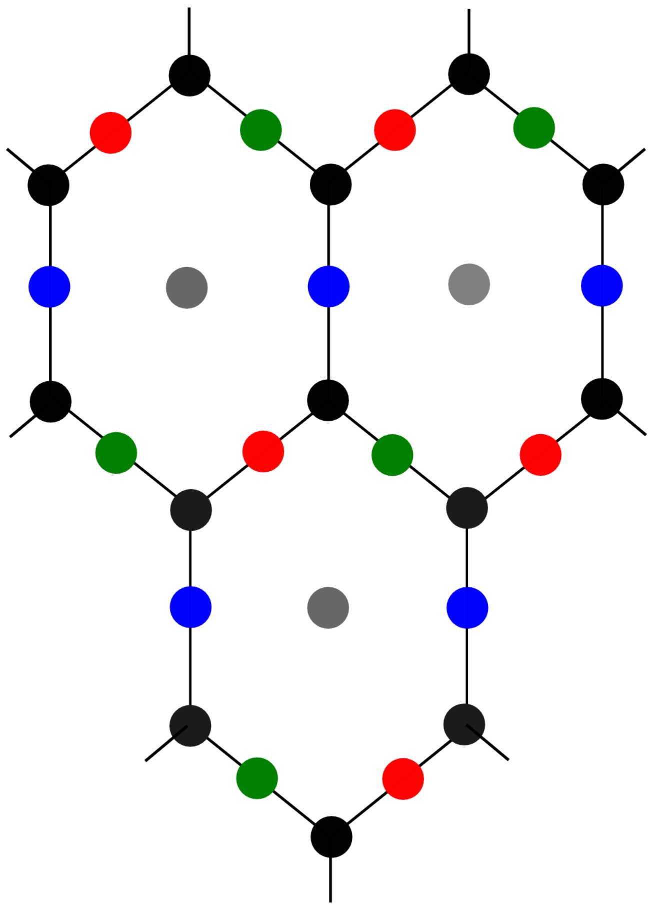


FIG. 1. The code is defined on a hexagonal lattice, of which a section is shown. Vertex qubits are shown in black and hexagon in grey. Edge qubits are red, green or blue depending on whether the edge is associated with a XX, YY or ZZ measurement, respectively.

MATCHING CODE

The grid shown in Fig. 1 is an example of a matching code [15], which is a type of quantum error correcting code [16] that is based on the honeycomb lattice model [4]. It is a variant of the surface code [1].

In this code there is a qubit located at each vertex, edge and hexagon. The vertex qubits are those that are truly part of the code. The edge and hexagon qubits are there to make measurements of the vertex qubits.

We associate a measurement with each edge. Each of these measures a collective property of the two qubits on the vertices connected by the edge.

For the vertical edges we have a so-called ZZ measurement. There are two

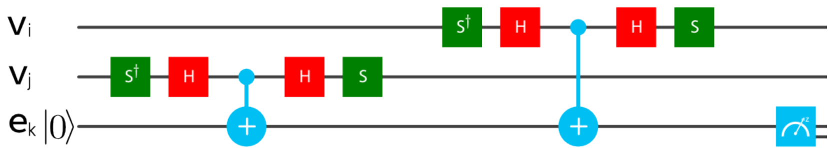


FIG. 2. In order to implement an ZZ measurement on two vertex qubits V_i and V_j using edge qubit e_k, the above circuit is applied using only the blue gates. The first CNOT copies the Z basis state of v_j to e_k. The second adds the Z basis state of v_i. If their states in the Z basis are the same, the sum is even and so e_k is left in the state $|0\rangle$. If they are different, the sum is odd and so e_k is left in the state $|1\rangle$. The measurement of e_k then outputs the corresponding bit value 0 or 1. An XX measurement is the same, except that the red gates are added. These rotate the control end of the CNOTs such that they look at the X basis states, rather than the Z. For YY both the red and green gates are added to rotate to the Y basis.

possible outcomes, which correspond to whether or not the state of the qubits different in the Z basis. If both qubits are $|0\rangle$, or both are $|1\rangle$, the measurement outputs 0 to show there is no difference. The same will happen for a superposition of both being $|0\rangle$ and both being $|1\rangle$, and the measurement will not disturb this superposition. On the other hand, if one qubit is $|0\rangle$ and the other is $|1\rangle$, the measurement will output 1 to show that there is a difference.

The other two edges correspond to XX and YY measurements. These are the same as a ZZ measurement, except that they compares whether the states are the same (output 0) or different (output 1) in the X and Y basis, respectively. The circuits that implement these measurements are shown in Fig. 2.

Measurements that share a vertex qubit do not commute. This means that making a measurement of one will affect the outcome of the other. For example, suppose we prepare a state such that the ZZ measurement on the light blue edge of Fig. 1 will give the outcome 0 with certainty. We then measure the XX shown in light red. Subsequently measuring ZZ will then randomly output 0 or 1, due to the effect of the non-commuting XX. Further explanation of this can be found in [17].

The measurements of the hexagons similarly have output either 0 or 1. These measure a collective property of all six qubits on the vertices around the hexagon. However, we need not consider these measurements explicitly for what follows.

All edge measurements commute with all hexagon measurements. We wish to find a set of measurements that all commute with each other, and for which there are as many measurements in the set as there are vertices. The simplest choice is to take all ZZ measurements and all hexagon measurements.

These will form the so-called *stabilizer*, which defines our quantum code. We will prepare and work with states for which the outcome is 0 to all stabilizer measurements. Measuring any 1 will then be clue that an error has occurred. However, we will not be considering error correction in what follows.

ANYONS IN A QUANTUM CODE

We consider states for which all stabilizer measurements yield the outcome 0 with certainty. Through the effects of noise, or of our own effort, we can then create states in for which some measurements yield 1. However, it is impossible to create a single isolated 1 in some area by solely manipulating qubits in its neighbourhood. Instead, such local manipulations will always cause at least two stabilizers to yield 1. This is similar to the property of particles, which cannot be created or annihilated in isolation. At least one other particle is required to serve as its antiparticle.

It has been found that stabilizers yielding a 1 behave as particles in all other ways also. They are therefore quasiparticles that arise from collective properties of the underlying qubits. And these quasiparticles behave as anyons.

The quasiparticles associated with hexagons correspond to the e and m anyons of the surface codes [1]. However, we will not be concerned with these in what follows.

The quasiparticles residing on the edge stabilizers correspond to fermions. In fact, they behave exactly the same as the ψ fermions of the Ising model. It is therefore interesting to determine whether these can be caused to decay into a pair of Majoranas, and whether these Majoranas can be moved, braided and used as non-Abelian anyons.

This can indeed be done. By changing the definition of the stabilizer close to the ψ we wish to split, we can push the two constituent Majoranas apart. This is shown in Fig. 3, and proven rigorously in [4, 15]. This is equivalent to the twist deformations of [2].

While performing this process, it is important to keep track of the results of the new stabilizers. Each result of 1 implies that the Majorana being moved has decayed into a Majorana and a ψ during the process, and so must be recombined. To avoid the need to account for such stray fermions, we will consider only results for which these measurements all yield the result 0.

MINIMAL EXCHANGE OF MAJORANAS

The process shown in Fig. 3 exchanges two Majoranas. The process of this exchange was introduced in [15], but is also explained in the caption of the figure.

The implementation of the exchange acts non-trivially only on three vertex qubits and two edge qubits. We may therefore truncate the lattice to this small area, and use a five qubit process to implement the exchange.

The truncated system is shown in Fig. 4. In common with other truncated surface code experiments [19–21], not all stabilizers have full support on the truncated area. They must therefore be reduced to the part with support on this area. Most notably, the three vertical links incident on the area only have one vertex within it. The ZZ measurements required for the full system then become single Z measurements. The fermionic occupancy of these three links

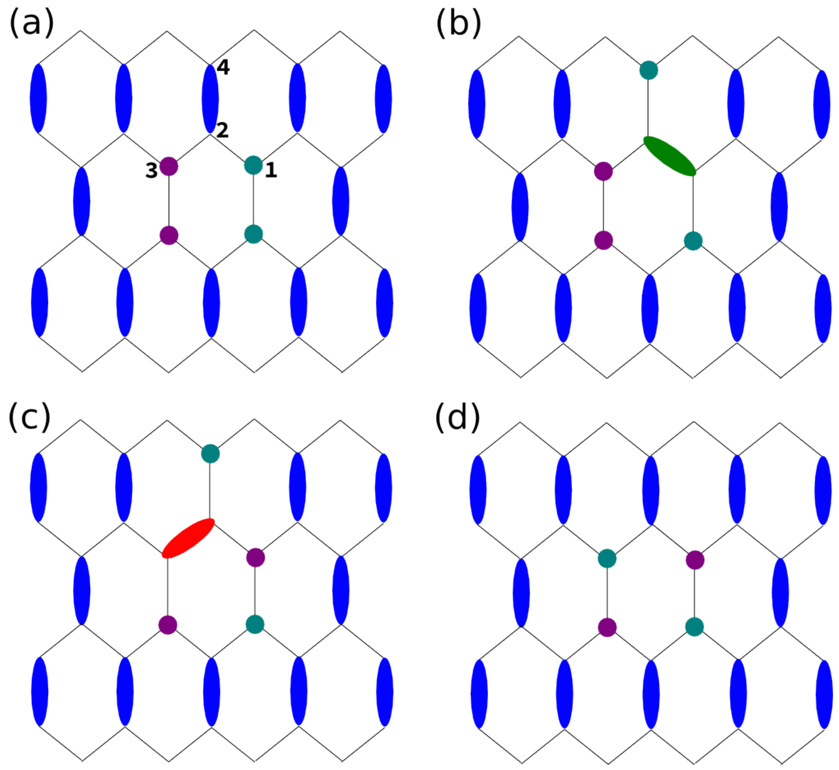


FIG. 3. In (a) we see the code with ZZ stabilizers shown in blue. The corresponding pairs of Majoranas can be associated with the vertices. These two pairs are shown explicitly for two ZZ stabilizers shown in purple and teal. (b) To move the green Majorana on vertex v_1 we add the adjacent YY measurement to the stabilizer. Since this does not commute with the ZZ measurement above, it is also removed from the stabilizer. The YY measurement binds the Majoranas on vertices v_1 and v_2 into a well-defined fermionic mode, whereas that on v_4 becomes unbound. The Majorana initially at v_1 is therefore effectively teleported to v_4 [15, 18]. (c) An XX measurement is similarly used to teleport the purple Majorana initially at v_3 to v_1. (d) Finally the initial ZZ measured, and so returned to the stabilizer. This moves the teal Majorana from v_4 to v_3, completing the exchange.

therefore corresponds exactly to the Z basis state of the three vertex qubits. Accordingly, the initial state is simply that for which all vertex qubits are $|0\rangle$. The initial state of the edge qubits, as always, should also be $|0\rangle$.

The most straightforward implementation of this process is shown in Fig. 5. Here the two edge qubits are used to make the required XX and YY measurements.

This circuit cannot be run on the IBM 5Q processor due to its restricted topology. Because of this we must use a single qubit as the intermediary for both the XX and ZZ measurements. In order for the results of the two mea-

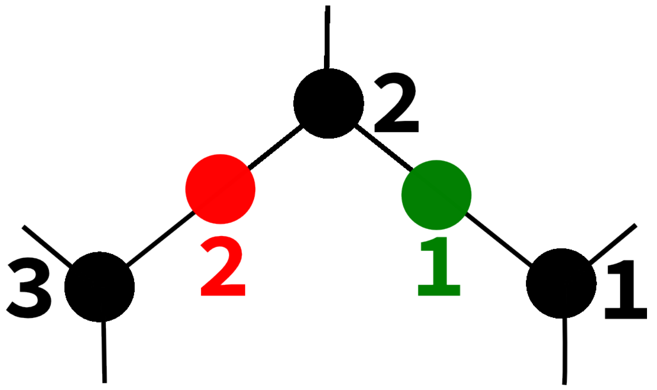


FIG. 4. The system used for the exchange is reduced to the five qubits on which the process acts non-trivially. The ZZ measurements are reduced to single Z measurements on the vertex qubits. The pairs of Majoranas that residing on these edges therefore now reside fully on the vertices. The Z measurements determine whether they would combine to annihilate or form a fermion, just as the ZZ measurements did previously. Other edge operators remain unchanged.

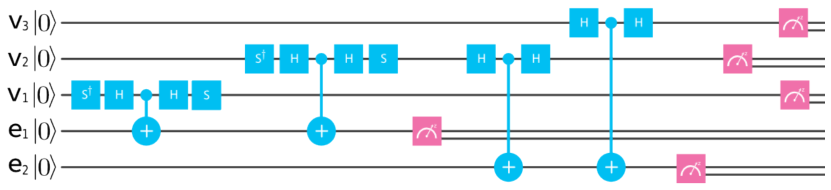


FIG. 5. The ideal circuit first performs the YY measurement of v_1 and v_2 using e_1, and then the XX of v_2 and v_3 using e_2. The Z measurement of v_2 is then performed. Afterwards, Z measurements are made on v_1 and v_3 to verify the expected correlations.

surements to be distinguished, the result of the YY measurement is copied onto an otherwise unused qubit using a CNOT.

It is also important to apply the circuit as quickly as possible, and to assign the qubits to their tasks based on the noise levels of their entangling gates and also their lifetimes. Doing do results in the final circuit as shown in Fig. 6.

RESULTS

The circuit of Fig. 6 was implemented using the IBM 5Q processor, using the interface provided by the IBM quantum experience. The success of the circuit is categorized by the correlation function,

$$C = P(00) + P(11) - P(01) - P(10).$$

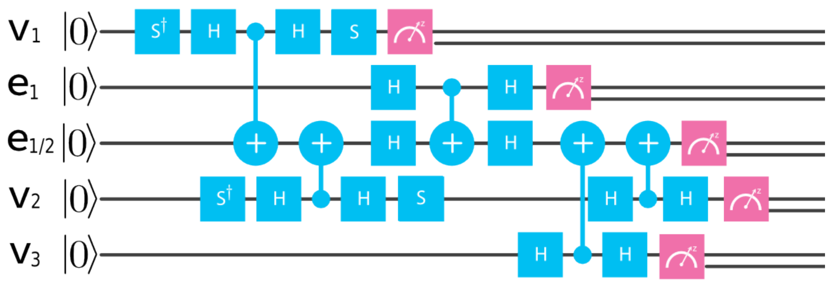


FIG. 6. The YY measurement is made using $e_{1/2}$. But rather than measuring this qubit, the result is copied to e_1 for measurement. The XX measurement is then made using $e_{1/2}$. The measurement result of this will be the summ of the XX and YY measurements, but the YY outcome can be inferred using that of XX. Each operation is applied as soon as the previous one on the same qubit is complete in order to avoid unnecessary noise. The qubits Q_0 to Q_1 of the processor are listed from top to bottom. The choice of which should play the role of each v_j and e_j was determined by the T_1, T_2 and e_g^{jk} parameters from the $2016-09-21$ calibration, in order to minimize noise.

Here $P(01)$ is the probability that the measurement of v_1 yields 0 and v_3 yields 1, both in the case of no stray fermions. The theoretical prediction for the ideal case is of perfect correlations, corresponding to $C = 1$. In contrast $C = 0$ would correspond to a lack of correlations and $C = -1$ to perfect anticorrelation.

A simulation of the circuit under noiseless conditions showed exactly the result expected: $C = 1$. A simulation under realistic conditions yields $C = 0.454$.

Running the circuit for 24576 shots on the IBM 5Q processor yields $C = 0.530$. This is in good agreement with the theoretical prediction made under realistic conditions. It is also well above zero, and so clearly demonstrates the expected correlations.

One possible problem with this result is that decay to the $|0\rangle$ state on v_1 and v_3 could lead to a false positive. To test this, tomography on the state is performed. Specifically, one can consider the Majoranas to encode a single logical qubit. The exchange has the effect of rotating this qubit to a Y basis state. The logical X operator corresponds to the product of the XX and YY link operators, and so to an YZX operator on v_1, v_2 and v_3. Respectively. Since the logical Z is simply a single Z on v_1 or v_3, the Y will be XZX or YZY respectively.

Since these three qubit measurements are performed at readout, superpositions in the system need not be preserved. We may therefore do three single qubit measurements and calculate the corresponding correlators afterwards. This simply requires additional rotations on v_1 and v_3 of Fig 6 to access the correct basis. The results for each are obtained from 8192 shots.

With these results, we can reconstruct the density matrix of the resulting state and compute its fidelity to the required Y basis state. It is found that this fidelity is 70.6% when using YZY. The closest point on the surface of the Bloch sphere has a fidelity of 94.0% to the required state. This is therefore a very encouraging result.

When using XZX the state obtained has a fidelity of only 54.6%, and that of the closest point on the Bloch sphere is 67.4%. The poor result in this case is most likely due to the longer time needed to measure the qubit $v_1 = Q_0$, which has the lowest lifetime of all the qubits.

All data can be found at [22].

CONCLUSION

The exchange of two Majoranas causes the state of two Majorana pairs with definite fusion result to become one with indefinite but correlated results. In this work we showed that this could be realized on a surface code based architecture using only five qubits. The experiment was then performed using the IBM 5Q processor. The results obtained demonstrate the expected correlations.

With higher fidelity qubits, this work could be extended by applying two exchanges. The effect of a full braid of one Majorana around another could then be demonstrated, as well as the effect of undoing the first exchange by one in the opposite direction. The orthogonality of the results in these two cases will provide an even starker demonstration of the braiding of non-Abelian anyons.

It would also be interesting to see this experiment reproduced with spin qubits. This could use the process proposed in [23] or the system introduced in [24].

ACKNOWLEDGEMENTS

We acknowledge use of the IBM Quantum Experience for this work. The views expressed are those of the author and do not reflect the official policy or position of IBM or the IBM Quantum Experience team. The author was supported by the NCCR QSIT.

[1] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, J. Math. Phys. **43**, 4452 (2002).
[2] H. Bombin, Phys. Rev. Lett. **105**, 030403 (2010).
[3] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton (2016), arXiv:1609.04673.
[4] A. Kitaev, Ann. Phys. **321**, 2 (2006).
[5] J. R. Wootton, *Anyons* (2016), URL `http://decodoku.blogspot.ch/2016/06/anyons.html`.
[6] A. Y. Kitaev, Ann. Phys. **303**, 2 (2003).
[7] J. R. Wootton, J. Burri, S. Iblisdir, and D. Loss, Phys. Rev. X **4**, 011051 (2014).
[8] C. G. Brell, S. Burton, G. Dauphinais, S. T. Flammia, and D. Poulin, Phys. Rev. X **4**, 031058 (2014).

[9] A. Hutter and J. R. Wootton, Phys. Rev. A **93**, 042327 (2016).

[10] J.-S. Xu, K. Sun, Y.-J. Han, C.-F. Li, J.K. Pachos, and G.-C. Guo, Nat. Comms. **7**, 13194 (2016).

[11] R. Pawlak, M. Kisiel, J. Klinovaja, T. Meier, S. Kawai, T. Glatzel, D. Loss, and E. Meyer, npj Quant. Inf. **2**, 16035 (2016).

[12] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000).

[13] J. R. Wootton, *Qubits with the simplest maths possible* (2016), URL `http://decodoku.blogspot.ch/2016/02/the-maths-of-qubits.html`.

[14] IBM, *Quantum experience* (2016), URL `https://quantumexperience.ng.bluemix.net/qstage/#/community`.

[15] J. R. Wootton, Journal of Physics A: Mathematical and Theoretical **48**, 215302 (2015).

[16] D. A. Lidar and T. A. Brun, eds., *Quantum Error Correction* (Cambridge University Press, Cambride, UK, 2013).

[17] J. R. Wootton, *Fun measurements for pairs of qubits* (2016), URL `http://decodoku.blogspot.ch/2016/04/fun-measurements-for-pairs-of-qubits.html`.

[18] P. Bonderson, M. Freedman, and C. Nayak, Phys. Rev. Lett. **101**, 010501 (2008).

[19] J. Kelly *et al.*, Nature **519**, 66 (2014).

[20] A. D. Corcoles *et al.*, Nature Communications **6** (2015).

[21] D. Nigg, M. Müller, E. A. Martinez, P. Schindler, M. Hennrich, T. Monz, M. A. Martin-Delgado, and R. Blatt, Science **345**, 302 (2014).

[22] J. R. Wootton, *Raw data* (2016), URL `https://www.dropbox.com/sh/2o50l9tcl5l0962/AADUv46t_NooxkrnqzdXH2jva?dl=0`.

[23] H.-A. Engel and D. Loss, Science **309**, 586 (2005), ISSN 0036-8075.

[24] M. Veldhorst, C. H. Yang, J. C. C. Hwang, W. Huang, J. P. Dehollain, J. T. Muhonen, S. Simmons, A. Laucht, F. E. Hudson, K. M. Itoh, et al., Nature **526**, 410 (2015), ISSN 0028-0836.

Chapter 4

Hexagonal matching codes with 2-body measurements

Abstract

Matching codes are stabilizer codes based on Kitaev's honeycomb lattice model. The hexagonal form of these codes are particularly well-suited to the heavy-hexagon device layouts currently pursued in the hardware of IBM Quantum. Here we show how the stabilizers of the code can be measured solely through the 2-body measurements that are native to the architecture. The process is then run on 27 and 65 qubit devices, to compare results with simulations for a standard error model. It is found that the results correspond well to simulations where the noise strength is similar to that found in the benchmarking of the devices. The best devices show results consistent with a noise model with an error probability of around $1.5\% - 2\%$.

Publication Information

Originally published in *Journal of Physics A: Mathematical and Theoretical*, Volume 55, Number 29 (2022) [8].

Note that the references of this chapter are self-contained, and not included in the overall references.

INTRODUCTION

The future of quantum computation is built on quantum error correction [1]. It is therefore vital to design and test quantum hardware in a way that ensures compatibility with this process. It is for this reason that recent years have seen many examples of experiments testing the fundamental components of quantum error correction (see [2–8] from the past two years, and references therein), as well as codes designed specifically for hardware architectures currently in development [9–11]. This work represents both of these approaches, with a proof-of-principle implementation of a code designed for current IBM Quantum hardware.

In particular we consider matching codes [12]. These are the Abelian phase of Kitaev's honeycomb lattice model [13], reimagined as a family of stabilizer codes with similar properties to surface codes [14, 15]. They can be defined on any trivalent lattice, but we will consider hexagonal lattices in this work.

Specifically, we consider a code defined on the lattice shown in Fig. 1(a). This is a so-called heavy-hexagonal lattice of qubits [9] , in which a qubit is placed on each vertex and each edge of a hexagonal lattice. The edges of the lattice are labelled x, y and z depending on their orientation. For each edge we define a link operator, $\sigma^\alpha \otimes \sigma^\alpha$, that acts on the two vertex qubits. Here $\alpha \in \{x, y, z\}$ is the link type.

We also associate an operator with each plaquette. This is the product of the link operators that form the boundary of the plaquette. It therefore corresponds to the following tensor product of Paulis on the six code qubits around the plaquette.

$$W = \sigma_0^x \sigma_1^y \sigma_2^z \sigma_3^x \sigma_4^y \sigma_5^z \qquad (1)$$

Here the qubit numbering is that shown in Fig. 1(c)

The stabilizer of the code is generated by the plaquette operators along with a commuting subset of the link operators. For concreteness, in this work we will specifically consider one such subset: that of the z-link operators.

Though the storage and processing of logical information is not within the scope of the current work, it is worth commenting on how this can be implemented in these codes. Logical information could be stored in the stabilizer space of the code, whose nature will depend on the boundary conditions of the lattice. However, it can also be done through careful choice of which link operators to use as stabilizers. For each pair of vertices that are not touched by a link stabilizer, a two-fold degeneracy in the stabilizer space will be opened. The code distance corresponds to the distance between these 'defects'. Furthermore these defects can be moved by changing the set of link operators used as stabilizers [12, 16]. They act as Majorana modes under braiding, allowing the implementation of some Clifford gates in a manner similar to code deformation in the surface code [17].

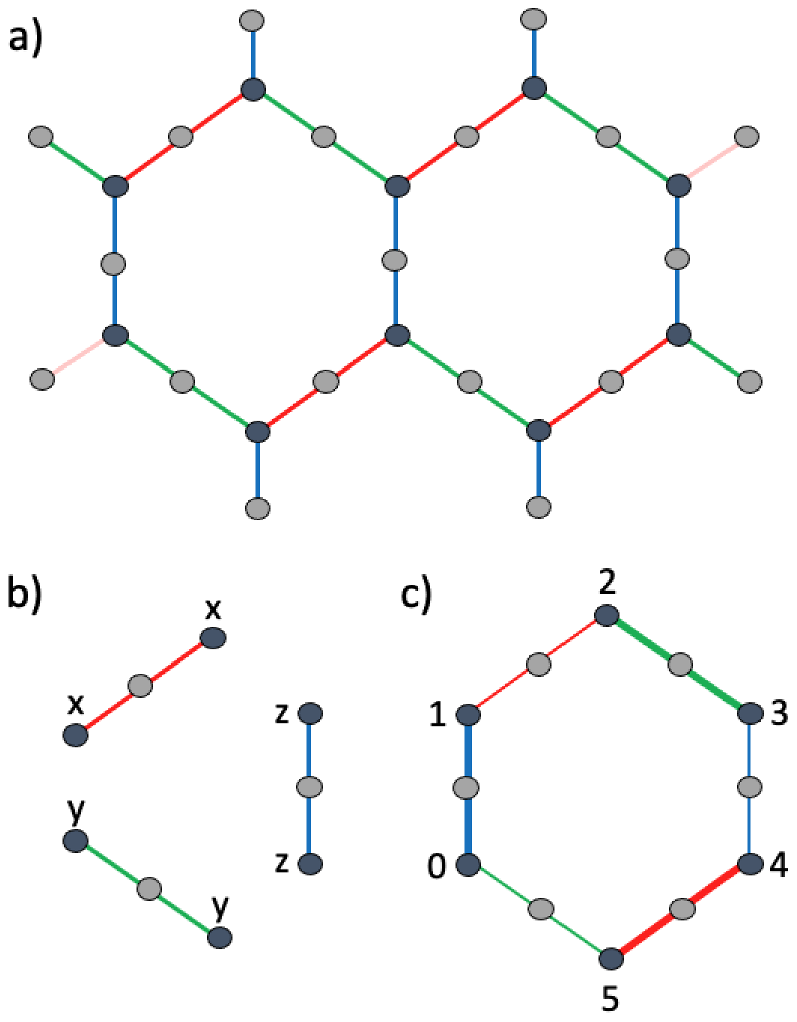


FIG. 1. The layout of qubits and definition of link operators in a hexagonal matching code.

MEASURING STABILIZERS WITH 2-BODY MEASUREMENTS

As with any stabilizer code, the stabilizer generators of the code must be constantly measured. Since the link operators are hermitian, they can be interpreted as observables and measured. Projections onto the eigenspaces of these operators is entangling in general, and so the measurement process requires entangling operations. The simplest means is to use an additional auxiliary qubit. This is entangled with the qubits of the link and then directly measured to give the result. The circuits required for x-, z- and y-link measurements are shown in Fig. 2. The connectivity required for this circuit means that the edge qubits in the heavy-hexagonal code are perfectly placed to be the auxilliary qubit for each link. The stabilizer generators that are link operators can therefore be directly measured in this way.

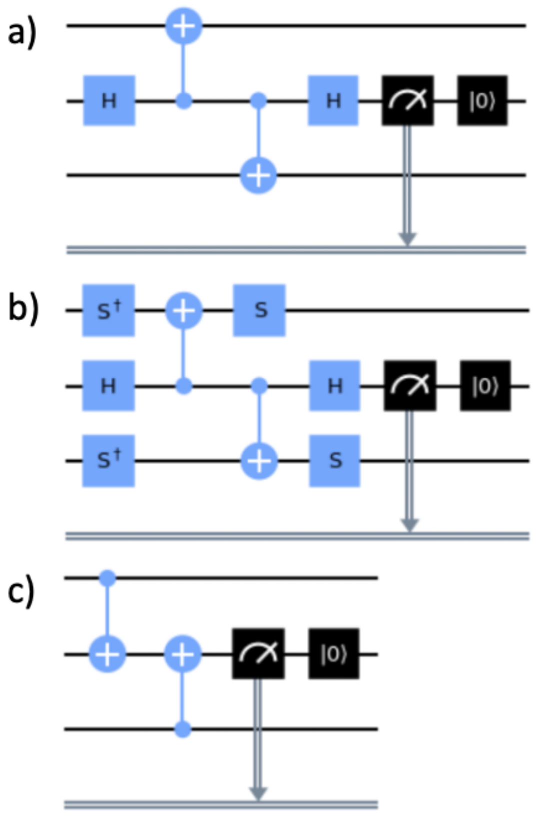


FIG. 2. Circuits to perform parity measurements for (a) x-links, (b) y-links and (c) z-links. The top and bottom qubits in the circuits are the pair for which the parity is measured. The middle qubit is an auxilliary, which should be initialized in the $|0\rangle$ state.

Unfortunately, the 6-body plaquette operators cannot be measured so easily. However, since the plaquette operators can be expressed as products of link operators, it is natural to expect that there could be a way to deduce their measurement from a set of link operator measurements.

The simplest way to do this is to split the 6 edges around each plaquette into two groups. In Fig. 1(c), group a is shown by thin lines, and group b by thick lines. The products of all link operators for each group are

$$\begin{aligned} V_b &= (\sigma_3^z\sigma_4^z)(\sigma_5^y\sigma_0^y)(\sigma_1^x\sigma_2^x), \\ V_a &= (\sigma_0^z\sigma_1^z)(\sigma_2^y\sigma_3^y)(\sigma_4^x\sigma_5^x) \end{aligned} \qquad (2)$$

It is important to note the following features of these operators:

1. They commute with the plaquette operator, which is equal to their product;

2. They commute with all individual link operators around the plaquette;

3. They commute with each other.

From point 1 it follows that measurement of V_a and V_b can be used to deduce the measurement result for the plaquette operator by summing their results modulo 2. From point 2 it follows that measuring the individual link operators in a group can be used to deduce the result for a measurement of the V operator of that group by summing their results modulo 2. From points 2 and 3 it follows that both V_a and V_b can be sequentially measured in this way, without the measurement of the latter invalidating the result of the former. Combining these, we find that the measurement of the plaquette operator can be achieved measured by measuring the link operators of first one group, and then the other, and computing the sum of all results modulo 2.

Note that this process will not commute with link stabilizers incident upon the plaquette. They will therefore need to be subsequently measured and returned to the stabilizer space. The act of measuring a plaquette operator is therefore effectively a form of code deformation [18], which will have the effect of removing the affected stabilizer generators are removed from the stabilizer and replacing them by their product (with with the process does commute). This can lead to a decrease in the distance of the code, since it possible for errors to take shortcuts across the larger stabilizer generator without being detected. To ensure that the distance is not reduced to zero, the plaquettes will need to be measured in shifts such that sets of affected link operators do not overlap. Between the shifts, in which subsets of the plaquettes are measured, measurements of all link stabilizers are made.

A remaining question is such measurements of plaquettes are compatible with fault-tolerance. The fact that these measurements disturb z-link stabilizers means that there is effectively an additional source of noise. This is very strong (effectively applying x- and y-link operators with probability 1/2 whenever one is measured), but also extremely localized (affecting only the measured plaquettes). Since the use of shifts means that each bout of this noise affects only non-overlapping regions, and since it is fully detected by the z-link stabilizers, a suitably calibrated decoded should be able to account for this additional noise source, with the effective loss of distance described above being the main deleterious effect.

EFFECTS OF NOISE

The process described above will now be implemented on multiple suitable IBM Quantum devices with the heavy-hexagon layout. These are the 27 qubit Falcon processors `ibm_cairo`, `ibm_hanoi`, `ibmq_montreal` and `ibmq_toronto`, and the 65 qubit Hummingbird processors `ibmq_brooklyn` and `ibmq_manhattan`. The device layouts are shown in Fig. 3.

Note in Fig. 3 (a) that a subset of the z-links are highlighted. These are those that are incident upon plaquettes that will be measured, but for which the entire z-link is not present on the device. In these cases, the usual $\sigma^z \otimes \sigma^z$ operator can be truncated to just a σ^z operator on the corresponding qubit shown in purple. The measurement of the link operator is therefore achieved by a simple measurement of this qubit. Similar truncated z-links also exist for the device in Fig. 3 (b), where the corresponding qubits are again shown in purple.

The process is repeated over T measurement rounds, with $T = 3$ used. Within each round are the different shifts of plaquette measurements, with each shift following by the measurement of all z-link stabilizers. Each shift consists of the measurement of all link operators within subset a of the corresponding plaquettes, followed by the measurement of those from subset b.

When applied to the standard initial state (of all qubits in state $|0\rangle$), the plaquette operators will not have a definite initial value. The result from the first round will therefore be random for each plaquette. The results will the remain the same for subsequent rounds, except when disturbed by noise. The z-link stabilizers, however, will be affected by the measurements of x- and y-link operators. However, consider the product of all z-link operators incident upon a plaquette, as highlighted for one of the plaquettes in Fig. 3 (b). This product commutes with all the x- and y-link operators of the plaquette. The value of this product immediately before and after the measurement of the corresponding plaquette should therefore be invariant, except when disturbed by noise.

Given these considerations, we will calculate the two quantities for each plaquette, one based on the results of plaquette operator measurements and the other based on the results of z-link measurements. For the former, we consider the probability that the value of each plaquette operator remains the same as for the previous round. This done for each plaquette and each round except the first. The average over all these values, $\langle p_W \rangle$, is then calculated. Similarly we calculate a quantity $\langle p_Z \rangle$ based on the probability that the product of all z-link operators incident upon a plaquette changes values.

The process will also be run for simulated noise. This will be done for the 27 qubit case only. The noise model is the standard one used for threshold calculations of the surface code [19]: every that can go wrong does so with some a probability p. Specifically, bit flip noise with probability p is applied after each reset and prior to each measurement, and two qubit depolarizing noise is applied to each two qubit. Note that additional noise is not applied to the single qubit gates around CX gates. Instead, the combination of these gates is regarded as a single two-qubit gate with a single error.

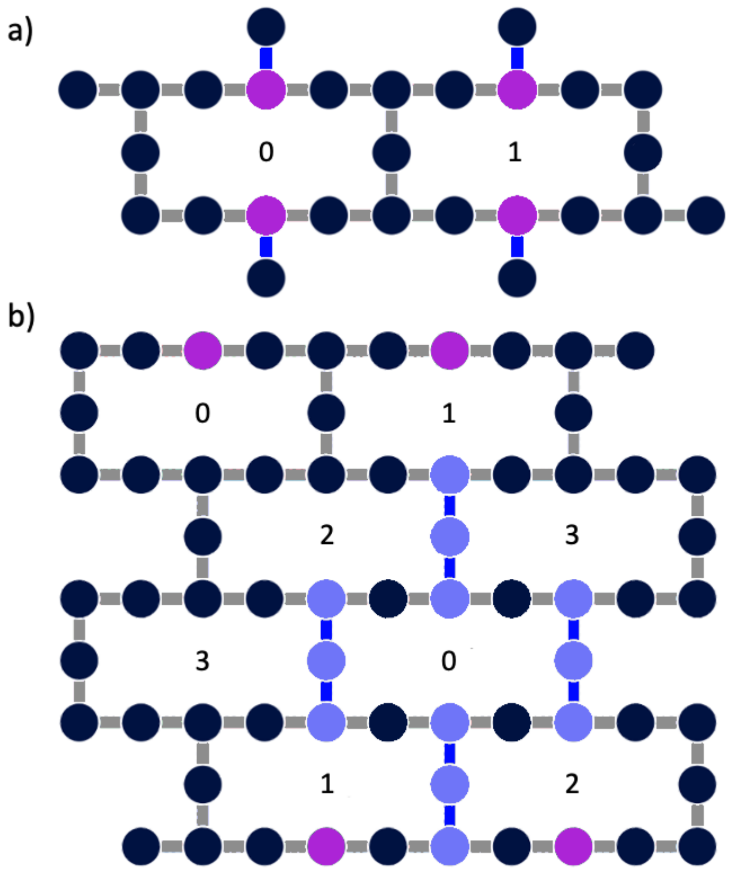


FIG. 3. Layout of (a) 27 qubit Falcon processors and (b) 65 qubit Hummingbird processors. The plaquettes are split into four 'shifts', such that all plaquettes within each group can be measured simultaneously. Each plaquette is labelled by which shift its measurement is a part of.

Note that, since the process is restricted to stabilizer states, noiseless simulations can be highly efficient. As such, the process was run for both the 27 and 65 qubit cases to verify that the calculated quantities are indeed invariant in the noiseless case.

For the runs on real hardware, regular benchmarking of the devices offers measured noise probabilities that can be compared with those of the error model. These probabilities are collected for both preparation and measurement noise on each qubit, as well as the CX noise on all pairs. Specifically, probabilities considered are as follows.

- Preparation noise: For each qubit, the probability `prob_meas1_prep0`.
- Measurement noise: For each qubit, the measurement error probability.

- CX noise: For each pair of qubits for which a CX is possible, the CX error probability.
- Idle noise: For each qubit, the error probability for an identity gate over the timescale of measurement and reset. This is estimated as

$$1 - (1 - 2\,p_{\text{id}})^{(t_{\text{meas}}+t_{\text{reset}})/t_{\text{id}}},$$

  Here p_{id} is the error over the standard timescale t_{id} of an identity gate. The above expression gives the probability for an odd number of such errors over the $t_{\text{meas}} + t_{\text{reset}}$ timescale of measurement and reset.

The mean and standard deviation of this combined list of probabilities is shown for each device in the table below, along with the values of the quantum volume [20].

Device	$\langle p \rangle$	σ	QV
`ibm_cairo`	1.47 %	1.23 %	64
`ibm_hanoi`	1.55 %	1.61 %	64
`ibmq_brooklyn`	4.73 %	5.00 %	32
`ibmq_montreal`	4.41 %	6.00 %	128
`ibmq_toronto`	5.17 %	6.45 %	32
`ibmq_manhattan`	18.3 %	31.9 %	32

Here we see that there is a high degree of variation in the probabilities, with the standard deviation on the order of $\langle p \rangle$ in all cases. We also see that there is not a strong correlation between $\langle p \rangle$ and the quantum volume. This is because the value of $\langle p \rangle$ is strongly affected by the probability of error an idle qubit while another is being measured and reset, whereas such processes do not occur in the calculation of the quantum volume. Also $\langle p \rangle$ is calculated using errors from across the whole device, whereas the quantum volume calculation can be done on an optimal subset of qubits.

The results of noisy runs, for both simulations and quantum hardware, are shown in Fig. 4. The simulations show that the probabilities $\langle p_W \rangle$ and $\langle p_Z \rangle$ increase with p and converge to $\langle p_W \rangle = \langle p_Z \rangle = 0.5$, as would be expected. For any run we see that $\langle p_W$ is consistently higher than $\langle p_Z \rangle$. This will be due to the fact that there is more time between two subsequent measurements of the stabilizer operators than of the z-links, and so more errors. The value of $\langle p_W \rangle$ approaches the convergence point already at around $p = 1.5\,\%$. For $\langle p_Z \rangle$, the results are easily distinguishable from the convergence point up to around $p = 3\,\%$.

A further data point not shown here is for `ibmq_manhattan`. This has $\langle p \rangle = 18.5\%$, which is well above those for the devices shown in the graph. The values of $\langle p_W \rangle$ and $\langle p_Z \rangle$ are at the convergence point, with $\langle p_W \rangle = 0.500$ and $\langle p_Z \rangle = 0.498$.

For the noisiest devices tested (`ibmq_brooklyn`, `ibmq_montreal`, `ibmq_toronto` and `ibmq_manhattan`) the results show $\langle p_W \rangle \approx \langle p_Z \rangle \approx 0.5$, and so are consistent

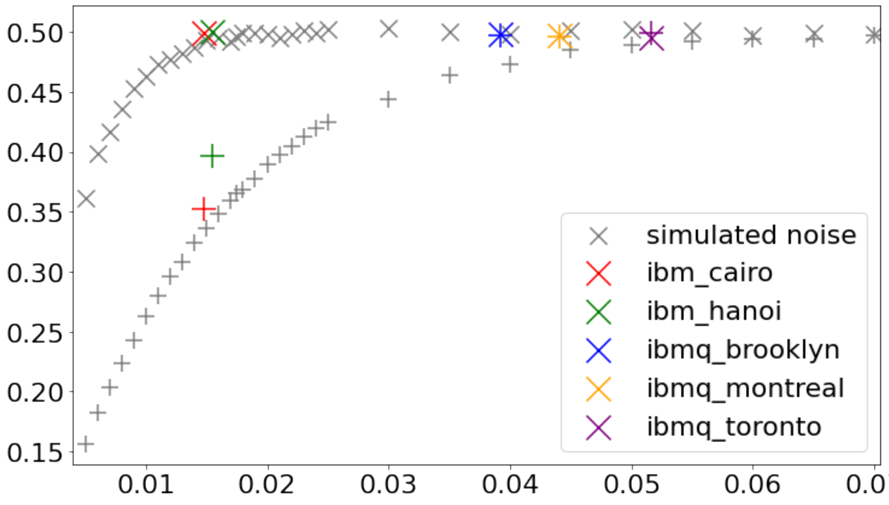


FIG. 4. The values $\langle p_W \rangle$ (marked as $\times$) and $\langle p_Z \rangle$ (marked as $+$) plotted against noise strength. For simulated runs, the noise parameter p is used, For runs on quantum hardware, the mean value $\langle p \rangle$ is used.

with the convergence point. This is consistent with the strength of the noise in these devices, which all have $\langle p \rangle > 4\,\%$.

For `ibm_cairo` and `ibm_hanoi` the results are clearly distinguishable from the convergence point, with $\langle p_Z \rangle < 0.4$ in both cases. The result for `ibm_cairo` is in good agreement with simulations for $p \approx 1.5\%$, which is consistent with the $\langle p \rangle = 1.47\,\%$ for this device. For `ibm_hanoi` the result agrees best with simulations for $p \approx 2\%$. Though this is higher than might be expected given the $\langle p \rangle = 1.55\,\%$ of this device, the high variation in the error strengths on mean that this is not a great surprise.

CONCLUSIONS AND OUTLOOK

The codes implemented in this work effectively consist of 2 independent stabilizer measurements per plaquette, per round. This means four per round for the Falcon devices, and 16 per round for the Hummingbird device. As such, these experiments are among the largest scale quantum error correction experiments yet implemented. Nevertheless, the scope of this study was not to use the codes to store or protect logical qubits. Instead it was to assess the effects of noise on the measurements of stabilizers, and to benchmark current quantum hardware by comparing results with simulations.

The results show that the various devices tested exhibit results consistent with a range of different error rates, from around 1.5% in the the best case of `ibm_cairo` to 5% or more for the worst case of `ibmq_toronto`. The former extreme shows good progress towards the well-known threshold of 0.5% -1% for the standard surface code [19, 21]. It is unfortunate that the only 65 qubit devices tested show results from the convergence point. Nevertheless, it is to be expected that initial versions of larger devices will exhibit higher error rates.

The figures of merit introduced here will continue to be applied as further quantum hardware is developed, allowing a holistic assessment of their capabilities in a concrete example of quantum error correction. Of particular interest will be the continued development of larger devices, such as the 65 qubit Hummingbird processors, to see evidence of low effective error rates. Also, the limit to $T = 3$ measurement rounds in this work was due to current limitations in the classical control software. In future we hope to greatly extend this, to also assess any effect of increasing circuit depth on the error rate.

Of course, there is also the need to further develop the understanding of the code itself. This includes developing a tailored decoder, and determining the threshold of the code. This will be done in conjunction with the development of the hardware, to work towards an eventual demonstration of a logical qubit with the code.

The data for all results in this paper are available at [22]

Acknowledgements: We acknowledge support from the Swiss National Science Foundation through NCCR SPIN.

[1] D. A. Lidar and T. A. Brun, eds., *Quantum Error Correction* (Cambridge University Press, Cambride, UK, 2013).

[2] J. R. Wootton, Quantum Science and Technology **5**, 044004 (2020).

[3] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, N. Lacroix, G. J. Norris, M. Gabureac, C. Eichler, and A. Wallraff, Nature Physics **16**, 875 (2020).

[4] A. Erhard, H. Poulsen Nautrup, M. Meth, L. Postler, R. Stricker, M. Stadler, V. Negnevitsky, M. Ringbauer, P. Schindler, H. J. Briegel, R. Blatt, N. Friis, and T. Monz, Nature **589**, 220 (2021).

[5] M. Gong, X. Yuan, S. Wang, Y. Wu, Y. Zhao, C. Zha, S. Li, Z. Zhang, Q. Zhao, Y. Liu, F. Liang, J. Lin, Y. Xu, H. Deng, H. Rong, H. Lu, S. C. Benjamin, C.-Z. Peng, X. Ma, Y.-A. Chen, X. Zhu, and J.-W. Pan, National Science Review (2021), 10.1093/nsr/nwab011, nwab011, https://academic.oup.com/nsr/advance-article-pdf/doi/10.1093/nsr/nwab011/35931214/nwab011.pdf.

[6] Z. Chen, K. J. Satzinger, J. Atalaya, A. N. Korotkov, A. Dunsworth, D. Sank, C. Quintana, M. McEwen, R. Barends, P. V. Klimov, S. Hong, C. Jones, A. Petukhov, D. Kafri, S. Demura, B. Burkett, C. Gidney, A. G. Fowler, H. Putterman, I. Aleiner, F. Arute, K. Arya, R. Babbush, J. C. Bardin, A. Bengtsson, A. Bourassa, M. Broughton, B. B. Buckley, D. A. Buell, N. Bushnell, B. Chiaro, R. Collins, W. Courtney, A. R. Derk, D. Eppens, C. Erickson, E. Farhi, B. Foxen, M. Giustina, J. A. Gross, M. P. Harrigan, S. D. Harrington, J. Hilton, A. Ho, T. Huang, W. J. Huggins, L. B. Ioffe, S. V. Isakov, E. Jeffrey, Z. Jiang, K. Kechedzhi, S. Kim, F. Kostritsa, D. Landhuis, P. Laptev, E. Lucero, O. Martin, J. R. McClean, T. McCourt, X. Mi, K. C. Miao, M. Mohseni, W. Mruczkiewicz, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Newman, M. Y. Niu, T. E. O'Brien, A. Opremcak, E. Ostby, B. Pató, N. Redd, P. Roushan, N. C. Rubin, V. Shvarts, D. Strain, M. Szalay, M. D. Trevithick, B. Villalonga, T. White, Z. J. Yao, P. Yeh,

A. Zalcman, H. Neven, S. Boixo, V. Smelyanskiy, Y. Chen, A. Megrant, and J. Kelly, "Exponential suppression of bit or phase flip errors with repetitive error correction," (2021), arXiv:2102.06132 [quant-ph].

[7] J. Hilder, D. Pijn, O. Onishchenko, A. Stahl, M. Orth, B. Lekitsch, A. Rodriguez-Blanco, M. Müller, F. Schmidt-Kaler, and U. Poschinger, "Fault-tolerant parity readout on a shuttling-based trapped-ion quantum computer," (2021), arXiv:2107.06368 [quant-ph].

[8] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz, "Realization of real-time fault-tolerant quantum error correction," (2021), arXiv:2107.07505 [quant-ph].

[9] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, Phys. Rev. X **10**, 011022 (2020).

[10] M. B. Hastings and J. Haah, "Dynamically generated logical qubits," (2021), arXiv:2107.02194 [quant-ph].

[11] C. Gidney, M. Newman, A. Fowler, and M. Broughton, "A fault-tolerant honeycomb memory," (2021), arXiv:2108.10457 [quant-ph].

[12] J. R. Wootton, Journal of Physics A: Mathematical and Theoretical **48**, 215302 (2015).

[13] A. Kitaev, Ann. Phy. **321**, 2 (2006).

[14] A. Y. Kitaev, Ann. Phys. **303**, 2 (2003).

[15] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, J. Math. Phys. **43**, 4452 (2002).

[16] J. R. Wootton, Quantum Science and Technology **2**, 015006 (2017).

[17] H. Bombin, Phys. Rev. Lett. **105**, 030403 (2010).

[18] H. Bombin and M. A. Martin-Delgado, J. Phys. A **42**, 095302 (2009).

[19] A. M. Stephens, Phys. Rev. A **89**, 022321 (2014).

[20] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, Phys. Rev. A **100**, 032328 (2019).

[21] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Phys. Rev. A **86**, 032324 (2012).

[22] J. R. Wootton, "The data includes pickled job, backend and results objects for all relevant jobs. currently at https://github.ibm.com/jwo/matching-code-paper, but this will be moved to the public github post-approval." (2021).

Chapter 5

Measurements of Floquet code plaquette stabilizers

Abstract

The recently introduced Floquet codes have already inspired several follow up works in terms of theory and simulation. Here we report the first preliminary results on their experimental implementation, using IBM Quantum hardware. Specifically, we implement the stabilizer measurements of the original Floquet code based on the honeycomb lattice model, as well as the more recently introduced Floquet Color code. The stabilizers of these are measured on a variety of systems, with the rate of syndrome changes used as a proxy for the noise of the device.

Publication Information

This chapter was made available on the arXiv [11] but is intended otherwise intended to be published only in this thesis.

Note that the references of this chapter are self-contained, and not included in the overall references.

INTRODUCTION

Floquet codes are a concept that has recently emerged in quantum error correction [1]. The first explicit example was the honeycomb Floquet code [1–5], based on Kitaev's honeycomb lattice model [6]. The concept was then extended with two additional codes. First was the $e \leftrightarrow m$ automorphism code [7], based on a generalization of honeycomb Floquet code. Next was the Floquet Color code [8–10], which can be derived from Color codes [11].

Both the original honeycomb Floquet code and the Floquet Color code share similarities in implementation. Specifically, both are based on the measurement of two-body observables corresponding to edges in a hexagonal lattice. This makes them both particularly well suited to the heavy-hexagon architecture of IBM Quantum processors [12], in which qubits reside on both the vertices and links of a hexagonal lattice. These can then be used as the required code qubits and auxiliary qubits respectively, where the latter are used within the circuits to measure two-body observables (see Fig. 1).

Most of the available hardware is not sufficient to perform a sensible analysis of the logical subspace. The 27 qubit *Falcon* devices, for example allow only two plaquettes to be realized. Nevertheless, the hardware is sufficient to measure the stabilizers, and assess how they detect noise on the devices. Here we will perform this for the honeycomb Floquet code and Floquet Color codes.

HONEYCOMB FLOQUET CODES

Kitaev's honeycomb lattice is a well-studied model in both condensed matter and quantum information [6]. It is defined on a hexagonal lattice with qubits on the vertices, as shown in Fig. 2. The links are labelled x, y and z depending on orientation. Each is assigned a two-body link operator $\sigma^\alpha \otimes \sigma^\alpha$ on the two adjacent vertex qubits, where $\alpha \in \{x, y, z\}$ is the link type.

Important to any study of this model are the plaquette operators, since they commute with all the link operators. Each is defined as the product of link operators around a corresponding plaquette. They can be expressed as

$$W = \sigma_0^x \sigma_1^y \sigma_2^z \sigma_3^x \sigma_4^y \sigma_5^z \tag{1}$$

using the numbering of Fig. 2(c).

The most obvious means to convert this model to a quantum error correcting code is arguably to perform the two-body parity measurements corresponding to the link operators, treating them as the gauge operators of a subsystem code. The plaquettes then form the corresponding stabilizer generator. Unfortunately, such a code has a trivial logical subspace [13]. However, two separate means to carve out a non-trivial subspace were introduced in the last year. One of these was the honeycomb Floquet code, and the other was an approach based on matching codes [14]. In both, a particular order is used for the measurement of the gauge operators in order to effectively create a logical subspace.

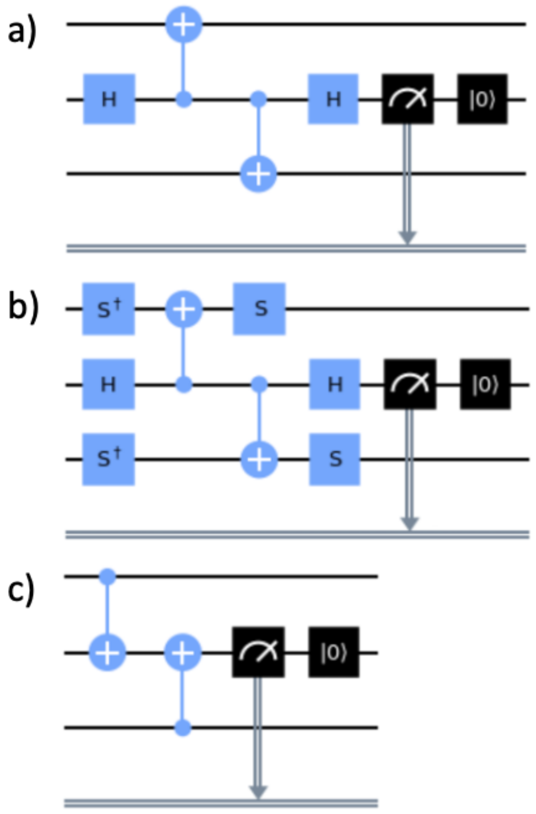


FIG. 1. Circuits to measure operators for (a) x-links, (b) y-links and (c) z-links. The two vertex qubits for which the parity is measured are at the top and bottom of each circuit. The auxiliary is in the middle. This should be initialized in the $|0\rangle$ state. If it remains in the output state of a previous measurement, the end result will be the XOR of the two.

For the honeycomb Floquet codes, the measurement schedule is defined using a tri-colouring of the plaquettes and links of the hexgaonal lattice. This is done such that any link of a given colour connects two plaquettes of that colour, and forms part of the boundary of two plaquettes of different colours. The colouring used for the plaquettes of the devices used are shown in Fig. 3

Measuring stabilizer changes

Measurement of the gauge operators is done by measuring the link operators of each colour in succession. We will consider the order red-green-blue by default. The results of any two successive rounds can be used to infer the results

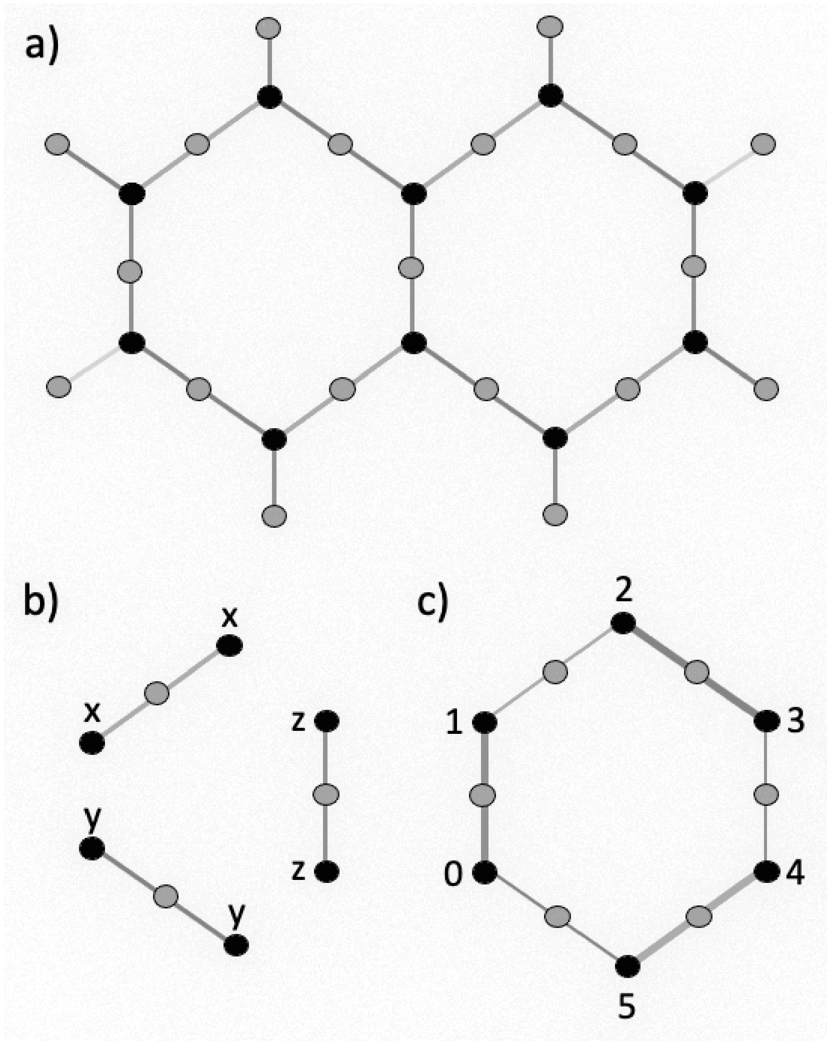


FIG. 2. (a) Portion of the lattice on which the honeycomb lattice model and derived codes are defined. Vertex qubits are shown as black dots. Grey dots are auxiliary qubits that can be used in the derived codes. (b) Definition of the link operators. (c) Definition of the plaquette operators.

for a subset of stabilizers. This is done by simply adding their results modulo 2. For the initial red and green rounds, it is the blue plaquettes for which results can be inferred. This is because the boundary of blue plaquettes is formed by red and green links. After the subsequent round of blue link measurements, results from the green and blue rounds can similarly be used to infer results for the red plaquettes, and so on.

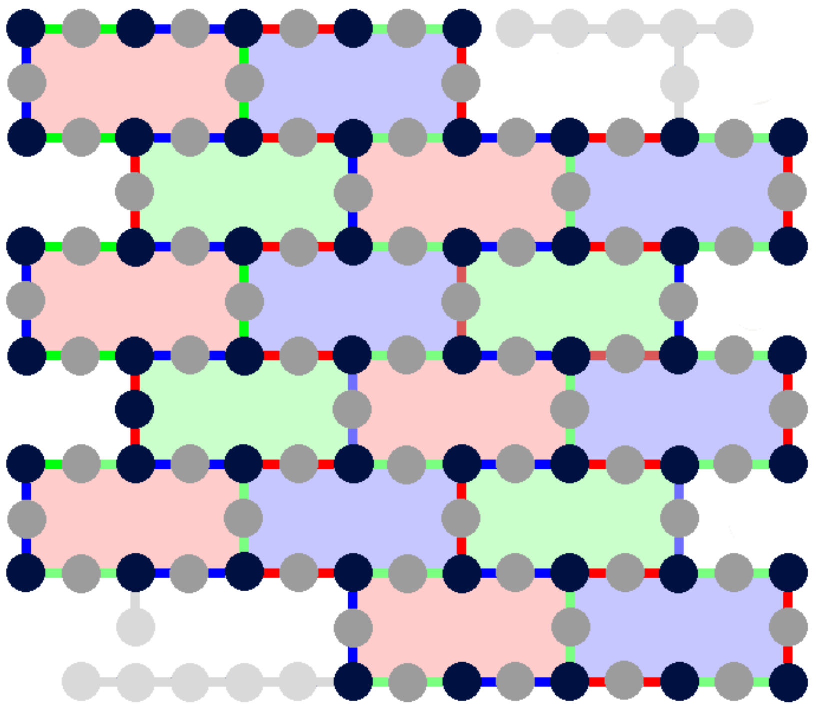


FIG. 3. Colouring of plaquettes and links for the `ibm_washington` device, which is a 127 qubit *Eagle* processor. Qubits shown in black are those of the code, those in grey are auxiliary qubits used during gauge operator measurements, and those in light grey are unused. The plaquettes of the 27 qubit *Falcon* devices and the 65 qubit *Hummingbird* processor are subsets of those shown here. The *Falcon* devices support two plaquettes, corresponding to a red and blue pair. The *Hummingbird* processor supports 8 plaquettes, corresponding to four rows of pairs alternating between red and blue, and green and red.

Assuming that the state begins in a product state, the initial measurement of each plaquette will typically give a random result. In the noiseless case, the next measurement would repeat the same result. A mismatch between subsequent measurements of the same plaquette is therefore a detection of an error. To perform such syndrome comparisons on all plaquettes, at least seven rounds of link operator measurements must be performed. It is this minimal seven round case that we will consider here.

We will also consider the case in which resets are not applied after measurement of the auxiliary qubits. The results for each link qubit measurement is then actually the parity of all measurements of that link qubit so far. The first syndrome change for red and blue plaquettes can then be calculated directly from the results of the second instance of the pertinent link operators. For the green plaquettes, only results from the second instance of the blue link measurements are required, but results from the first and third instance of the red link measurements must be combined.

Results

The seven round process is run for both IBM Quantum devices and simulated noise. For the former, an example of a scheduled circuit is shown in Fig. 4 for `ibmq_cairo`. The circuits typically settle into three distinct layers. The first consists of the two qubit gates required for the first three rounds of link operator measurement, followed by the measurements on auxilliary qubits for all. This is then repeated for the next three rounds. The final layer is similar, but consists only of the final round of link operator measurements. Note that there is significant idle time for the qubits, caused by both measurement and the scheduling of two qubit gates. The latter is exacerbated by the mismatch in two qubit gate times. In the first layer, this can mitigated to a degree by delaying the first gate for as long as possible for some qubits. This is because idling minimally effects qubits in their initialized state.

For the simulations, only the 27 qubit case is considered. The noise model is a variation of the standard one used for surface code threshold calculations [15], in which everything that can go wrong does so with some a probability p. Specifically, bit flips after reset and prior to measurement occur with probability p, and depolarizing noise is applied to each two qubit gate and during idle times with probability p. The idling error is applied during measurement to the unmeasured qubits. It is also applied to each qubit at the beginning of each layer, as a proxy for the idling during two qubit gates.

Whether for quantum devices or simulation, the results are processed to calculate a syndrome change detection rate for each plaquette. This is the fraction of shots for which a change in value was detected for the two measurements of the corresponding plaquette. Since such a change in value occurs only in the presence of errors, this rate will be zero for $p = 0$. It can then be expected to converge to the maximally random value of $1/2$ as p increases. The value found for quantum devices will give some sense of how noisy the syndrome extraction process is.

For comparison of IBM Quantum devices with simulations, we will calculate an average of the noise probabilities from the devices obtained in benchmarking. Specifically we consider the following probabilities.

- Preparation noise: The probabilities `prob_meas1_prep0` for each qubit.
- Measurement noise: The measurement error probability for each qubit.
- CX noise: The CX error probabilities.
- Idle noise: For each qubit, the error probability for an identity gate over the timescale of measurement and the longest CX gate. This is extrapolated from the probability p_{id} of an error over the standard timescale t_{id} of an identity gate [14].

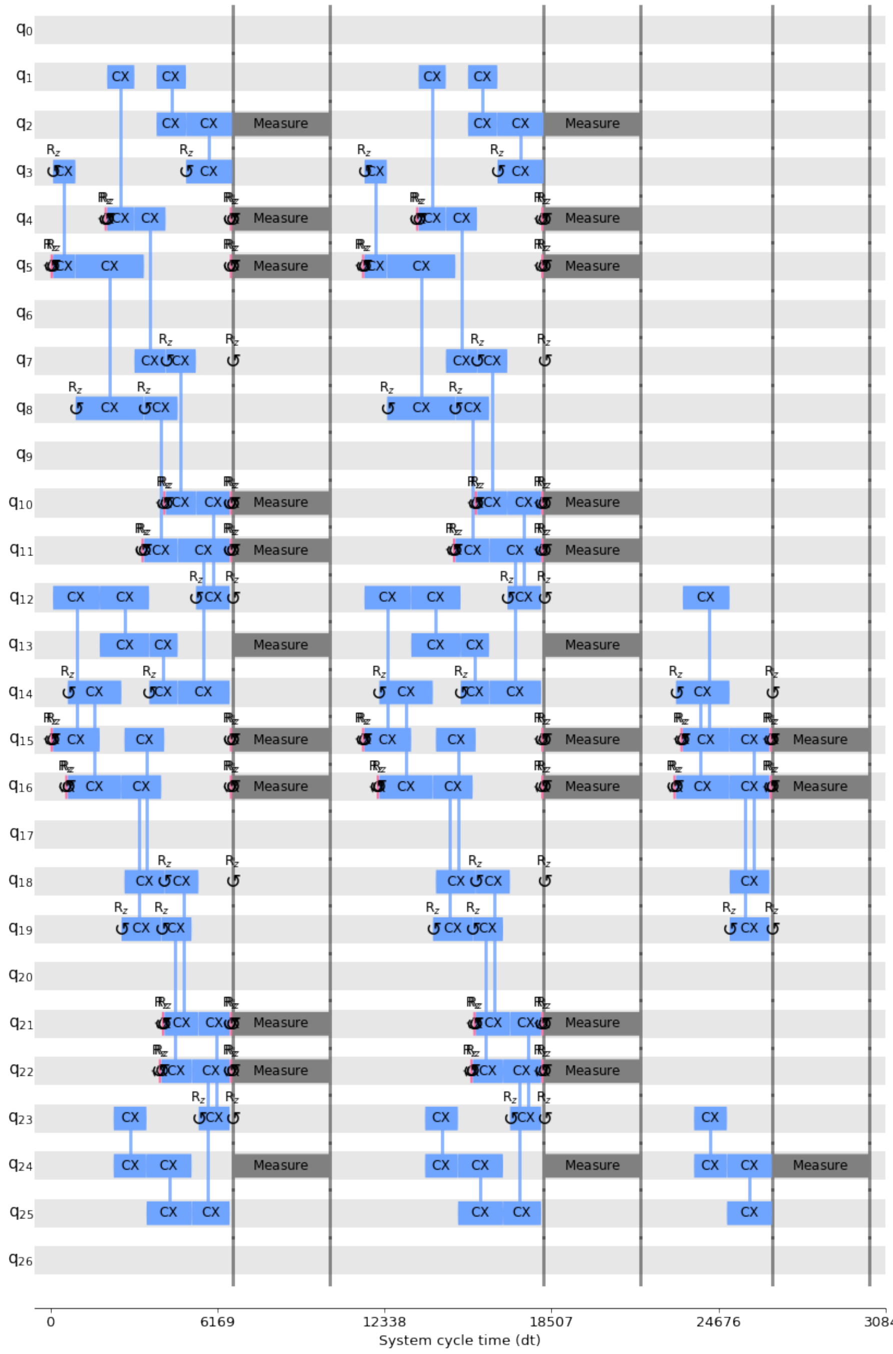


FIG. 4. A scheduled circuit for `ibmq_cairo`. The sample time for this device is $dt = 0.22$ns. Though circuit does not include dynamical decoupling for clarity, the circuits as run include a pair of `x` gates during each delay.

The mean and standard deviation of this set of probabilities is shown for the devices in table below, along with the the quantum volume [16]. These values are derived from the most recent benchmarking data at the time when the results were taken.

Device	$\langle p \rangle$	σ	QV
`ibm_hanoi`	1.32 %	1.39 %	64
`ibm_auckland`	1.50 %	1.71 %	64
`ibmq_kolkata`	1.76 %	3.14 %	128
`ibm_peekskill`	1.81 %	2.37 %	-
`ibmq_montreal`	2.26 %	2.09 %	128
`ibmq_mumbai`	3.00 %	1.73 %	128
`ibmq_brooklyn`	3.02 %	3.33 %	32
`ibmq_washington`	3.13 %	5.39 %	64
`ibmq_toronto`	4.72 %	6.37 %	32

The results are shown in Fig. 5. Results from both simulations and quantum hardware show the expected behaviour as the error rate increases. For `ibmq_toronto`, the relatively high noise rate results in completely random syndrome change detection rates. For all other cases, however, results depart significantly from the maximally random value, with ever lower average error rates achieving ever lower syndrome change detection rates. The best results are comparable with a gate error of $p = 2\%$ for the simple noise model used in the simulations.

FLOQUET COLOR CODES

Measuring stabilizer changes

The Floquet Color codes are also defined using the tri-colouring of the plaquettes and links of the hexagonal lattice shown in Fig. 3 For each link we assign two parity operators: $\sigma^\alpha \otimes \sigma^\alpha$ for $\alpha \in \{x, z\}$. For each plaquette we define two plaquette operators,

$$W^\alpha = \sigma_0^\alpha \sigma_1^\alpha \sigma_2^\alpha \sigma_3^\alpha \sigma_4^\alpha \sigma_5^\alpha, \quad \alpha \in \{x, z\}, \tag{2}$$

using the numbering of Fig. 2(c).

Note that the set of parity operators here do not commute with the plaquette operators. This code therefore does not correspond to a subsystem code in which the link operators are gauge operators and the plaquette operators are stabilizers. Nevertheless, the roles they play are equivalent.

Each round of measurements consists of measuring a single link type of a single colour of links, alternating between the two link types and through the three colours from round to round. A complete cycle therefore takes 6 rounds [8–10].

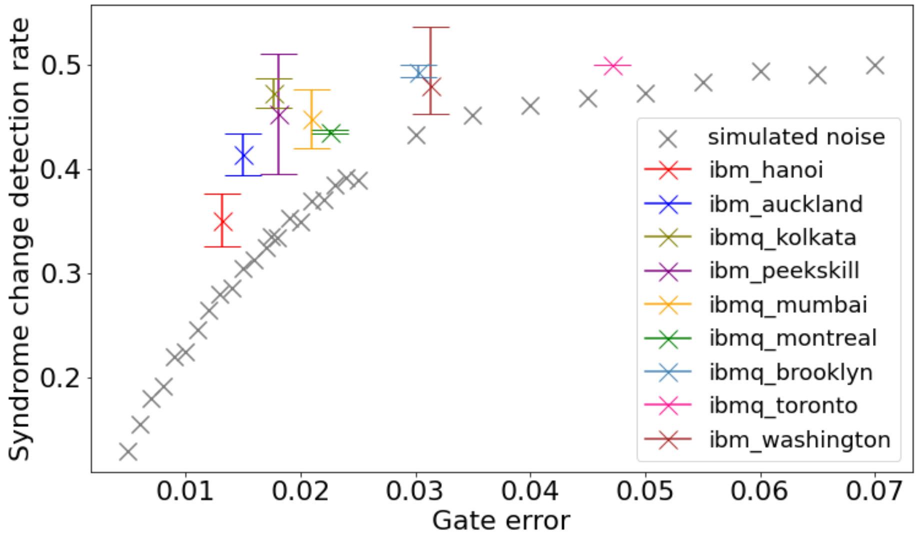


FIG. 5. The values of the syndrome change detection rate plotted against noise strength (p for simulated runs and $\langle p \rangle$ for quantum hardware). For simulated results, the mean syndrome change detection rate over all plaquettes is shown. For results from quantum hardware, the mean, maximum and minimum over all plaquettes are shown. Specifically, the central point is the mean, and the bars extend down to the minimum and up to the maximum.

The measurement outcome for a red W^α plaquette operator can be deduced from the α-link measurements for both green and blue links, and therefore twice per cycle. However, note that the red W^α plaquette operator does not commute with the $\neg\alpha$-link operators for red links. The outcome of the red W^α plaquettes following the measurement of red $\neg\alpha$-link operators will therefore produce random results, uncorrelated with any previous results. However, the subsequent red W^α plaquettes will not be so disturbed, and therefore should repeat the same result. A mismatch between subsequent measurements of the same W^α on the same plaquette is therefore a detection of an error. These syndrome comparisons form the basis of error detection in the code, since any measured syndrome change is a signature of an error. Equivalent arguments can be made for the blue and green plaquettes.

As with the honeycomb Floquet code, the scheduled circuit will settle into distinct layers in which each consist of three rounds of link operator measurements. This is because such sets of three rounds directly measure different auxiliary qubits, allowing the measurement gates to be performed relatively simultaneously. Each set of three rounds performs both types of plaquette measurement on both types of plaquette. A single six-round cycle is therefore sufficient to determine syndrome changes for several types of W^α. However, due to the scheduling of the non-commuting link measurements, ten rounds are required for syndrome changes of all [9, 10],. Note that we do not consider changes in syndrome from the initialized value, which is possible for some of the W^z stabilizers. This is because we are interested in the typical syndrome changes within a computation, rather than this edge case.

Results

The ten round process is run for both IBM Quantum devices and simulated noise. The layers within the scheduled circuits are visually similar to those of the honeycomb Floquet code shown in Fig. 4, and so have similar considerations concerning idling times. The same noise model as before is therefore used for the simulations. The quantities measured in these runs are the syndrome change detection rate for each plaquette W^α.

The comparison noise probabilities are calculated in the same way as before. However, since results were taken on different days, the benchmarking data from which the values are derived are not the same. The relevant set of probabilities for these results is therefore shown for the devices in table below.

Device	$\langle p \rangle$	σ	QV
`ibm_hanoi`	1.40 %	1.63 %	64
`ibmq_kolkata`	1.47 %	1.80 %	128
`ibm_auckland`	1.82 %	3.36 %	64
`ibm_peekskill`	2.16 %	3.06 %	-
`ibmq_montreal`	2.26 %	2.09 %	128
`ibmq_mumbai`	2.38 %	2.04 %	128
`ibmq_brooklyn`	2.78 %	2.96 %	32
`ibm_washington`	3.10 %	4.78 %	64
`ibmq_montreal`	4.36 %	7.82 %	128

The results are shown in Fig. 6. Results from both simulations and quantum hardware show the expected behaviour as the error rate increases. For the simulations, we see that the syndrome detection rate for a given p is lower than that for the honeycomb Floquet codes. Nevertheless, results from quantum hardware show similarities between the two codes For example, devices with $\langle p \rangle \approx 2\%$ have a syndrome detection rate of just over 0.4 for both.

CONCLUSIONS

Here we report results from preliminary implementations of two forms of Floquet code. The results can be used to get a sense of both how well current hardware can implement such codes, as well as how well the codes perform.

From the perspective of the hardware, we find that current IBM Quantum devices are already up to the task of performing syndrome measurements for

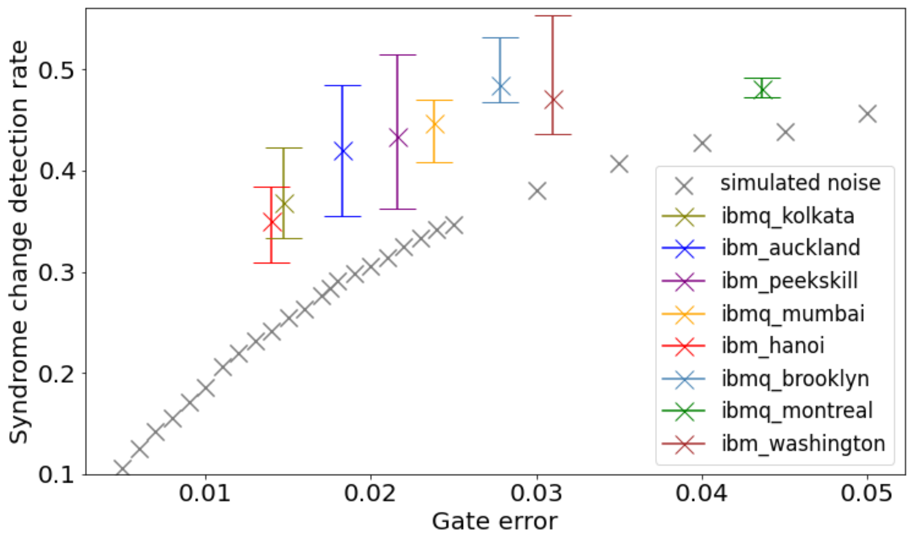


FIG. 6. The values of the syndrome change detection rate plotted against noise strength (p for simulated runs and $\langle p \rangle$ for quantum hardware). For simulated results, the mean syndrome change detection rate over all plaquettes is shown. For results from quantum hardware, the mean, maximum and minimum over all plaquettes are shown. Specifically, the central point is the mean, and the bars extend down to the minimum and up to the maximum.

these codes. Specifically, the syndrome change detection rate is significantly below the maximally random value of 0.5 in almost all cases. Nevertheless, it is clear that the noise level has only just reached the point where a clear signal is possible. Indeed, the clarity of the signal was marginal for the similar study performed in [14], which was for a different code but with much of the same hardware. Much of the improved performance seen in the current work is dependent on suitable use of dynamical decoupling. It will therefore not only be interesting to see progress made in future devices, but also to see how much further current devices can be pushed.

From the perspective of the codes, the results do not clearly distinguish between the two. For the simulations, the syndrome change detection rates for the Floquet Color code are slightly lower than for the honeycomb Floquet code. However, this advantage does not also appear for runs on quantum hardware, at least to an appreciable degree.

This work is not intended to give a definitive assessment of either the codes or the hardware. Instead, these results are a reference by which we can compare future progress.

The source code and data for all results in this paper is available at [17].

Acknowledgements: This work was partially supported by IARPA under

LogiQ (contract W911NF-16-1-0114). All statements of fact, opinion or conclusions contained herein are those of the authors and should not be construed as representing the official views or policies of the US Government. The software used was created as part of a project supported as a part of NCCR SPIN, a National Centre of Competence in Research, funded by the Swiss National Science Foundation (grant number 51NF40-180604). The author would like to thank Benjamin Brown and Markus Kesselring for discussions of their work.

[1] M. B. Hastings and J. Haah, Quantum **5**, 564 (2021), ISSN 2521-327X, URL `https://doi.org/10.22331/q-2021-10-19-564`.

[2] C. Gidney, M. Newman, A. Fowler, and M. Broughton, Quantum **5**, 605 (2021), ISSN 2521-327X, URL `https://doi.org/10.22331/q-2021-12-20-605`.

[3] J. Haah and M. B. Hastings, Quantum **6**, 693 (2022), ISSN 2521-327X, URL `https://doi.org/10.22331/q-2022-04-21-693`.

[4] C. Gidney, M. Newman, and M. McEwen, Quantum **6**, 813 (2022), ISSN 2521-327X, URL `https://doi.org/10.22331/q-2022-09-21-813`.

[5] A. Paetznick, C. Knapp, N. Delfosse, B. Bauer, J. Haah, M. B. Hastings, and M. P. da Silva, *Performance of planar floquet codes with majorana-based qubits* (2022), URL `https://arxiv.org/abs/2202.11829`.

[6] A. Kitaev, Ann. Phy. **321**, 2 (2006).

[7] D. Aasen, Z. Wang, and M. B. Hastings, Phys. Rev. B **106**, 085122 (2022), URL `https://link.aps.org/doi/10.1103/PhysRevB.106.085122`.

[8] B. J. Brown, *Anyon condensation and the color code*, doi:10.26081/K6F65V (2022).

[9] M. Davydova, N. Tantivasadakarn, and S. Balasubramanian, *Floquet codes without parent subsystem codes* (2022), arXiv:2210.02468.

[10] M. S. Kesselring, J. C. M. de la Fuente, F. Thomsen, J. Eisert, S. D. Bartlett, and B. J. Brown (in preparation).

[11] H. Bombin and M. A. Martin-Delgado, Phys. Rev. Lett. **97**, 180501 (2006), URL `https://link.aps.org/doi/10.1103/PhysRevLett.97.180501`.

[12] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, Phys. Rev. X **10**, 011022 (2020), URL `https://link.aps.org/doi/10.1103/PhysRevX.10.011022`.

[13] M. Suchara, S. Bravyi, and B. Terhal, Journal of Physics A: Mathematical and Theoretical **44**, 155301 (2011), URL `https://doi.org/10.1088/1751-8113/44/15/155301`.

[14] J. R. Wootton, Journal of Physics A: Mathematical and Theoretical **55**, 295302 (2022), URL `https://dx.doi.org/10.1088/1751-8121/ac7a75`.

[15] A. M. Stephens, Phys. Rev. A **89**, 022321 (2014), URL `https://link.aps.org/doi/10.1103/PhysRevA.89.022321`.

[16] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, Phys. Rev. A **100**, 032328 (2019), URL `https://link.aps.org/doi/10.1103/PhysRevA.100.032328`.

[17] J. R. Wootton, `https://github.com/quantumjim/data4papers`.

Part II

Repetition Codes

Chapter 6

Introduction

The repetition code is the simplest possible example of error correction. It is the method that is naturally employed even by those with no concept of information theory: if you think there is some chance that your message won't be heard, then repeat it! Nevertheless, it can also be expressed naturally within the framework of quantum error correction [13], as a particular example of a stabilizer code [14]. It can therefore be implemented using the standard methodology of stabilizer codes.

Strictly speaking, repetition codes have a distance $d = 1$. This suggests that only a single error is enough to destroy any stored information. However, this is only true for *quantum* information. For example, for the standard encoding of the repetition code, just a single phase flip error is enough to cause a phase flip of the stored logical qubit. However, that qubit is protected against bit flips up to a distance of n, the number of qubits in the code. The code is therefore very poor at storing a logical qubit, but gives unmatched performance for storing a logical *bit*. As such, when we refer to the distance of a repetition code, it is typically the distance for the stored classical information that we refer to.

Furthermore, repetition codes are very flexible regarding the arrangement of the physical qubits. In fact, the standard arrangement is simply a line of qubits for which entangling gates can be performed between each pair of neighbours. Whatever the coupling map of a device, extensive examples of such lines can typically be found.

The above mentioned properties make repetition codes a simple, powerful and flexible example of quantum error correction in action. As such, they are the a perfect way to test the capabilities of a small to medium sized device. With 50 qubits, for example, the largest surface code that could be implemented is one with $d = 5$ [15]. Though the implementation of the former represents a very important step towards fault-tolerance, the latter allows the prospect of much larger distances to be probed with the same resources.

The minimal implementation of a repetition code that can detect and correct errors is that with $d = 3$: a line of three code qubits alternating with two auxiliary qubits. The five qubits required overall mean that such a code could be implemented on the 5 qubits available on the cloud in 2016. However, in 2017 the number of available qubits increased to 16. These were mostly laid out with a ladder connectivity (though with sone missing links), meaning that a line could be found that covered all 16 qubits. With this, repetition codes of up to $d = 8$ could be implemented. This allowed codes of different sizes to be compared, to determine whether quantum error correction could really deliver on its promise that logical errors would be ever more suppressed for larger codes. This experiment was performed and reported in [16], reproduced here as Chapter 7. Additional analysis of the results was also performed in [17].

The results presented in the above works were obtained during summer 2017. Towards the end of that year, the device was returned to room temperature and then again cooled. After this process, the full intended ladder connectivity was available. However, other changes to the qubits could also be seen. In particular, much higher physical error rates were found, and the expected suppression of logical errors with increasing code distance was not. These results were presented to IBM Quantum, and found to be due to frequency collisions. It was to address this issue that the heavy hexagonal architecture was developed [7], which is still pursued by IBM Quantum today. Of course, these results were not to only sign of frequency crowding observed, and were not the only reason for the change of architecture. Nevertheless, this example shows that it is important to test current devices with all the biggest and best examples of quantum error correction possible, to ensure that the direction pursued in hardware is compatible with the requirements of scalability.

The above experiment was tailored to the device on which it ran: 16 qubits with a particular layout, and with no capacity to reuse qubits after measurements. To allow bigger and better experiments, it became necessary to create more professional, adaptable and reusable software. This was one of the motivations for the `topological_codes` package, created as a part of Qiskit Ignis [18], which in turn is part of IBM Quantum's open-source quantum SDK [19]. This was made to automate the creation of circuits for repetition code experiments, allowing them to be adapted to different numbers and layouts of qubits, as well as supporting multiple measurement rounds. It also allowed different choices of encoding for the code: either the standard encoding that protects against bit flip errors, or an alternative that protects against bit flip errors. The package also included the software required to analyze the results from the circuits by performing decoding. It has since been expanded and incorporated into the new part of Qiskit for quantum error correction: Qiskit QEC [20].

A tutorial for this package, as well as to the basics of quantum error correction in general, was written as a chapter of the Qiskit textbook [21]. This included examples of the creation and analysis of repetition code circuits, with results from an implementation of IBM Quantum's `ibmq_rochester` device. This allowed the implementation of a $d = 22$ code, with 43 qubits in all, which was the largest repetition code that had been implemented at the time. The textbook chapter was also published as a paper [22], and is reproduced here as Chapter 8.

The experimental results in the above work centered around measurement of the logical error rate for codes of different distances. There the aim is to determine whether the logical error rate decreases for increasing d. Specifically, it is expected that the decrease is exponential, with an increasingly potent decay as the error rates of the physical qubits also decrease. The measurement of logical error rates is done by sampling many runs of the code, and determining whether a logical error occurs for each. The exponential decay therefore implies an exponential sampling complexity. The study of repetition codes purely through the analysis of logical error rates is therefore not scalable.

This has now become a problem for experiments with IBM Quantum hardware, which has continued to improve in all important directions: increasing the number of qubits, decreasing the error rates and allowing the reuse of qubits after measurement. These advances both allow and necessitate more detailed analysis of experimental results. In particular, note that errors at different locations within a circuit lead to different signatures in the final syndrome. It is this property that allows these errors to be detected and corrected. However, it also allows them to be benchmarked. By analyzing the syndrome outcomes from many shots, it is possible to determine the probabilities that errors occur during each point in the circuit for each qubit. This gives a more detailed picture of errors than those obtained by simply benchmarking each qubit or each gate in isolation [23]. They also provide a direct account of how errors occur while a device is involved in a large-scale quantum error correction process. Such a study is performed on a range of devices in Chapter 9. In particular, $d = 3$ repetition codes are implemented in order to analyze the errors of the central qubit, looking at whether these correspond to those expected from the T_1 and T_2 times of the qubit in isolation.

Chapter 7

A repetition code of 15 qubits

Abstract

The repetition code is an important primitive for the techniques of quantum error correction. Here we implement repetition codes of at most 15 qubits on the 16 qubit *ibmqx3* device. Each experiment is run for a single round of syndrome measurements, achieved using the standard quantum technique of using ancilla qubits and controlled operations. The size of the final syndrome is small enough to allow for lookup table decoding using experimentally obtained data. The results show strong evidence that the logical error rate decays exponentially with code distance, as is expected and required for the development of fault-tolerant quantum computers. The results also give insight into the nature of noise in the device.

Publication Information

Originally published in *Physical Review A*, Volume 97, Issue 5 (2018) [16], with co-author Daniel Loss.

Note that the references of this chapter are self-contained, and not included in the overall references.

INTRODUCTION

The development of quantum computing is entering an exciting new era. Prototype quantum processors based on various physical architectures are beginning to appear, such as those superconducting qubits [1, 2], trapped ions [3] and spin qubits [4]. Such devices are too small and noisy to fully realize the promise of quantum computation. Many more qubits and the ability to achieve fault-tolerance are required before the age of quantum computers truly dawns. Nevertheless, current and near future devices can and have been used to generate many significant proof-of-principle results.

An important challenge of this new era is to benchmark and compare quantum processors. A quantitative starting point for this is the 'Quantum Volume'[5], designed to capture not only the number of qubits in a device, but also some idea of the circuit depth that can be achieved before the effects of noise dominate.

More detail on the capabilities of a device can be obtained by running simple programs. An obvious choice would be to implement small instances of algorithms intended for large fault-tolerant devices, such as Shor's [6] or Grover's [7] algorithms. However, a better insight would arguably come from algorithms that have been specifically designed to work on small and noisy devices [8].

Fortunately, there are already a class of protocols designed specifically for noisy systems: those of quantum error correction [9]. Many experiments have already been done based on tasks in this area, such as error detection [3, 10, 11], correction [1, 12] and proof-of-principle tests of the required techniques [2, 13, 14].

Several of these experiments have involved large Hilbert spaces. However, most of these have either not allowed both detection and correction [13], or not used the standard paradigm of encoding in a many qubit system [12]. Apart from these, the largest number of qubits used so far for the detection and correction of errors was a repetition code of 9 qubits [1]. This code is capable of both detecting and correcting errors on a logical *bit* value stored in an array of qubits. Note that, unlike most quantum error correcting codes, this does not allow a logical *qubit* to be stored. However, the connectivity required to implement codes which store qubits is beyond most current devices [15, 16].

Specifically, if the qubits of a device are represented as the vertices of a graph, and edges are placed between all pairs for which entangling gates can be directly performed, full quantum error correcting codes require this graph to be at least a two dimensional planar lattice. Repetition codes, however have much more amenable needs: All they require is a line. This code therefore represents the forefront of current implementations of quantum fault-tolerance, and is an important means to benchmark current devices.

At the time producing this study, the largest quantum processor was the *ibmqx3* of IBM [17]. This is a 16 qubit device whose connectivity is described by the 2×8 square lattice shown in Fig. 1. Since repetition code can only be defined on an odd number of qubits, this device can be used to implement a repetition code of up to 15 qubits. It is such an implementation that we consider in this work. By doing so, we can look for evidence of one of the key

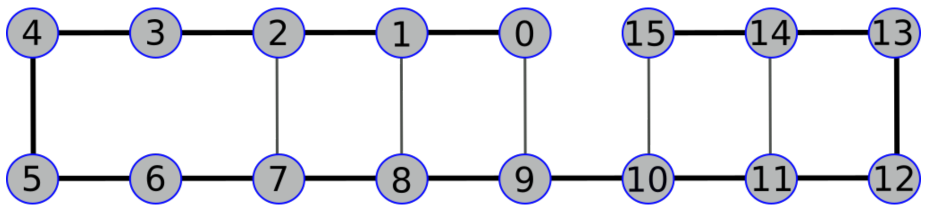


FIG. 1. The layout of the *ibmqx3* device, with the numbering of qubits used in this study. Lines connect pairs of qubits for which a CNOT can be performed. Thick black lines are show for the CNOTs used in our implementation of the repetition code.

assumptions and requirements of quantum error correction, namely that the logical error rate decays exponential with code distance.

This exponential decay is typically dependent on another assumption: that noise acts with finite probability on only a finite number of qubits. This could be violated by sufficiently large correlations between code qubits [18, 19], or by the operations used to implement the code inadvertently allowing noise to spread [20]. In such cases, the noise threshold required for successful error correction would become zero, significantly limiting the degree to which errors can be suppressed. Such effects could be effectively ruled out by demonstrating that the exponential decay of logical error rate persists to arbitrarily large code distances. In this study we will consider this to a limited extent by considering a range of possible code sizes, but it is beyond the scope of this work to fully rule out such effects.

THE REPETITION CODE

The repetition code stores classical information through repetition. A code with distance d stores the bit value $b \in \{0, 1\}$ by simply repeating the value b across d bits.

For example, this encoding step correponds to the following for $d = 5$,

$$0 \rightarrow 00000, \;\; 1 \rightarrow 11111.$$

To implement this code using a quantum device, we simply replace the bits with qubits. The value 0 is then stored with the state $|\,0\rangle^{\otimes d}$, and 1 is stored with $|\,1\rangle^{\otimes d}$.

For the example of $d = 5$, this is then,

$$0 \rightarrow |\,00000\rangle\,, \;\; 1 \rightarrow |\,11111\rangle\,.$$

If no errors occurred, the result at output would be to simply regain the initial $|\,0\rangle^{\otimes d}$ or $|\,1\rangle^{\otimes d}$. Decoding the stored bit value in this case would trivially be an inversion of the encoding, such as,

$$|\,00000\rangle \rightarrow 0, \;\; |\,11111\rangle \rightarrow 1.$$

When errors do occur, the result will most likely be a bit string with a mixture of 0s and 1s. Decoding is then typically done by majority voting: If the error rate is low, the majority of qubits can still be expected to be correct. Deducing that the majority value is the one that was encoded will therefore allow the encoded information to be retrieved in most cases. The probability that the decoding is incorrect is known as the logical error probability.

For example, suppose errors are bit flips which occur with probability $p < 0.5$. If the result $|\,01000\rangle$ is obtained for $d = 5$, it could have resulted from two possible processes. One is that a logical 0 was encoded, followed by an error on the second qubit. The other is than a 1 was encoded, and errors occurred on all but the second qubit. The probability of the former is much higher than that of the latter, due to the much smaller number of errors. So the decoding would be $|\,01000\rangle \rightarrow 0$.

For the error rate to be low enough for good decoding to be possible, the stored information must be retrieved quickly after being first encoded. However, error correction typically aims to store information over long time scales. When the repetition code is implemented classically, this can be achieved by periodically measuring the bits of the code to keep track of errors as they occur.

This tactic is not compatible with the needs of quantum error correction, in which our techniques should not be allowed to collapse a superposition of a stored 0 and a stored 1, which would be encoded in the code qubits as $\alpha\,|\,0\rangle^{\otimes d} + \beta\,|\,1\rangle^{\otimes d}$. We therefore need a corresponding method that detects errors, but does not need to readout the stored bit at the same time. This process corresponds making to so-called 'stabilizer measurements'[21], which lie at the heart of most prominent quantum error correcting codes.

Stabilizer measurements for the repetition code are achieved using the circuit of Fig. 2. In addition to the d code qubits, $d{-}1$ ancilla qubits are used. The code and ancilla qubits are arranged alternately on a line. Each pair of neighbouring code qubits therefore always has an ancilla located between them. CNOT gates are then performed, with a code qubit as control and an ancilla qubit as target in each case. For each ancilla, a CNOT is applied with both neighbours. The end effect of this on the ancilla does not depend on the encoded bit value. It depends only on whether the computational basis states of the neighbouring code qubits agree (as they should due to the repetition) or not (a signature of error). By measuring the ancillas we can then extract information about errors, without collapsing any encoded superposition.

This portion of the circuit (CNOTs and ancilla measurements) can be repeated indefinitely to keep track of errors as they arise. When it is time to read out the bit value, the code qubits should also be measured. The entire output can then be used to deduce the original intended value. As long as the probability of error between ancilla measurement rounds is sufficiently low, the probability of a logical error will decay exponentially as d is increased.

The decoding in this case can no longer be done simply using majority voting. Instead a decoding method such as the Blossom algorithm for minimum weight perfect matching can be used [22], as can other methods designed for the case of the surface code with perfect syndrome measurements [23–25].

The use of such algorithms is not required if the output is sufficiently small (due to small d and few ancilla measurement rounds). In such cases, a look-up table decoder can be calculated from experimental data.

Specifically, the look-up tables are conditional probability distributions $\pi(R|E)$, where R is a bit string of respresenting the final output of a code, and $E \in \{0,1\}$ is the encoded bit value. These distributions can be determined experimentally for codes of limited size using the results of many runs. Once known, they can be used to decode an arbitrary output R. This is done simply by finding the value of E for which $\pi(R|E)$ is highest. This then gives the most likely value for the encoded bit, assuming that the priors for the two values are equal, and so is taken to be the decoded bit value.

The probabilities for logical errors can be determined from the same look-up tables used for the decoding. For a given encoded bit value E, the probability $P(R|E)$ that the result R occurs and causes a logical error is clearly

$$P(R|E) = \begin{cases} \pi(R|E), & \text{if } \pi(R|\neg E) > \pi(R|E). \\ \pi(R|E)/2, & \text{if } \pi(R|\neg E) = \pi(R|E). \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The second case here reflects the possibility that the decoding is ambiguous, and so the decoded value is chosen randomly.

The total probability for a logical error for a given encoded bit value E can then be obtained by summing over all possible outcomes

$$P(E) = \sum_R P(R|E). \tag{2}$$

In our experiment we consider codes with a maximum of $d = 8$ (and so 15 qubits in total). Also, due to restrictions of the API for the IBM Quantum Experience [26], it is not possible to measure the ancilla qubits repeatedly. We therefore do only a single round of CNOTs and ancilla measurements, the latter of which are done simultaneously with the final readout of code qubits. The circuit applied is therefore exactly that shown in Fig. 2 for $d = 3$, or generalizations thereof for higher d. Due to this limited size, we perform the lookup table decoding described above.

Note that the limited sample size will lead to these results being approximate. Specifically, the method stated above will strictly function as a lower bound due to the possibility of overfitting. An alternative method producing an upper bound is investigated in an alternative version of the source code for this study, which can be found at [27]. This is found to give the same qualitative features as the results presented in this work, but is quantitatively different.

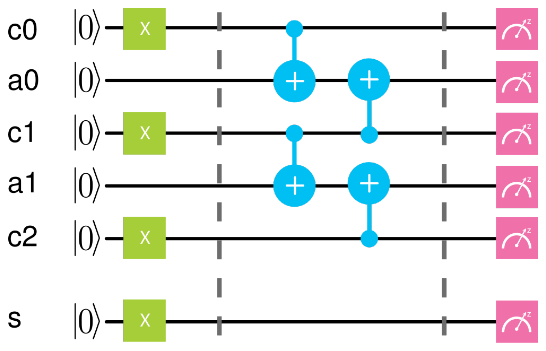


FIG. 2. The basic circuit for a $d = 3$ repetition code. The case shown is that for which the encoded logical bit value is 1. This value is encoded in the three code qubits ($c0$, $c1$ and $c2$) by rotating the initial $|0\rangle$ value to $|1\rangle$ using an X gate. The case for an encoded logical 0 is the same, but without the X gates. The qubit s is not part of the code, but is used to compare the code against the case of encoding the same bit value in a single qubit. The qubits 0 to 5, according to the numbering of Fig. 1, are used to implement this code.

CONDITIONS FOR SUCCESS

In order to determine how well the device implements the repletion code, concrete conditions for success must be defined.

The most straightforward is to compare a bit stored in a code with one stored in only a single qubit. The promise of quantum error correction is that the non-local storage of information across many qubits is more reliable than the single qubits themselves. We must therefore confirm that this the case. To do this, each run of the repetition code with a given encoded bit value is accompanied by a single qubit in which the same value is encoded. This additional qubit, which we will refer to as qubit s, is shown at the bottom of Fig. 2.

Another obvious test is to ensure that the probability of a logical error does indeed decrease with d, as expected. This should certainly be the trend for large increments d, though exceptions may be found between closely related values. These could give an interesting insight in to the nature of noise on the device. Also, note that the minimum number of errors required to cause incorrect decoding is $\lceil d/2 \rceil$ (the number of flips on code qubits required to change (for odd d) or create ambiguity regarding (for even d) the majority. Since this number is the same for each odd d and the even $d+1$ that follows it, there might not be a significant decrease in the logical error probability between these pairs.

For an additional test, note that the output of the code qubits alone is enough to perform decoding. This would not be true if many rounds of CNOTs and ancilla measurements were applied. In this case, the amount of noise built up on the code qubits would be too high for reliable decoding, and only with the history of ancilla measurement results can the original encoded value be deduced. However, in the case of the single ancilla measurement round as we consider, the noise level should be low enough to allow decoding using the code qubit results alone.

The result of this is that the code could satisfy the previously mentioned conditions (the code performing better than a single qubit, and ever better as d increases) even if the CNOTs completely failed to occur. This would in no way be any proof-of-principle of quantum error correction, and so these conditions are clearly not sufficient to ensure success.

The use of the CNOTs and ancilla measurements in our experiment has both benefits and drawbacks. The former is due to the extra information that can be extracted about the errors that occur. The latter is due to the additional noise suffered by the code qubits due to imperfections in the CNOTs. For a proof-of-principle demonstration of quantum error correction, it must be shown that the benefits significantly outweigh the drawbacks.

To do this, we compare results for two types of decoding. One is 'full decoding' in which the look-up tables of Eq. 1 use the results from both the code and ancilla qubits (and therefore each R is a string of $2d-1$ bits). The other is 'partial decoding', in which only results from the code qubits are used to construct the look-up tables (and so each R is a d bit string). Since only the former benefits from the syndrome measurement round, since this records additional information about the errors on the ancilla qubits, it should lead to significantly lower logical error probabilities. By showing this, we would demonstrate that the effect of the CNOTs and ancilla measurements in extracting additional information truly has a powerful effect. Note that this form of test was originally proposed in [28] for the surface code.

RESULTS

Repetition codes of size between $d = 3$ and $d = 8$ were studied. For each, the lookup tables where populated using 8192 samples. Note that this is significantly less than the total possible number of measurement outcomes, 2^{2d-1}, for $d = 8$ and not significantly greater than that for $d = 6$ or $d = 7$. Statistical inaccuracies in the lookup table can therefore be expected to affect the quality of the decoding. In order to get an idea of the extent of this effect, logical error probabilities for each case are calculated from 10 different runs and a mean is taken. The standard deviation of these values is used as an estimate of error.

Simulated runs were also used to produce data that could be used for comparison. These runs were done in the same way as above, but only up to $d = 6$.

The simulations also include artificially introduced noise, since they would be otherwise perfect. This was done using partial rotations about the X axis (which rotates between $|\,0\rangle$ and $|\,1\rangle$). The rotation angle depends on whether the qubit was initialized in state $|\,0\rangle$ or $|\,1\rangle$ at the encoding step. For the former, an angle of $\pi/20$ was used. For the latter, $\pi/10$ was used. This mimics the greater probability of $|\,1\rangle \rightarrow |\,0\rangle$ transitions in the real device. Noise was added to all qubits at three points: immediately after encoding, between the two rounds of

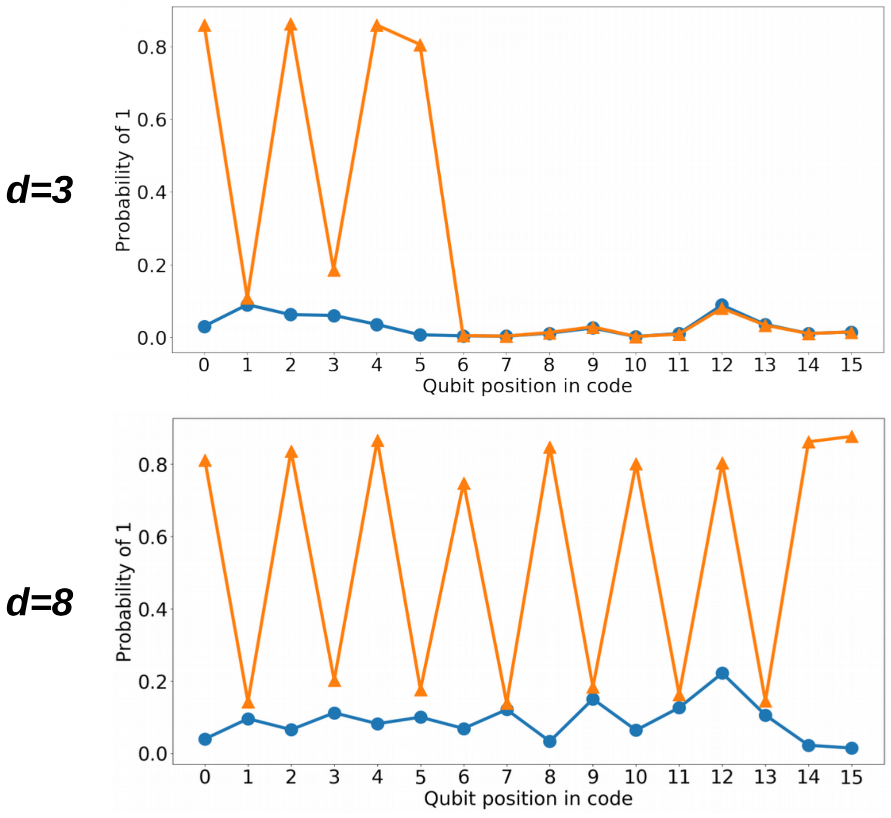


FIG. 3. The state of each qubit at the end of the process, characterized by the probability of the outcome $|1\rangle$ when measured. The extreme examples of $d = 3$ and $d = 8$ are shown. Results for the encoded bit value 0 are shown in blue (circle markers), and those for 1 are in orange (triangle markers). Qubits alternate between code and ancilla qubits, finally ending in a code qubit at $2d - 2$. Qubit s is located at $2d - 1$. Any remaining qubits are unused.

CNOTs and immediately before measurement. This noise mechanism is chosen for its simple implementation, and is not expected to accurately reproduce the true noise processes in the real device. The values of $\pi/20$ and $\pi/10$ are chosen because they reproduce some values and features of the results from the real device.

Full details of the implementation for both the real device and the simulation, including source code and raw data, can be found at [27].

The results in Fig. 3 show the state of each qubit at the end of the process. In the case of no noise, all qubit states should be $|0\rangle$, except for code qubits and qubit s when the bit value 1 encoded. The results show good agreement to these expectations. Qubits expected to be $|0\rangle$ typically have a fidelity of more than

90%, and those expected to be $|1\rangle$ have a fidelity of more than 80%. For the bit stored in the single qubit s, the lowest probability of a logical error is around 0.5% for a stored 0 and 10% for a stored 1. The values for the simulator show good agreement to those for the real device, though this is due only to the fact that the simulated noise parameters were chosen to obtain such an agreement.

The results in Fig. 4 show the logical error probabilities as a function of code distance. These probabilities are found to be much lower than the corresponding values for the qubit s in the case of an encoded bit value of 0. The effect is not so strong for the encoded 0, however this is due to the fact that the noise tends cause qubits to decay towards $|0\rangle$ in all cases, which happens to be advantageous when 0 is encoded.

The results show that the trend is indeed for the error probability to decrease with code distance, as we would expect. A notable exception is found at $d = 6$ for an encoded bit value 0. However, note that this is the first code that uses the qubit numbered 9 in Fig. 3, which can be seen to be particularly noisy. Also, the error probability after full decoding is significantly less than that after partial decoding in almost all cases, which is again in line with expectations.

Fit to an exponential decay

To further analyse these trends, the data is fit to an exponential decay. The simplest possible model of the repetition code is used to avoid over fitting, or unstable fits. This model has a single free parameter, p, which corresponds to the probability of a bit flip error for each qubit. Decoding is done simply by using the majority voting of the code qubits The logical error probability for a given code distance d is therefore the probability of such errors on at least half of the qubits within the code, and so decays exponentially with a factor of $[p/(1-p)]^{\lceil d/2 \rceil}$. Note that the supremum here results in this factor being equal for each odd d and the corresponding $d+1$, due to the lack of a clear majority in the even case.

This simple model corresponds most closely to the case considered for partial decoding, and so is directly used to fit this data. The results are shown in Fig. 5. The fitting was done using a least squares method on the logarithms of the logical error probabilities. The values for the fitting parameter were found to be 0.088 ± 0.001 for encoded 0 and 0.102 ± 0.001 for encoded 1. These values represent the combined effect of preparation, measurement and entangling gate errors, which are each measured to be on the order of 1% using randomized benchmarking [17]. The fitting parameters are therefore certainly of the right order to represent their combined effect. However, the possibility that correlated noise can also form a significant contribution cannot be ruled out.

For the fitting of data from full decoding, let us consider two extremes. Firstly, consider the case for which all CNOTs and ancilla qubits are perfect. Full decoding would then effectively factor into two rounds of error correction, each of which can be modelled in the same way as partial decoding. We will refer to the values of the physical error probability for these rounds as p_0 and p_1, respectively. Clearly $p = p_0(1-p_1)+p_1(1-p_0)$, though we will use the approximation $p = p_0 + p_1$ for simplicity.

The second extremal case is that for which the syndrome measurements result in no useful information being placed on the ancillas. This could either be due to the CNOTs being completely ineffective, or the ancillas being completely decohered. In this case, the ancilla results can be ignored. Full decoding could then be treated as a single round of partial decoding.

We will therefore fit the data for full decoding to two rounds of the simple model. This will be done by first assuming that one round has the error probability $p_0 = p$, found from the fit to the partial decoding data, and the other has no errors ($p_1 = 0$). The fit will then optimize over all other possible p_0 and p_1 given the $p = p_0 + p_1$ constraint.

The resulting p_0 and p_1 can then be used to determine the degree to which we can factorize the error correction into two rounds. If one of these probabilities is found to dominate, it would suggest that the effectiveness of the syndrome measurement approaches the worst case scenario described above.

A more even split would support the notion that the syndrome measurement round is indeed effective. However, it is not a complete proof. Similar results could occur if the CNOTs were ineffective, but suffered correlated noise. Full proof of effectiveness would therefore require a deeper understanding of the effects of noise in the system.

Fitting the data for the full decoding in this way yields $p_0 = 0.054 \pm 0.001$ and $p_1 = 0.034 \pm 0.001$ for stored 0 and $p_0 = p_1 = 0.051 \pm 0.001$ for stored 1. These do show a clear sign of factoring into two distinct rounds, which strengthens the argument that the syndrome measurement round significantly increases the performance of the code.

The data can be seen to show a clear agreement with the exponential decay of the fit lines. However, the data and fitted lines do differ by a typical factor of around 2. This is due to disagreement in the form of the even/odd effects. In some cases, such as partial decoding for an encoded 0, the data and fit show opposite even/odd effects. This is likely due to biased noise changing the nature of the decoding, as will be discussed below. The even/odd effects in the data also appear to be less prominent, with partial decoding for an encoded 1 as the clearest example. These differences could be accounted for by fitting to a more complex model that accounts for biased and correlated noise. However, this would increase the number of fitting parameters. A full study of these effects must therefore be deferred to future experiments with larger devices, such that more data points can be taken. Future experiments would also benefit from greater access to the raw data from devices, rather than the post-processed outputs of 0 and 1 as supplied currently.

Analysis of finite size effects

An exception to the decay of the logical error rate can be seen in Fig. 4 for $d = 3$ and an encoded bit value of 0. This occurs both for the real device and the simulator. It may seem counterintuitive that better results could come from ignoring some information. However, this effect is due to the way the probabilities are calculated and the biased nature of the noise.

For example, consider an extreme case for which noise on the code qubits causes very strong relaxation, such that the code qubits always end in state $|0\rangle$ when 0 was encoded, and almost always end in state $|0\rangle$ even when 1 is encoded. Partial decoding using only these results would then lead to the decoding guessing that 0 was encoded in almost all cases. There would then never be a logical error for an encoded 0, but a logical error would be almost certain for an encoded 1.

More informed decoding could be achieved using the ancilla results. If these show signs that extensive errors have occurred, it would be likely that the code qubits start in in state $|1\rangle$. Many cases for which a logical error would have occurred when using partial decoding could then be successfully decoded using this full decoding. However, for some cases, such as many measurement errors on ancilla qubits, it is possible for an encoded 0 to be misidentified as a 1. The use of a less biased decoder therefore may be more effective in overall, but it can be less effective for the specific case of an encoded 0.

The reason for the effect in this specific case can be seen from Fig. 6. Here results are shown for $d = 3$ and $d = 6$. These graphs show how probable it is to get each possible number of the outcome 1 in the output of the code qubits. For an encoded 0 it is most probable to have a small number of 1s, since each is an error deviating from the perfect output in which all are 0. For an encoded 1 the output is most likely to have a large number of 1s, since each 0 would be an error in this case.

For unbiased noise, the crossover between the curves for encoded 0 and encoded 1 would occur at $d/2$. Any output with less 1s than this should therefore be decoded as a 0, and any with more should be decoded as a 1. For codes with even d, for which it is possible for the number of 1s in the output to be the marginal value of $d/2$, the decoding can be chosen randomly in this case.

For the biased noise, as present in the real device and our simulations, the decoding can deviate from this simple majority voting. This can be seen in the results for $d = 6$. The crossover point has shifted, resulting in $d/2$ no longer being the marginal case that would result for unbiased noise. Instead, having this number of 1s in the output is recognized as being a strong indicator that the encoded bit value was 1 and would be decoded accordingly. Similar effects occur for all other code distances, with the exception of $d = 3$

For $d = 3$ there is a similar shift of the crossover point. This shift is by approximately the same fraction of d as in the $d = 6$ case. However, the smaller nature of the code means that there is less freedom to alter the decoding accordingly. In fact, since finding two 1s in the output is still slightly more likely to correspond to an encoded 0, the optimal decoding will still correspond to majority voting.

Nevertheless, outputs with two ones are found to be very close to being a marginal case, with encoded 0 only being slightly more likely. Decoding these as 0 in all cases will therefore lead to a unfair advantage for this encoded value, causing the feature found in Fig. 4.

CONCLUSIONS

The logical error probabilities of the code sizes considered were found to agree with expectations in most cases, showing that current technology is certainly capable of achieving this simple example of quantum error correction on a relatively large scale. The exceptions found shed light on the nature of noise in the system, with the bias induced by relaxation being the dominant effect.

The quantum part of the code is the mapping of information about errors to ancilla qubits via controlled operations. This is a central technique of quantum error correction. Due to only a single round of ancilla measurements being used, it was possible to compare the effectiveness of decoding both with and without the ancilla results. This allowed direct insight into the effectiveness of the quantum part. It was found that it did indeed allow for significantly better results.

The next major goal of experimental quantum error correction is to build a logical qubit that can be stored as successfully as the logical bit here. The analysis used in this paper, such as the lookup table decoding and comparison of full and partial decoding, would be just as valid in that case [28]. It would therefore be highly interesting to see corresponding results in future.

The most recent version of the Jupyter notebook containing source code for this project can be found at [27]. The raw data is also provided, allowing the analysis of this paper to be repeated and expanded upon. The version of the notebook on which this publication is based is included with the arXiv source files.

ACKNOWLEDGEMENTS

This work was supported by the Swiss National Science Foundation and the NCCR QSIT. JRW thanks Hanhee Paik for comments on the manuscript.

The IBM Quantum Experience was used to produce results for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Quantum Experience team.

[1] J. Kelly *et al.*, Nature **519**, 66 (2014).
[2] M. Takita, A. D. Córcoles, E. Magesan, B. Abdo, M. Brink, A. Cross, J. M. Chow, and J. M. Gambetta, Phys. Rev. Lett. **117**, 210505 (2016).
[3] N. M. Linke *et al.* (2016), arXiv:1611.06946.
[4] T. F. Watson *et al.* (2017), arXiv:1708.04214.
[5] L. Bishop (2017), YouTube video, accessed August 2017, URL `https://www.youtube.com/watch?v=-7L5o-mzLqU`.

[6] P. W. Shor, SIAM J. Comp. **26**, 1484 (1997).

[7] L. Grover, in *28th Annual ACM Symposium on the Theory of Computing (STOC)* (1996), p. 212.

[8] J. R. Wootton (2017), GitHub repository, accessed August 2017, URL `https://github.com/decodoku/A_Game_to_Benchmark_Quantum_Computers`.

[9] D. A. Lidar and T. A. Brun, eds., *Quantum Error Correction* (Cambridge University Press, Cambride, UK, 2013).

[10] C. Vuillot (2017), arXiv:1705.08957.

[11] M. Takita *et al.* (2017), arXiv:1705.09259.

[12] N. Ofek, A. Petrenko, R. Heeres, P. Reinhold, Z. Leghtas, B. Vlastakis, Y. Liu, L. Frunzio, S. M. Girvin, L. Jiang, et al., Nature **536**, 441 (2016).

[13] D. Nigg, M. Müller, E. A. Martinez, P. Schindler, M. Hennrich, T. Monz, M. A. Martin-Delgado, and R. Blatt, Science **345**, 302 (2014).

[14] J. R. Wootton, Quantum Science and Technology **2**, 015006 (2017).

[15] J. M. Gambetta, J. M. Chow, and M. Steffen, npj Quantum Information **3**, 2 (2017), ISSN 2056-6387.

[16] X. Bermudez, *et al.* (2017), arXiv:1705.02771.

[17] IBM (2017), 16-qubit backend: IBM QX team, "ibmqx3 backend specification,", URL `https://ibm.biz/qiskit-ibmqx3`.

[18] A. G. Fowler and J. M. Martinis, Phys. Rev. A **89**, 032316 (2014).

[19] A. Hutter and D. Loss, Phys. Rev. A **89**, 042334 (2014).

[20] J. Helsen, M. Steudtner, M. Veldhorst, and S. Wehner, Quantum Science and Technology (2018).

[21] D. Gottesman, Phys. Rev. A **54**, 1862 (1996).

[22] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, J. Math. Phys. **43**, 4452 (2002).

[23] A. Hutter, J. R. Wootton, and D. Loss, Phys. Rev. A **89**, 022326 (2014).

[24] S. Bravyi, M. Suchara, and A. Vargo, Phys. Rev. A **90**, 032326 (2014).

[25] G. Duclos-Cianci and D. Poulin, Quant. Inf. Comp. **14**, 0721 (2014).

[26] IBM (2017), QISKit API: GitHub repository, accessed August 2017, URL `https://github.com/QISKit/qiskit-api-py`.

[27] J. R. Wootton (2017), GitHub repository, accessed August 2017, URL `https://github.com/decodoku/repetition_code`.

[28] J. R. Wootton, A. Peter, J. R. Winkler, and D. Loss, Phys. Rev. A **96**, 032338 (2017).

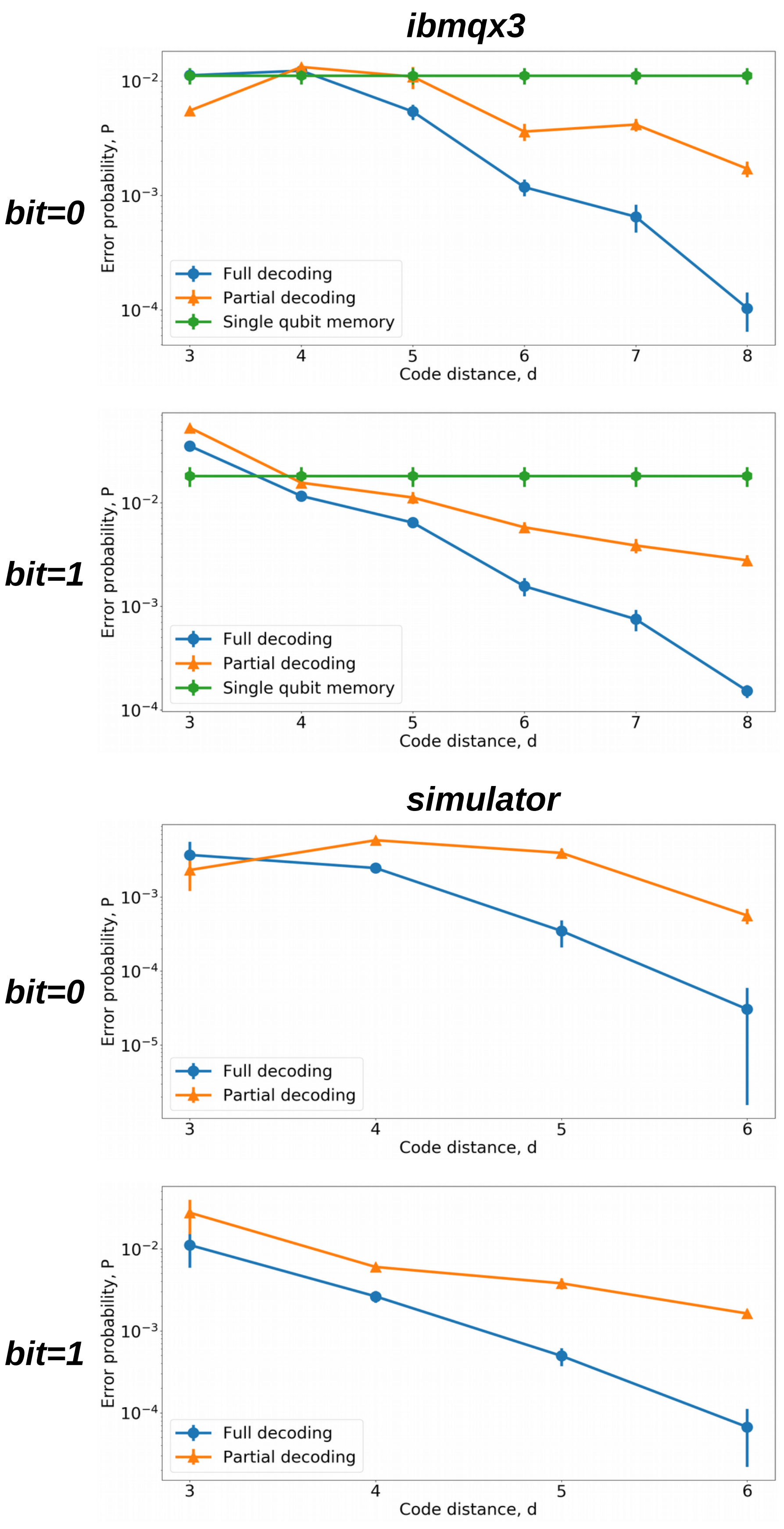


FIG. 4. Probabilities of logical errors for both full and partial decoding. As a comparison, the minimum value of the single qubit memory from all code sizes is plotted across the graph.

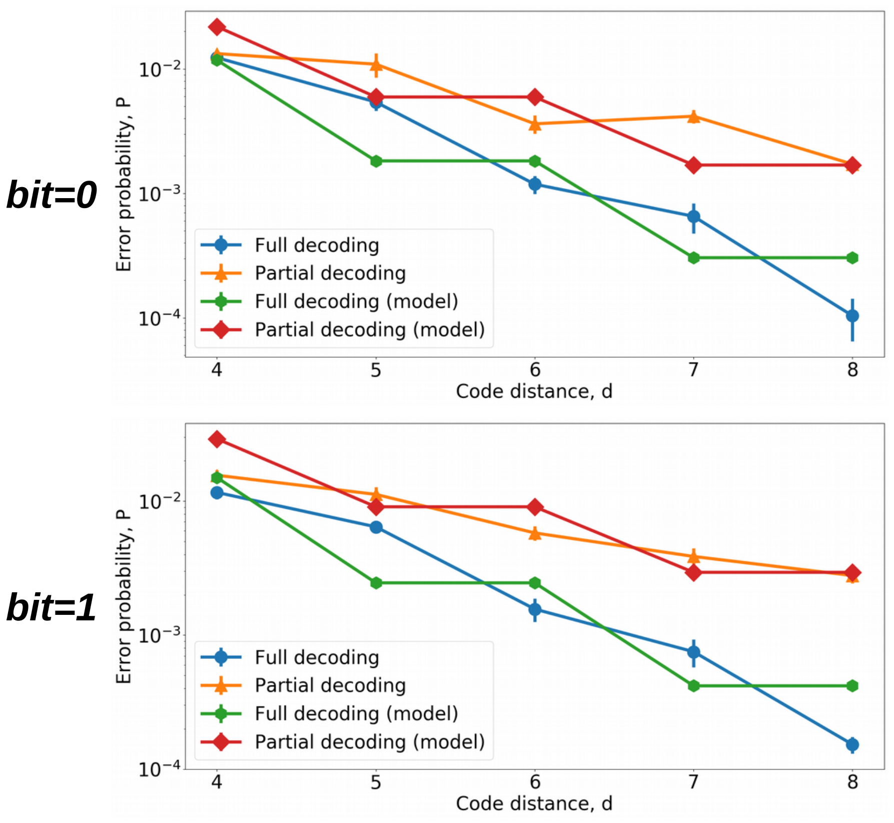


FIG. 5. Probabilities of logical errors for both full and partial decoding, as shown in Fig. 4, with additional fit lines for an exponential decay. The results for $d = 3$ have been omitted to reduce finite size effects.

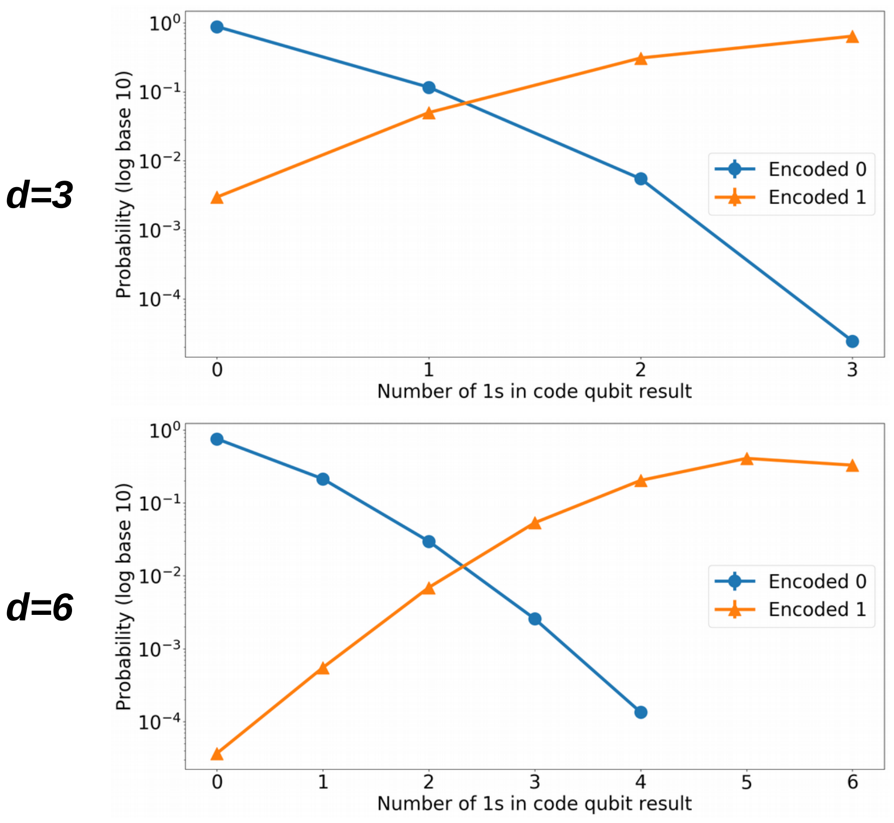


FIG. 6. Probabilities for different numbers of 1s in the output of the code qubits.

Chapter 8

Benchmarking near-term devices with quantum error correction

Abstract

Now that ever more sophisticated devices for quantum computing are being developed, we require ever more sophisticated benchmarks. This includes a need to determine how well these devices support the techniques required for quantum error correction. In this paper we introduce the `topological_codes` module of Qiskit-Ignis, which is designed to provide the tools necessary to perform such tests. Specifically, we use the `RepetitionCode` and `GraphDecoder` classes to run tests based on the repetition code and process the results. As an example, data from a 43 qubit code running on IBM's *Rochester* device is presented.

Publication Information

Originally published in *Quantum Science and Technology*, Volume 5, Number 4 (2018) [22].

Note that the references of this chapter are self-contained, and not included in the overall references.

I. INTRODUCTION

Software comes in many forms. The most prominent forms of software for classical computers are dedicated to applications, in which the device performs a useful task for the end user. Though this is also the goal of quantum software, there will instead be a heavy focus on benchmarking, testing and validation of quantum devices in the near-term. The `topological_codes` module of Qiskit Ignis is one means by which this can be done. In this paper we introduce this new module, and describe its implementation and the methodology behind it.

Quantum software is based on the idea of encoding information in qubits. Most quantum algorithms developed over the past few decades have assumed that these qubits are perfect: they can be prepared in any state we desire, and be manipulated with complete precision. Qubits that obey these assumptions are often known as *logical qubits*.

The last few decades have also seen great advances in finding physical systems that behave as qubits with ever greater fidelity. However, the imperfections can never be removed entirely. These qubits will always be much too imprecise to serve directly as logical qubits. Instead, we refer to them as *physical qubits*.

In the current era of quantum computing, we seek to use physical qubits despite their imperfections, by designing custom algorithms and using error mitigation[1–3]. For the future era of fault-tolerance, however, we must find ways to build logical qubits from physical qubits. This will be done through the process of quantum error correction [4], in which logical qubits are encoded in a large numbers of physical qubits. The encoding is maintained by constantly putting the physical qubits through a highly entangling circuit. Auxiliary degrees of freedom are then constantly measured, to detect signs of errors and allow their effects to be removed.

Because of the vast amount effort required for this process, most operations performed in fault-tolerant quantum computers will be done to serve the purpose of error detection and correction. The logical operations required for quantum computation are essentially just small perturbations to the error correction procedure. As such, as we benchmark our progress towards fault-tolerant quantum computation, we must keep track of how well our devices perform error correction.

Various experiments testing the ideas behind quantum error correction have already been performed [5–16]. These include several experiments based on repetition codes [5, 13, 14]. This is the simplest example of error detection and correction that can be done using the standard techniques of quantum stabilizer codes [17]. Though not a true example of quantum error correction - it uses physical qubits to encode a logical *bit*, rather than a qubit - it serves as a simple guide to all the basic concepts in any quantum error correcting code. Its requirements in terms of qubit number and connectivity are very flexible, allowing it to be straightforwardly implemented on almost any device. This makes it an excellent general-purpose benchmark.

In this paper we will provide a simple introduction to the code, and show how to run instances of it on current prototype devices using the open-source *Qiskit* framework [18]. Specifically, we will use the `topological_codes` module of Qiskit-Ignis, which provides tools to create the quantum circuits required for simple quantum error correcting codes, as well as process the results.

II. INTRODUCTION TO THE REPETITION CODE

A. The basics of error correction

The basic ideas behind error correction are the same for quantum information as for classical information. This allows us to begin by considering a very straightforward example: speaking on the phone. If someone asks you a question to which the answer is ‘yes’ or ‘no’, the way you give your response will depend on two factors:

- How important is it that you are understood correctly?
- How good is your connection?

Both of these can be paramaterized with probabilities. For the first, we can use P_a, the maximum acceptable probability of being misunderstood. If you are being asked to confirm a preference for ice cream flavours, and don't mind too much if you get vanilla rather than chocolate, P_a might be quite high. If you are being asked a question on which someone's life depends, however, P_a will be much lower.
For the second we can use ρ, the probability that your answer is garbled by a bad connection. For simplicity, let's imagine a case where a garbled 'yes' doesn't simply sound like nonsense, but sounds like a 'no'. And similarly a 'no' is transformed into 'yes'. Then ρ is the probability that you are completely misunderstood.
A good connection or a relatively unimportant question will result in $\rho < P_a$. In this case it is fine to simply answer in the most direct way possible: you just say 'yes' or 'no'.
If, however, your connection is poor and your answer is important, we will have $\rho > P_a$. A single 'yes' or 'no' is not enough in this case. The probability of being misunderstood would be too high. Instead we must encode our answer in a more complex structure, allowing the receiver to decode our meaning despite the possibility of the message being disrupted. The simplest method is the one that many would do without thinking: simply repeat the answer many times. For example say 'yes, yes, yes' instead of 'yes' or 'no, no no' instead of 'no'.
If the receiver hears 'yes, yes, yes' in this case, they will of course conclude that the sender meant 'yes'. If they hear 'no, yes, yes', 'yes, no, yes' or 'yes, yes, no', they will probably conclude the same thing, since there is more positivity than negativity in the answer. To be misunderstood in this case, at least two of the replies need to be garbled. The probability for this, P, will be less than ρ. When encoded in this way, the message therefore becomes more likely to be understood. The code cell below shows an example of this.

```python
p = 0.01
P = 3 * p**2 * (1-p) + p**3 # probability of 2 or 3 errors
print('Probability of a single reply being garbled:',p)
print('Probability of a the majority of three replies being␣
 ↪garbled:',P)
```

The output obtained from running the above program snippet is as follows (from henceforth, any such output is displayed directly beneath the cell that it pertains to).

```
Probability of a single reply being garbled: 0.01
Probability of a the majority of three replies being garbled:
0.00029800000000000003
```

If $P < P_a$, this technique solves our problem. If not, we can simply add more repetitions. The fact that $P < \rho$ above comes from the fact that we need at least two replies to be garbled to flip the majority, and so even the most likely possibilities have a probability of $\sim \rho^2$. For five repetitions we'd need at least three replies to be garbled to flip the majority, which happens with probability $\sim \rho^3$. The value for P in this case would then be even lower. Indeed, as we increase the number of repetitions, P will decrease exponentially. No matter how bad the connection, or how certain we need to be of our message getting through correctly, we can achieve it by just repeating our answer enough times.
Though this is a simple example, it contains all the aspects of error correction.

- There is some information to be sent or stored: In this case, a 'yes' or 'no'.

- The information is encoded in a larger system to protect it against noise: In this case, by repeating the message.
- The information is finally decoded, mitigating for the effects of noise: In this case, by trusting the majority of the transmitted messages.

This same encoding scheme can also be used for binary, by simply substituting `0` and `1` for 'yes' and 'no. It can therefore also be easily generalised to qubits by using the states $|0\rangle$ and $|1\rangle$. In each case it is known as the *repetition code*. Many other forms of encoding are also possible in both the classical and quantum cases, which outperform the repetition code in many ways. However, its status as the simplest encoding does lend it to certain applications. One is exactly what it is used for in Qiskit: as the first and simplest test of implementing the ideas behind quantum error correction.

B. Correcting errors in qubits

We will now implement these ideas explicitly using Qiskit. To see the effects of imperfect qubits, we can simply use the qubits of the prototype devices. We can also reproduce the effects in simulations. The function below creates a simple noise models in order to do the latter. The noise models it creates go beyond the simple case discussed earlier, of a single noise event which happens with a probability ρ. Instead we consider two forms of error that can occur. One is a gate error: an imperfection in any operation we perform. We model this here in a simple way, using so-called depolarizing noise. The effect of this will be, with probability ρ_{gate} ,to replace the state of any qubit with a completely random state. For two qubit gates, it is applied independently to each qubit. The other form of noise is that for measurement. This simply flips between a `0` to a `1` and vice-versa immediately before measurement with probability ρ_{meas}.

```
from qiskit.providers.aer.noise import NoiseModel
from qiskit.providers.aer.noise.errors import pauli_error,␣
 ↪depolarizing_error

def get_noise(rho_meas,rho_gate):

    error_meas = pauli_error([('X',rho_meas), ('I', 1 -␣
 ↪rho_meas)])
    error_gate1 = depolarizing_error(rho_gate, 1)
    error_gate2 = error_gate1.tensor(error_gate1)

    noise_model = NoiseModel()
    # measurement error is applied to measurements
    noise_model.add_all_qubit_quantum_error(error_meas,␣
 ↪"measure")
    # single qubit gate error is applied to x gates
    noise_model.add_all_qubit_quantum_error(error_gate1, ["x"])
    # two qubit gate error is applied to cx gates
    noise_model.add_all_qubit_quantum_error(error_gate2, ["cx"])

    return noise_model
```

With this we'll now create such a noise model with a probability of 1% for each type of error.

```
noise_model = get_noise(0.01,0.01)
```

Let's see what affect this has when try to store a `0` using three qubits in state $|0\rangle$. We'll repeat the process `shots=1024` times to see how likely different results are.

```
from qiskit import QuantumCircuit, execute, Aer

# initialize circuit with three qubits in the 0 state
qc0 = QuantumCircuit(3,3,name='0')

qc0.measure(qc0.qregs[0],qc0.cregs[0]) # measure the qubits

# run the circuit with th noise model and extract the counts
counts = execute( qc0, Aer.get_backend('qasm_simulator'),
                  noise_model=noise_model,
                  shots=1024).result().get_counts()

print(counts)
```

```
{'110':2, '001': 9, '100': 11, '010': 14, '000': 988}
```

Note that the `shots=1024` argument here is actually the default argument for the `execute` function, and so it nexed not be included (unless a different number of shots is required). As such, it will not be included in future code snippets.
Here a set of typical results are shown. Results will vary for different runs, but will be qualitatively the same. Specifically, almost all results still come out `'000'`, as they would if there was no noise. Of the remaining possibilities, those with a majority of 0s are most likely. Much less than 10 of the 1024 samples will come out with a majority of 1s. When using this circuit to encode a 0, this means that $P < 1\%$
Now let's try the same for storing a 1 using three qubits in state $|1\rangle$.

```
# initialize circuit with three qubits in the 0 state
qc1 = QuantumCircuit(3,3,name='0')
qc1.x(qc1.qregs[0]) # flip each 0 to 1

qc1.measure(qc1.qregs[0],qc1.cregs[0]) # measure the qubits

# run the circuit with th noise model and extract the counts
counts = execute( qc1, Aer.get_backend('qasm_simulator'),
                  noise_model=noise_model).result().get_counts()

print(counts)
```

```
{'110': 15, '011': 17, '111': 974, '101': 18}
```

The number of samples that come out with a majority in the wrong state (0 in this case) is again much less than 100, so $P < 1\%$. Whether we store a 0 or a 1, we can retrieve the information with a smaller probability of error than either of our sources of noise.
This was possible because the noise we considered was relatively weak. As we increase p_{meas} and p_{gate}, the higher the probability P will be. The extreme case of this is for either of them to have a 50/50 chance of applying the bit flip error, `x`. For example, let's run the same circuit as before but with $p_{meas} = 0.5$ and $p_{gate} = 0$.

```
noise_model = get_noise(0.5,0.0)
counts = execute( qc1, Aer.get_backend('qasm_simulator'),
                  noise_model=noise_model).result().get_counts()
print(counts)
```

```
{'000': 123, '001': 125, '011': 121, '100': 131, '010': 124,␣
  ↪'110': 130, '111':
140, '101': 130}
```

With this noise, all outcomes occur with equal probability, with differences in results being due only to statistical noise. No trace of the encoded state remains. This is an important point to consider for error correction: sometimes the noise is too strong to be corrected. The optimal approach is to combine a good way of encoding the information you require, with hardware whose noise is not too strong.

C. Storing qubits

So far, we have considered cases where there is no delay between encoding and decoding. For qubits, this means that there is no significant amount of time that passes between initializing the circuit, and making the final measurements.
However, there are many cases for which there will be a significant delay. As an obvious example, one may wish to encode a quantum state and store it for a long time, like a quantum hard drive. A less obvious but much more important example is performing fault-tolerant quantum computation itself. For this, we need to store quantum states and preserve their integrity during the computation. This must also be done in a way that allows us to manipulate the stored information in any way we need, and which corrects any errors we may introduce when performing the manipulations.
In all cases, we need account for the fact that errors do not only occur when something happens (like a gate or measurement), they also occur when the qubits are idle. Such noise is due to the fact that the qubits interact with each other and their environment. The longer we leave our qubits idle for, the greater the effects of this noise becomes. If we leave them for long enough, we'll encounter a situation like the $p_{meas} = 0.5$ case above, where the noise is too strong for errors to be reliably corrected.
The solution is to keep measuring throughout. No qubit is left idle for too long. Instead, information is constantly being extracted from the system to keep track of the errors that have occurred.
For the case of classical information, where we simply wish to store a `0` or `1`, this can be done by just constantly measuring the value of each qubit. By keeping track of when the values change due to noise, we can easily deduce a history of when errors occurred.
For quantum information, however, it is not so easy. For example, consider the case that we wish to encode the logical state $|+\rangle$. Our encoding is such that

$$|0\rangle \rightarrow |000\rangle, \quad |1\rangle \rightarrow |111\rangle.$$

To encode the logical $|+\rangle$ state we therefore need

$$|+\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \rightarrow \frac{1}{\sqrt{2}}\left(|000\rangle + |111\rangle\right).$$

With the repetition encoding that we are using, a z measurement (which distinguishes between the $|0\rangle$ and $|1\rangle$ states) of the logical qubit is done using a z measurement of each physical qubit. The final result for the logical measurement is decoded from the physical qubit measurement results by simply looking which output is in the majority.
As mentioned earlier, we can keep track of errors on logical qubits that are stored for a long time by constantly performing z measurements of the physical qubits. However, note that this effectively corresponds to constantly performing z measurements of the logical qubit. This is fine if we are simply storing a `0` or `1`, but it has undesired effects

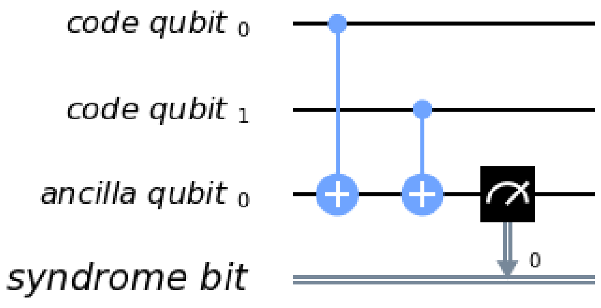
if we are storing a superposition. Specifically: the first time we do such a check for errors, we will collapse the superposition.

This is not ideal. If we wanted to do some computation on our logical qubit, or if we wish to perform a basis change before final measurement, we need to preserve the superposition. Destroying it is an error. But this is not an error caused by imperfections in our devices. It is an error that we have introduced as part of our attempts to correct errors. And since we cannot hope to recreate any arbitrary superposition stored in our quantum computer, it is an error that cannot be corrected.

For this reason, we must find another way of keeping track of the errors that occur when our logical qubit is stored for long times. This should give us the information we need to detect and correct errors, and to decode the final measurement result with high probability. However, it should not cause uncorrectable errors to occur during the process by collapsing superpositions that we need to preserve.

The way to do this is with the following circuit element.

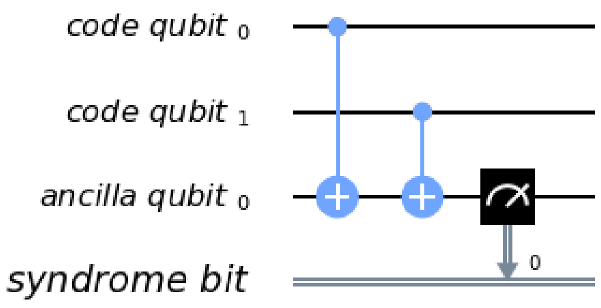
```
from qiskit import QuantumRegister, ClassicalRegister

cq = QuantumRegister(2,'code\ qubit\ ')
lq = QuantumRegister(1,'ancilla\ qubit\ ')
sb = ClassicalRegister(1,'syndrome\ bit\ ')
qc = QuantumCircuit(cq,lq,sb)
qc.cx(cq[0],lq[0])
qc.cx(cq[1],lq[0])
qc.measure(lq,sb)
qc.draw(output='mpl')
```



Here we have three physical qubits. Two are called ‘code qubits’, and the other is called an ‘ancilla qubit’. One bit of output is extracted, called the syndrome bit. The ancilla qubit is always initialized in state $|0\rangle$. The code qubits, however, can be initialized in different states. To see what affect different inputs have on the output, we can create a circuit `qc_init` that prepares the code qubits in some state, and then run the circuit `qc_init+qc`.

First, the trivial case: `qc_init` does nothing, and so the code qubits are initially $|00\rangle$.

```
qc_init = QuantumCircuit(cq)

(qc_init+qc).draw(output='mpl')
```

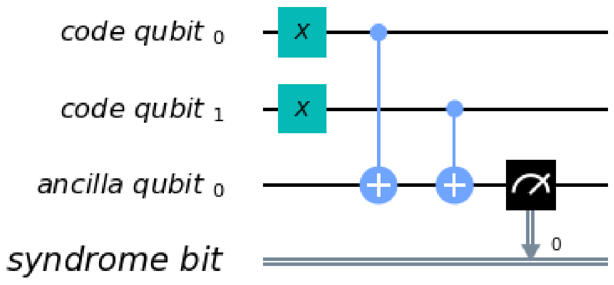


```
counts = execute( qc_init+qc, Aer.get_backend('qasm_simulator')
                ).result().get_counts()
print('Results:',counts)
```

```
Results: {'0': 1024}
```

The outcome, in all cases, is 0.
Now let's try an initial state of $|11\rangle$.

```
qc_init = QuantumCircuit(cq)
qc_init.x(cq)

(qc_init+qc).draw(output='mpl')
```

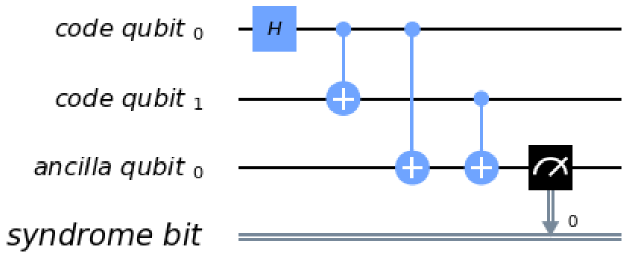


```
counts = execute( qc_init+qc, Aer.get_backend('qasm_simulator')
                ).result().get_counts()
print('Results:',counts)
```

```
Results: {'0': 1024}
```

The outcome in this case is also always 0. Given the linearity of quantum mechanics, we can expect the same to be true also for any superposition of $|00\rangle$ and $|11\rangle$, such as the example below.

```
qc_init = QuantumCircuit(cq)
qc_init.h(cq[0])
qc_init.cx(cq[0],cq[1])

(qc_init+qc).draw(output='mpl')
```

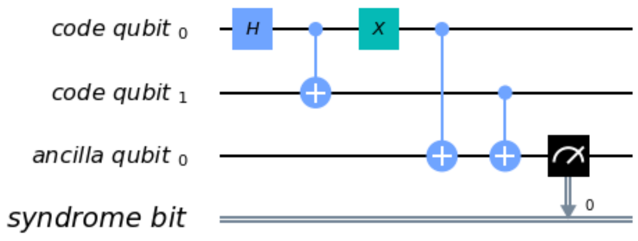


```
counts = execute( qc_init+qc, Aer.get_backend('qasm_simulator')
                ).result().get_counts()
print('Results:',counts)
```

```
Results: {'0': 1024}
```

The opposite outcome will be found for an initial state of $|01\rangle$, $|10\rangle$ or any superposition thereof.

```
qc_init = QuantumCircuit(cq)
qc_init.h(cq[0])
qc_init.cx(cq[0],cq[1])
qc_init.x(cq[0])

(qc_init+qc).draw(output='mpl')
```



```
counts = execute( qc_init+qc, Aer.get_backend('qasm_simulator')
                ).result().get_counts()
print('Results:',counts)
```

```
Results: {'1': 1024}
```

In such cases the output is always `'1'`.

This measurement is therefore telling us about a collective property of multiple qubits. Specifically, it looks at the two code qubits and determines whether their state is the same or different in the z basis. For basis states that are the same in the z basis, like $|00\rangle$ and $|11\rangle$, the measurement simply returns `0`. It also does so for any superposition of these. Since it does not distinguish between these states in any way, it also does not collapse such a superposition.

Similarly, For basis states that are different in the z basis it returns a `1`. This occurs for $|01\rangle$, $|10\rangle$ or any superposition thereof.

Now suppose we apply such a 'syndrome measurement' on all pairs of physical qubits in our repetition code. If their state is described by a repeated $|0\rangle$, a repeated $|1\rangle$, or any superposition thereof, all the syndrome measurements will return `0`. Given this result, we will know that our states are indeed encoded in the repeated states that we want them to be, and can deduce that no errors have occurred. If some syndrome measurements return `1`, however, it is a signature of an error. We can therefore use these measurement results to determine how to decode the result.

D. Quantum repetition code

We now know enough to understand exactly how the quantum version of the repetition code is implemented

We can use it in Qiskit by importing the required tools from Ignis.

```
from qiskit.ignis.verification.topological_codes import␣
 ↪RepetitionCode
from qiskit.ignis.verification.topological_codes import␣
 ↪lookuptable_decoding
from qiskit.ignis.verification.topological_codes import␣
 ↪GraphDecoder
```

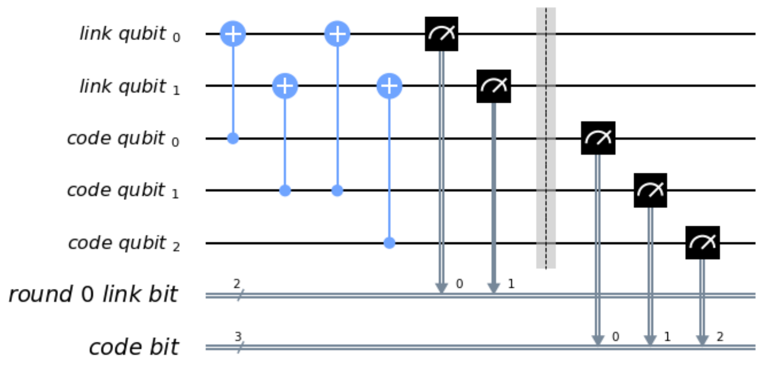
We are free to choose how many physical qubits we want the logical qubit to be encoded in. We can also choose how many times the syndrome measurements will be applied while we store our logical qubit, before the final readout measurement. Let us start with the smallest non-trivial case: three repetitions and one syndrome measurement round. The circuits for the repetition code can then be created automatically from the using the `RepetitionCode` object from Qiskit-Ignis.

```
n = 3
T = 1

code = RepetitionCode(n,T)
```

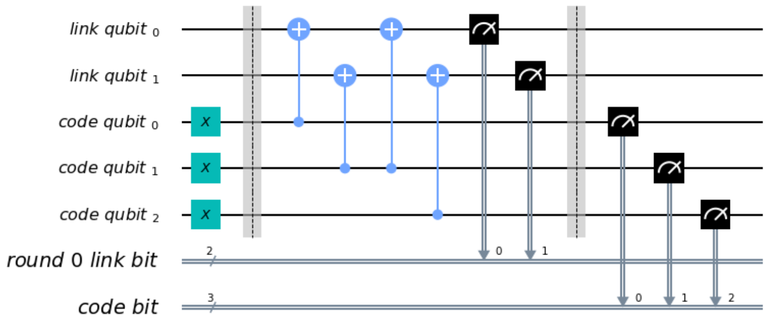
With this we can inspect various properties of the code, such as the names of the qubit registers used for the code and ancilla qubits.

The `RepetitionCode` contains two quantum circuits that implement the code: One for each of the two possible logical bit values. Here are those for logical `0` and `1`, respectively.

```
# this bit is just needed to make the labels look nice
for reg in code.circuit['0'].qregs+code.circuit['1'].cregs:
    reg.name = reg.name.replace('_','\ ') + '\ '

code.circuit['0'].draw(output='mpl')
```



```
code.circuit['1'].draw(output='mpl')
```



In these circuits, we have two types of physical qubits. There are the 'code qubits', which are the three physical qubits across which the logical state is encoded. There are also the 'link qubits', which serve as the ancilla qubits for the syndrome measurements.

Our single round of syndrome measurements in these circuits consist of just two syndrome measurements. One compares code qubits 0 and 1, and the other compares code qubits 1 and 2. One might expect that a further measurement, comparing code qubits 0 and 2, should be required to create a full set. However, these two are sufficient. This is because the information on whether 0 and 2 have the same z basis state can be inferred from the same information about 0 and 1 with that for 1 and 2. Indeed, for n qubits, we can get the required information from just $n-1$ syndrome measurements of neighbouring pairs of qubits.

Running these circuits on a simulator without any noise leads to very simple results.

```
def get_raw_results(code,noise_model=None):

    circuits = code.get_circuit_list()
    job = execute( circuits, Aer.get_backend('qasm_simulator'),
                  noise_model=noise_model )
    raw_results = {}
    for log in ['0','1']:
        raw_results[log] = job.result().get_counts(log)
    return raw_results

raw_results = get_raw_results(code)
for log in raw_results:
    print('Logical',log,':',raw_results[log],'\n')
```

```
Logical 0 : {'000 00': 1024}

Logical 1 : {'111 00': 1024}
```

Here we see that the output comes in two parts. The part on the right holds the outcomes of the two syndrome measurements. That on the left holds the outcomes of the three final measurements of the code qubits.
For more measurement rounds, $T = 4$ for example, we would have the results of more syndrome measurements on the right.

```
code = RepetitionCode(n,4)

raw_results = get_raw_results(code)
for log in raw_results:
    print('Logical',log,':',raw_results[log],'\n')
```

```
Logical 0 : {'000 00 00 00 00': 1024}

Logical 1 : {'111 00 00 00 00': 1024}
```

For more repetitions, $n = 5$ for example, each set of measurements would be larger. The final measurement on the left would be of n qubits. The T syndrome measurements would each be of the $n-1$ possible neighbouring pairs.

```
code = RepetitionCode(5,4)

raw_results = get_raw_results(code)
for log in raw_results:
    print('Logical',log,':',raw_results[log],'\n')
```

```
Logical 0 : {'00000 0000 0000 0000 0000': 1024}

Logical 1 : {'11111 0000 0000 0000 0000': 1024}
```

E. Lookup table decoding

Now let's return to the $n = 3$, $T = 1$ example and look at a case with some noise.

```
code = RepetitionCode(3,1)

noise_model = get_noise(0.05,0.05)

raw_results = get_raw_results(code,noise_model)
for log in raw_results:
    print('Logical',log,':',raw_results[log],'\n')
```

```
Logical 0 : {'011 10': 1, '001 10': 5, '111 00': 2, '010 01':␣
 ↪24, '000 11': 6,
'101 00': 2, '100 01': 2, '100 11': 1, '100 10': 7, '001 11': 2,␣
 ↪'010 00': 52,
'001 01': 2, '110 01': 4, '000 00': 642, '000 01': 64, '001 00':␣
 ↪49, '010 10':
5, '000 10': 73, '011 00': 5, '111 01': 1, '011 01': 1, '101 01':
 ↪ 1, '100 00':
67, '110 00': 5, '010 11': 1}

Logical 1 : {'011 10': 13, '001 10': 3, '111 00': 594, '010 01':␣
 ↪2, '011 11': 2,
'111 10': 65, '101 00': 55, '100 11': 3, '101 10': 11, '001 11':␣
 ↪1, '101 11':
16, '001 01': 1, '010 00': 9, '110 01': 22, '111 11': 9, '000␣
 ↪00': 1, '110 11':
4, '001 00': 7, '010 10': 6, '110 10': 2, '011 00': 44, '111 01':
 ↪ 61, '011 01':
9, '101 01': 17, '100 00': 4, '110 00': 63}
```

Here we have created `raw_results`, a dictionary that holds both the results for a circuit encoding a logical `0` and `1` encoded for a logical `1`.
Our task when confronted with any of the possible outcomes we see here is to determine what the outcome should have been, if there was no noise. For an outcome of `'000 00'` or `'111 00'`, the answer is obvious. These are the results we just saw for a logical `0` and logical `1`, respectively, when no errors occur. The former is the most common outcome for the logical `0` even with noise, and the latter is the most common for the logical `1`. We will therefore conclude that the outcome was indeed that for logical `0` whenever we encounter `'000 00'`, and the same for logical `1` when we encounter `'111 00'`.
Though this tactic is optimal, it can nevertheless fail. Note that `'111 00'` typically occurs in a handful of cases for an encoded `0`, and `'000 00'` similarly occurs for an encoded `1`. In this case, through no fault of our own, we will incorrectly decode the output. In these cases, a large number of errors conspired to make it look like we had a noiseless case of the opposite logical value, and so correction becomes impossible.
We can employ a similar tactic to decode all other outcomes. The outcome `'001 00'`, for example, occurs far more for a logical `0` than a logical `1`. This is because it could be caused by just a single measurement error in the former case (which incorrectly reports a single `0` to be `1`), but would require at least two errors in the latter. So whenever we see `'001 00'`, we can decode it as a logical `0`.
Applying this tactic over all the strings is a form of so-called 'lookup table decoding'. Whenever an output string is obtained, it is compared to a large body of results for

known logical values. Then most likely logical value can then be inferred. For many qubits, this quickly becomes intractable, as the number of possible outcomes becomes so large. In these cases, more algorithmic decoders are needed. However, lookup table decoding works well for testing out small codes.
We can use tools in Qiskit to implement lookup table decoding for any code. For this we need two sets of results. One is the set of results that we actually want to decode, and for which we want to calculate the probability of incorrect decoding, P. We will use the `raw_results` we already have for this.
The other set of results is one to be used as the lookup table. This will need to be run for a large number of samples, to ensure that it gets good statistics for each possible outcome. We'll use `shots=10000`.

```
circuits = code.get_circuit_list()
job = execute( circuits, Aer.get_backend('qasm_simulator'),
               noise_model=noise_model, shots=10000 )
table_results = {}
for log in ['0','1']:
    table_results[log] = job.result().get_counts(log)
```

With this data, which we call `table_results`, we can now use the `lookuptable_decoding` function from Qiskit. This takes each outcome from `raw_results` and decodes it with the information in `table_results`. Then it checks if the decoding was correct, and uses this information to calculate P.

```
P = lookuptable_decoding(raw_results,table_results)
print('P =',P)
```

```
P = {'0': 0.0238, '1': 0.0237}
```

Here we see that the values for P are lower than those for ρ_{meas} and ρ_{gate}, so we get an improvement in the reliability for storing the bit value. Note also that the value of P for an encoded `1` is higher than that for `0`. This is because the encoding of `1` requires the application of `x` gates, which are an additional source of noise.

F. Graph theoretic decoding

The decoding considered above produces the best possible results, and does so without needing to use any details of the code. However, it has a major drawback that counters these advantages: the lookup table grows exponentially large as code size increases. For this reason, decoding is typically done in a more algorithmic manner that takes into account the structure of the code and its resulting syndromes.
The `topological_codes` module is designed to support multiple codes that share the same structure, and therefore can be decoded using the same methods. These methods are all based on similar graph theoretic minimization problems, where the graph in question is one that can be derived from the syndrome. The repetition code is one example that can be decoding in this way, and it is with this example that we will explain the graph-theoretic decoding in this section. Other examples are the toric and surface codes[19, 20], 2D color codes[21, 22] and matching codes[23]. All of these are examples of so-called topological quantum error correcting codes, which led to the name of the module. However, note that not all topological codes are compatible with such a decoder. Also, some non-topological codes will be compatible (such as the repetition code).
To find the the graph that will be used in the decoding, some post-processing of the syndromes is required. Instead of using the form shown above, with the final measurement of the code qubits on the left and the outputs of the syndrome measurement rounds on

the right, we use the `process_results` method of the code object to rewrite them in a different form.
For example, below is the processed form of a `raw_results` dictionary, in this case for $n = 3$ and $T = 2$. Only results with 50 or more samples are shown for clarity.

```
code = RepetitionCode(3,2)

raw_results = get_raw_results(code,noise_model)

results = code.process_results( raw_results )

for log in ['0','1']:
    print('\nLogical ' + log + ':')
    print('raw results       ', {string:
 ↪raw_results[log][string] for string in raw_results[log] if␣
 ↪raw_results[log][string]>=50 })
    print('processed results ', {string:results[log][string]␣
 ↪for string in results[log] if results[log][string]>=50 })
```

```
Logical 0:
raw results        {'000 00 00': 485, '000 00 01': 55}
processed results  {'0 0  00 00 00': 485, '0 0  01 01 00': 55}

Logical 1:
raw results        {'111 10 00': 51, '111 01 00': 57, '111 00␣
 ↪00': 455, '111 00
10': 51}
processed results  {'1 1  00 10 10': 51, '1 1  00 01 01': 57, '1␣
 ↪1  00 00 00':
455, '1 1  10 10 00': 51}
```

Here we can see that `'000 00 00'` has been transformed to `'0 0  00 00 00'`, and `'111 00 00'` to `'1 1  00 00 00'`, and so on.
In these new strings, the `0 0` to the far left for the logical `0` results and the `1 1` to the far left of the logical `1` results are the logical readout. Any code qubit could be used for this readout, since they should (without errors) all be equal. It would therefore be possible in principle to just have a single `0` or `1` at this position. We could also do as in the original form of the result and have n, one for each qubit. Instead we use two, from the two qubits at either end of the line. The reason for this will be shown later. In the absence of errors, these two values will always be equal, since they represent the same encoded bit value.
After the logical values follow the $n-1$ results of the syndrome measurements for the first round. A `0` implies that the corresponding pair of qubits have the same value, and `1` implies they they are different from each other. There are $n-1$ results because the line of d code qubits has $n-1$ possible neighbouring pairs. In the absence of errors, they will all be `0`. This is exactly the same as the first such set of syndrome results from the original form of the result.
The next block is the next round of syndrome results. However, rather than presenting these results directly, it instead gives us the syndrome change between the first and second rounds. It is therefore the bitwise `OR` of the syndrome measurement results from the second round with those from the first. In the absence of errors, they will all be `0`.
Any subsequent blocks follow the same formula, though the last of all requires some comment. This is not measured using the standard method (with a link qubit). Instead

it is calculated from the final readout measurement of all code qubits. Again it is presented as a syndrome change, and will be all `0` in the absence of errors. This is the $T+1$-th block of syndrome measurements since, as it is not done in the same way as the others, it is not counted among the T syndrome measurement rounds.

The following examples further illustrate this convention.

Example 1: `0 0  0110 0000 0000` represents a $d=5$, $T=2$ repetition code with encoded `0`. The syndrome shows that (most likely) the middle code qubit was flipped by an error before the first measurement round. This causes it to disagree with both neighboring code qubits for the rest of the circuit. This is shown by the syndrome in the first round, but the blocks for subsequent rounds do not report it as it no longer represents a change. Other sets of errors could also have caused this syndrome, but they would need to be more complex and so presumably less likely.

Example 2: `0 0  0010 0010 0000` represents a $d=5$, $T=2$ repetition code with encoded `0`. Here one of the syndrome measurements reported a difference between two code qubits in the first round, leading to a `1`. The next round did not see the same effect, and so resulted in a `0`. However, since this disagreed with the previous result for the same syndrome measurement, and since we track syndrome changes, this change results in another `1`. Subsequent rounds also do not detect anything, but this no longer represents a change and hence results in a `0` in the same position. Most likely the measurement result leading to the first `1` was an error.

Example 3: `0 1  0000 0001 0000` represents a $d=5$, $T=2$ repetition code with encoded `1`. A code qubit on the end of the line is flipped before the second round of syndrome measurements. This is detected by only a single syndrome measurement, because it is on the end of the line. For the same reason, it also disturbs one of the logical readouts.

Note that in all these examples, a single error causes exactly two characters in the string to change from the value they would have with no errors. This is the defining feature of the convention used to represent the syndrome in `topological_codes`. It is used to define the graph on which the decoding problem is defined.

Specifically, the graph is constructed by first taking the circuit encoding logical `0`, for which all bit values in the output string should be `0`. Many copies of this are then created and run on a simulator, with a different single Pauli operator inserted into each. This is done for each of the three types of Pauli operator on each of the qubits and at every circuit depth. The output from each of these circuits can be used to determine the effects of each possible single error. Since the circuit contains only Clifford operations, the simulation can be performed efficiently.

In each case, the error will change exactly two of the characters (unless it has no effect). A graph is then constructed for which each bit of the output string corresponds to a node, and the pairs of bits affected by the same error correspond to an edge.

The process of decoding a particular output string typically requires the algorithm to deduce which set of errors occurred, given the syndrome found in the output string. This can be done by constructing a second graph, containing only nodes that correspond to non-trivial syndrome bits in the output. An edge is then placed between each pair of nodes, with an corresponding weight equal to the length of the minimal path between those nodes in the original graph. A set of errors consistent with the syndrome then corresponds then to finding a perfect matching of this graph. To deduce the most likely set of errors to have occurred, a good tactic would be to find one with the least possible number of errors that is consistent with the observed syndrome. This corresponds to a minimum weight perfect matching of the graph [20].

Using minimal weight perfect matching is a standard decoding technique for the repetition code and surface codes [20, 24], and is implemented in Qiskit Ignis. It can also be used in other cases, such as Color codes, but it does not find the best approximation of the most likely set of errors for every code and noise model. For that reason, other decoding techniques based on the same graph can be used. The `GraphDecoder` of Qiskit Ignis calculates these graphs for a given code, and will provide a range of

methods to analyze it. At time of writing, only minimum weight perfect matching is implemented.
Note that, for codes such as the surface code, it is not strictly true than each single error will change the value of only two bits in the output string. A σ^y error, for example would flip a pair of values corresponding to two different types of stabilizer, which are typically decoded independently. Output for these codes will therefore be presented in a way that acknowledges this, and analysis of such syndromes will correspondingly create multiple independent graphs to represent the different syndrome types.

III. RUNNING A REPETITION CODE BENCHMARKING PROCEDURE

We will now run examples of repetition codes on real devices, and use the results as a benchmark. First, we will briefly summarize the process. This applies to this example of the repetition code, but also for other benchmarking procedures in `topological_codes`, and indeed for Qiskit Ignis in general. In each case, the following three-step process is used.

1. A task is defined. Qiskit Ignis determines the set of circuits that must be run and creates them.
2. The circuits are run. This is typically done using Qiskit. However, in principle any service or experimental equipment could be interfaced.
3. Qiskit Ignis is used to process the results from the circuits, to create the output required for the given task.

For `topological_codes`, step 1 requires the type and size of quantum error correction code to be chosen. Each type of code has a dedicated Python class. A corresponding object is initialized by providing the parameters required, such as `n` and `T` for a `RepetitionCode` object. The resulting object then contains the circuits corresponding to the given code encoding simple logical qubit states (such as $|0\rangle$ and $|1\rangle$), and then running the procedure of error detection for a specified number of rounds, before final readout in a straightforward logical basis (typically a standard $|0\rangle/|1\rangle$ measurement).
For `topological_codes`, the main processing of step 3 is the decoding, which aims to mitigate for any errors in the final readout by using the information obtained from error detection. The optimal algorithm for decoding typically varies between codes.
The decoding is done by the `GraphDecoder` class. A corresponding object is initialized by providing the code object for which the decoding will be performed. This is then used to determine the graph on which the decoding problem will be defined. The results can then be processed using the various methods of the decoder object.
In the following we will see the above ideas put into practice for the repetition code. In doing this we will employ two Boolean variables, `step_2` and `step_3`. The variable `step_2` is used to show which parts of the program need to be run when taking data from a device, and `step_3` is used to show the parts which process the resulting data. Both are set to false by default, to ensure that all the program snippets below can be run using only previously collected and processed data. However, to obtain new data one only needs to use `step_2 = True`, and perform decoding on any data one only needs to use `step_3 = True`.

```
step_2 = False
step_3 = False
```

To benchmark a real device we need the tools required to access that device over the cloud, and compile circuits suitable to run on it. These are imported as follows.

```
from qiskit import IBMQ
from qiskit.compiler import transpile
from qiskit.transpiler import PassManager
```

We can now create the backend object, which is used to run the circuits. This is done by supplying the string used to specify the device. Here the 53 qubit `'ibmq_rochester'` device is used. Of the publicly accessible devices, `'ibmq_16_melbourne'` can also be used. This has 15 active qubits at time of writing. To gather new from this device, the device name below is the only thing that would need to be changed.

```
device_name = 'ibmq_rochester'

if step_2:

    IBMQ.load_account()

    for provider in IBMQ.providers():
        for potential_backend in provider.backends():
            if potential_backend.name()==device_name:
                backend = potential_backend

    coupling_map = backend.configuration().coupling_map
```

When running a circuit on a real device, a transpilation process is first implemented. This changes the gates of the circuit into the native gate set implement by the device. In some cases these changes are fairly trivial, such as expressing each Hadamard as a single qubit rotation by the corresponding Euler angles. However, the changes can be more major if the circuit does not respect the connectivity of the device. For example, suppose the circuit requires a controlled-NOT that is not directly implemented by the device. The effect must be then be reproduced with techniques such as using additional controlled-NOT gates to move the qubit states around. As well as introducing additional noise, this also delocalizes any noise already present. A single qubit error in the original circuit could become a multiqubit monstrosity under the action of the additional transpilation. Such non-trivial transpilation must therefore be prevented when running quantum error correction circuits.

Tests of the repetition code require qubits to be effectively ordered along a line. The only controlled-NOT gates required are between neighbours along that line. Our first job is therefore to study the coupling map of the device, shown in Fig. 1, and find a line.

For Melbourne it is possible to find a line that covers all 15 qubits. The choice one specified in the list `line` below is designed to avoid the most error prone `cx` gates. For the 53 qubit *Rochester* device, there is no single line that covers all 53 qubits. Instead we can use the following choice, which covers 43.

```
if device_name=='ibmq_16_melbourne':
    line = [13,14,0,1,2,12,11,3,4,10,9,5,6,8,7]
elif device_name=='ibmq_rochester':
    line = [10,11,17,23,22,21,20,19,16,7,8,9,5,0,1,
            2,3,4,6,13,14,15,18,27,26,25,29,36,37,38,
            41,50,49,48,47,46,45,44,43,42,39,30,31]
```

Now we know how many qubits we have access to, we can create the repetition code objects for each code that we will run. Note that a code with `n` repetitions uses n code qubits and $n-1$ link qubits, and so $2n-1$ in all.

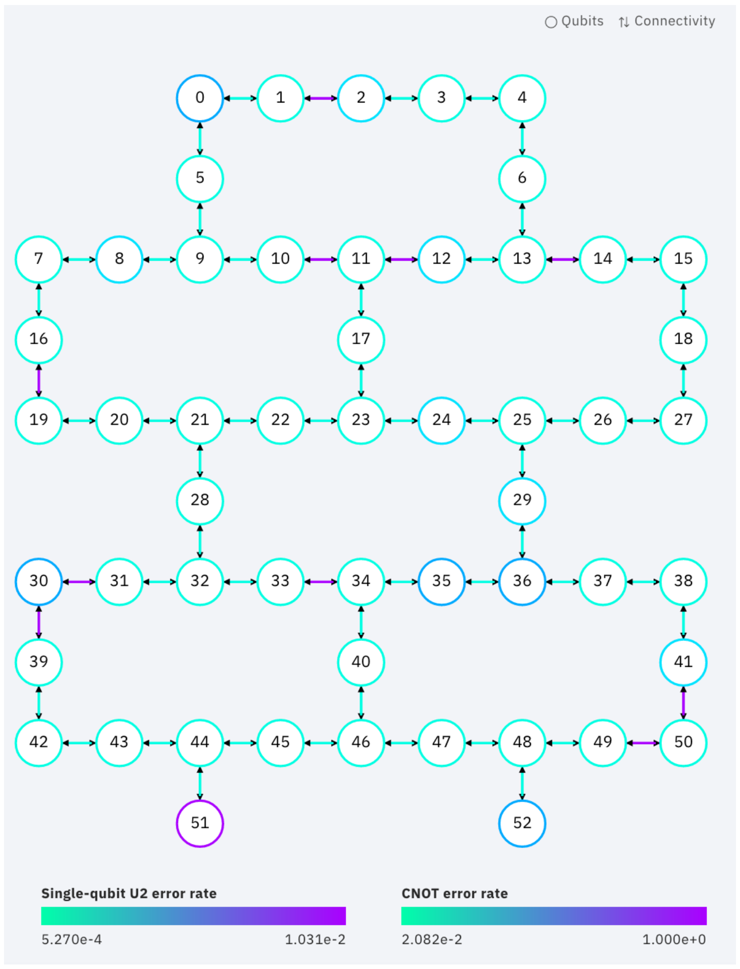


FIG. 1: The layout of the *Rochester* device. Colours represent error probabilities for controlled-NOTs and readout on qubits.

```
n_min = 3
n_max = int((len(line)+1)/2)

code = {}

for n in range(n_min,n_max+1):
    code[n] = RepetitionCode(n,1)
```

Before running the circuits from these codes, we need to ensure that the transpiler knows which physical qubits on the device it should use. This means using the qubit of `line[0]` to serve as the first code qubit, that of `line[1]` to be the first link qubit, and so on. This is done by the following function, which takes a repetition code object and a `line`, and creates a Python dictionary to specify which qubit of the code corresponds to which element of the line.

```
def get_initial_layout(code,line):
    initial_layout = {}
    for j in range(n):
        initial_layout[code.code_qubit[j]] = line[2*j]
    for j in range(n-1):
        initial_layout[code.link_qubit[j]] = line[2*j+1]
    return initial_layout
```

Now we can transpile the circuits, to create the circuits that will actually be run by the device. A check is also made to ensure that the transpilation indeed has not introduced non-trivial effects by increasing the number of qubits. Furthermore, the compiled circuits are collected into a single list, to allow them all to be submitted at once in the same batch job.

```
if step_2:

    circuits = []
    for n in range(n_min,n_max+1):
        initial_layout = get_initial_layout(code[n],line)
        for log in ['0','1']:
            circuits.append( transpile(code[n].circuit[log],
                                       backend=backend,
                                       ␣
 ↪initial_layout=initial_layout) )
            num_cx = dict(circuits[-1].count_ops())['cx']
            assert num_cx==2*(n-1), str(num_cx) + ' instead of␣
 ↪' + str(2*(n-1)) + ' cx gates for n = ' + str(n)
```

We are now ready to run the job. As with the simulated jobs considered already, the results from this are extracted into a dictionary `raw_results`. However, in this case it is extended to hold the results from different code sizes. This means that `raw_results[n]` in the following is equivalent to one of the `raw_results` dictionaries used earlier, for a given `n`.

```
if step_2:

    job = execute(circuits,backend,shots=8192)

    raw_results = {}
    j = 0
    for n in range(n_min,n_max+1):
        raw_results[n] = {}
        for log in ['0','1']:
            raw_results[n][log] = job.result().get_counts(j)
            j += 1
```

It can be convenient to save the data to file, so that the processing of step 3 can be done or repeated at a later time.

```
if step_2: # save results
    with open('results/raw_results_'+device_name+'.txt', 'w')␣
 ↪as file:
        file.write(str(raw_results))
elif step_3: # read results
    with open('results/raw_results_'+device_name+'.txt', 'r')␣
 ↪as file:
        raw_results = eval(file.read())
```

As was described previously, some post-processing of the syndromes is required to find the the graph that will be used in the decoding. This is done using the `process_results` method of each repetition code object `code[n]`. Specifically, we use it to create a results dictionary `results[n]` from each `raw_results[n]`.

```
if step_3:
    results = {}
    for n in range(n_min,n_max+1):
        results[n] = code[n].process_results( raw_results[n] )
```

The decoding also needs us to set up the `GraphDecoder` object for each code. The initialization of these involves the construction of the graph corresponding to the syndrome, as described in the last section.

```
if step_3:
    dec = {}
    for n in range(n_min,n_max+1):
        dec[n] = GraphDecoder(code[n])
```

Finally, the decoder object can be used to process the results. Here the default algorithm, minimum weight perfect matching, is used. The end result is a calculation of the logical error probability. This is simply the probability that the decoded output is `1` when the encoded value was `0`, or vice-versa.

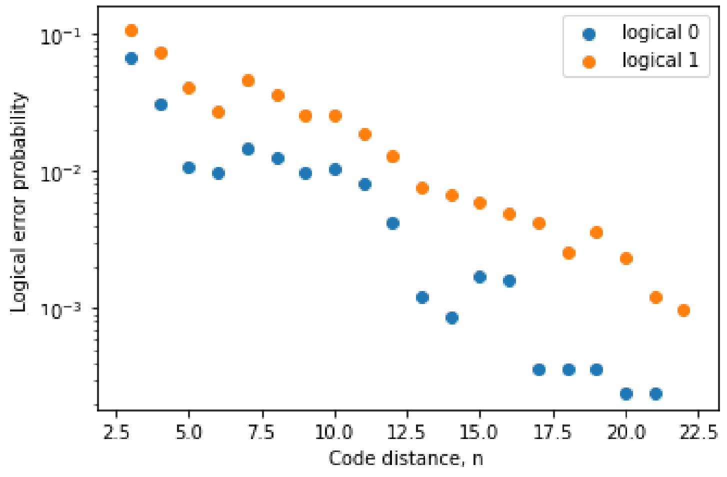
When running step 3, the following snippet also saves the logical error probabilities. Otherwise, it reads in previously saved probabilities.

```
if step_3:

    logical_prob_match = {}
    for n in range(n_min,n_max+1):
        logical_prob_match[n] = dec[n].
 ↪get_logical_prob(results[n])

    with open('results/logical_prob_match_'+device_name+'.txt',␣
 ↪'w') as file:
        file.write(str(logical_prob_match))

else:
    with open('results/logical_prob_match_'+device_name+'.txt',␣
 ↪'r') as file:
        logical_prob_match = eval(file.read())
```

The resulting logical error probabilities are displayed in the following graph, which uses a log scale on the y axis. We would expect that the logical error probability decays exponentially with increasing *n*. If this is the case, it is a confirmation that the device is compatible with this basis test of quantum error correction. If not, it implies that the qubits and gates are not sufficiently reliable.

Fortunately, the results from this device does show the desired trend. Deviations can be observed, however, which can serve as a starting point for investigations into exactly why the device behaves as it does. However, in this paper we simply seek to show an example of the `topological_codes` module in action, and will not perform more in-depth analysis.

```
import matplotlib.pyplot as plt
import numpy as np

x_axis = range(n_min,n_max+1)
P = { log: [logical_prob_match[n][log] for n in x_axis] for log␣
 ↪in ['0', '1'] }

ax = plt.gca()
plt.xlabel('Code distance, n')
plt.ylabel('Logical error probability')
ax.scatter( x_axis, P['0'], label="logical 0", s=512)
ax.scatter( x_axis, P['1'], label="logical 1", s=512)
ax.set_yscale('log')
ax.set_ylim(ymax=1.5*max(P['0']+P['1']),
            ymin=0.75*min([ p for p in P['0']+P['1'] if p>0 ]))
plt.legend()

plt.show()
```



Another insight we can gain is to use the results to determine how likely certain error processes are to occur.
To see how this can be done, recall that each node in the syndrome graph corresponds to a particular syndrome measurement being performed at a particular point within the circuit. A pair of nodes are connected by an edge if and only if a single error, occurring on a particular qubit at a particular point within the circuit, can cause the value of both to change.
For any such pair of adjacent nodes, we will specifically consider the values C_{11} and C_{00} , where the former is the number of counts in `results[n]['0']` corresponding to the syndrome value of both adjacent nodes being 1, and the latter is the same for them both being 0.
The most likely cause for each event recorded in C_{11} is the occurrence of the error corresponding to the edge between these two nodes. Also, it is most likely that this error has not occurred for each event recorded in C_{00}. As such, to first order, we can state that

$$\frac{p}{1-p} \approx \frac{C_{11}}{C_{00}}$$

Where p is the probability of the error corresponding to the edge between these two nodes.
For example, suppose that one of the nodes we are considering corresponds to the syndrome measurement of code qubits 0 and 1 in the first round, and the other corresponds to the same for code qubits 1 and 2. If both syndrome measurements output `0`, it is most likely that none of these three qubits suffered an error (during the initial state preparation or during the controlled-NOTs required for the syndrome measurements). If both syndrome measurements output `1`, it is most likely that a single error occurred on the shared qubit 2. The probability p in this case would therefore be that for such an error on qubit 2, either in preparation or during the controlled-NOTs.
The decoder object has a method `weight_syndrome_graph` which determines these ratios, and assigns each edge the weight $-\ln(p/(1-p))$. By employing this method and inspecting the weights, we can easily retrieve these probabilities.

```
import pandas as pd

if step_3:

    dec[n_max].weight_syndrome_graph(results=results[n_max])

    probs = []
    for edge in dec[n_max].S.edges:
        ratio = np.exp(-dec[n_max].S.
 ↪get_edge_data(edge[0],edge[1])['distance'])
        probs.append( ratio/(1+ratio) )

    probs_summary = pd.Series(probs).describe().to_dict()

    with open('results/probs_summary_'+device_name+'.txt', 'w')␣
 ↪as file:
        file.write(str(probs_summary))

else:

    with open('results/probs_summary_'+device_name+'.txt', 'r')␣
 ↪as file:
        probs_summary = eval(file.read())

for stat in probs_summary:
    print(stat+':',probs_summary[stat])
```

```
count: 85.0
mean: 0.15019428099809715
std: 0.11236995205935389
min: 0.026545002073828285
25%: 0.05599450360700791
50%: 0.11155859846959323
75%: 0.20286006128702758
max: 0.41629711751662973
```

These probabilities come from a novel way of benchmarking the device, that is distinct from currently used methods. Since no other set of available error probabilities are exactly equivalent, it is not possible to do a direct comparison. Nevertheless, since the results from the standard benchmarking of the device can be easily obtained from the `backend` object, we will generate a similar summary. Specifically, we will look at the readout errors and controlled-NOT gate errors.

```
if step_3:

    gate_probs = []
    for j,qubit in enumerate(line):

        gate_probs.append( backend.properties().
 ↪readout_error(qubit) )

        cx1,cx2 = 0,0
        if j>0:
            gate_probs.append(backend.properties().
 ↪gate_error('cx',[qubit,line[j-1]]) )
        if j<len(line)-1:
            gate_probs.append(backend.properties().
 ↪gate_error('cx',[qubit,line[j+1]]) )

    gate_probs_summary = pd.Series(gate_probs).describe().
 ↪to_dict()

    with open('results/gate_probs_summary_'+device_name+'.txt',␣
 ↪'w') as file:
        file.write(str(gate_probs_summary))

else:

    with open('results/gate_probs_summary_'+device_name+'.txt',␣
 ↪'r') as file:
        gate_probs_summary = eval(file.read())

for stat in gate_probs_summary:
    print(stat+':',gate_probs_summary[stat])
```

```
count: 127.0
mean: 0.06850161154431056
std: 0.05891171361248407
min: 0.019374999999999992
25%: 0.03708774898932393
50%: 0.046021125148897085
75%: 0.0857536432141715
max: 0.33624999999999994
```

Note that, despite the different methods used to obtain these values, there is excellent agreement for the minimum and first quartile. However, the values from the repetition code do show more of a skew to larger values in the upper quartiles. This could be due to the fact that repetition code values are obtained from an approximation that is less accurate for higher error probabilities. The possibility of using a better approximation will therefore be investigated in future versions of `topological_codes`, as will further study of how the results of this benchmark relates to the standard one.

IV. CONCLUSIONS

In this paper we have provided an introduction to the repetition code and its implementation in Qiskit. It forms part of the `topological_codes` module of Qiskit-Ignis.

This module has been designed to have a modular form, allowing new codes and decoders to be added with relative ease. It will be expanded to include more codes, such as the 17 qubit variant of the surface code [25, 26]. It will also be expanded to introduce new decoders, such as Bravyi-Haah [27] and methods devised as part of the 'Decodoku' citizen science project [28]. There are also many other possible ways to expand the module. One aim of this paper is to serve as a starting point for anyone interested in making any contributions to this open-source project.

Results were shown from an example run from a 43 qubit code running on IBM's *Rochester* device. This showed what can be done using just a few lines of code. However, much more could be done to fully probe the working of a device using the repetition code. The most obvious would be to use different possible choices for `line`, to fully map out where the errors come from and what effect they have. This paper also serves to serve as a starting point for such endeavours, since anyone with an internet connection can now use the tool to probe the 15 qubits of IBM's publicly accessible *Melbourne* device.

V. ACKNOWLEDGEMENTS

This paper was based on resources developed for the Qiskit tutorials and textbook [29]. The author would therefore like to thank all members of the Qiskit community.

[1] K. Temme, S. Bravyi, and J. M. Gambetta, Phys. Rev. Lett. **119**, 180509 (2017).
[2] S. Endo, S. C. Benjamin, and Y. Li, Phys. Rev. X **8**, 031027 (2018).
[3] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, "Software Mitigation of Crosstalk on Noisy Intermediate-Scale Quantum Computers," (2020), arXiv:2001.02826.
[4] D. A. Lidar and T. A. Brun, eds., *Quantum Error Correction* (Cambridge University Press, Cambride, UK, 2013).
[5] J. Kelly *et al.*, Nature **519**, 66 (2014).
[6] D. Ristè, S. Poletto, M. Z. Huang, A. Bruno, V. Vesterinen, O. P. Saira, and L. DiCarlo, Nature Communications **6**, 6983 (2015).
[7] A. D. Corcoles *et al.*, Nature Communications **6** (2015), doi:10.1038/ncomms7979.
[8] M. Takita, A. D. Córcoles, E. Magesan, B. Abdo, M. Brink, A. Cross, J. M. Chow, and J. M. Gambetta, Phys. Rev. Lett. **117**, 210505 (2016).
[9] J. R. Wootton, Quantum Science and Technology **2**, 015006 (2017).
[10] M. Takita, A. W. Cross, A. D. Córcoles, J. M. Chow, and J. M. Gambetta, Phys. Rev. Lett. **119**, 180501 (2017).
[11] N. M. Linke, M. Gutierrez, K. A. Landsman, C. Figgatt, S. Debnath, K. R. Brown, and C. Monroe, Science Advances **3** (2017), 10.1126/sciadv.1701074.
[12] C. Vuillot, Quant. Inf. Comp. **18**, 0949 (2018).
[13] J. R. Wootton and D. Loss, Phys. Rev. A **97**, 052313 (2018).
[14] Y. Naveh, E. Kashefi, J. R. Wootton, and K. Bertels, in *Proceedings of the 2018 Design, Automation & Test in Europe (DATE)* (2018).
[15] M. Gong *et al.*, "Experimental verification of five-qubit quantum error correction with superconducting qubits," (2019), arXiv:1912.09410.
[16] C. Kraglund Andersen *et al.*, "Repeated Quantum Error Detection in a Surface Code," (2019), arXiv:1907.04507.
[17] D. Gottesman, Phys. Rev. A **54**, 1862 (1996).
[18] H. Abraham, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, G. Alexandrowics, E. Arbel, A. Asfaw, C. Azaustre, AzizNgoueya, P. Barkoutsos, G. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, L. S. Bishop, S. Bosch, D. Bucher, CZ, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, R. Chen, J. M. Chow, C. Claus, C. Clauss, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, Cryoris, C. Culver, A. D. Córcoles-Gonzales, S. Dague, M. Dartiailh, DavideFrr, A. R. Davila, D. Ding, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, E. Eastman, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, P. M. Fernández, A. H. Ferrera, A. Frisch, A. Fuhrer, M. GEORGE,

I. GOULD, J. Gacon, Gadi, B. G. Gago, J. M. Gambetta, L. Garcia, S. Garion, Gawel-Kus, J. Gomez-Mosquera, S. de la Puente González, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, I. Haide, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, A. Javadi-Abhari, Jessica, K. Johns, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, Knabberjoe, A. Kovyrshin, V. Krishnan, K. Krsulich, G. Kus, R. LaRose, R. Lambert, J. Latone, S. Lawrence, D. Liu, P. Liu, P. B. Z. Mac, Y. Maeng, A. Malyshev, J. Marecek, M. Marques, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, S. Meesala, A. Mezzacapo, R. Midha, Z. Minev, M. D. Mooring, R. Morales, N. Moran, P. Murali, J. Müggenburg, D. Nadlinger, G. Nannicini, P. Nation, Y. Naveh, Nick-Singstock, P. Niroula, H. Norlen, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, S. Oud, D. Padilha, H. Paik, S. Perriello, A. Phan, M. Pistoia, A. Pozas-iKerstjens, V. Prutyanov, D. Puzzuoli, J. Pérez, Quintiii, R. Raymond, R. M.-C. Redondo, M. Reuter, D. M. Rodríguez, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, N. Sathaye, B. Schmitt, C. Schnabel, T. L. Scholten, E. Schoute, I. F. Sertage, N. Shammah, Y. Shi, A. Silva, Y. Siraichi, I. Sitdikov, S. Sivarajah, J. A. Smolin, M. Soeken, D. Steenken, M. Stypulkoski, H. Takahashi, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, D. Vogt-Lee, C. Vuillot, J. Weaver, R. Wieczorek, J. A. Wildstrom, R. Wille, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, D. Yeralin, J. Yu, C. Zachow, L. Zdanski, Zoufalc, anedumla, azulehner, bcamorrison, brandhsn, chlorophyll zz, dime10, drholmie, elfrocampeador, faisaldebouni, fanizzamarco, gruu, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, mrossinek, neupat, ordmoj, sethmerkel, strickroman, sumitpuri, tigerjack, toural, willhbang, yang.luh, and yotamvakninibm, "Qiskit: An open-source framework for quantum computing," (2019).

[19] A. Y. Kitaev, Ann. Phys. **303**, 2 (2003).

[20] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, J. Math. Phys. **43**, 4452 (2002).

[21] H. Bombin and M. A. Martin-Delgado, Phys. Rev. Lett. **97**, 180501 (2006).

[22] N. Delfosse, Phys. Rev. A **89**, 012317 (2014).

[23] J. R. Wootton, Journal of Physics A: Mathematical and Theoretical **48**, 215302 (2015).

[24] A. G. Fowler, Phys. Rev. Lett. **109**, 180502 (2012).

[25] Y. Tomita and K. M. Svore, Phys. Rev. A **90**, 062320 (2014).

[26] J. R. Wootton, A. Peter, J. R. Winkler, and D. Loss, Phys. Rev. A **96**, 032338 (2017).

[27] S. Bravyi and J. Haah, Phys. Rev. Lett. **111**, 200501 (2013).

[28] J. R. Wootton, "Getting the public involved in Quantum Error Correction," (2017), arXiv:1712.09649.

[29] A. Asfaw, L. Bello, Y. Ben-Haim, S. Bravyi, L. Capelluto, A. C. Vazquez, J. Ceroni, J. Gambetta, S. Garion, L. Gil, S. D. L. P. Gonzalez, D. McKay, Z. Minev, P. Nation, A. Phan, A. Rattew, J. Schaefer, J. Shabani, J. Smolin, K. Temme, M. Tod, and J. Wootton., "Learn quantum computation using qiskit," (2020).

Chapter 9

Syndrome-Derived Error Rates as a Benchmark of Quantum Hardware

Abstract

Quantum error correcting codes are designed to pinpoint exactly when and where errors occur in quantum circuits. This feature is the foundation of their primary task: to support fault-tolerant quantum computation. However, this feature could used as the basis of benchmarking: By analyzing the outputs of even small-scale quantum error correction circuits, a detailed picture can be constructed of error processes across a quantum device. Here we perform an example of such an analysis, using the results of small repetition codes to determine the error rate of each qubit while idle during a syndrome measurement. This provides an idea of the errors experienced by the qubits across a device while they are part of the kind of circuit that we expect to be typical in fault-tolerant quantum computers.

Publication Information

This chapter was made available on the arXiv [23], but is intended otherwise intended to be published only in this thesis.

Note that the references of this chapter are self-contained, and not included in the overall references.

INTRODUCTION AND MOTIVATION

The development of quantum computers requires the benchmarking and validation of quantum hardware [1]. When aiming towards the long-term goal of fault-tolerant and scalable quantum computation, it is crucial for this benchmarking to account for quantum error correction: probing to what degree its requirements are met, and whether it does indeed provide the functionality that it is designed for.

It is with this motivation that many experiments have been performed as proofs-of-principle for the components of quantum error correction circuits as well as demonstrating error suppression in certain restricted cases (see [2–8] for selected recent examples, and references therein). The most recent examples have made the major breakthrough of demonstrating surface codes of distance 3 [9, 10].

In such demonstrations, the metric of success is often defined at a large-scale level. For example, the repetition code implementations of [11, 12] look at the probability of a logical error for increasing code distances. These codes restricted by the fact that only a limited subset of errors can be corrected. However, the code distance for this subset can be made very large even with current quantum hardware. Specifically, for n total qubits used the distance will be $d \approx n/2$. Since the probability of a logical error decays exponentially with d, the number of samples required to estimate this probability must therefore increase exponentially. Analysis of progress using only the calculation of this value is therefore not scalable.

It is for this reason that we will turn our attention to the microscopic scale. Quantum error correction codes constantly output syndrome information, which is explicitly designed to detect errors, and pinpoint exactly where and when single error events occur. In the ideal case, we deduce exactly which kind of Pauli error occurred on a specific qubit at a specific point within the circuit. This information is available on a shot-by-shot basis, allowing for the computation to correct or otherwise avoid the effects of the error. Nevertheless, we can also make use of the information when comparing results from many shots. This can be used to compile information about the likelihood of different kinds of error, giving us probabilities for different error events at specific points within a quantum error correction circuit.

These syndrome-derived error rates have already been well-studied as a means to provide accurate weights within a decoding procedure (such as in [13]). However, relatively little attention has been given to their potential for benchmarking. The exceptions being [12] and the functionality within Qiskit that it documents [14], on which this current work is also built, and the presentation of such error rates in recent papers by Google [3, 15]

The fact that these error rates are tailored to specific points within a circuit, and that they can only be measured within the context of a large multi-qubit circuit, stands in contrast to many other benchmarking techniques. In randomized benchmarking [16], for example, results are tailored to specific qubits and specific gate type. Also, the benchmarked gates are implemented in isolation by default. For performance in the context of more complex circuits, we must assume that the same results still hold, or design more sophisticated benchmarks (such as ones designed to be sensitive to cross-talk [17]).

The advantage of the syndrome-derived error rates is that such assumptions or upgrades are not required. From the context of fault-tolerant quantum computation, the most important error rates are those of qubits that are actively part of an error correction circuit. By using the syndrome from such a circuit to analyze the errors within that very circuit, we obtain the most accurate assessment of error rates within that circuit at the microscopic level. Of course, we can still only assume that the same qubits would experience equivalent error rates if used for a different quantum error correcting code. However, this is a much more reasonable assumption than for error rates calculated for a qubit analyzed in isolation.

Different points within a circuit will be subject to different forms of noise. Each error rate will therefore have contributions from multiple different error processes. In this work we will therefore focus on one particular point within a particular circuit: the idling of the central qubit of a $d = 3$ repetition code during measurement. This will allow us to add delay gates to increase the idle time, and therefore increase the effects of relaxation noise in particular, to see the effects of T_1 and T_2 times within the error rates.

CHARACTERIZING CODE QUBIT IDLING ERRORS

A minimal set of experiments

In a quantum error correcting code, different physical qubits play different roles. Some are the ‘code qubits’ (also known as ‘data qubits’), whose collective properties are used to store and manipulate quantum information. Others are ‘auxiliary qubits’, used to mediate multi-qubit measurements. Circuits for quantum error correction consist of a constantly repeating schedule of these multi-qubit measurements, which typically involve a set of two-qubit gates between code and auxiliary qubits, followed by measurements of the auxiliaries. This measurement is typically the longest duration gate within circuit. In designing a circuit, it can be advantageous to find a way to keep the code qubits active while some of the auxiliary qubits are being measured. This is to avoid the errors that would occur due to the qubit being idle. Nevertheless, the idle time for a code qubit during measurement of its auxiliaries is a common feature of quantum error correcting circuits. As such, whether we are seeking to avoid them or accepting their presence, it is important to determine the potency of errors on code qubits during measurement of auxiliaries.

To do this we use repetition codes implemented along a line, in the same manner as in [12]. A distance d repetition code of this form has d code qubits and $d-1$ auxiliary qubits. There are therefore $2d-1$ qubits in total, alternating along a line between code and auxiliary qubits. Of the code qubits, the $d-2$ in the bulk are part of two distinct syndrome measurements. The auxiliary qubits for these are the two neighbours of each code qubit. The two code qubits on the ends of the line are part of only one syndrome measurement and therefore have less detailed information extracted about their errors.

Our approach is to select a single qubit on a device whose errors we wish to examine. Since we wish to examine the errors for a code qubit while idle, the selected qubit must obviously be a code qubit of the code we implement. To get sufficiently detailed information, it will also need to be a code qubit within the bulk of the code (not one involved in less syndrome measurements due to being part of the edge). The minimal repetition code to have at least one bulk code qubit is that with $d = 3$, where it is the central qubit in a line of five.

When implementing the code, we will run a predetermined number of syndrome measurement rounds. During the first round, errors detected on code qubits could come from a variety of sources, most notably incorrect initialization of the qubit, and imperfections in the two-qubit gates used for the syndrome measurements. The code qubits will subsequently experience their first errors due to delay while the auxiliary qubits are being measured. These will be detected by the second and all subsequent rounds of syndrome measurement, alongside the aforementioned imperfections in the two-qubit gates. During final readout, the code qubits are also measured and the results are used to infer a final effective syndrome measurement round. Results from this round will additionally detect measurement errors for the code qubits. In order to concentrate on errors during the idle time, we therefore require $T \geq 2$ syndrome measurement rounds. Results from the first and (effective) final round will not be used.

Repetition codes only detect a single type of error, whose nature depends on the definition of the code. For bit flip codes, a logical `0` is encoded on the code qubits as $|0\rangle^{\otimes d}$ and 1 is $|1\rangle^{\otimes d}$. These will detect bit flips during the idle time, i.e. errors composed of the Paulis X and Y. For phase flip codes the codewords are $|+\rangle^{\otimes d}$ and $|-\rangle^{\otimes d}$, and it is the phase flips Y and Z that are detected. A fully general single qubit error could be composed of X, Y and Z, but no single repetition code can detect a full set of errors in a single instance. However, by performing experiments with both bit flip and phase flip codes we can find a full set of error probabilities.

Given the above considerations, we now have a minimal set of experiments to probe the idling errors of a single code qubit during measurement of its auxiliaries. We will use $d = 3$ repetition codes for $T = 2$ syndrome measurement rounds, with the selected qubit in the centre. We extract the probability for an error on the central qubit during the first syndrome measurement round. This is done for both the bit flip encoding and phase flip encoding to get probabilities for both bit and phase flips.

Constructing the circuits

For any given device, we must first determine which qubits can be analyzed in the manner described above. This will depend on the coupling graph, in which the qubits are vertices and edges correspond to pairs of qubits for which the two qubit `cx` gate can be applied. A qubit can only serve as the central qubit of a $d = 3$ code if there exists a line of five qubits on this graph, centered on the chosen qubit. Examples of such lines are shown in Fig. 1.

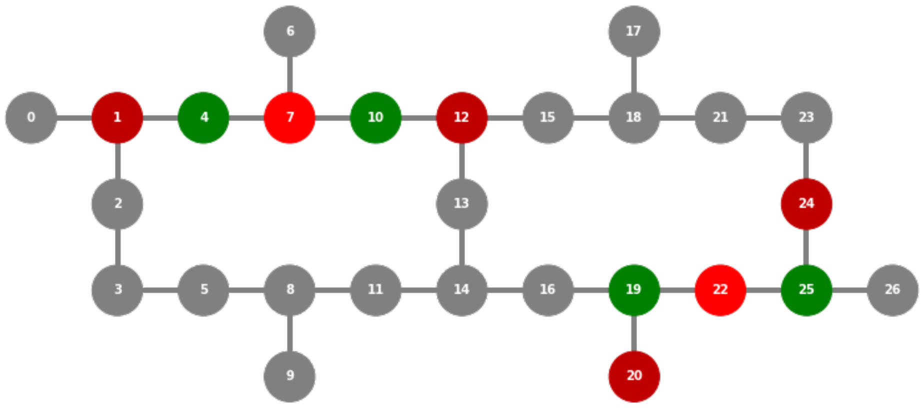


FIG. 1. Two lines of five qubits are shown on the coupling map of IBM Quantum *Falcon* devices. For each, code qubits are shown in red and auxiliary qubits in green. One of these lines is the unique one required to benchmark qubit 7 on this device. The other is one of several options for qubit 22.

For some qubits, there is only one possible line of five qubits for which it sits at the center. For others, there are multiple options. For example, Fig. 1 shows the line $20-19-22-25-24$, which can be used to benchmark qubit 22. However, this could equally begin at 16 rather than 20, and end at 26 rather than 24.

In such cases where there is more than one option, more information is required. Our experiments are performed on IBM Quantum hardware via Qiskit. As such, there is an array of benchmarking data available regarding each device, which is updated during regular calibrations. In particular we will make use of the measured T_1 and T_2 times for each qubit, and the error probability of each `cx` gate.

The `cx` gate errors are what we use to determine the optimal line. Specifically, two quantities are determined for the `cx` gates within each line:

- The maximum error rate for `cx` gates that involve the central qubit;
- The maximum error rate for `cx` gates within the line as a whole.

The chosen line is that which first minimizes the former, and then minimizes the latter. All lines involving a `cx` with an error rate of over 0.5 are discarded, since these typically signal a poorly calibrated gate.

The 'leaf qubits' around the edge, which couple only to a single neighbour, cannot serve as the central qubit in any such code. We therefore cannot study these in the same manner. Also, if the quality of some `cx` gates are sufficiently low, we may also be blocked from benchmarking further qubits.

An example of a circuit implemented along the line is shown in Fig. 2. Such circuits contain multiple points at which qubits are idle, such as while waiting for a neighbour to complete an entangling gate or measurement. To increase performance, we can insert dynamical decoupling sequences during all idle times. When doing so, we insert a specific CPMG sequence [18, 19]: a pair of `x` gates.

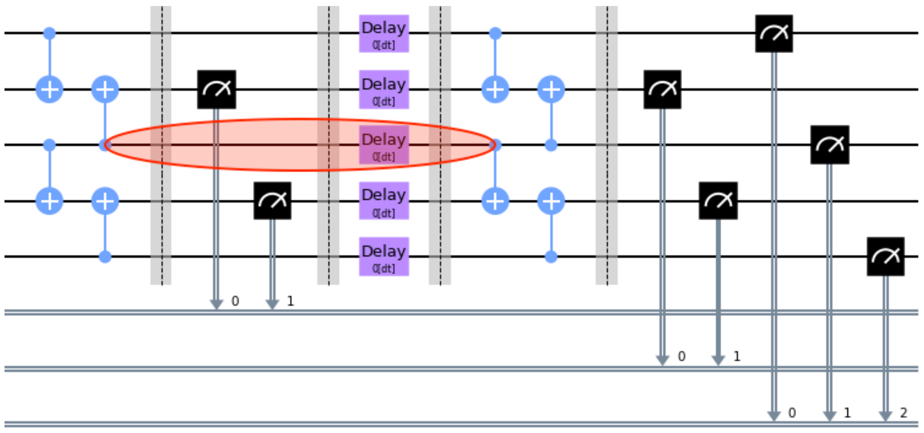


FIG. 2. The circuit used for codes with the bit flip encoding and a logical `0`. The position of the `delay` gates show where delays can be added to increase the effects of relaxation noise and dephasing. The errors measured by our experiments are those corresponding to the highlighted part of the circuit.

COMPARING RESULTS WITH WITH T_1 AND T_2

Before blindly accepting the error probabilities calculated from syndrome measurement results, it is important to determine if they produce results in line with expectations. For this we can greatly increase the amount of idling error by increasing the idling time. This is done by simply introducing a delay after each set of measurements. For a sufficiently long delay time, the errors due to idling will be greatly dominant over those from imperfections in the two qubit gates. It should therefore be possible to estimate the error probabilities from standard benchmarking data. Specifically: the T_1 and T_2 times.

A useful feature of the repetition code is that the code qubits are in a product state. It is therefore simple to determine the expected errors from T_1 and T_2. Specifically, for a total idle time of t for the code qubits, we can derive the following properties of $P_{0\to1}$ (the probability that a $|0\rangle$ will flip to $|1\rangle$), $P_{1\to0}$ (vice-versa), and $P_{+\leftrightarrow-}$ (the probability of flipping between $|+\rangle$ and $|-\rangle$).

$$\begin{aligned} P_{0\to 1}(t) + P_{1\to 0}(t) &= 1 - e^{-t/T_1}, \\ \frac{P_{0\to 1}(t)}{P_{1\to 0}(t)} &= \frac{1-p_0}{p_0}, \\ P_{+\leftrightarrow -}(t) &= \frac{1-e^{-t/T_2}}{2}. \end{aligned} \tag{1}$$

Here p_0 is the probability that the qubit will be found in the $|0\rangle$ state when the it reaches equilibrium. A well made qubit typically decays to a good approximation of the $|0\rangle$ state, and so $p_0 \approx 1$. This implies $P_{0\to 1} \approx 0$ and $P_{1\to 0} \approx 1 - e^{-t/T_1}$.

Before applying these probabilities to our states, we must account for the possibility of dynamical decoupling. Though methods for this vary, typically they will cause a qubit during a delay to spend half its time in a flipped state. For the bit flip encoding, this means that $t/2$ will be spend as both $|0\rangle$ and $|1\rangle$ regardless of the logical value. The relevant probability of a bit flip for either logical value during a delay is then,

$$P_{0\leftrightarrow 1}(t) \approx P_{0\to 1}(t/2) + P_{1\to 0}(t/2) = 1 - e^{-(t/2)/T_1}.$$

For phase flips, the use of some form of dynamical decoupling is already assumed when using the T_2 time. The probability $P_{+\leftrightarrow -}(t)$ as stated above therefore applies in this case. For a circuit without dynamical decoupling, dephasing would instead be characterized by a shorter timescale T_2^*. This accounts for additional effects that are echoed out by the dynamical decoupling.

When adding a long delay, we will use $t = T/8$ where T is the relevant timescale for the code being run. Specifically, $T = T_1$ For the bit flip encoding and $T = T_2$ for the phase flip encoding. With this delay time, we find,

$$P_{1\to 0} \approx 11.8\%, \quad P_{0\leftrightarrow 1} \approx 6.1\%, \quad P_{+\leftrightarrow -} \approx 5.9\%. \tag{2}$$

Without dynamical decoupling, the value of $P_{+\leftrightarrow -}$ cannot be predicted from T_2 alone. However, we can expect that it will be comparatively high.

Note that these numbers are for the probabilities of a flip during the delay alone. There is also the probability of a flip during the circuit before and after the delay. Indeed this latter probability is what we otherwise wish to measure. However, we can assume that it will be relatively small in comparison. We therefore do not expect the measured flip probabilities to exactly align with the values above. Instead these values provide a guide to what kind of results we can regard as reasonable.

Codes were run on `ibm_hanoi` to compare with these values. Both bit and phase flip encodings were run, both with and without dynamical decoupling. This was done for all the qubits on the device that could be placed at the center of a five qubit line. The median values found for the above probabilities were,

$$\hat{P}_{1\to 0} = 16.0\%, \quad \hat{P}_{0\leftrightarrow 1} = 9.6\%, \quad \hat{P}_{+\leftrightarrow -} = 25.0\%. \tag{3}$$

The first of these was determined from a run without dynamical decoupling, and the latter two with dynamical decoupling.

The values for $\hat{P}_{1\to 0} = 16.0\%$ and $\hat{P}_{1\leftrightarrow 0} = 9.6\%$ are obviously larger than the guide values, but are nevertheless well within a factor of two. We therefore regard them as being within expectations. However, the value of $\hat{P}_{+\leftrightarrow -}$ is significantly worse than the guide value of 5.9%. The full set of probabilities for each qubit in this case is shown in Fig. 3, which shows that several qubits have even more extreme examples.

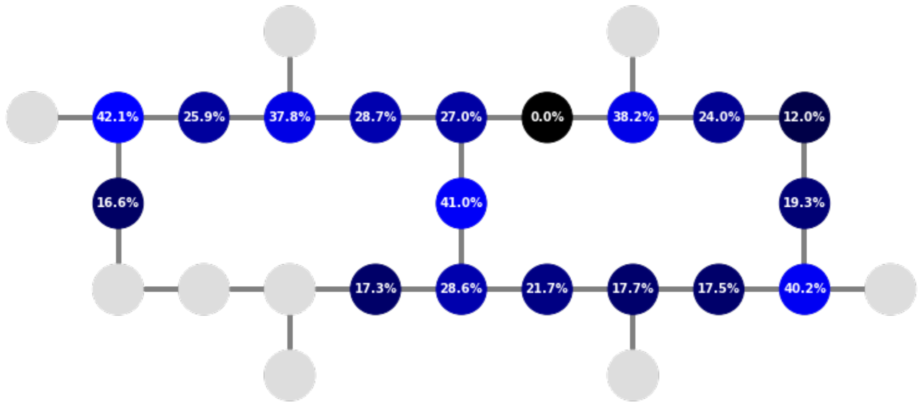


FIG. 3. The error probabilities $P_{+\leftrightarrow -}$ for an additional delay time $t_{delay} = T/8$ and dynamical decoupling applied all qubits. The brightness of the qubits is proportional to the values.

One possible mechanism that could cause additional dephasing is that described in [20], where it was shown that decay events can cause phase noise on neighbouring qubits. In our case the neighbours of all code qubits are auxiliary qubits. The circuit only flips these to the $|1\rangle$ state when an error is detected. They should therefore mostly be in the $|0\rangle$ state, which would make them immune from relaxation errors. Since they should always be in either the $|0\rangle$ state or $|1\rangle$ state during delays, they are also immune to dephasing. The application of dynamical decoupling to these qubits will therefore have only negative effects. The biggest being the increased probability of relaxation errors when holding the qubit in the $|1\rangle$ state for long times, and therefore increased phase noise on the neighbouring code qubits. Our use of dynamical decoupling on all qubits may therefore have in fact caused this much increased dephasing.

Motivated by this, the experiments were run with dynamical decoupling only on code qubits. In this case we found $\hat{P}_{+\leftrightarrow -} = 7.95\%$, which is in line with expectations. The full set of probabilities for each qubit in this case is shown in Fig. 4.

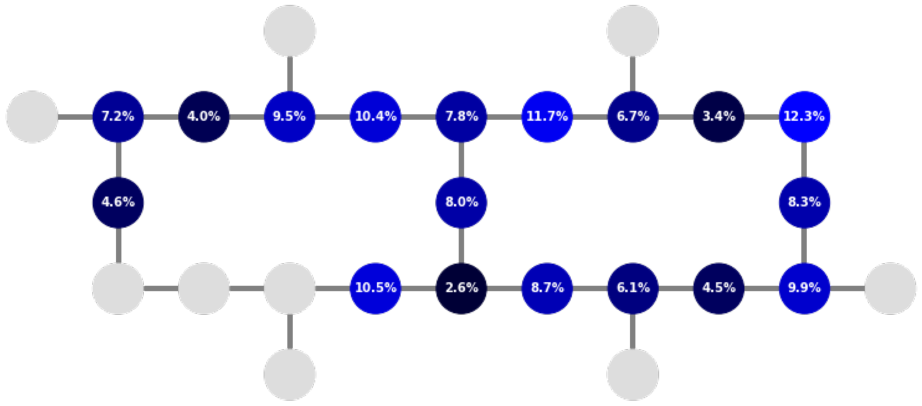


FIG. 4. The error probabilities $P_{+\leftrightarrow-}$ for an additional delay time $t_{delay} = T/8$ and dynamical decoupling applied only to code qubits.

Though implementing dynamical decoupling on the auxiliary qubits was a mistake, it was one that proved instructive. The complex noise triggered by a seemingly innocuous over-application of dynamical decoupling teaches an important lesson: we cannot assume that errors will always be described by a naive combination of T_1, T_2 and probabilities from randomized benchmarking. Measuring syndrome-derived error rates is therefore a means by which we can find and highlight these mysteries.

RESULTS FROM BENCHMARKING DEVICES

In the case of dynamical decoupling applied to the code qubits alone, we have now found that the syndrome-derived error rates are largely within expectations. Given that we can expect to find sensible results, we can now proceed to our main goal: determine the idling errors of a single code qubit during measurement of its auxiliaries. For this we run the same circuits as above, but without the additional delay time. The results are shown in Fig. 5. The expected contributions from the T_1, T_2 and `cx` errors, obtained from each device via Qiskit, are shown in Fig. 6.

With these results, we can summarize the status of the worst and best qubits on each device (from the perspective of the syndrome-derived error rates). The numbering used for the qubits here is that seen in Fig. 1.

- `ibm_hanoi`: The worst qubit is 23, which has relatively long T_1 and T_2, but one of the worst `cx` gates; One of the best qubits is 4, which has a relatively long T_1 and typical `cx` gates for the device.

- `ibm_auckland`: The worst qubit is 15, which has a short T_1, a typical T_2 and two of the worst `cx` gates. The best qubit is 7, which has a relatively typical T_1, a long T_2, and good `cx` gates.

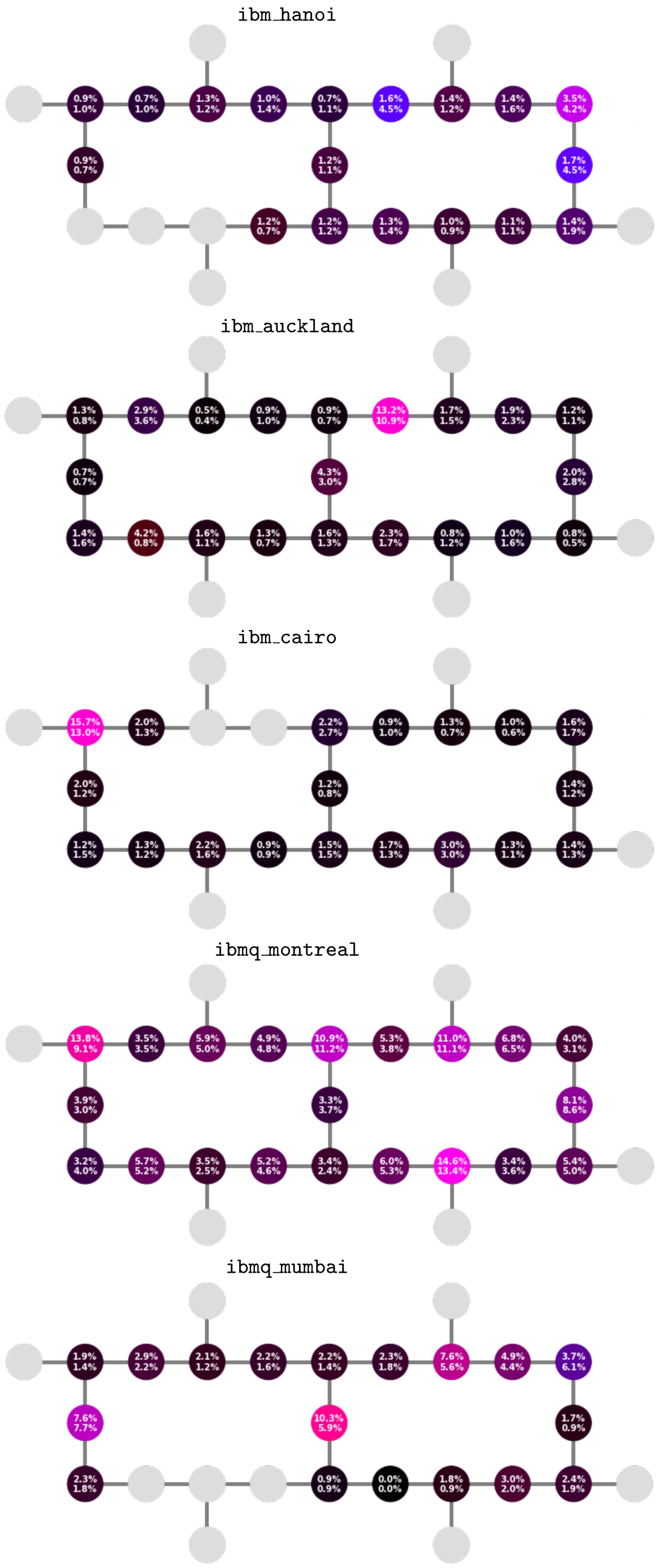


FIG. 5. This figure shows the probabilities $P_{0\leftrightarrow 1}$ and $P_{+\leftrightarrow -}$ measured on the qubits of a range of devices. The values are represented both in text ($P_{0\leftrightarrow 1}$ listed first and $P_{+\leftrightarrow -}$ listed second), and in the colour of the qubit. For the latter, $P_{0\leftrightarrow 1}$ and $P_{+\leftrightarrow -}$ correspond to the brightness of the red and blue channels, respectively, with the maximum probability on the device used for the maximum brightness.

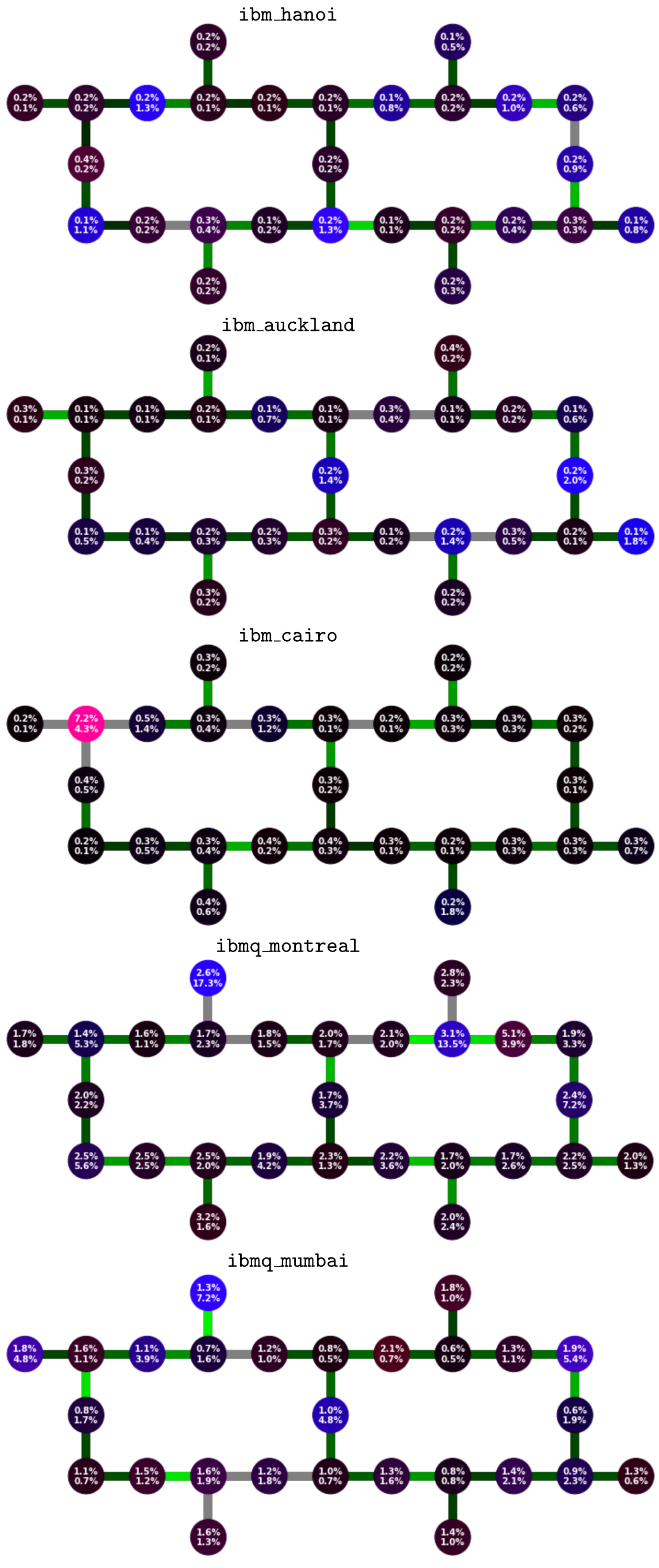


FIG. 6. The figure shows values for the probabilities $P_{0\leftrightarrow 1}$ and $P_{+\leftrightarrow -}$, calculated according to the T_1 and T_2 times of the qubits and the readout time on the device. The values are represented as in Fig. 5. The error rate for `cx` gates is represented by the colour of the corresponding link, with bright green corresponding to a 2% error rate in all cases. Any `cx` with a higher error rate is shown in grey.

- `ibm_cairo`: The worst qubit is 1, which has short T_1 and T_2 and two of the worst `cx` gates. One of the best qubits is 15, with relatively long T_1 and T_2, but also has one of the worst `cx` gates.
- `ibmq_montreal`: The worst qubits are 1 and 19, despite having relatively typical T_1, T_2 and `cx` gates. The best qubit is 14, which has good T_1, T_2 and `cx` gates.
- `ibmq_mumbai`: The worst qubit is 13, which has among the worst T_1 ad T_2 times. The best is 16 which has relatively typical T_1 and T_2, and relatively good `cx` gates.

Overall the results suggest that qubits with long T_1 and T_2 and low errors for `cx` gates will do well, and those for which either or both are very bad will do poorly. However, exceptions do occur, with the results from `ibmq_montreal` in particular showing very poor behaviour from qubits whose T_1, T_2 appear typical for the device. The results therefore suggest an optimistic but cautionary outlook: the T_1, T_2 will give a good guide to the performance of a qubit within a quantum error correcting code, but direct measurement of the syndrome-derived error rates is required to ensure all is as it should be.

CONCLUSIONS

The implementation of small quantum error correcting codes is already well underway. Those experiments bring a wealth of syndrome information, which could provide interesting insights into the qubits of the devices. If the reader takes anything away from this paper, it should be that calculating and presenting these syndrome derived error rates should become standard practice. This would allow devices to be compared in a manner that lies between the extremes of randomized benchmarking and quantum volume [21], and which directly benchmarks performance towards large-scale quantum error correction.

In this work we investigated a minimal implementation of such a study, assessing qubits across a device with tailored $d = 3$, $T = 2$ repetition codes. Continuing to calculate these probabilities will provide users of quantum devices with a novel set of benchmarks for each device, giving a sense of how qubits will fare within complex circuits with mid-circuit measurement.

However, even these calculations are small and simple in comparison with the large-scale error correction of fault-tolerance. This work is therefore just the first step towards the continual assessment of quantum hardware through syndrome-derived error rates.

The source code and data for all results in this paper is available at [22].

ACKNOWLEDGEMENTS

The author would like to thank Daniel Miller for useful conversations.

Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-21-1-0002. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Part of the software used was created as part of a project supported by the NCCR SPIN, a National Centre of Competence in Research, funded by the Swiss National Science Foundation (grant number 51NF40-180604).

[1] J. Eisert, D. Hangleiter, N. Walk, I. Roth, D. Markham, R. Parekh, U. Chabaud, and E. Kashefi, Nature Reviews Physics **2**, 382 (2020).

[2] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, N. Lacroix, G. J. Norris, M. Gabureac, C. Eichler, and A. Wallraff, Nature Physics **16**, 875 (2020).

[3] Z. Chen, K. J. Satzinger, J. Atalaya, A. N. Korotkov, A. Dunsworth, D. Sank, C. Quintana, M. McEwen, R. Barends, P. V. Klimov, S. Hong, C. Jones, A. Petukhov, D. Kafri, S. Demura, B. Burkett, C. Gidney, A. G. Fowler, A. Paler, H. Putterman, I. Aleiner, F. Arute, K. Arya, R. Babbush, J. C. Bardin, A. Bengtsson, A. Bourassa, M. Broughton, B. B. Buckley, D. A. Buell, N. Bushnell, B. Chiaro, R. Collins, W. Courtney, A. R. Derk, D. Eppens, C. Erickson, E. Farhi, B. Foxen, M. Giustina, A. Greene, J. A. Gross, M. P. Harrigan, S. D. Harrington, J. Hilton, A. Ho, T. Huang, W. J. Huggins, L. B. Ioffe, S. V. Isakov, E. Jeffrey, Z. Jiang, K. Kechedzhi, S. Kim, A. Kitaev, F. Kostritsa, D. Landhuis, P. Laptev, E. Lucero, O. Martin, J. R. McClean, T. McCourt, X. Mi, K. C. Miao, M. Mohseni, S. Montazeri, W. Mruczkiewicz, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Newman, M. Y. Niu, T. E. O'Brien, A. Opremcak, E. Ostby, B. Pató, N. Redd, P. Roushan, N. C. Rubin, V. Shvarts, D. Strain, M. Szalay, M. D. Trevithick, B. Villalonga, T. White, Z. J. Yao, P. Yeh, J. Yoo, A. Zalcman, H. Neven, S. Boixo, V. Smelyanskiy, Y. Chen, A. Megrant, and J. Kelly, "Exponential suppression of bit or phase errors with cyclic error correction," (2021).

[4] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz, "Realization of real-time fault-tolerant quantum error correction," (2021), arXiv:2107.07505 [quant-ph].

[5] J. Hilder, D. Pijn, O. Onishchenko, A. Stahl, M. Orth, B. Lekitsch, A. Rodriguez-Blanco, M. Müller, F. Schmidt-Kaler, and U. Poschinger, "Fault-tolerant parity readout on a shuttling-based trapped-ion quantum computer," (2021), arXiv:2107.06368 [quant-ph].

[6] M. Gong, X. Yuan, S. Wang, Y. Wu, Y. Zhao, C. Zha, S. Li, Z. Zhang, Q. Zhao, Y. Liu, F. Liang, J. Lin, Y. Xu, H. Deng, H. Rong, H. Lu, S. C. Benjamin, C.-Z. Peng, X. Ma, Y.-A. Chen, X. Zhu, and J.-W. Pan, National Science Review (2021), 10.1093/nsr/nwab011.

[7] A. Erhard, H. Poulsen Nautrup, M. Meth, L. Postler, R. Stricker, M. Stadler, V. Negnevitsky, M. Ringbauer, P. Schindler, H. J. Briegel, R. Blatt, N. Friis, and T. Monz, Nature **589**, 220 (2021).

[8] E. H. Chen, T. J. Yoder, Y. Kim, N. Sundaresan, S. Srinivasan, M. Li, A. D.

Córcoles, A. W. Cross, and M. Takita, Phys. Rev. Lett. **128**, 110504 (2022).

[9] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, G. J. Norris, C. K. Andersen, M. Müller, A. Blais, C. Eichler, and A. Wallraff, Nature **605**, 669 (2022).

[10] Y. Zhao, Y. Ye, H.-L. Huang, Y. Zhang, D. Wu, H. Guan, Q. Zhu, Z. Wei, T. He, S. Cao, F. Chen, T.-H. Chung, H. Deng, D. Fan, M. Gong, C. Guo, S. Guo, L. Han, N. Li, S. Li, Y. Li, F. Liang, J. Lin, H. Qian, H. Rong, H. Su, L. Sun, S. Wang, Y. Wu, Y. Xu, C. Ying, J. Yu, C. Zha, K. Zhang, Y.-H. Huo, C.-Y. Lu, C.-Z. Peng, X. Zhu, and J.-W. Pan, "Realization of an error-correcting surface code with superconducting qubits," (2021).

[11] J. R. Wootton and D. Loss, Phys. Rev. A **97**, 052313 (2018).

[12] J. R. Wootton, Quantum Science and Technology **5**, 044004 (2020).

[13] S. T. Spitz, B. Tarasinski, C. W. J. Beenakker, and T. E. O'Brien, Advanced Quantum Technologies **1**, 1800012 (2018).

[14] Qiskit Contributors, "Qiskit: An open-source framework for quantum computing," (2021).

[15] Google Quantum AI, "Suppressing quantum errors by scaling a surface code logical qubit," (2022).

[16] E. Magesan, J. M. Gambetta, and J. Emerson, Phys. Rev. Lett. **106**, 180504 (2011).

[17] D. C. McKay, A. W. Cross, C. J. Wood, and J. M. Gambetta, "Correlated randomized benchmarking," (2020).

[18] H. Y. Carr and E. M. Purcell, Phys. Rev. **94**, 630 (1954).

[19] S. Meiboom and D. Gill, Review of Scientific Instruments **29**, 688 (1958), https://doi.org/10.1063/1.1716296.

[20] P. Jurcevic and L. C. G. Govia, "Effective qubit dephasing induced by spectator-qubit relaxation," (2022).

[21] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, Phys. Rev. A **100**, 032328 (2019).

[22] J. R. Wootton, `https://github.com/quantumjim/data4papers`.

Part III

Beyond Quantum Error Correction

The previous two parts of this thesis covered work that was based on quantum error correction. However, the advent of cloud quantum computing has the potential to change how researchers work in all aspects of quantum computation. This section explores work done in other fields, namely education, outreach and the novel field of quantum procedural generation.

Each of the following chapters begins with a list of relevant papers by the author, along with a reproduction of their abstracts. The contributions of these papers, as well as their relevance to the theme of this thesis, are then summarized and discussed.

Chapter 10

Education

10.1 Abstracts

[21] Teaching quantum computing with an interactive textbook

James R. Wootton, Francis Harkins, Nicholas T. Bronn, Almudena Carrera Vazquez, Anna Phan, Abraham T. Asfaw

Quantum computing is a technology that promises to offer significant advantages during the coming decades. Though the technology is still in a prototype stage, the last few years have seen many of these prototype devices become accessible to the public. This has been accompanied by the open-source development of the software required to use and test quantum hardware in increasingly sophisticated ways. Such tools provide new education opportunities, not just for quantum computing specifically, but also more broadly for quantum information science and even quantum physics as a whole. In this paper we present a case study of one education resource which aims to take advantage of the opportunities: the open-source online textbook 'Learn Quantum Computation using Qiskit'. An overview of the topics covered is given, as well as an explanation of the approach taken for each.

[24] Quantum Games and Interactive Tools for Quantum Technologies Outreach and Education

Zeki C. Seskir, Piotr Migdał, Carrie Weidner, Aditya Anupam, Nicky Case, Noah Davis, Chiara Decaroli, İlke Ercan, Caterina Foti, Paweł Gora, Klementyna Jankiewicz, Brian R. La Cour, Jorge Yago Malo, Sabrina Maniscalco, Azad Naeemi, Laurentiu Nita, Nassim Parvin, Fabio Scafirimuto, Jacob F. Sherson, Elif Surer, James Wootton, Lia Yeh, Olga Zabello, Marilú Chiofalo

In this article, we provide an extensive overview of a wide range of quantum games and interactive tools that have been employed by the community in recent years. The paper presents selected tools, as described by their developers. The list includes Hello Quantum, Hello Qiskit, Particle in a Box, Psi and Delta, QPlayLearn, Virtual Lab by Quantum Flytrap, Quantum Odyssey, ScienceAtHome, and The Virtual Quantum Optics Laboratory. Additionally, we present events for quantum game development: hackathons, game jams, and semester projects. Furthermore, we discuss the Quantum Technologies Education for Everyone (QUTE4E) pilot project, which illustrates an effective integration of these interactive tools with quantum outreach and education activities. Finally, we aim at providing guidelines for incorporating quantum games and interactive tools in pedagogic materials to make quantum technologies more accessible for a wider population.

10.2 Qiskit Textbook

The author has given lectures at the University of Basel since 2012. At the height of this activity, between 2013 and 2017, two courses were given per year: one primarily on quantum information theory, and the other on quantum computation.

The author was then tasked with writing an online textbook after moving to IBM Research in 2018. This was to be designed such that it could be used to supplement university courses on quantum computation. This would be a 'Qiskit textbook', which would allow the student to get hands-on with the topic by using the Qiskit SDK [19] for quantum circuits.

The initial chapters of this textbook, written between late 2018 and early 2019, were based on the author's course on quantum computation. The version of the course delivered in 2019 was then based off of the textbook. This allowed the contents of the textbook to be refined, and ensure with experience that it can indeed be used as the basis of a course.

The textbook was written to provide a thorough grounding in the mathematics behind quantum gates and algorithms, but to also give hands-on experience of creating and running quantum circuits using IBM Quantum's *Qiskit* software development kit. This was not possible in 2012, when the first course was designed. At that time, it was very much a theory course rather than a practical one. This evolved after cloud quantum computers became available in 2016. Even as early as 2017, exercises were given based on using these devices, mitigating for their noise, and writing circuits to run on simulators. This approach to the course is what made it a perfect basis for the Qiskit textbook, and allowed it to naturally transition to be based on the Qiskit textbook in turn. A full account of the Qiskit textbook and the thinking behind it is given in [21].

10.3 Hello Quantum and Hello Qiskit

A collaboration with IBM Research beginning in 2017 similarly led to a synergy with with the author's teaching. The aim of the collaboration was to create a game which would introduce people to some aspect of quantum computation. The final product was a puzzle game, based on a visualization of two qubit states and gates.

The main version of this, known as 'Hello Quantum' and distributed as an iOS and Android app, was presented as an informal game with optional reading material available after playing. However, throughout development there was always a more pedagogical version, in which some aspect of quantum gates was explained through each puzzle. This began as a simple command line prototype, and evolved into the Jupyter notebook implementation that is currently integrated into the Qiskit textbook. This version, known as 'Hello Qiskit' has been used as part of the author's courses since 2017, as an exercise that can build up intuition before formally teaching the mathematics of qubits. Some statistics on the usage of these can be found in [24].

These pedagogical tools were designed to fulfill a need created by cloud quantum computers. A new audience were now able to interact with quantum computers, who had no idea how one Hadamard gate could make things random while who removed the randomness, or how surrounding a controlled-NOT with Hadamards could turn it around. Many of such counter-intuitive effects can be found in the simplest quantum circuits: those with Clifford gates and at most two qubits. These are, however, the simplest circuits to simulate, and also to visualize. This is the core of 'Hello Quantum' and 'Hello Qiskit'. They provide a simple introduction, mostly limited to Clifford gates, via an intuitive visualization. By doing so, they allow newcomers to quantum computing to understand what they are doing, at least during their first attempts at creating and running the simplest of circuits.

Chapter 11

Outreach

11.1 Abstracts

[25] Getting the public involved in Quantum Error Correction
James R. Wootton

The Decodoku project seeks to let users get hands-on with cutting-edge quantum research through a set of simple puzzle games. The design of these games is explicitly based on the problem of decoding qudit variants of surface codes. This problem is presented such that it can be tackled by players with no prior knowledge of quantum information theory, or any other high-level physics or mathematics. Methods devised by the players to solve the puzzles can then directly be incorporated into decoding algorithms for quantum computation. In this paper we give a brief overview of the novel decoding methods devised by players, and provide short postmortem for Decodoku v1.0-v4.1.

[26] Benchmarking of quantum processors with random circuits
James R. Wootton

Quantum processors with sizes in the 10-100 qubit range are now increasingly common. However, with increased size comes increased complexity for benchmarking. The effectiveness of a given device may vary greatly between different tasks, and will not always be easy to predict from single and two qubit gate fidelities. For this reason, it is important to assess processor quality for a range of important tasks. In this work we propose and implement tests based on random quantum circuits. These are used to evaluate multiple different superconducting qubit devices, with sizes from 5 to 19 qubits, from two hardware manufacturers: IBM Research and Rigetti. The data is analyzed to give a quantitive description of how the devices perform. We also describe how it can be used for a qualititive description accessible to the layperson, by being played as a game.

[27] The History of Games for (Quantum) Computers

James R. Wootton

Computer games are not just one application of computers, they are a multitude. A wide variety of computational tasks are combined, all running as fast as possible, to deliver the best possible experience to the player. It is reasonable to expect that, somewhere in this maelstrom of computation, we can find something that quantum computers will excel at. If the quest to combine games and quantum computers was a game itself, it would be one of open-world exploration. Though there may be no well-defined, concrete goals to achieve, there is something that will guide us: this game is a sequel. By looking at how we combined games with computers the first time, we can get some idea of what we can expect in this quantum successor.

[28] Defining Quantum Games

Laura Piispanen, Marcel Pfaffhauser, Annakaisa Kultima, James R. Wootton

In this article, we explore the concept of quantum games and define quantum games as any type of playable games that are related to or reference quantum physics through any of three proposed aspects. The rise of the quantum computers has made it possible to think about a new wave of computer games, namely quantum computer games, games on quantum computers. But at the same time, there are also various games that are exploring quantum mechanics and related topics through digital, analogue and hybrid means. In this article we go through the emerging body of quantum games, the history of quantum games and the different ways a game may be considered a quantum game. For this we propose three dimensions for analysing and defining the phenomenon of quantum games: the perceivable dimension of quantum games, the dimension of quantum technologies and the dimension of scientific purposes.

11.2 Decodoku

Quantum error correction requires us to solve puzzles: for a given syndrome, what errors caused them and how to we correct for their effects. These puzzles are not themselves quantum in nature, but are instead based on simple principles. The decoding of surface codes, for example, is primarily a matter of pairing up nodes on a decoding graph. With a good visual representation, the problem of decoding can therefore be easily and effectively conveyed to a broad audience.

It was this insight on which the 'Decodoku' project was based [25]. This was conducted primarily in 2016, funded by a director's reserve grant from the NCCR QSIT. The core of the project was a set of apps for both iOS and Android. These presented decoding problems for topological codes as games, which could then be played by the general public. By playing the game and developing a successful strategy, the players essential devise their own decoding algorithm. This then allows them to directly participate in quantum computation research themselves.

Specifically, the games were based on the decoding of qudit quantum double models. For these it is not possible to apply common and effective decoding methods such as minimum weight perfect matching. Instead, more heuristic methods are often employed. Furthermore, little research was actively underway at the time to provide highly effective algorithms for these cases. These decoding problems were therefore perfectly suited to a citizen science approach: players could develop heuristic methods for decoding that could compare well to those of experts, without the fear of experts overshadowing the efforts of the players during the course of the project.

Of course, the design of an algorithm requires more than just playing a game. The next step is then for the players to reflect upon the strategy that they have created, and be able to explain it. No in-app software was provided to automate this process, or to collect data from players. Instead, supplying information about their algorithm was entirely up to the players, and they could do so in whatever manner they chose. For the handful of players who chose to participate at this level, most chose to do so via an exchange of emails. However, the top scorer was a student at the University of Basel, allowing an in-person meeting. The information from all these exchanges was then consolidated and used to design a 'bot' for an updated version of the game, would would provide hints to players. The principles behind this decoding strategy was also explained in the paper that resulted from the project [25].

One participant also chose to go one step further, reproducing the puzzles programmatically in order to write his own decoder using a genetic algorithm [29]. This also visualized the decoders process of deciding how to proceed with the decoding. By studying this work, the author was able to reproduce similar decoding behaviour using a more standard HDRG strategy. This novel decoder was the most concrete advance that emerged from the project [25].

A further aim of the project was to demystify quantum computation by presenting an aspect of research in an accessible way. In doing so it was hoped that players would realize that the realm of quantum physics is not entirely the arcane and inaccessible world they may have been led to believe, and might consider studying the subject in future.

The Decodoku project was conceived of and mostly implemented before the advent of cloud quantum computers. The motivation for this project was therefore not influenced by the availability of such devices. In fact, it was inspired by a desire to close the distance between cutting-edge research and the public that funded it. The project has therefore been revived as a way of teaching about error correction for Qiskit QEC and presenting experimental results for repetition code experiments [20].

11.3 Quantum games

The first games for conventional computers were not made primarily for fun, but for education and training. Early examples were done to demonstrate technology at trade shows, as part of research into human computer interaction, and as a way of getting hands-on experience with using a new device. In a similar way, games could also prove useful during the development of quantum computers. These would not just be games *about* quantum computation, as with Hello Qiskit and Decodoku, but games that actually require the creation and execution of quantum circuits. The advent of cloud quantum computers provided the perfect opportunity to explore this possibility.

This was the motivation behind what were arguably the first games made for quantum computers: 'Cat-Box-Scissors' and 'Battleships with Partial Not Gates'. These are simple games that use quantum hardware, but were made more to be explained than to be played. As straightforward examples of quantum software, they formed the basis of blog posts which explained some of the basic principles of quantum computation. In summary, they used properties of single qubit gates to implement the central game mechanic. In 'Battleships with Partial Not Gates', for example, a quantum circuit is built up round-by-round through the actions taken by the players. The current circuit is run at the beginning of each round, and the results inform the actions taken by the players. The full game is a quantum-classical hybrid, but the quantum circuit provides a crucial role by manipulating in-game data.

Games using only few qubits can, of course, easily be played on a simulator rather than real hardware. Indeed, many later quantum games made by the author and others were designed with this specifically in mind, since simulation of small circuits is much faster than waiting on results from real devices. However, the peculiarities of real devices can also provide a unique challenge within the game. For example, 'Cat-Box-Scissors', is a variant of 'Rock-Paper-Scissors', for which the only strategy beyond pure randomness is to learn the biases in the randomness of your opponent. Since the device serves as the opponent in 'Cat-Box-Scissors', it provides an opportunity for the player to realize that current qubits suffer from noise that biases towards a particular outcome. This idea was expanded upon in 'Quantum Awesomeness' [26], in which the noise of a device explicitly affected how playable the game was. The intention was to give non-experts an intuitive way to assess the quality of a device in action, without needing to understand a set of benchmarking values.

Another possibility to learn via quantum games is by making them. By implementing a simple game mechanic using single or two qubit gates, someone learning about quantum computing can hope to achieve at least a little practical experience, which may help them as they dive into the technicalities later. This was the idea behind the game jams and workshops hosted by many organizations across the world, such that the 'Quantum Game Jam' series started by the University of Turku in 2014 and which has incorporated cloud quantum computing in recent years. In such events, it is easy for non-experts to find and follow the path of least resistance: simply using the quantum circuit to generate random numbers. To encourage and facilitate higher level usage, a method for single qubit procedural generation was developed [27]. This was simple enough to explain and understand within a short workshop, ran fast enough to be simulable on a microprocessor, and could deterministically produce interesting landscapes that could be used within a game.

Through work by the author and others, there are now many quantum games using many different approaches and made for various purposes. In depth explanations of those made by the author, and the historical context behind such them, is presented in [27]. A categorization current quantum games was given in [28]. A collection of some of the games, including some by the author, are available online at [30].

Chapter 12

Procedural generation

12.1 Abstracts

[31] A quantum procedure for map generation
James R. Wootton

Quantum computation is an emerging technology that promises a wide range of possible use cases. This promise is primarily based on algorithms that are unlikely to be viable over the coming decade. For near-term applications, quantum software needs to be carefully tailored to the hardware available. In this paper, we begin to explore whether near-term quantum computers could provide tools that are useful in the creation and implementation of computer games. The procedural generation of geopolitical maps and their associated history is considered as a motivating example. This is performed by encoding a rudimentary decision making process for the nations within a quantum procedure that is well-suited to near-term devices. Given the novelty of quantum computing within the field of procedural generation, we also provide an introduction to the basic concepts involved.

[32] Procedural generation using quantum computation
James R. Wootton

Quantum computation is an emerging technology that promises to be a powerful tool in many areas. Though some years likely still remain until significant quantum advantage is demonstrated, the development of the technology has led to a range of valuable resources. These include publicly available prototype quantum hardware, advanced simulators for small quantum programs and programming frameworks to test and develop quantum software. In this provocation paper we seek to demonstrate that these resources are sufficient to provide the first useful results in the field of procedural generation. This is done by introducing a proof-of-principle method: a quantum generalization of a blurring process, in which quantum interference is used to provide a unique effect. Through this we hope to show that further developments in the technology are not required

before it becomes useful for procedural generation. Rather, fruitful experimentation with this new technology can begin now.

[33] Investigating the usefulness of Quantum Blur
James R. Wootton and Marcel Pfaffhauser

hough some years remain before quantum computation can outperform conventional computation, it already provides resources that can be used for exploratory purposes in various fields. This includes certain tasks for procedural generation in computer games, music and art. The so-called 'Quantum Blur' method represents the first step on this journey, providing a simple proof-of-principle example of how quantum software can be useful in these areas today. Here we analyse the 'Quantum Blur' method and compare it to conventional blur effects. This investigation was guided by discussions with the most prominent user of the method, to determine which features were found most useful. In particular we determine how these features depend on the quantum phenomena of superposition and entanglement.

[34] Quantum Natural Language Generation on Near-Term Devices
Amin Karamlou, James R. Wootton, Marcel Pfaffhauser

The emergence of noisy medium-scale quantum devices has led to proof-of-concept applications for quantum computing in various domains. Examples include Natural Language Processing (NLP) where sentence classification experiments have been carried out, as well as procedural generation, where tasks such as geopolitical map creation, and image manipulation have been performed. We explore applications at the intersection of these two areas by designing a hybrid quantum-classical algorithm for sentence generation. Our algorithm is based on the well-known simulated annealing technique for combinatorial optimisation. An implementation is provided and used to demonstrate successful sentence generation on both simulated and real quantum hardware. A variant of our algorithm can also be used for music generation. This paper aims to be self-contained, introducing all the necessary background on NLP and quantum computing along the way.

12.2 Procedural Generation

The field of procedural generation is concerned with the algorithmic generation of content [35]. The nature of this content can vary widely, but can include images, music, maps or text. Procedural generation is probably best known for its use in computer games, where it can be used to generate novel worlds, levels or puzzles for the player.

The wide scope of the field means that a wide variety of computational methods are used for different tasks. These can vary from the relatively simple, such as texture generation using noise functions [36], to problems that would ideally require the solution of complex constraint satisfaction problems [35]. This makes the field well-suited to explore with quantum computation: even current and near-term devices can perform relevant tasks at the simple end of the field, and the ability to solve complex constraint satisfaction problems will benefit the field when fault-tolerant quantum computers are realized. We can therefore hope to find and explore ever more sophisticated examples of quantum procedural generation as technology progresses. Of course, any uses with current and near-term devices would not represent examples of quantum advantage [37]. Nevertheless, they would already provide examples of quantum computers in use for computational tasks.

The author's interest in procedural generation was initially sparked a topic covered in the last chapter: the idea of learning about quantum computation through making simple games. An example of using quantum circuits to generate textures was created for this purpose. This was inspired by the use of blur effects in early techniques for procedural generation [35]. It used an amplitude encoding to encode images within a register of qubits, able to store 2^n pixels within n qubits. This encoding was chosen to pack as much as possible within a small number of qubits, both to allow easy simulation as well as implementation on publicly available cloud quantum computers. The exact mapping between qubit states and pixel coordinates was chosen to maintain a relationship between Hamming distance for bit strings in the former with distance between pixels in the latter. Specifically, neighbouring pixels in the image are encoded by strings that differ on only one bit. This means that small angle rotations induce a blur-like effect. Given this behaviour, the method was named 'Quantum Blur' [31].

Quantum Blur was intended as a simple example of how a method for procedural generation could be natively developed as a quantum circuit. It served as an initial proof-of-principle for the use of quantum computational techniques in procedural generation, as well as a tool that beginners could use and learn from in quantum game jams and similar events. However, it has since found use in professional contexts by artists [33]. Specifically, by the artists Roman Lipski [38] and Libby Heaney [39]. It has also been used in a commercial computer game by indie studio MiTale [40].

Quantum Blur was primarily an example of using quantum *software* for procedural generation, since results are more easily and accurately obtained by simulating the circuits than by running them on cloud quantum computers.

The next step was then to find a method for quantum procedural generation that was primarily designed to run on current hardware. This would need to be designed with the abilities and restrictions of near-term devices in mind. Specifically:

- Qubits are arranged on two-dimensional surface with two qubit gates possible between certain pairs of neghbouring qubits;
- Circuits are composed of single and two-qubit gates only, each of which contribute noise;
- Final output bit strings are highly unreliable due to noise, while expectation values can be regarded as more resilient.

These factors mean, for example, than any method requiring a device-wide Fourier transform would be unsuitable. Instead, the method must be built explicitly using the available components.

The approach taken was, essentially, to build an AI for a *Civilization*-like game. In this there are n nations, one for each qubit. These nations have starting positions on a map that reflect the positions of the qubits on their device, such that neighbouring nations correspond to pair of qubits for which two-qubit gates are possible. The start with a small amount of territory around their initial position, and take turns to place cities within their territory. The placement of these cities then redraws the borders between nations. This can result in a nation gaining unclaimed territory, or territory being exchanged between nations. The extreme case of the latter is where a city changes hands, due to its territory becoming dominated by another nation.

The decisions made by nations are determined by a quantum circuit, which is run at the beginning of each round of moves. More specifically, in order to calculate the single qubit expectation values and two-qubit expectation values between neighbours that are required, many copies of the circuit are run different measurements appended to each [41]. These values are used to determine whether the nation will act in an aggressive, defensive or explorative manner, and against which other nation that action will be directed in the former cases. Given this choice of strategy, a position for city placement is then determined classically. At the end of each round, the effects of these moves on the borders is calculated. Based on this information, new gates are added to the circuit. These are chosen such that a nation will become more aggressive or defensive against particular neighbours when gaining and losing territory and cities. These additions to the circuits are implemented entirely via single and two qubit gates, for each nation or pair of nations, respectively.

Though this process has been described in the above as a game, note that it is a game played only by the AI. Running the game is a hybrid quantum-classical process in which a circuit is continually updated and run, with the results influencing the game map, and the changes to the map influencing the changes made to the circuit. Thoughout this process, a map is generated which represents a living history of the nations it depicts, and their interactions over time. This map, and the history of the locations within it, is then the content that has been procedurally generated.

The method used for this AI-based approach is similar to that proposed for imaginary time evolution in [42], demonstrating that even methods designed for very different use cases could be put to use for quantum procedural generation. This insight inspired a third example of quantum procedural generation. Specifically, quantum tools designed to analyze natural language [43, 44] were used to instead generate sentences [34]. The input for this method is a target condition that the generated sentence should satisfy, such as the topic. An iterative process is then applied, starting with a naively generated initial sentence that is then analyzed by the existing quantum tools. This determines the closeness of the current sentence to the target condition. If it does not satisfy the condition, small changes are made and the sentence is analyzed again. The combination of a classical simulated annealing algorithm and the quantum analysis then guide the sentence towards the required condition.

All these methods were specifically inspired by the constraints of near-term devices, and a desire to do as much as possible with them. They show that the quantum computation resources we have now are not just limited to proofs-of-principle, but can be used to create complex and interesting content. We may not yet have quantum advantage, but current quantum hardware can still inspire creativity.

Bibliography

[1] A. Kitaev, "Anyons in an exactly solved model and beyond," *Annals of Physics*, vol. 321, pp. 2–111, jan 2006.

[2] H. Bombin, "Topological order with a twist: Ising anyons from an abelian model," *Physical Review Letters*, vol. 105, jul 2010.

[3] J. R. Wootton, "A family of stabilizer codes for D(Z2) anyons and majorana modes," *Journal of Physics A: Mathematical and Theoretical*, vol. 48, p. 215302, may 2015.

[4] B. Srivastava, A. F. Kockum, and M. Granath, "The xyz2 hexagonal stabilizer code," *Quantum*, vol. 6, p. 698, apr 2022.

[5] IBM Quantum, "IBM Quantum Composer and Lab." *https://quantum-computing.ibm.com.*

[6] J. R. Wootton, "Demonstrating non-abelian braiding of surface code defects in a five qubit experiment," *Quantum Science and Technology*, vol. 2, p. 015006, mar 2017.

[7] A. C. Paul Nation, Hanhee Paik and Z. Nazario, "The IBM Quantum heavy hex lattice." *https://research.ibm.com/blog/heavy-hex-lattice.*

[8] J. R. Wootton, "Hexagonal matching codes with two-body measurements," *Journal of Physics A: Mathematical and Theoretical*, vol. 55, p. 295302, jul 2022.

[9] M. B. Hastings and J. Haah, "Dynamically generated logical qubits," *Quantum*, vol. 5, p. 564, oct 2021.

[10] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton, "Poking holes and cutting corners to achieve clifford gates with the surface code," *Phys. Rev. X*, vol. 7, p. 021029, May 2017.

[11] J. R. Wootton, "Measurements of floquet code plaquette stabilizers." *https://arxiv.org/abs/2210.13154*, 2022.

[12] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, sep 2019.

[13] D. Lidar and T. Brun, eds., *Quantum Error Correction.* Cambridge University Press, 2013.

[14] D. Gottesman, "Stabilizer codes and quantum error correction." *https://arxiv.org/abs/quant-ph/9705052*, 1997.

[15] Google Quantum AI, "Suppressing quantum errors by scaling a surface code logical qubit," 2022.

[16] J. R. Wootton and D. Loss, "Repetition code of 15 qubits," *Phys. Rev. A*, vol. 97, p. 052313, May 2018.

[17] Y. Naveh, E. Kashefi, J. R. Wootton, and K. Bertels, "Theoretical and practical aspects of verification of quantum computers," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 721–730, 2018.

[18] Qiskit Contributors, "Qiskit Ignis." *https://github.com/qiskit/qiskit-ignis*, 2021.

[19] Qiskit Contributors, "Qiskit: An open-source framework for quantum computing." *doi:10.5281/zenodo.2573505*, 2021.

[20] Qiskit Contributors, "Qiskit QEC." *https://github.com/qiskit/qiskit-qec*, 2022.

[21] J. R. Wootton, F. Harkins, N. T. Bronn, A. C. Vazquez, A. Phan, and A. T. Asfaw, "Teaching quantum computing with an interactive textbook," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 385–391, 2021.

[22] J. R. Wootton, "Benchmarking near-term devices with quantum error correction," *Quantum Science and Technology*, vol. 5, p. 044004, jul 2020.

[23] J. R. Wootton, "Syndrome-derived error rates as a benchmark of quantum hardware." *https://arxiv.org/abs/2207.00553*, 2022.

[24] Z. C. Seskir, P. Migdał, C. Weidner, A. Anupam, N. Case, N. Davis, C. Decaroli, İ . Ercan, C. Foti, P. Gora, K. Jankiewicz, B. R. L. Cour, J. Y. Malo, S. Maniscalco, A. Naeemi, L. Nita, N. Parvin, F. Scafirimuto, J. F. Sherson, E. Surer, J. Wootton, L. Yeh, O. Zabello, and M. Chiofalo, "Quantum games and interactive tools for quantum technologies outreach and education," *Optical Engineering*, vol. 61, jul 2022.

[25] J. R. Wootton, "Getting the public involved in quantum error correction," in *Proceedings of GSGS'17*, July 2017.

[26] J. R. Wootton, "Benchmarking of quantum processors with random circuits." *https://arxiv.org/abs/1806.02736*, 2018.

[27] J. R. Wootton, *The History of Games for (Quantum) Computers*, pp. 345–367. Cham: Springer International Publishing, 2022.

[28] L. Piispanen, M. Pfaffhauser, A. Kultima, and J. R. Wootton, "Defining quantum games." *https://arxiv.org/abs/2206.00089*, 2022.

[29] P. Cochin, "Genetic quantum correction." *https://github.com/sneakyweasel/genetic-quantum-correction* , 2017.

[30] S. Maniscalco, C. Foti, G. GarcÍa-Pérez, M. Rossi, D. Cavalcanti, B. Sokolov, R. Maniscalco, and M. Chiofalo, "QPlayLearn." *https://qplaylearn.com/*.

[31] J. R. Wootton, "A quantum procedure for map generation," in *2020 IEEE Conference on Games (CoG)*, IEEE, aug 2020.

[32] J. R. Wootton, "Procedural generation using quantum computation," in *International Conference on the Foundations of Digital Games*, ACM, sep 2020.

[33] J. R. Wootton and M. Pfaffhauser, "Investigating the usefulness of quantum blur." *https://arxiv.org/abs/2112.01646*, 2021.

[34] A. Karamlou, J. Wootton, and M. Pfaffhauser, "Quantum natural language generation on near-term devices," in *Proceedings of the 15th International Conference on Natural Language Generation*, (Waterville, Maine, USA and virtual meeting), pp. 267–277, Association for Computational Linguistics, July 2022.

[35] T. Short and T. Adams, eds., *Procedural Generation in Game Design.* A K Peters/CRC Press, June 2017.

[36] K. Perlin, "An image synthesizer," SIGGRAPH '85, (New York, NY, USA), p. 287–296, Association for Computing Machinery, 1985.

[37] J. Preskill, "Quantum computing and the entanglement frontier." *https://arxiv.org/abs/1203.5813*, 2012.

[38] Qiskit Blog, "Making The Invisible Visible: A New Exhibition of Quantum Art." *https://medium.com/qiskit/making-the-invisible-visible-a-new-exhibition-of-quantum-art-ba1540b9ba82*, 2021.

[39] Libby Heaney, "Art shared on Twitter." *https://twitter.com/LibbyHeaney/status/1187108052887113729*, 2021.

[40] Qiskit Blog, "Upcoming Video Game Will Generate New Levels Using Qiskit and a Quantum Simulator." *https://medium.com/qiskit/upcoming-video-game-will-generate-new-levels-using-qiskit-and-a-quantum-simulator-b47dfc911234*, 2020.

[41] G. Garcí a-Pérez, M. A. C. Rossi, B. Sokolov, E.-M. Borrelli, and S. Maniscalco, "Pairwise tomography networks for many-body quantum systems," *Physical Review Research*, vol. 2, jun 2020.

[42] M. Motta, C. Sun, A. T. K. Tan, M. J. O'Rourke, E. Ye, A. J. Minnich, F. G. S. L. Brandão, and G. K.-L. Chan, "Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution," *Nature Physics*, vol. 16, pp. 205–210, nov 2019.

[43] K. Meichanetzidis, S. Gogioso, G. de Felice, N. Chiappori, A. Toumi, and B. Coecke, "Quantum natural language processing on near-term quantum computers," *Electronic Proceedings in Theoretical Computer Science*, vol. 340, pp. 213–229, sep 2021.

[44] D. Kartsaklis, I. Fan, R. Yeung, A. Pearson, R. Lorenz, A. Toumi, G. de Felice, K. Meichanetzidis, S. Clark, and B. Coecke, "lambeq: An efficient high-level python library for quantum nlp," 2021.