Fast decoders for qudit topological codes

Hussain Anwar[1,2,5]**, Benjamin J Brown**[3,5]**, Earl T Campbell**[4] **and Dan E Browne**[2]

[1] Department of Mathematical Sciences, Brunel University, Uxbridge, Middlesex UB8 3PH, UK
[2] Department of Physics and Astronomy, University College London, Gower Street, London WC1E 6BT, UK
[3] Quantum Optics and Laser Science, Blackett Laboratory, Imperial College London, London SW7 2AZ, UK
[4] Dahlem Center for Complex Quantum Systems, Freie Universitát Berlin, D-14195 Berlin, Germany
E-mail: hussain.anwar@brunel.ac.uk

Abstract

Qudit toric codes are a natural higher-dimensional generalization of the well-studied qubit toric code. However, standard methods for error correction of the qubit toric code are not applicable to them. Novel decoders are needed. In this paper we introduce two renormalization group decoders for qudit codes and analyse their error correction thresholds and efficiency. The first decoder is a generalization of a 'hard-decisions' decoder due to Bravyi and Haah (arXiv:1112.3252). We modify this decoder to overcome a percolation effect which limits its threshold performance for many-level quantum systems. The second decoder is a generalization of a 'soft-decisions' decoder due to Poulin and Duclos-Cianci (2010 *Phys. Rev. Lett.* **104** 050504), with a small cell size to optimize the efficiency of implementation in the high dimensional case. In each case, we estimate thresholds for the uncorrelated bit-flip error model and provide a comparative analysis of the performance of both these approaches to error correction of qudit toric codes.

Keywords: topological error correcting codes, toric code, qudits, thresholds, renormalization group decoders

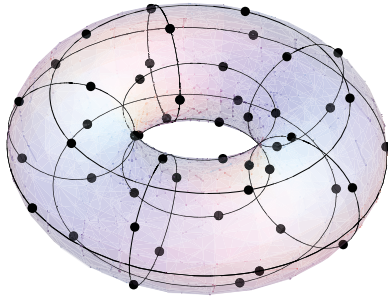[5] These authors contributed equally to this paper.

Figure 1. In a toric code, qudits (black dots) are attached to the edges (black lines) of a square lattice on the surface of a torus. Errors are detected by measuring the code's stabilizer generators—operators which act locally on four qudits associated with each plaquette and vertex.

1. Introduction

The study of quantum error correction and fault-tolerant quantum computation [1–3] for qubit systems is very well established, and the combination of topological codes [4] for robust error tolerance and magic state distillation [5] for universality has become a leading framework for fault-tolerant quantum computation [6–9].

In contrast to qubit systems, fault-tolerant quantum computation with systems of dimension d higher than 2 are less well understood. Only recently were the first magic state distillation schemes for qudit systems developed, which demonstrated improved distillation thresholds and reduced overhead compared with their qubit counterparts [10, 11]. Also, recently the first decoders for qudit topological codes were proposed [12], providing the ingredients for a fault-tolerant qudit computation.

Topological codes were introduced by Kitaev in one of his seminal papers [4], establishing a framework encompassing both qubit and qudit variants. In particular, the toric code places qudits on the surface of a torus, as illustrated in figure 1. Also notable is the planar code, which has similar properties but can be physically realized in two-dimensions [13, 14]. These codes are topological in nature as the quantum information is encoded in degrees of freedom which are independent of the local physics of the code.

For a topological code to protect quantum information, physical errors must be detected and corrected at a sufficient rate to prevent the errors from accumulating and causing an unwanted logical error. An important part of error correction is the *decoder*, the classical algorithm which, given the output of the error detecting measurements (the *syndrome set*) computes a correction operator which should restore the quantum code to its original state. The mapping between syndromes and errors can never be one-to-one (even in classical codes), so a good decoder will output a correction operator which has a high likelihood of successful error correction. Comparing decoders, however, is subtle. Whilst some decoders may achieve the highest thresholds (*optimal* decoders) others may run faster and at more favourable computational cost.

In this work, we implement and refine two types of decoders for the qudit generalization of Kitaev's toric code [15]. One of the motivations behind exploring qudit systems is that stabilizer measurements have more outcomes, and so provide more information with which to determine the error locations. If this extra information is exploited correctly, then improvement in

performance vis-à-vis qubit codes can be observed [16]. Studies of the thermal stability and Hamiltonian realization of the qudit toric code also indicate some advantages in using qudit systems [17]. Moreover, the extra levels in qudit systems allow for the enhancement of quantum memories in two-dimensions via the insertion of domain walls [19, 20], which is not possible in qubit codes [21].

For the *qubit* toric code a variety of decoding algorithms have been studied. The early extensive study by Dennis *et al* [22] demonstrated how the decoding problem for the qubit toric code undergoing independent X and Z errors can be mapped to a statistical–mechanical model, the random-bond Ising model (RBIM). Remarkably, they showed that the optimal error threshold, $p_{\text{th}}^{\text{opt}}$, for this model directly maps to a phase transition point in the RBIM, known as the Nishimori point [23], which had already been identified numerically to be around 10.9% [24–27]. Moreover, Dennis *et al* observed that this optimal threshold lay very close to an important quantity arising in quantum Shannon theory called the *hashing bound threshold* p_{th}^{H} [28] (see appendix B). In the qubit case, and for the independent X and Z error model, the hashing bound threshold is $p_{\text{th}}^{\text{H}} = 11.0028\%$. Further work by Takeda and Nishimori [29] showed that the similarity between the optimal and the hashing bound thresholds applies to more general statistical–mechanical models. They made a conjecture in statistical mechanics terms, which when translated to topological codes, implies that the optimal code threshold should coincide with the hashing bound threshold for a variety of error models, i.e. $p_{\text{th}}^{\text{opt}} = p_{\text{th}}^{\text{H}}$. Whether this conjecture holds (either exactly or approximately) remains an open question, but numerical results so far have supported it [30].

Thresholds are dependent on the error model chosen. In this paper, unless explicitly stated otherwise, all thresholds reported are for the independent X and Z error model, defined in section 3. Note also that the reported threshold values are code thresholds and not fault-tolerance thresholds and that error-free syndrome measurement and correction is assumed.

The most typically employed decoder for the qubit toric code is the minimum-weight perfect matching algorithm (MWPMA), which has an efficient implementation based on Edmonds' blossom algorithm [31]. This algorithm has been extensively studied and its threshold has been estimated to be 10.3% [32]. As we will see below, however, the MWPMA is not suitable for decoding $d > 2$ qudit toric codes.

Recently, a number of alternative decoders have been proposed. Of particular relevance to this paper are the two decoders which utilize renormalization group (RG) ideas, proposed by Duclos-Cianci and Poulin [33, 34], and by Bravyi and Haah [35]. While both of these decoders employ RG techniques, they do so in very different ways. To distinguish between them, we shall use the terminology suggested by Duclos-Cianci and Poulin [36]. We will refer to the first as the *soft-decisions*-RG (SDRG) decoder and the second as the *hard-decisions*-RG (HDRG) decoder. These decoders will be detailed in sections 4 and 5, and the reasons for this terminology will become clear. These decoders have gained great attention recently due to the fact that they have a greater flexibility over MWPMA [37, 38] and yet achieve comparable thresholds. The thresholds that were obtained for the qubit toric code by the SDRG (using 2×1 cells) and HDRG decoders are 8.2% and 6.7%, respectively. Notably, higher thresholds can be achieved using the SDRG decoder by increasing the cell size [33].

In section 4 we generalize the HDRG decoder to decode the qudit toric code. We will show that the construction of this decoder has no dependence on the qudit dimension, which
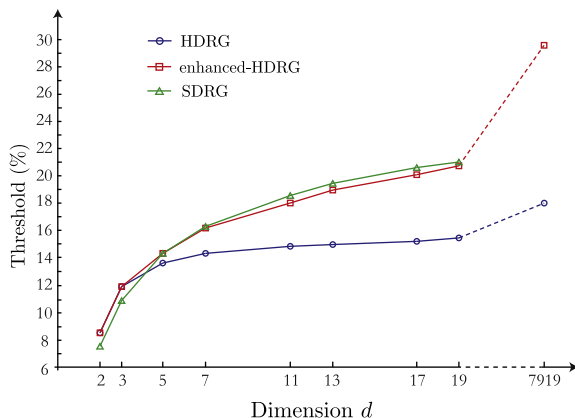
Figure 2. A comparison between the thresholds obtained using the HDRG, enhanced-HDRG and SDRG decoders presented in this paper. The dimension 7919 is the 1000th prime number used here to demonstrate the independence of the HDRG decoder on the dimension of the qudits. The error bars have been omitted for clarity.

allows us to obtain a numerical estimate of the threshold for any dimension. Some of the refinements of our qudit decoder also have benefits for the qubit code. We will demonstrate that the original qubit HDRG decoder of [35] can be improved so that a higher threshold of 8.4% is achieved. In the limit of high dimensions this decoder reaches a saturating threshold of about 18%. We discover that this behaviour is due to a percolation effect, where thresholds achieved by this decoder are always upper bounded by the *syndrome* percolation threshold. To beat this upper bound, we introduce an *initialization step* into the algorithm which disrupts this percolation effect and enhances the performance of the decoder such that, for high dimensional systems, thresholds as high as 30% are obtained. We refer to the HDRG decoder when augmented with the initialization step as the *enhanced*-HDRG decoder.

In section 5 we develop a variant of the SDRG decoder for toric codes of any dimension. Recently, Duclos-Cianci and Poulin generalized the SDRG decoder for qudit codes [12]. They studied the relationship between the hashing bound threshold and the thresholds achieved by their decoder, finding, strikingly, that their thresholds approximate a constant fraction of the hashing bound threshold. Due to the fact that their decoder has a run-time complexity that depends on the dimension as $O\left(d^7\right)$, they were not able to investigate this behaviour beyond $d = 6$. The version of the SDRG decoder that we develop has a different scaling behaviour with a dimension dependence of $O\left(d^4\right)$. The smaller cell size we use leads to lower thresholds, but allows us to analyse much higher dimensional systems. This enables us to estimate thresholds for systems of dimensions up to $d = 19$.

We have summarized the thresholds obtained by our HDRG, enhanced-HDRG and SDRG decoders in figure 2. This paper is structured as follows. In section 2 we review the qudit toric code. In section 3 we describe the noise model formally and describe the numerical method we adopt to estimate the thresholds. Sections 4 and 5 describe our generalization of the HDRG and SDRG decoders, respectively. Finally, we compare and discuss these two decoders in section 6.
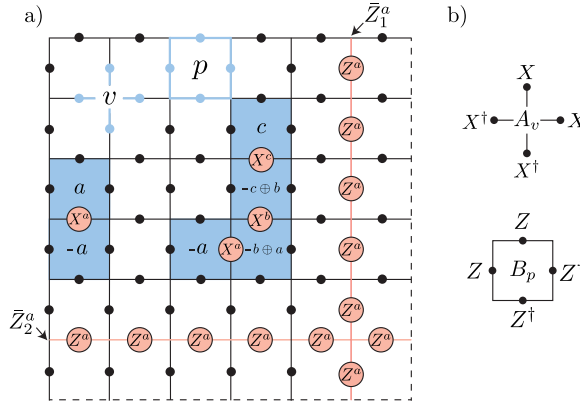
Figure 3. (a) The primal lattice of the toric code with qudits depicted by black dots, and a couple of examples showing how the plaquette syndromes are generated for a set of X errors. The two logical operators $\bar{Z}_1^a$ and $\bar{Z}_2^a$ correspond to the two non-contractible loops on the primal lattice. The dashed lines indicate the periodic boundaries. (b) The qudit vertex and plaquette operators.

2. Qudit toric code

The toric code is a stabilizer error-correcting code described within the stabilizer formalism [39, 40]. Here, we consider a toric code consisting of a lattice of d-level quantum systems, *qudits*, where $d \geqslant 2$. For simplicity, we will restrict our discussion to prime dimensions. Our definitions are valid in both the $d = 2$ and $d > 2$ case.

The single qudit Pauli group, $\mathcal{P}_d$, is generated by

$$ X = \sum_{j \in \mathbb{Z}_d} |j \oplus 1\rangle \langle j|, \quad Z = \sum_{j \in \mathbb{Z}_d} \omega^j |j\rangle \langle j|, \tag{1} $$

where the addition '$\oplus$' is carried out modulo d, and $\omega = e^{2\pi i/d}$ [41]. These operators satisfies the commutation relation $XZ = \omega^{-1}ZX$.

In the case where qudit dimension $d > 2$, unitaries X and Z are not Hermitian, but, possessing orthogonal eigenspaces, they still can be interpreted as observable measurements whose measurement outputs are labelled by complex eigenvalues ω^k. As a short-hand we shall usually denote outcome ω^k simply by integer k.

The n-qudit Pauli group $\mathcal{P}_d^n$ is generated by the n-fold tensor product of single qudit Pauli operators. A stabilizer code is defined by an Abelian subgroup $\mathcal{S}$ of the Pauli group $\mathcal{P}_d^n$, such that it contains the identity operator $\mathbb{1}$ but not any other phase multiples of the identity, i.e. $\omega^j \mathbb{1} \notin \mathcal{S}$ for all non-zero j. The common $+1$ eigenspace of S forms a protected subspace called the code-space, and the elements of $\mathcal{S}$ are called the stabilizers of the code.

The qudit toric code is defined on a square lattice of $L \times L$ vertices with periodic boundary conditions, where we have a qudit on each of the edges of the lattice, see figure 3(a). The stabilizer group of the qudit toric code is generated by two types of stabilizers, the *vertex* operators A_v and the *plaquette* operators B_p. There is an A_v stabilizer for each vertex v and a B_p stabilizer for each plaquette p of the lattice. We show examples of A_v and B_p operators in figure 3(b). The qudit toric code encodes two logical qudits. The operators acting within the

code-space on the logical information are string-like operators that have support on non-contractible loops of the lattice. We show examples of logical operators $\bar{Z}_1^a$ and $\bar{Z}_2^a$ in figure 3(a). The conjugate $\bar{X}_j^a$ logical operators have support over non-contractible loops on the *dual* lattice. Logical operators commute with the stabilizer group, but are not members of $\mathcal{S}$. We shall label the initial encoded state, which we wish to preserve, as $\left| \psi_{\text{init}} \right\rangle$.

The vertex and plaquette operators collectively comprise the *check* operators for the code. These are the operators which are measured to determine the error syndromes. We shall call the integer outcome a of a measured check operator a *syndrome*. If this outcome is zero we call this a trivial syndrome. The combined outputs of all check operators we shall call the syndrome set. The syndrome set thus comprises an integer a for each vertex v and plaquette p check operator, such that $a = 0, 1,..., d - 1$ correspond to the eigenvalues ω^a of the measured operator.

After measuring the set of stabilizer generators, errors are projected to Pauli errors and the lattice will be in the state $E \left| \psi_{\text{init}} \right\rangle$ such that $E \in \mathcal{P}_d^n$. In common with other CSS codes [2, 3], we can deal with X-type and Z-type errors on the toric code independently. Here we will focus on X errors, however the behaviour of Z errors is equivalent with respect to the dual lattice. The relationship between the errors and the syndromes is crucial to the design of the decoder, in essence the decoder is trying to invert this mapping.

If a toric code state undergoes a single X^a error, as shown for example in figure 3(a), then a pair of non-zero syndromes are generated with values a and $-a$, where negative numbers are defined modulo d. For multiple errors the integer syndromes combine *additively*. For example, the three errors X^a, X^b and X^c in figure 3(a) generate syndromes $-b \oplus a$ and $-c \oplus b$ on the plaquettes between pairs of errors. Note that in general, any set of syndrome outcomes generated by a set of errors will sum to zero (e.g. for a single error $a + (-a) = 0$).

We see that if adjacent errors correspond to equal (in some directions) and opposite (in other directions) powers of X, then the intermediate syndrome cancels to the zero outcome. It is this behaviour which strongly distinguishes qubit toric codes from $d > 2$ qudit codes. For qubits, Pauli operators X and Z are self-inverse, which means that intermediate syndromes always cancel. A string of errors will only generate non-trivial syndromes at its ends. For $d > 2$, however, the syndrome between adjacent errors will cancel only in the special case that adjacent errors are equal (or opposite, depending on the direction). This is the principle reason why decoders designed for the qubit code will often not immediately generalize to the $d > 2$ setting.

The task of the decoder, then, is to derive a correction operator $F^\dagger$ which returns the state to the code-space. Given any correction operator $F^\dagger$ which (trivially) returns the logical operator to the code-space, and given any logical operator on the code-space L, then the combined operator $LF^\dagger$ will also return the system to its code-space. Indeed, given any set of errors E and any logical operator L, the error set EL will return an identical syndrome. It is this degeneracy between the errors and the syndromes which makes decoding non-trivial. For a successful correction, F must be chosen such that $F^\dagger E$ acts trivially on the encoded logical information, i.e. implements the identity operator and not a non-trivial logical operator.

Considering Z operators alone, the logical operators on the toric code are of the form $\bar{Z}_1^j \bar{Z}_2^k$ for all j and k integers between $0 \leqslant j, k \leqslant d - 1$. There are thus d^2 equivalence classes of correction operators for any given syndrome. It is the role of the decoder to choose the most

appropriate correction operator given the underlying error model. We call a decoder with the highest probability of choosing the correct equivalence class of correction operators the optimal decoder.

It is sometimes useful to adopt *quasi-particle* terminology to describe syndromes on a toric code. We interpret a single syndrome outcome a as representing a quasi-particle of *charge a*, with $-a$ outcome representing the anti-particle of a. The set of quasi-particles generated by any set of errors is then of neutral charge, and sets of quasi-particles of overall neutral charge can be *annihilated* by the application of an appropriate correction operator.

The mathematical structure of the toric code can be elegantly and concisely described using the mathematics of *homology*. Homology theory provides a concise and exact representation of the (otherwise somewhat complicated) relationship between errors and syndromes, and the logical operators on the code-space. Since we do not expect the typical reader of this paper to be familiar with this field, we have aimed to refrain from using homological terms, as much as we can, in the main text and provide a primer on homology in appendix A. Some aspects of the SDRG decoder, however, are most cleanly expressed in homology terms and these shall be explained at the beginning of section 5. A powerful aspect of homology is that the relationship between errors and syndromes on the toric code is the same for $d = 2$ and all higher values of d.

It is worth noting that we do not choose to generalize the MWPMA. In the qubit case, the MWPMA typically consists of constructing a complete weighted graph where the vertices are non-trivial syndromes and the weight of the edges is the shortest Manhattan distance between the vertices. Then using Edmonds' algorithm [31, 42] the perfect matching of minimum weight can be efficiently determined. In quasi-particle language MWPMA works by associating pairs of syndromes, which together have neutral charge. It then finds the minimal weight correction operators to annihilate each pair. In higher dimensions, however, the charge of the vertices ranges across the set $\{1, ..., d - 1\}$. Any neutral set (not just a pairing) of clusters should be considered for annihilation. Thus to find the lowest weight error correction chains for the qudit code requires an algorithm, which must minimize weights on a hypergraph whose hyperedges consist of all charge neutral subsets of vertices. Minimum weight hypergraph matching is in general an NP-Hard problem [43]. We therefore do not expect good speed performance for such a decoder, and have not pursued it here.

3. Noise model and threshold estimation

In this section we define the noise model used in this paper, and review the numerical methods we use to estimate the thresholds. We work with the independent error model, otherwise known as the uncorrelated error model, which is widely used in other studies of the toric code [12, 22]. The principle benefits of this model are its simplicity and its direct mapping to statistical mechanics models (e.g. the RBIM and Potts gauge glass models).

In the independent error model, we treat *X*-type and *Z*-type errors as separate processes which act independently on each physical qudit. Each channel has a simple definition. For *X*-type errors, with probability $1 - p$ no error occurs to the qudit, and with probability $p/(d - 1)$ the error operator X^j is applied, where j is an integer between $1 < j \leqslant d - 1$. The *Z*-type errors occur according to an analogous channel with the same probability p. The important feature of

this error model is that all powers of X occur with equal likelihood, and X and Z errors are uncorrelated.

We will estimate these thresholds numerically via a Monte Carlo simulation. For a single Monte Carlo sample, we initiate the lattice in the pure state of the code-space, fix p and then generate a random error configuration using the above noise model. The syndromes of the error configuration are then measured and fed to the decoder. The decoding algorithm will return a Pauli correction operator $F^{\dagger}$, that will return the system to the code-space.

In the simulation we repeat this procedure N times for a given p, and we evaluate the success probability p_{succ} as the fraction of times the decoder succeeds. The standard deviation in the estimated success probability is $\sigma = \sqrt{p_{\text{succ}}\left(1 - p_{\text{succ}}\right)/N}$. To determine the threshold, we plot p versus p_{succ} for different lattice sizes as shown, for example, by figures 5 and 13. The threshold p_{th} is defined to be the *point* at which the success probability curves intersect in the limit $L \to \infty$. In other words, the threshold represents the point below which arbitrarily high p_{succ} can be achieved provided that the lattice is made large enough. However, in the actual simulations, the data points can only be obtained for a relatively small lattice sizes L, and such lattices are subject to small system size effects, which can affect the evaluation of p_{th}. This is easily seen in the $L = 16$ curve of figure 5.

To account for the small system size effects, we estimate p_{th} by using the fitting proposed by Harrington *et al* [32]. In this fitting, *all* the data points are fitted to the curve

$$p_{\text{succ}} = A + Bx + Cx^2 + DL^{-1/\mu}, \tag{2}$$

where $x = \left(p - p_{th}\right)L^{1/\nu}$, as shown, for example, in the boxed plot in figure 5. The last term in the fitting, $DL^{-1/\mu}$, accounts for the the small size effects. We can see that, in the limit of $L \to \infty$, this term tends to 0, where μ is positive[6].

In the next two sections we introduce two types of decoders, namely, the HDRG and SDRG decoders, suitable for decoding the qudit toric code of any dimension. We describe earlier formulations of these decoders, our refinements to them and the resulting thresholds achieved for the independent error model.

4. HDRG decoder

The HDRG decoder was introduced by Bravyi and Haah in [35] as an efficient decoder for general local topological codes [44, 45], where MWPMA is an unsuitable decoder. For the qubit toric code, they obtain a threshold of 6.7% using the independent noise model. The construction of this decoder built upon previous work by Harrington [46] and Dennis [47]. In the first sub-section of this section, we present a refined version of this decoder and show that these refinements achieve higher thresholds for the toric code.

---

[6] We have used the **NonlinearModelFit** function in Mathematica to estimate the fitting parameters $\left\{A, B, C, D, p_{th}, \nu, \mu\right\}$. In particular we have used the options 'BestFitParameters' to extract the parameter estimates, and 'ParameterErrors' to estimate the standard deviation error in each parameter.

4.1. Decoder description

The HDRG decoder has a simple and elegant intuition behind its construction, and before we introduce it formally we shall give a heuristic description of how it works. If error rates are low, errors will typically be sparsely distributed. Each cluster of errors will generate a cluster of non-zero syndrome measurements in its immediate vicinity. Identifying and annihilating such small clusters of syndromes is the heart of Bravyi and Haah's HDRG decoding technique.

The decoder considers the syndromes on the lattice over many levels of decoding. Each level of decoding is associated with a geometric measure of distance on the square lattice, such that the distance gets bigger as the levels increase. At each level, the non-trivial syndromes are divided into clusters which are disjoint with respect to this measure. In other words, in each cluster, all of the syndromes are separated from at least one other syndrome of the cluster by a distance no greater than that determined by the decoding level. If the sum of all the syndromes within a cluster is zero (modulo *d*), i.e. if the cluster is neutral, then the syndromes of the clusters are annihilated locally with respect to the cluster. Clusters whose total syndromes are non-zero are referred to as *charged* clusters. Charged clusters are passed to the next level until ultimately they become part of a neutral cluster which is annihilated. Next, we will define and explain all the aspects of this decoder more rigorously.

Without loss of generality we shall consider only the plaquette syndromes, with the analogous vertex formalism being obvious. Firstly, we review some distance measures between plaquettes labelled as $\vec{x} = (x_1, x_2)$ and $\vec{y} = (y_1, y_2)$. The Manhattan (or taxi-cab) distance is $D^{(1)}(\vec{x}, \vec{y}) = |x_1 - y_1| + |x_2 - y_2|$. Our second distance is the Max distance, and is $D^{(\infty)}(\vec{x}, \vec{y}) = \max\left\{|x_1 - y_1|, |x_2 - y_2|\right\}$. Both can be used to define balls of a certain radius, centred on a plaquettes *p*, such that

$$B_r^{(1)}(\vec{x}) = \left\{\vec{y} \,\middle|\, D^1(\vec{x}, \vec{y}) \leqslant r\right\}, \tag{3}$$

$$B_r^{(\infty)}(\vec{x}) = \left\{\vec{y} \,\middle|\, D^\infty(\vec{x}, \vec{y}) \leqslant r\right\}. \tag{4}$$

Although called balls, the Max distance generates squares and the taxi-cab distance picks out diamonds. However, primarily we are interested in regions that combine these two regions. For any integers *r* and *s*, we define regions $\mathcal{R}_{r,s}$ that when centred on a point $\vec{x}$ are

$$\mathcal{R}_{r,s}(\vec{x}) = B_{r+s}^{(1)}(\vec{x}) \bigcap B_r^{(\infty)}(\vec{x}), \tag{5}$$

and so simply the intersection of two balls with different metrics. The first few instances are shown in figure 4, and clearly we only need to consider $s \leqslant r$. Note that the regions are symmetric, so if $\vec{x} \in \mathcal{R}_{r,s}(\vec{y})$ then $\vec{y} \in \mathcal{R}_{r,s}(\vec{x})$ and when this happens we say $\vec{x}$ and $\vec{y}$ are (r,s)-connected.

Furthermore, we need a notion of connection for a *cluster C*, or set, of plaquettes. Firstly, we define connected paths in *C*. A path γ in *C* is an *ordered* subset of *C*, such that $\gamma = \left\{\vec{x}^{(1)}, \vec{x}^{(2)}, ..., \vec{x}^{(n+1)}\right\}$, and we define it to be an (r,s)-path if for all *j*, $\vec{x}^{(j)}$ and $\vec{x}^{(j+1)}$ are (r,s)-connected. Now, we say the cluster is an (r,s)-cluster if for all $\vec{x}, \vec{y} \in C$ there exists an (r,s)-path in *C* starting at $\vec{x}$ and ending at $\vec{y}$. The intuition behind considering these particular regions
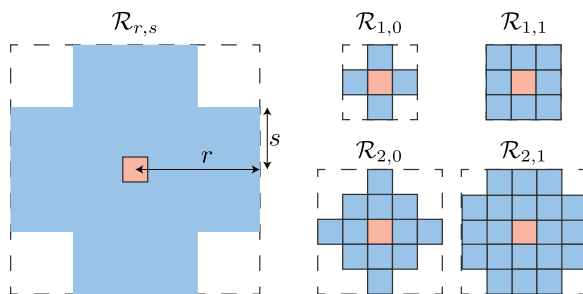
Figure 4. The refined regions $\mathcal{R}_{r,s}$ on a taxi-cab geometry (left-hand side) with examples of the first four levels (right-hand side).

is to take into account some of the degeneracy in the errors creating the syndromes. We will discuss this point in more detail in the next section.

These geometric concepts can be used to explain the decoding scheme. Given the measurement data of all the plaquettes, if plaquette $\vec{x}$ has measurement outcome $m_{\vec{x}}$, then information is conveyed by the ordered pair $(\vec{x}, m_{\vec{x}})$, and the full list of charged plaquettes is $\mathcal{W} = \left\{ (\vec{x}, m_{\vec{x}}) \big| m_{\vec{x}} \neq 0 \right\}$. Similarly, a charged cluster is a subset of the full charge distribution, $C \subset \mathcal{W}$, where we use a different script to indicate the presence of charge information. A cluster is said to be *neutral* if the *total* charge is zero, so that $\sum_C m_{\vec{x}} = 0$ modulo d. Neutral clusters can always be annihilated by transporting and fusing the syndromes within the cluster until the total charge disappears. When doing so, we update the plaquette information from $\mathcal{W}$ to $\mathcal{W}'$, such that the annihilated neutral cluster C is no longer contained in $\mathcal{W}'$. Despite there being many different ways to annihilate the charge, all of these are equivalent assuming only that the cluster is small compared to the lattice and the charges are transported within the cluster. The intuition behind the HDRG decoder is that if such small clusters are annihilated locally, then the resultant correction chains, combined with the actual error chain, will form a trivial loop of errors. That is, a stabilizer element of the code and so equivalent to the identity on the code-space.

The complete set $\mathcal{W}$ can always be partitioned into a set of *disjoint* clusters $\tilde{\mathcal{W}} = \{C_1, C_2, ..., C_n\}$, for some n, and where $\mathcal{W} = C_1 \cup C_2 \cup ... \cup C_n$. We say a particular partition $\tilde{\mathcal{W}}$ is an (r,s)-partition if both the following conditions are satisfied:

(a) *Connectivity:* every charged cluster in the partition is an (r,s)-cluster;
(b) *Maximality:* for any distinct pair of charged clusters in the partition, C_j and $C_{k \neq j}$, we find that $C_j \cup C_k$ is not an (r,s)-cluster.

Maximality tells us there is no suitable path between the disjoint clusters, and so they could not be merged into a single cluster. Furthermore, whenever the connectivity condition is met, but maximally fails, there exists another partition that does fulfil both conditions using fewer charged clusters.

As stated previously, the HDRG decoder involves multiple levels of decoding. Each decoding level l is associated with a choice of regions $\mathcal{R}_{r,s}$. At the first level we begin with $(r,s) = (1, 0)$. The parameters increase iteratively, such that for $l + 1$, first we try to increase s

by 1, but if $s = r$ we instead increase r by one and reset s to zero. The relation between the level number and the distance parameters can be determined with simple calculation to be $r(r + 1)/2 + s$. The decoder performs the following, beginning with the first level $l = 1$:

 (i) *Clustering*: find an (r,s)-partition of $\mathcal{W}_l$.

 (ii) *Neutral annihilation*: for every neutrally-charged cluster in the partition, for instance C_j, find a Pauli correction e_j with support entirely on C_j that annihilates all the syndromes within the cluster.

(iii) *Refresh*: record the collective Pauli correction $\prod_j e_j$ and update the syndrome information to $\mathcal{W}_{l+1}$. If $\mathcal{W}_{l+1}$ is non-empty, then repeat at next level $l = l + 1$.

It is helpful to refer to individual levels of the decoder as sub-protocols that we label $\mathfrak{D}_l$. Any charged cluster that cannot be annihilated completely by $\mathfrak{D}_l$, is therefore left for the next higher level of decoding $\mathfrak{D}_{l+1}$. The higher levels will have larger regions and therefore any charged clusters will eventually be combined and form bigger neutral clusters which can then be annihilated. Also, notice that in the HDRG construction the correction chains are determined during the neutral annihilation step at every level of decoding. In classical coding theory, this is a typical feature of what is known as a *hard-decision decoder* [48, 49]. Later, in section 4.4, we will introduction an initialization step that is not part of the above main loop and occurs before $\mathfrak{D}_1$. The initialization step performs some syndrome manipulation to edit $\mathcal{W}$ prior to use in the first level of the decoder.

There are several crucial differences between our version of the HDRG decoder described above and the original decoder by Bravyi and Haah [35]. First, the distance measure in [35] is the Max distance $D^{(\infty)}$. Recall that a ball of radius r in the Max distance is denoted B_r^∞, and Bravyi and Haah use such a region at level-r of the decoder where r grows *exponentially* with the decoder level. The refined metric we choose here includes all such regions, since $\mathcal{R}_{r,0} = B_r^\infty$, but our decoder will consider additional intermittent levels corresponding to $\mathcal{R}_{r,s\neq0}$. In addition, our metric grows as $O\left(r^2\right)$ as the decoding level increases. It is important to point out that in [35] Bravyi and Haah presented a proof for the existence of a threshold for their HDRG decoder. The proof makes use of the exponential increase of their $D^{(\infty)}$ metric, and it is not clear to us that it can be extended to the linearly growing refined metric we consider. However, as we will see, our numerics demonstrate clearly the existence of a threshold. Second, their decoder declares failure and aborts if the area of a cluster is larger than half the lattice size. The idea behind this requirement is that annihilating such large clusters would very likely lead to a logical error. In our decoder we do not enforce this requirement because, as we will see, in higher dimensions the syndrome tends to percolate at high enough error probability, and we would like to investigate how this decoder behaves in such regimes. For this reason, in our implementation here, in the annihilation step we simply combine the syndromes within a cluster with their first possible local neighbours. Similar variations of the HDRG decoder have been proposed very recently in [50–52].

The construction of the HDRG decoder has no dependence on the actual charge of the syndromes. All that matters in the clustering step is the fact that the syndromes have non-zero charge. Similarly, in the neutral annihilation step the syndromes in a neutral cluster are combined with their local neighbours systematically regardless of their actual charge. As a
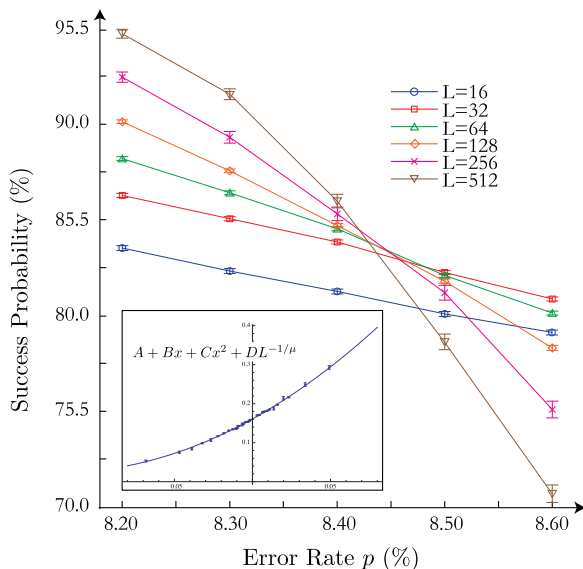
Figure 5. The success probability of the HDRG decoder for the qubit case. The data points are generated with $N = 10^5$ samples for $L \in \{16, 32, 64, 128\}$ and $N = 10^4$ samples for $L \in \{256, 512\}$. The error bars are are taken to be 2σ. The boxed plot shows the data fitting, where $x = (p - p_{th})L^{1/\nu}$, $\nu = 1.85 \pm 0.04$ and $\mu = 0.46 \pm 0.06$.

consequence, the run-time complexity of the HDRG decoder depends only on the number of non-trivial syndromes. Note that the *density* of syndromes increases as the qudit dimension increases. This is because in the qudit case, the probability of two neighbouring errors with opposite weights to occur diminishes quickly as the dimension increases, and hence any shared syndrome will almost always be detected (see figure 3). Therefore, the exact run-time complexity needs to capture the relation between the number of syndromes and the qudit dimension, which is not a trivial task. Nevertheless, our numerical analysis shows that this dependence on the qudit dimension is negligible and for practical purposes can be ignored.

It is not hard to derive an upper bound for the worst case run-time complexity for our HDRG decoder. At each decoding level, the complexity of each of the clustering and the neutral annihilation steps is at most $O(L^2)$. Since in our construction we have considered a quadratically increasing level, then at most $O(L^2)$ levels of decoding are needed. As a result, our HDRG decoder has a worst run-time complexity of $O(L^6)$.

4.2. Threshold estimation

In this section we present the results of the Monte Carlo simulation for the HDRG decoder. We begin with the qubit case before moving to higher dimensions. We plot the success probability curves for the qubit case in figure 5. Using the fitting described in equation (2), we estimate the threshold to be $8.4\% \pm 0.01$.

Recall that the threshold achieved by the original HDRG decoder in [35] was 6.7%. The improvement in the threshold achieved by our HDRG decoder is mainly due to the refined set of regions $\mathcal{R}_{r,s}$, which we have adopted in favour over using just the Max distance, to account for
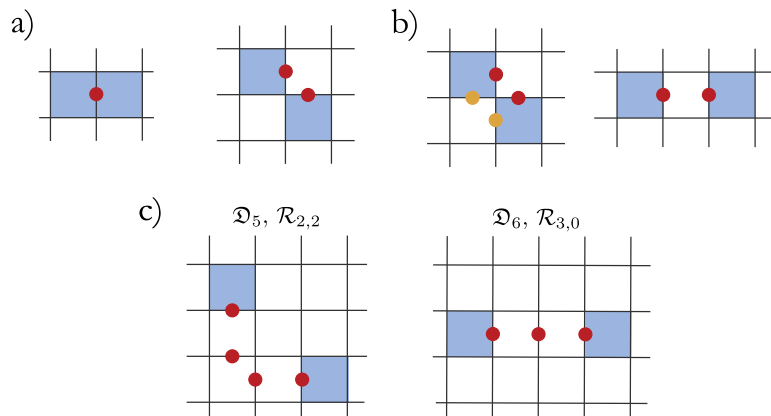
Figure 6. The regions $\mathcal{R}_{r,s}$ distinguishes between (a) and (b), whereas the D^∞-metric does not. (c) The optimal distance measure will switch levels 5 and 6.

some of the degeneracy of difference error configurations. To demonstrate this point, we consider two simple examples. Figure 6(a) shows two plaquettes created by one and two errors. Clearly the single error is more likely to occur in comparison to two neighbouring errors. However, with the D^∞-metric the plaquettes in both cases will be connected at the first level. But the regions $\mathcal{R}_{r,s}$ distinguish between the two cases, and they will be connected at two separate decoding levels, namely $\mathfrak{D}_1$ and $\mathfrak{D}_2$. Also, figure 6(b) shows two cases of two plaquettes created by two errors. For the first case, there are two errors for which the set of successful recovery operations are identical. Hence, the first case is more probable to occur since it has double the degeneracy. The regions $\mathcal{R}_{r,s}$ better account for this degeneracy by again treating these cases in two separate levels, namely $\mathfrak{D}_2$ and $\mathfrak{D}_3$. The overall effect of such refinement is to create finer clusters which would lead to better error correction during the annihilation step.

   The above observations suggest that to improve our decoder further one can consider a different sequence of regions. Such distance measure is optimal in the sense that it will always connect syndromes that can be created by fewer errors and higher degeneracy first. It is not hard to see that such improvement will switch, for example, level $\mathfrak{D}_5$ with level $\mathfrak{D}_6$, because the latter will connect syndromes created by fewer errors as shown in figure 6(c). Our approach, however, was easier to implement, and we leave such further improvement for a future investigation. The above ideas of improving a decoder performance by accounting for degeneracy has been considered for other decoders too. For instance, in [53] Stace and Barrett demonstrated how the threshold achieved by the MWPMA can be improved from 10.3% to 10.6% by including the degeneracy in the weights of the edges to refine the outcome of the perfect matching.

   The thresholds of the remaining prime dimensions are plotted in figure 7. To demonstrate that a numerical estimate of the threshold can be obtained for any dimension we have chosen the 1000th prime number $d = 7919$ to represent the limit of high d. As can be seen from figure 7, the threshold increases monotonically with qudit dimension and reaches a saturating value of about 18%. We have discovered that the reason for this behaviour is due to a percolation effect, which we will discuss next.
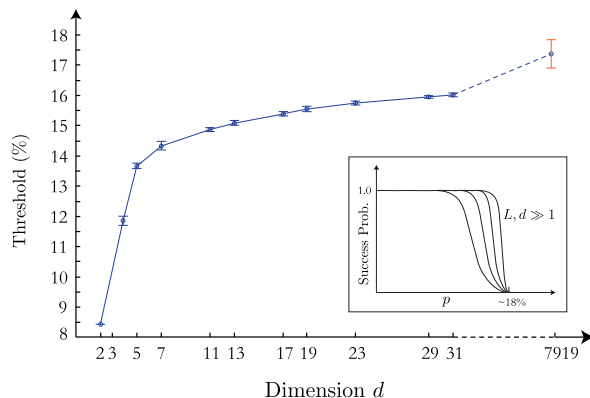
Figure 7. The threshold values of the HDRG decoder for prime dimensions with 2σ error bars. The boxed plot is an illustrative figure of the behaviour of the success probability in the limit of high d.
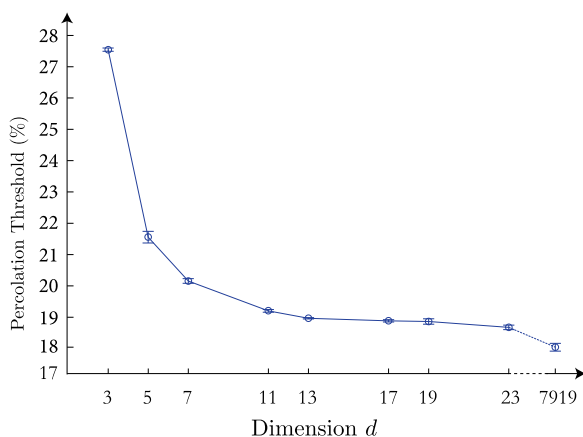

Figure 8. Syndrome percolation threshold for prime dimensions with 2σ error bars.

It is not hard to see that for a given error rate the density of syndromes increases as the dimension of the qudits increase. In fact, as we will show in the next section, for any given prime dimension $d \geqslant 3$, there exists a unique threshold error rate at which the syndromes percolate the lattice. In other words, above this threshold the syndromes will always span the lattice completely. We refer to this threshold as the *syndrome percolation threshold*, denoted here by $p_{\text{th}}^{\text{syn}}$. We will provide numerical estimates of this threshold in the next section. We will find that it decreases as the dimension increases until it reaches a constant value of about 18% in the limit of high d, see figure 8.

The syndrome percolation has severe consequences for the HDRG decoder. For any error rate $p > p_{\text{th}}^{\text{synd}}$ there will be one percolating neutral cluster at the first level $\mathfrak{D}_1$ of decoding. The HDRG will try to annihilate the syndromes with their nearest-neighbours and will most likely fail. This suggests that we cannot expect the HDRG decoder to achieve a threshold higher than the percolation threshold, because the success probability curves must diminish above the percolation threshold. Indeed this is what we observe in the limit of high d, as illustrated by the boxed plot in figure 7. The point of intersection of the curves (which defines the threshold)

intersects the *x*-axis at the value of the percolation threshold. The actual curves (omitted here) are too noisy around the syndrome percolation threshold, for this reason we have indicated by the red error bar the range at which the actual curves intersect in the limit of high *d*. Furthermore, our numerical analysis shows that if we ignore the small lattice sizes, then the curves of the large lattice sizes clearly cross at single point around 18%.

The conclusion of the above discussion is that in the limit of high *d* we expect the syndrome percolation threshold to be a close upper bound to the threshold achieved by the HDRG decoder. Next, we outline some basic concepts in percolation theory and show how the syndrome percolation thresholds can be estimated.

4.3. Syndrome percolation thresholds

Percolation theory is the study of connectivity and transport on random graphs [54–56]. A standard percolation model consists of a random graph whose *sites* are distributed in space, and the *bonds* connect neighbouring sites only. We are mainly interested in the percolation behaviour on a two-dimensional regular graph, and in particular the regular square graph. There are typically two stochastic mechanisms associated with each graph structure: either the sites of the graph are fixed in space and bonds are made randomly on them, or sites are random in space and the bonds are determined on the neighbouring sites.

For instance, in the random vertex model, each site is 'empty' with probability *p* and otherwise it is 'occupied'. For each instance, percolation occurs if there is a nearest neighbour path that spans the graph using only occupied sites. The key result of percolation theory is that there exists a threshold, $p_{th}^{site} = 59.27\%$ [55], above which the probability of percolation approaches unity with increasing lattice size, and below threshold the percolation probability vanishes in the large lattice limit. A similar phenomenon occurs when randomly removing graph bonds, and has an analytic threshold of $p_{th}^{bond} = 50\%$.

On the square lattice of the toric code, the bonds correspond to the qudits on the edges of the lattice and the sites correspond to the vertices/plaquette operators. In our discussion here, we are interested in the *syndrome* percolation threshold of the toric code. This is not equivalent to site percolation because the syndromes are created in pairs by qudit errors on the lattice edges. Given a syndrome $\mathcal{W}$ we say that it percolates the lattice if there is a nearest neighbour path in $\mathcal{W}$ that spans the lattice. Using our terminology, a nearest neighbour path is a $(1, 0)$-path in $\mathcal{W}$. There have been studies of site percolation with distant neighbouring interaction [57, 58], but to our knowledge there have not been investigations where bonds interact with sites in the manner defined by the toric code. Also, there does not appear to be an analytic method that can determine the syndrome percolation threshold precisely from the known theory on the bond and site percolation.

We resort to estimating the syndrome percolation numerically via Monte Carlo simulations. The simulation is straightforward and it is very similar to that described in section 3 in estimating the error correction threshold of a general decoder. For a given dimension *d*, error rate *p*, and lattice size *L*, we generate a qudit lattice such that each qudit suffers an error with probability *p*. The syndromes are then calculated. If the syndromes percolate, then the simulation will be declared successful, otherwise it is a failure. This procedure is then repeated *N* times, and the success probability is evaluated as the fraction of times the simulation has succeeded. The

simulation is repeated for a fixed range of p for different lattice sizes. The threshold is determined as the point of intersection of the different success probability curves.

The numerical estimates for syndrome percolation thresholds obtained are presented in figure 8. As can be seen from this figure, there does not exist a syndrome percolation threshold for the qubit case, a fact that can be understood as follows. Consider the probability that a plaquette (the toric code analogue of a site) is non-trivial given that the four neighbouring qubits independently suffer an error with a probability p. This probability can easily be shown to be $4p\left(1-p\right)^3 + 4p^3\left(1-p\right)$. This expression is symmetric about $p = 0.5 \pm c$, where c is a constant between $0 \leqslant c \leqslant 0.5$. This indicates that the profile of the success probability curve versus the error rate p has a bell shape about $p = 0.5$, and therefore prohibiting the existence of a unique threshold point above which the lattice always percolates. However, for the remaining prime dimensions, such symmetry does not exist and we always observe a threshold. We see that the syndrome percolation threshold decreases monotonically with the qudit dimension, and in the limit of high d it reaches a constant value of about 18%. This confirms the conclusion of the last section in that the syndrome percolation threshold is an upper bound for the HDRG decoder. In the next section we will show how the HDRG decoder can be enhanced to beat this upper bound.

4.4. Beating the percolation threshold

To overcome the syndrome percolation threshold we introduce an *initialization step* $\mathcal{I}_{r,s}$ that enhances the performance of the HDRG decoder. Although this step is not efficient, we show that for a sufficiently high d it can boost the threshold to about 30% at a computationally feasible cost. The initialization step is designed to dissect any percolating cluster into a more sparse set of clusters *before* running the HDRG decoder. It achieves this by using a brute force method in finding any neutral *sub-clusters* within a percolating cluster. The sub-clusters are then annihilated before running the HDRG decoder. We have constructed the initialization step to search for the sub-clusters systematically by utilizing similar concepts as those used in the HDRG decoder. For example, the subscripts r and s of each step $\mathcal{I}_{r,s}$ take the same increasing integers as previously defined, and here they quantify the *depth* of searching for the neutral sub-clusters in the lattice.

Each initialization step $\mathcal{I}_{r,s}$ consists of a series of *initialization levels* $\mathcal{L}_{r,s}$ that systematically search for neutral sub-clusters. More precisely, in this construction, each step $\mathcal{I}_{r,s}$ simply involves running all the initialization levels in ascending order such that $\mathcal{I}_{r,s} = \left\{ \mathcal{L}_{1,0}, \mathcal{L}_{1,1},..., \mathcal{L}_{r,s} \right\}$. Loosely speaking, the subscripts r and s of each level $\mathcal{L}_{r,s}$ quantify the 'size' of the regions to search over for any neutral sub-clusters; we will expand on this point shortly. Each level $\mathcal{L}_{r,s}$ is associated with a *set* of syndromes $Q_{r,s}$. The set $Q_{r,s}$ consists of the syndromes at the outer layer of $\mathcal{R}_{r,s}$, such that

$$Q_{r,s}(\vec{x}) = \begin{cases} \text{for } s = 0, \ \mathcal{R}_{r,s}\backslash\mathcal{R}_{r-1,r-1}, \\ \text{for } s > 0, \ \mathcal{R}_{r,s}\backslash\mathcal{R}_{r,s-1}, \end{cases} \tag{6}$$

where '$A\backslash B$' just means in A but not in B. This is more easily shown by the examples in figure 9. We denote the elements of the set $Q_{r,s}$ by q_k, and by definition, each set has either four or eight syndromes. Next, we describe how the searching procedure at each initialization level
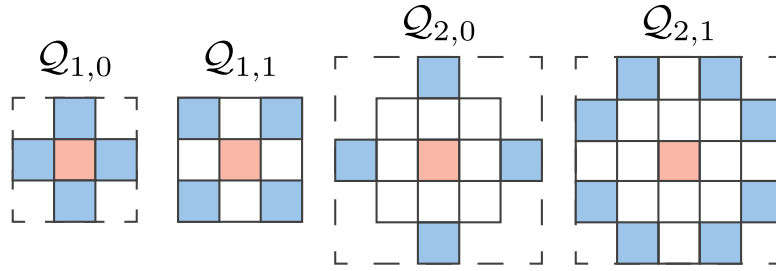
Figure 9. The set of syndromes $\mathcal{Q}_{r,s}$ for the first four initialization levels $\mathcal{L}_{r,s}$. The red square is the syndrome u and the blue squares are the q_k syndromes at the outer layer of $\mathcal{R}_{r,s}$.

works, and again without loss of generality we will limit the discussion to the plaquette operators. We will denote the set of all plaquettes by $\mathcal{U} = \{u_1, u_2, ..., u_{L^2}\}$.

At each level $\mathcal{L}_{r,s}$, the search for sub-clusters is performed by starting at a plaquette u_j in the lattice (regardless if it is charged or not) and then for each $q_k \in \mathcal{Q}_{r,s}$, we construct a *search rectangle* $\mathcal{T}$, which is defined as the *minimum* size rectangle that encloses syndromes u_j and q_k. In other words, the plaquettes u_j and q_k form the opposite corners of the search rectangle. Inside $\mathcal{T}$, we define a *search-path* τ as any $(1, 0)$-path in $\mathcal{T}$ that starts at u_j and ends at q_k. There are many such paths, and by construction, every path will contain $|\tau| = (r + s + 1)$ elements in total. We denote the set of all possible search-paths in $\mathcal{T}$ by $T = \{\tau_1, ..., \tau_{|T|}\}$, where $|T|$ is the total number of possible paths. From a pure geometric point of view, a rectangle consisting of $a \times b$ plaquettes has $|T| = (a + b)!/a!b!$ possible paths connecting its corners. This expression was calculated by considering the equivalent problem of finding all the minimum paths between two points on a Manhattan geometry [59]. Our idea here is to treat each search-path as an independent sub-cluster, and the aim is to annihilate any neutral sub-clusters.

Based on the above definitions, we now summarise the searching routine of an initialization level $\mathcal{L}_{r,s}$ as follows. For each plaquette $u_j \in \mathcal{U}$ (starting with u_1):

 (i) Choose an element $q_j \in \mathcal{Q}_{r,s}$, and construct a search rectangle $\mathcal{T}$;
 (ii) Search for all possible sub-clusters $\tau_j \in T$ within $\mathcal{T}$ systematically. If any sub-cluster τ_j is found to be neutral, then annihilate τ_j and stop the search. Then start step 1 with the next plaquette $u_{j+1} \in \mathcal{U}$; else
 (iii) If no neutral sub-cluster were found, choose the next element $q_{j+1} \in \mathcal{Q}_{r,s}$ and repeat steps 1 and 2; else
 (iv) If there are no remaining syndromes $q_j \in \mathcal{Q}_{r,s}$, then the search has ended without finding a neutral sub-cluster for plaquette u_j. Start step 1 with the next plaquette $u_{j+1} \in \mathcal{U}$.

The above procedure is repeated until all the plaquettes $u_j \in \mathcal{U}$ have been searched. The overhead of this search procedure is proportional size of the search rectangle $|T|$, which is factorial in r and s. More precisely, for each initialization level $\mathcal{L}_{r,s}$, in the worst case scenario (where no neutral sub-clusters are found) the search takes αL^2 steps, where the constant
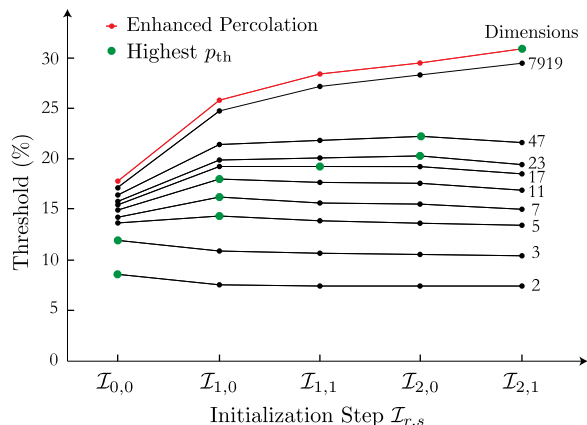
Figure 10. The thresholds for the enhanced-HDRG decoder with the first four initialization steps $\mathcal{I}_{r,s}$. The error bars and data of some prime dimensions are not included for clarity. The red curve is the enhanced-syndrome-percolation threshold in the limit of high d.

overhead $\alpha = (r+s)!/r!s!$. Although that seems to be completely inefficient (in the depth of search), the parameters r and s increase polynomially with the number of initialization levels, and hence for the first few levels the overhead α is small enough. As a result, running the above procedure for the first few initialization levels is still a computationally feasible task. It is important to notice that for each plaquette u_j the procedure stops once a neutral sub-cluster is found, and the worst case of not finding any neutral sub-clusters happens only when the dimensions d and the error rate p are sufficiently high.

The depth of searching for the neutral sub-clusters increases as the initialization levels increase in size. We propose an *enhanced*-HDRG decoder at depth (r,s) to consists of running the initialization step $\mathcal{I}_{r,s}$ followed by the HDRG decoder described in section 4.1.

The numerical estimates for the thresholds achieved by the enhanced-HDRG decoder for the first four initialization steps are summarized in figure 10. The thresholds for $\mathcal{I}_{0,0}$ corresponds to the HDRG decoder without any enhancement, with the corresponding thresholds previously presented in figure 7. For the qubit and qutrit cases we see that the thresholds decreases after the initialization steps are introduced. This is because for these low dimensions, finding a neutral sub-cluster is very probable, and hence the initialization step is in fact too destructive. As a result the clusters are divided into very sparse set of smaller clusters, and running the HDRG will end up connecting these sparse sets of clusters and causing more logical errors.

However, we start to observe improvement in the thresholds above the qutrit case. Notice that for all the listed first few primes dimensions, after some initialization step the thresholds start to decrease. This is also because after some depth of searching the initialization step becomes too destructive. In the limit of high d, we see that a threshold just under 30% can be achieved. The current shape of the curve indicate a potential increase in threshold with initialization step beyond $\mathcal{I}_{2,1}$, and we leave such investigation for a future work.

Finally, in the limit of high d, the saturating thresholds of the enhanced-HDRG decoder can also be explained by the syndrome percolation effect. We introduce the *enhanced-*syndrome-percolation threshold which is determined by simply running the initialization step

$\mathcal{I}_{r,s}$ followed by the syndrome percolation simulation described in section 4.3. The numerical estimates for the enhanced-syndrome-percolation thresholds are presented in figure 10 by the red curve. Our numerical analysis shows that the enhanced-HDRG decoder can reach the upper bound of the red line by ignoring small size effects and considering large lattice sizes only.

5. SDRG decoder

5.1. SDRG overview

In this section we study the SDRG decoder introduced by Duclos-Cianci and Poulin in [33, 34]. The SDRG decoder used here, developed independently of that used by Duclos-Cianci and Poulin in [12], differs from their approach in that we optimise the decoder for very high speed decoding at the expense of a reduced threshold. This enables us to probe thresholds up to very large d. Specifically, the decoder presented here has time complexity scaling as $O\left(d^4\right)$, which is comparably faster than the decoder presented in [12], which has time complexity $O\left(d^7\right)$. It will become clear during the presentation of the SDRG decoder from where these scalings arise. In this section we broadly review the techniques used in the SDRG decoder. Next, we introduce the specific implementation of the SDRG decoder we use. Finally, we discuss the thresholds obtained by this decoder.

It would be cumbersome to describe this decoder without employing homology terminology. In the following, for the non-expert reader, *homological equivalence* can be taken as equivalent to equivalence under multiplication by a member of the stabilizer group (or more precisely the stabilizer subgroup generated only by plaquette or vertex operators, depending on context). Two homologically equivalent objects are referred to as being *homologous*. A homology class is an equivalence class of operators equal up to a member of the vertex or plaquette stabilizer subgroup (as appropriate). We refer the reader who would like a precise definition of these terms to appendix A.

The SDRG decoder has a run-time complexity $O\left(L^2 \log L\right)$. It works by approximating the relative likelihood of different *homology classes* $\mathcal{H}$ of error configurations e with corresponding error operators $E = X\left(e\right)$, where we are using the notation

$$X\left(e\right) = \bigotimes_j X_j^{e_j}, \tag{7}$$

where $e = \left\{0,...,d-1\right\}^n$ is an *n*-dimensional vector.

We begin this section by considering first the exhaustive evaluation of the relative probabilities of different homology classes of error cofigurations, denoted $\mathbb{P}_{\mathcal{H}}$. A decoder that can evaluate this information can make the best informed decision on the most likely error configuration. We evaluate relative probabilities by summing the probabalities of all the error configurations $\mathbb{P}\left(e\right)$ for each homology class $\mathcal{H}$

$$\mathbb{P}_{\mathcal{H}} = \sum_{u \in \mathcal{H}} \mathbb{P}\left(u\right). \tag{8}$$

On the torus, we have d^2 distinct homology classes. Homology classes differ by addition of configurations of non-contractible loops, l, where, for example, we may have l such that
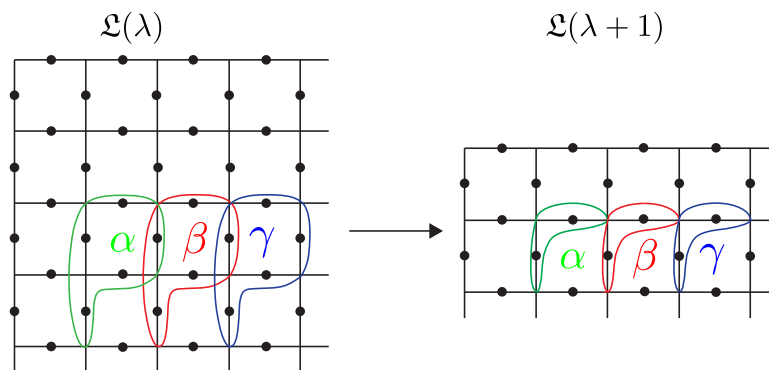
Figure 11. Three fixed, overlapping 2×1 cells, α, β and γ, of a 4×4 lattice. The cells coarse grain $\mathfrak{L}(\lambda)$ to the 4×2 lattice $\mathfrak{L}(\lambda + 1)$ in the implementation used here.

$\bar{X}_1 = X(l)$. The calculated probabilities of all homologous e for each $\mathcal{H}$ can then be used to produce a correction operator from the appropriate homology class to attempt to return the lattice to its initial state. This method of exhaustive decoding is not adopted because it is not efficient with system size. We find all the elements of a homology class by stabilizer deformations on the qudit toric code, where we have $O(L^2)$ stabilizers, we have therefore $O(d^{L^2})$ elements of every homology class. Summing over an exponential number of correction operators is clearly inefficient.

It is not necessary to consider all the error configurations within a homology class. Instead, we can consider probabilities of many 'sensible' error configurations which are likely to have occurred and still achieve respectable thresholds. The SDRG decoder uses RG methods to efficiently consider the probabilities of many sensible error configurations. It coarse grains syndromes and prior error probability distributions, or *priors*, over multiple scales using Bayesian inference methods. We label different length scales with an integer λ. The decoder coarse grains over $\sim O(\log L)$ levels, until it reaches the final coarse graining level which we label λ_0. The priors at level λ_0 correspond to approximate probabilities of the error configuration on the original lattice having come from particular homology classes.

We denote a lattice which contains both syndromes and priors at different scales by $\mathfrak{L}(\lambda)$. To efficiently coarse grain $\mathfrak{L}(\lambda)$, the SDRG dissects $\mathfrak{L}(\lambda)$ into small fixed *cells* of constant size. Each cell occupies a local connected area of $\mathfrak{L}(\lambda)$. Examples of three cells, α, β and γ are shown in green, red and blue respectively in figure 11. This cellular decomposition is then used to coarse grain $\mathfrak{L}(\lambda)$ to $\mathfrak{L}(\lambda + 1)$, shown on the right of figure 11. Syndromes of the coarse grained lattice $\mathcal{L}(\lambda + 1)$ are evaluated by summing the syndromes of each cell, and the priors of $\mathfrak{L}(\lambda + 1)$ correspond to probabilities that the syndrome of the cell is generated by an error chain from different homology classes of the cell. Each cell is decoded exhaustively. As the size of each cell is constant, and small, the time to decode a single cell is constant, and fast. The cells of $\mathfrak{L}(\lambda)$ are decoded in $O(L^2)$ time, with the capacity to be parallelized to constant time. After coarse graining to scale $\lambda_0 \sim O(\log L)$, we arrive at $\mathcal{L}(\lambda_0)$ whose syndrome is necessarily
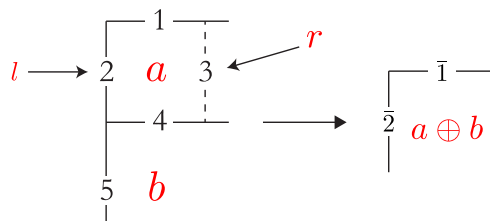
Figure 12. A five edge cell of $\mathfrak{L}(\lambda)$ which is renormalized to two edges of $\mathfrak{L}(\lambda + 1)$. The cell receives messages from its left and right neighbours at *l* and *r*.

vacuum and whose edge priors contain the probabilities that the syndromes were generated by an error configuration from different homology classes.

Coarse graining $\mathfrak{L}(\lambda)$ by exhaustively decoding individual cells will only give approximate priors for $\mathfrak{L}(\lambda + 1)$, as each cell only has access to restricted local information from the local region of the cell of $\mathfrak{L}(\lambda)$. In particular, at the boundaries where cells dissect the lattice, the approximation used is very poor. To overcome this, the SDRG decoder employs *belief propagation* to share information between neighbouring cells before renormalization takes place. The cells are chosen such that they contain overlapping edges with neighbouring cells, as in figure 11. Before the cells are renormalized, they pass *marginal messages* to other neighbouring cells. The messages take the form of a probability distribution, and describe the beliefs of a cell of what physical errors may have occurred on edges shared, given its syndrome information. In a similar spirit to exhaustively decoding each small cell, the marginal messages are also evaluated exhaustively over the cell in a constant time. Messages received from nearby cells are used to find better priors when the cells are coarse grained. In general, many messages can be shared between cells, where new messages are generated iteratively using previous messages. Multiple iterations of this step significantly enhance the performance of the SDRG decoder.

In the following sections, we describe how the renormalization step of the decoder works using messages that we assume have already been exchanged. We then explain how the messages are generated and passed, and we finally discuss the performance of this decoder.

5.2. Decoder implementation

The decoder will coarse grain the lattice $\mathfrak{L}(\lambda)$, to a lattice of fewer edges $\mathfrak{L}(\lambda + 1)$. In the decoder implementation used here, even and odd values of λ employ different shape renormalization cells. For even λ we use cells of 2×1 vertices, and for odd λ we use cells of 1×2 vertices. We describe in detail the coarse graining and belief propagation stages for a 2×1 cell as shown in figure 12, but the cell decompositions for odd or even λ are equivalent up to a reflection. We note that the cells used here are the smallest possible cells that can be used in such a decoder, which optimize the speed of the algorithm. In choosing this cell size, it is necessary to use different cell shapes at odd and even λ. A further detail of the message passing stage in the implementation used here is that cells pass messages only to left and right neighbouring cells for even λ, and to above and below neighbouring cells for odd λ. In a general implementation however, messages can be passed in all directions at all levels. Cells evaluate probabilities of their own homology classes, which become priors on the coarse grained lattice.

The decoder uses many cells at every λ. However, the action of a single cell of each $\mathfrak{L}(\lambda)$ is identical up to its input. In the following subsection we describe in detail the action of a single cell, and its two nearest neighbours, which is repeated over the entire lattice $\mathfrak{L}(\lambda)$ for all λ.

5.3. Renormalization cells

Each cell contains five edges, and two syndrome measurements, *a* and *b*, which are shown at the left of figure 12. As before, we denote operators of Pauli *X* operators with notation $X(e)$ where now error configuration *e* now only covers five edges indexed on the cell shown on the left of figure 12.

A cell will coarse grain its syndromes. For the plaquette operators we perform this coarse graining by moving syndrome *a* shown in figure 12 onto the face of syndrome *b*, such that the coarse grained syndrome will take the value $a \oplus b$. Coarse graining is achieved using the operator

$$T^a = X(at), \text{ where } t = (0, 0, 0, 1, 0). \tag{9}$$

The configuration *at* is a member of a homologous class of configurations which will have no errors on the edges of its corresponding coarse-grained cell. We change the class of the coarse-graining configurations to consider the probabilities of errors suffered on the coarse-grained edges of a cell using the logical operators

$$\overline{X}_1 = X(l_1), \text{ where } l_1 = (1, 0, 0, 1, 0), \tag{10}$$

$$\overline{X}_2 = X(l_2), \text{ where } l_2 = (0, 0, 0, 0, 1). \tag{11}$$

These configurations modify the class of a coarse-graining configuration because they represent error configurations that extend between different cells.

So far, we have specified three $X(e)$ operators (with $e = t, l_1, l_2$) in a renormalization cell, but need another two independent operators to form a complete basis for all possible errors. The remaining two operators are vertex operators truncated to the support of the cell. They are sometimes called gauge stabilizers in the literature. They are

$$S_1 = X(s_1), \text{ where } S_1 = (0, 1, 0, 1, d-1), \tag{12}$$

$$S_2 = X(s_2), \text{ where } S_2 = (0, 0, 1, d-1, 0). \tag{13}$$

Two error configurations *e* and e' are now homologous if they differ by a gauge stabilizer. Specifically, this is true if and only if there exists $\mu, \nu \in \mathbb{Z}_d$ such that $e = e' \oplus \mu s_1 \oplus \nu s_2$.

The cell only considers error configurations consistent with the syndrome. Given a particular measurement syndrome *a* we are interested in classes of homologous errors $\mathcal{H}(h_1, h_2)$ for $h_1, h_2 \in \mathbb{Z}_d$ where $\mathcal{H}(h_1, h_2)$ is the class containing elements homologous to *at* $\oplus h_1 l_1 \oplus h_2 l_2$. We shall calculate the relative likelihood of each of these classes as described in the next section.

Let us reflect for a moment on the last layer of renormalization. For $\mathfrak{L}(\lambda_0)$ we have a lattice with only two edges and a single syndrome. The syndrome has been generated by summing syndromes at different levels of renormalization in such a way that it equals the sum of syndromes over the whole lattice. Since the whole lattice is charge neutral we know that the last syndrome must be trivial, and so syndromes play no further role at this stage. Whereas the edges

and the probabilities of them carrying an error can be directly interpreted as the relative probabilities of each homology class of the original lattice at the microscopic scale, and hence we choose our recovery error from the most likely error class.

5.4. Coarse graining priors

In addition to syndrome information, each cell contains a set of prior probability distributions and messages received from cells to the left and right. Each edge, j, of $\mathfrak{L}(\lambda)$ contains a prior probability distribution $\mathbb{p}_j$ that takes as input $e_j \in \mathbb{Z}_d$ and outputs an estimated probability $\mathbb{p}_j(e_j)$ for an $X_j^{e_j}$ error. The initial lattice $\mathcal{L}(0)$ contains the original lattice syndrome and takes its priors from the error model described in section 3. Each message, $\mathbb{q}_{l,r}$, encodes beliefs calculated by neighbouring cells which share edges 2 and 3.

Based on these priors, a cell will evaluate $\mathbb{p}_{1'}$ and $\mathbb{p}_{2'}$ which are coarse grained priors to be used in $\mathfrak{L}(\lambda + 1)$. This is achieved by considering the probabilities of error configurations for different homology classes of the cell. First we find the probability of a particular error configuration

$$\mathbb{P}(e) = \mathbb{p}_1(e_1)\,\mathbb{q}_l(e_2)\,\mathbb{q}_r(e_3)\,\mathbb{p}_4(e_4)\,\mathbb{p}_5(e_5), \tag{14}$$

which evaluates the probability that an error configuration has occurred using priors $\mathbb{p}_j$ and messages $\mathbb{q}_{l,r}$.

However, we are actually interested in probabilities over a whole homology class $\mathcal{H}(h_1, h_2)$ and so

$$\mathbb{P}_{\mathcal{H}(h_1,h_2)} = \sum_{u\in\mathcal{H}(h_1,h_2)} \mathbb{P}(u). \tag{15}$$

Ideally, we would pass on all of this information to the next level of renormalization as it represents our belief of the joint probability distribution $\mathbb{p}_{1'} \times \mathbb{p}_{2'}$. However, our renormalization cells only accept input priors for individual edges, and these are given by the marginal distributions:

$$\mathbb{p}_{1'}(e_1) = \sum_{k\in\mathbb{Z}_d} \mathbb{P}_{\mathcal{H}(e_1,k)}, \tag{16}$$

$$\mathbb{p}_{2'}(e_2) = \sum_{k\in\mathbb{Z}_d} \mathbb{P}_{\mathcal{H}(k,e_2)}. \tag{17}$$

A smarter use of correlations, which we have discarded here, could lead to improved thresholds [60].

We see from equation (15) that we must sum over d^4 elements of $\mathcal{H}(h_1, h_2)$, generated by two stabilizer operators and two logical operators per cell. Similar to this, as we will see in the following section, belief propagation cells have the same time complexity. This shows explicitly the time complexity for the decoder of $O(d^4)$.

5.5. Belief propagation

To enhance the performance of the decoder, each cell is supplied with marginal messages from neighbouring cells. The messages correspond to the beliefs of a cell that *physical* errors have occurred on particular edges. The messages are calculated before each level of coarse graining. We label one cell β, and its left and right neighbours are labelled α and γ, as shown in figure 11. Each β prepares two messages which are the believed error distributions over the shared edges, 2 and 3 of figure 12, between neighbouring cells α and γ using the syndrome information of the cell. One message L is passed left to become q_r of cell α and the other, R, is passed right to become its q_l of cell γ. Keep in mind that each message is a list of d numbers, e.g. L is communicated as the vector $\left\{ L(0), L(1), ..., L(d-1) \right\}$. At the same time α and γ will respectively prepare messages q_l and q_r respectively for cell β. These messages are then exchanged for later message passing rounds or for use in coarse graining.

At the beginning of any level λ, all the messages $q_{l,r}$ are initialized to the uniform distribution. The messages are then evaluated to be the marginals over all homology classes

$$L(e_2) = \sum_{u \in \mathcal{H}} G(u) \delta_{u_2, e_2}, \tag{18}$$

$$R(e_3) = \sum_{u \in \mathcal{H}} H(u) \delta_{u_3, e_3}, \tag{19}$$

which are in terms functions G and H which return probabilities of error configurations that we define shortly. Here, $\mathcal{H}$ is the union over all $\mathcal{H}(h_1, h_2)$ and δ_{u_j, e_j} is an indicator function that equals unity when $u_j = e_j$ and zero otherwise. The presence of the indicator function ensures that we are calculating marginal probabilities. We can unpack this notation, by considering when the indicator function doesn't vanish to find the more explicit, but less compact, formulae

$$L(e_2) = \sum_{h_1, h_2, \nu} G(h_1 l_1 \oplus h_2 l_2 \oplus e_2 s_1 \oplus \nu s_2 \oplus at),$$

$$R(e_3) = \sum_{h_1, h_2, \mu} H(h_1 l_1 \oplus h_2 l_2 \oplus \mu s_1 \oplus e_3 s_2 \oplus at).$$

Conveniently, the only error configurations we consider that act on edges 2 and 3 are s_1 and s_2, respectively. This is why we are able to change $e_2(e_3)$ simply by adding the error configuration $s_1(s_2)$. Now, we reveal the functions upon which these equations depend

$$G(e) = \mathbb{p}_1(e_1) \mathbb{p}_2(e_2) q_r(e_3) \mathbb{p}_4(e_4) \mathbb{p}_5(e_5), \tag{20}$$

$$H(e) = \mathbb{p}_1(e_1) q_l(e_2) \mathbb{p}_3(e_3) \mathbb{p}_4(e_4) \mathbb{p}_5(e_5). \tag{21}$$

One may notice from G and H that the messages being passed left (right) do not evaluate new messages using messages received from the left (right). This is to avoid feedback, where messages are created using messages that have previously been sent.

It is easily seen that the computational complexity of evaluating a round of messages is the same as performing one coarse graining step. However, the improvement in threshold by applying belief propagation significantly enhances the threshold of the decoder, so it pays to spend a few rounds evaluating messages [33]. Further, short cuts can be found to evaluate future
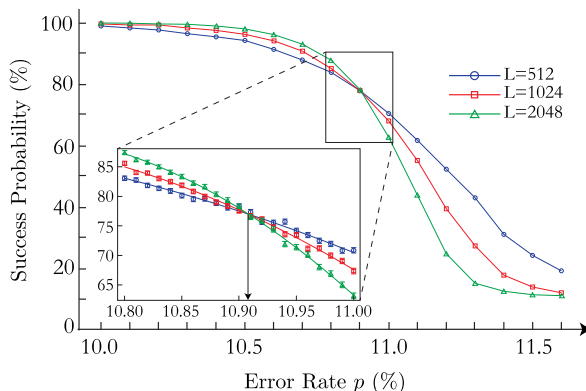
Figure 13. Threshold crossing of the SDRG decoder by plotting p_s versus p. The inset shows the data points collected close to the crossing point which were fitted to the fitting (2). Each data point is calculated using $N = 10^4$ Monte Carlo samples.

messages, after the first round of messages have been evaluated by simply performing updates on the previous messages, rather than evaluating new messages from scratch. This significantly speeds up the evaluation of messages. In the implementation of the SDRG decoder used here we use five rounds of message passing at each stage before performing a renormalization step. After a few rounds of message passing, messages tend to converge, and they are used to coarse grain the lattice in a renormalization step.

5.6. Threshold estimation and the hashing bound

In estimating the thresholds we have only considered the crossing of the three largest system sizes to reduce small system size effects, and we used the fitting in equation (2). An example is shown for the qutrit case in figure 13, where we use system sizes $L = 512$, 1024 and 2048 to find the crossing. The inset of figure 13 shows the points close to the crossing point we fit (2) to, as well as the fitting itself. We evaluate each p_{succ} using $N = 10^4$ samples. We calculate thresholds up to $d = 19$, however, due to the run time complexity the decoder shows in d, we reduce system sizes as we increase d. The $d = 19$ data point uses system sizes $L = 16$, 32 and 64 . The achieved thresholds are shown in figure 14.

The choice of small cells in the SDRG decoder means the thresholds are lower than those in [36]. However, this comes with the trade-off of impressive run times, which enables us to probe very large d. It is conjectured in [36] that the threshold will follow a constant fraction of the generalized Hashing bound with increasing d. However, we see that the obtained thresholds tend away from this limit. This can be partially explained by small system size effects, as we have to decrease the system sizes as we increase d.

6. Summary and discussion

In this paper we have introduced two efficient decoding algorithms for decoding the qudit toric code, and we have studied how their thresholds for the independent noise model vary as we change the local dimension of the qudits of the code.
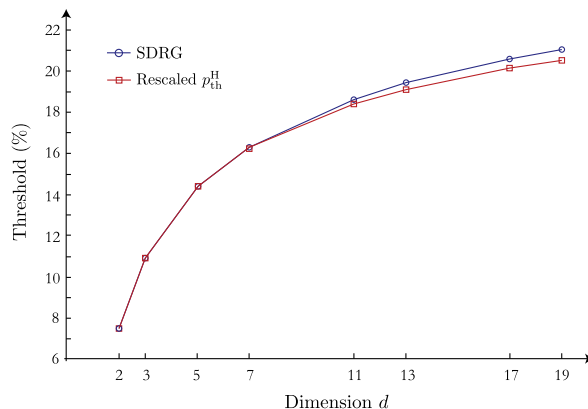
Figure 14. This graph shows thresholds for prime d for the SDRG (in blue), and a constant factor of ~0.68 of the generalized Hashing bound (in red).

We observed first of all that the HDRG decoder is restricted by the syndrome percolation threshold. For small d, the decoder is capable of exploiting the additional syndrome information provided by increasing d and the threshold increased. However, the decoder is unable to correct errors above an error rate of approximately 18%, where syndromes percolate across the lattice. We introduced an initialization step (in the enhanced-HDRG) which makes use of the charge information given by the syndrome, to annihilate small neutral sub-clusters of syndromes ameliorating the percolation effect. This enhancement enabled the HDRG decoder to achieve thresholds beyond the percolation limit.

Part of the simplicity of the HDRG algorithm is that it makes minimal utilization of charge information, in particular that the clusterization step is charge-blind. The initialization step in our enhanced algorithm incorporates more local charge information into the decoding and doing so enhances thresholds. It would be worthwhile to attempt to combine initialization and clusterization steps into a single algorithm which might combine computational efficiency with higher thresholds.

We study also the SDRG decoder which we optimized for high speed decoding. This decoder uses Bayesian inference methods to coarse grain probability distributions to efficiently find the probabilities that different errors have occurred on the lattice. This decoder considers probabilities of different error configurations at a microscopic level. This enables the decoder to overcome percolation thresholds in a natural way. Instead, we observe that this decoder maintains a threshold which is a constant factor from the optimal threshold for a non-degenerate code, which we should expect to achieve by exhaustive decoding.

While we see that the SDRG will typically outperform the HDRG decoder, we note that for low d the HDRG decoder performs comparably well to the SDRG decoder. This is remarkable given the comparative simplicity of the decoder. Moreover, we see that the enhanced-HDRG can continue to achieve thresholds that come close to matching those of the SDRG decoder.

We see a general trend of error threshold increasing with dimension d. This is in line with the conjecture that the optimal threshold should be close to or equal to the hashing bound threshold. The hashing bound threshold rises monotonically with d up to a maximal value of 50% for high d, and we see a corresponding monotonic rise in the thresholds for both decoders. Comparing the obtained thresholds with a rescaled hashing bound threshold in figure 15, we see
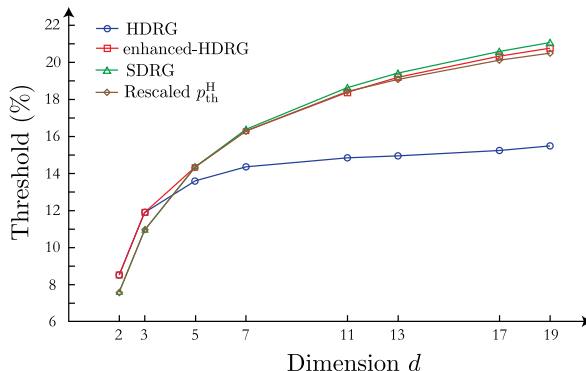
Figure 15. A comparison between the thresholds obtained using the HDRG, enhanced-HDRG and SDRG decoders presented in this paper, plotted, for comparison, against $0.69p_{th}^{H}$, the hashing bound threshold rescaled to 69% of its value. In [12] Duclos-Cianci and Poulin report that the thresholds for their SDRG decoder (with larger unit cells) are close to $0.81p_{th}^{H}$, independent of d.

further evidence of a phenomenon first reported by Duclos-Cianci and Poulin. For both decoders, the numerical threshold remains close to a constant factor (69%) of the hashing bound threshold, independent of d. If the conjecture that the hashing bound threshold approximates the optimal threshold is true, this would seem to imply that the decoder thresholds are reaching a constant fraction of the optimal threshold independent of d. We do not understand the origin of this effect, and investigating it with a wider range of noise models will be an avenue for future work.

It is pertinent to discuss some limitations of the noise model studied here. The independent noise model was chosen for its convenience and its connection to statistical mechanics models (the Potts gauge glass). However, it is not physically motivated, and certain aspects of it (equal probability of all powers of X and Z, and independence of X and Z errors) do not represent a noise model in nature. In future work, we will explore the performance of these decoders in more general noise models. In particular, for high d one would expect, for general noise channels and e.g. for the depolarizing channel, to see high correlations between X and Z noise. A decoder which took these correlations into account may therefore reach higher thresholds than one treating these as separate decoding problems. It is difficult to make a fair comparison between the noise thresholds of different d. In particular, in the independent noise model we have considered here, as d increases the total error probability is split between more and more individual noise processes. Thus the increased thresholds here must be partly attributed to this fact. Nevertheless, the increase in threshold probability for low d (e.g. from 2 to 3 or 5) is striking and coupled with the increased thresholds and yields observed in magic state distillation at these dimensions [11] may promise advantages in the implementation of quantum computation. However, to verify this promise further study is needed, in particular a full fault-tolerant analysis with a physically motivated noise model allowing fair comparison between schemes of different dimension.

The development of generalized decoding algorithms has provided analytical tools for the study of novel topological systems. For instance, recent developments in decoding algorithms [35] have given us a probe to study the properties of topological phases coupled to a thermal environment [35, 45]. In particular, the HDRG decoder developed here is used to study a 2D

topological phase with Hamiltonian defects [21]. Moreover, recent advances in decoding algorithms have demonstrated the capability to encode and read out quantum information in non-Abelian topological phases [51, 52] which shows promise towards the realization of fault-tolerant topological quantum computation. Further development using more specialized decoders may lead to more refined analysis of such fault-tolerant systems.

Our study shows that both SDRG and HDRG provide effective decoders in scenarios where the MWPMA is inappropriate. The simplicity of the HDRG, and the incorporation of a sub-lattice optimal decoder for the SDRG mean that both may be readily generalized to non-standard topological codes. The key advantage in the HDRG decoder is its light computational requirements. In scenarios where high threshold is important, however, for example, where reducing the code overhead [61, 62] is a key priority, the extra classical computational cost of a SDRG decoder, and in particular its ability to reach higher thresholds via larger cell sizes, may be a price worth paying. Overall, the diversity of efficient decoders provides a toolbox for further research into the new phenomena, new physics and potential advantages for quantum information offered by non-traditional and non-qubit topological codes.

Appendix A. Homology

A.1. Introduction

The algebra of the stabilizer group of the qudit toric code defined on the edges of a lattice is captured by *homology*. Homology is a framework for relating structures of different dimension via the concept of *cycles*, which has important applications in topology. We will not give a detailed and formal introduction to homology here, but instead introduce the key concepts needed for understanding homology in toric codes, in terminology accessible to the general physicist.

In short, homological equivalence of string-like operators supported on the edges of the toric code lattice, as used in the main text and in the literature, correspond to equivalence under multiplication by stabilizer operators—and hence two homologically equivalent operators are logically equivalent on the code-space of the toric code. While this definition will suffice for some readers, we invite those who would like a fuller introduction to homology to read on. For a formal introduction to homology as used in the topological code literature which does not take excursions into more general algebraic topology, we recommend chapter 3 of [63] or chapter 5 of [64].
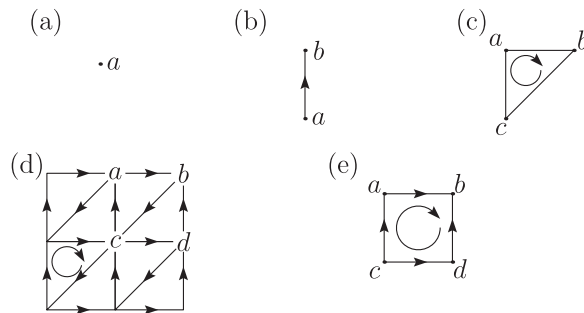
Figure A1. (a) A 0-simplex, *a*. (b) A 1-simplex. Its orientation is depicted with an arrow from vertex *a* to vertex *b*. (c) A clockwise oriented 2-simplex. (d) A triangulation of a manifold where 2-simplices have a uniform clockwise orientation. (e) A plaquette, the fundamental square object we use for manifold 'triangulation' throughout these notes.

A.2. Simplices and the triangulation of a manifold

The fundamental objects in *simplicial homology* which we describe here are *directed simplices*. A simplex is an *n*-dimensional generalization of a solid triangle. A 0-simplex is a vertex, a 1-simplex is a line and a 2-simplex is a triangle. We label vertices with letters. Vertex *a* is shown in figure A1(a). The term 'directed' means we assign orientations to the simplices. The 1-simplex shown in figure A1(b) is oriented from vertex *a* to vertex *b*, and the 2-simplex shown in figure A1(c) has a clockwise orientation from *a*, to *b*, to *c*, and back to vertex *a*.

We introduce the notation Δ_n, to denote an *n*-simplex. We extend this notation to include a direction. A 1-simplex running from point *a* to *b* shown in figure A1(b), is denoted $\Delta_1(a, b)$. The same simplex with opposite direction is written $\Delta_1(b, a)$, where we have permuted vertices *a* and *b*. The 2-simplex shown in figure A1(c) where the clockwise orientation follows the vertices in the sequence $a \rightarrow b \rightarrow c \rightarrow a$, is written $\Delta_2(a, b, c)$. We point out that the sequence of vertices $b \rightarrow c \rightarrow a \rightarrow b$ will describe the same oriented 2-simplex $\Delta_2(a, b, c)$, such that $\Delta_2(a, b, c) = \Delta_2(b, c, a)$. The orientation of a 2-simplex is changed by permuting a pair of vertices, for instance, $\Delta_2(a, c, b)$ has the opposite orientation to $\Delta_2(a, b, c)$. We should use the notation '$\Delta_0(a)$' to denote a 0-simplex, but for brevity with 0-simplices write only *a*.

Simplices can be used to describe topologically non-trivial manifolds. A manifold, such as the 2D surface of a torus, can be *triangulated*, meaning it can be divided into a set of oriented simplices. We show a triangulation of a 2D manifold in figure A1(d), where the triangulation includes all the directed 0-, 1- and 2-simplices shown in the diagram. We are free to assign all the 2-simplices of the triangulation a clockwise orientation.

A.3. Chains

Having introduced a simplicial triangulation of a manifold, it is now interesting to construct complex objects on a manifold composed of many simplices. General *n*-dimensional objects are known as *n-chains*. Such *n*-chains are linear combinations of *n*-simplices. We write an *n*-chain, *A*, as
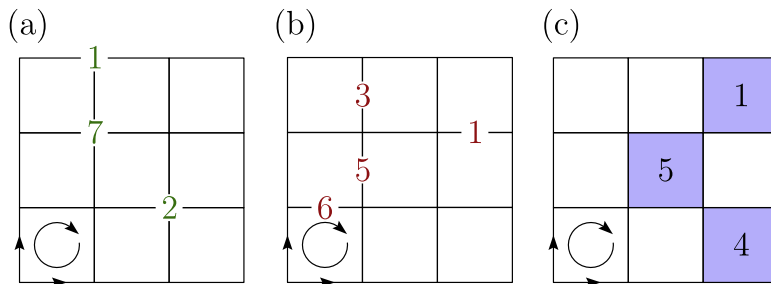
Figure A2. (a), (b) and (c) show examples of a 0-chain, a 1-chain and a 2-chain respectively on a square lattice. We orient 1- and 2-simplices uniformly. We mark the uniform orientation in the bottom-left corner of the lattice.

$$A = \sum_{\Delta} a_{\Delta} \Delta_n, \tag{A.1}$$

where we sum over all *n*-simplices of a triangulated manifold. In the present exposition we consider $a_{\Delta} \in \mathbb{Z}_d$. We are able to perform binary operations between chains. For example

$$A + B = \sum_{\Delta} (a_{\Delta} + b_{\Delta}) \Delta_n, \tag{A.2}$$

where we use $B = \sum_{\Delta} b_{\Delta} \Delta_n$. We remark that the additive inverse of a simplex is the same simplex with opposite orientation. For instance, $\Delta_1(a, b) = -\Delta_1(b, a)$, and $\Delta_2(a, b, c) = -\Delta_2(b, a, c)$.

Having introduced linear combinations of simplices, we are now able to define a *plaquette*, $\Xi(a, b, c, d)$, in terms of 2-simplices. The plaquette is the fundamental square object we use to describe the square toric code lattice, shown in figure A1(e). We consider once more the example triangulation shown in figure A1(d), we have

$$\Xi(a, b, c, d) = \Delta_2(a, b, c) + \Delta_2(c, b, d). \tag{A.3}$$

We compose an entire lattice of plaquettes. We find the plaquettes of the square decomposition by summing all the pairs of 2-simplices which share a *diagonal* bounding edge of the considered regularly triangulated manifold. In the next section we see from the example plaquette $\Xi(a, b, c, d)$ that the simplex $\Delta_1(b, c)$ is not included in its bounding set, thus eliminating the diagonal edges of figure A1(d) from the plaquette decomposition of the manifold.

It is useful to write arbitrary *n*-chains on the square lattice. We will see that such chains correspond to operators relevant to the qudit toric code. We give an example of an arbitrary 0-, 1- and 2-chain on the considered square lattice in figure A2. In these diagrams, and the diagrams we use throughout this appendix, we uniformly assign all plaquettes a clockwise orientation, and vertical(horizontal) edges are assigned an upwards(right) orientation, which we mark in the bottom left corner of each lattice diagram. The numbers then correspond to the coefficient of a given simplex for the described *n*-chain of using the defined orientations.

A.4. The boundary map

A key idea of homology is the *boundary*. A boundary of an *n*-simplex is a unique linear combination of $(n - 1)$ simplices. The boundary of $\Delta_1(a, b)$ shown in figure A1(b) contains two bounding vertices, *a* and *b*, and the boundary of a triangle, $\Delta_2(a, b, c)$, shown in figure A1(c), contains lines $\Delta_1(a, b)$, $\Delta_1(b, c)$ and $\Delta_1(c, a)$.

   To make the concept of a boundary rigorous, we define the boundary map δ_n. The boundary map is a linear map which takes an *n*-chain, *A*, and outputs an $(n - 1)$ chain which forms the boundary of *A*. We consider the examples we have introduced in this section.

   We first consider a single vertex, *a*, shown in figure A1(a). A vertex necessarily has no boundary

$$\delta_0[a] = 0. \tag{A.4}$$

The boundary map δ_1 acting on the 1-simplex $\Delta_1(a, b)$, shown in figure A1(b) returns

$$\delta_1\big[\Delta_1(a, b)\big] = b - a, \tag{A.5}$$

where the negative sign arises due to the orientation of $\Delta_1(a, b)$. The importance of signs will become clear in later sections where we consider *cycles*.

   The last boundary map relevant to us, δ_2, will output a linear combination of the edges. It is defined

$$\delta_2\big[\Delta_2(a, b, c)\big] = \Delta_1(a, b) - \Delta_1(a, c) + \Delta_1(b, c), \tag{A.6}$$

again, where it is very important to keep track of vertex order *a*, *b* and *c* to maintain consistency with the signs. One can easily check that the output $\delta_2\big[\Delta_2(a, b, c)\big]$ is independent of even (cyclic) permutations of vertices *a*, *b* and *c* using that $\Delta_1(a, b) = -\Delta_1(b, a)$.

   Finally, we consider the boundary of a plaquette (A.3), composed from 1-simplices which we denote ∂p. By linearity we have that

$$\partial p \equiv \delta_2\big[\Xi(a, b, c, d)\big] = \delta_2\big[\Delta_2(a, b, c)\big] + \delta_2\big[\Delta_2(b, d, c)\big]. \tag{A.7}$$

Then, using equation (A.6), it is easily checked that

$$\partial p = \Delta_1(a, b) + \Delta_1(c, a) - \Delta_1(d, b) - \Delta_1(c, d). \tag{A.8}$$

As the orientations of the two simplices of $\Xi(a, b, c, d)$ align, the boundary matches with our intuition and the $\Delta_1(b, c)$ does not appear in the boundary of the plaquette.

A.5. Cycles

We begin discussing cycles by considering the example of the boundary of a plaquette, calculated in equation (A.8). We calculate the boundary of this 1-chain using equation (A.5) to find

$$\delta_1[\partial p] = (b - a) - (c - a) + (d - b) - (d - c) = 0. \tag{A.9}$$

In homology any *n*-chain, *A*, such that $\delta_n[A] = 0$, is known as an *n-cycle*. Calculation (A.9) shows explicitly that the boundary of a plaquette is a 1-cycle. Indeed, one can verify in general
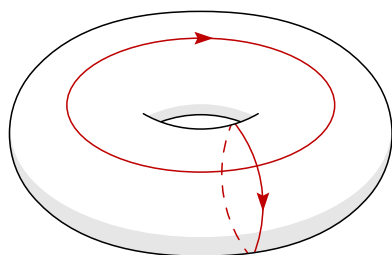
Figure A3. On a torus there are a number of ways one can construct cycles which do not enclose a boundary. Here are two examples. The cycles depicted here cannot be transformed to one another via the addition of a homologically trivial cycle.
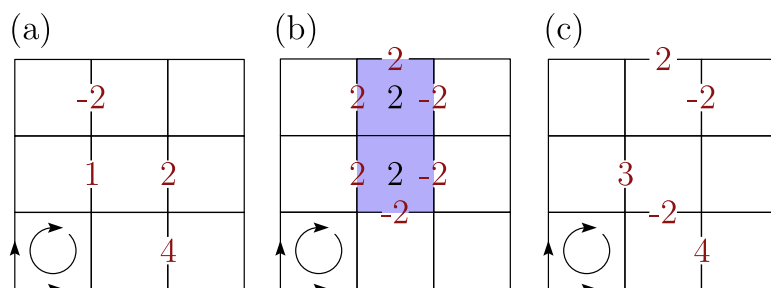

Figure A4. An example of homology. (a) Shows a 1-chain A. (b) A 1-cycle ∂H which bounds 2-chain H which is highlighted on blue plaquettes. (c) A 1-chain B homologous to A as $B = A + \partial H$.

that a boundary of any n-chain is a cycle. Written more rigorously, one can prove that $\delta_{n-1}\big[\delta_n[A]\big] = 0$ for any n-chain A.

Are all n-cycles the boundary of an $(n + 1)$ chain? The answer is no. Consider the loop indicated in figure A3. This chain has zero boundary, which can be verified algebraically. Nevertheless it encloses no 2-chain. If we try and 'fill out' the surface of the torus to enclose a chain, we will find that this covers the whole toric surface. The 2-chain covering the surface however, has no boundary. Hence this 1-cycle is not a boundary of any 2-chain.

The distinction between boundary cycles and non-boundary cycles is of central importance to homology theory (and to the toric code). Boundary cycles are known as 'homologically trivial' cycles. Otherwise, cycles are 'homologically non-trivial'.

A.6. Homological equivalence

The group of boundary cycles is used to define another central concept of homology theory, *homological equivalence*. We say two n-chains are homologically equivalent, or *homologous*, if they are equivalent up to addition of a boundary n-cycle. We provide an explicit example of two homologous 1-chains. We show in figures A4(a) and (c) the 1-chains A and B respectively. It's easily seen that the B differs from A by only a boundary, ∂H, shown in figure A4(b). The 1-cycle $\partial H = \delta_2[H]$ is clearly a boundary of the 2-chain h highlighted in blue on figure A4(b). We denote that two n-chains are homologous by the symbol '$\sim$', such that $A \sim B$. It is from this
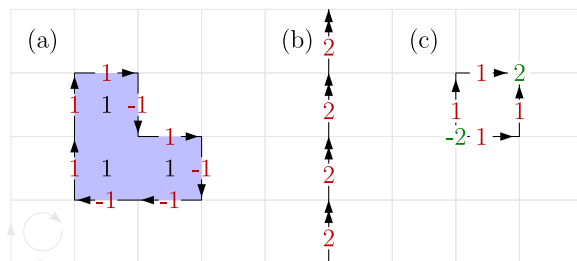
Figure A5. (a) A homologically trivial 1-cycle which is the boundary of the 2-chain highlighted in blue. (b) A homologically non-trivial 1-cycle *C*. (c) A 1-chain whose boundary is non-zero.

insight that we see why boundary *n*-cycles are known as 'homologically trivial'. All *n*-boundaries are homologous to the trivial *n*-chain, $A = 0$. It is in this sense that homologically trivial cycles are *contractible*, i.e. can be contracted to a point or the trivial *n*-chain. Non-trivial cycles such as those shown in red in figure A3 do not share this property. Indeed, these non-trivial cycles are known as *non-contractible*.

Before we move onto the final section of this appendix we remark that the group of non-trivial cycles of a triangulation of a manifold is known as the *first homology group*, and is a topological invariant used to classify manifolds.

A.7. Homology and the toric code

Here we arrive at the main section of the appendix where we show that operators acting on the code-space of the qudit toric code are elegantly characterized using concepts from homology.

We make use of the notation introduced in the main text to identify 1-chains $A = \sum_\Delta a_\Delta \Delta$ with Pauli operators $Z(A) = \otimes_\Delta Z_\Delta^{a_\Delta}$. The subscript Δ now indexes qudits lying on the edges of the toric code lattice.

We first consider the plaquette operators of the toric code. It is easily checked that plaquette stabilizers simply correspond to the boundary cycle of a plaquette, such that $B_p = Z(\partial p)$, where $\partial p = \delta_1 \left[\Xi(a, b, c, d) \right]$. In fact, it is easily checked that any operator of the form $Z(\partial A)$ where $\partial A = \delta_2(A)$ for any 2-chain A will act trivially on the code-space of the toric code. We show an example of such a boundary cycle in figure A5(a).

We consider next logical $\bar{Z}$ operators of the qudit toric code. These are easily identified with homologically non-trivial 1-cycles, such as *C*, shown in figure A5(b). A sensible encoding of the toric code might be chosen such that $\bar{Z}^2 = Z(C)$. All operators $Z(C')$ where $C' \sim C$ will act equivalently on the code space of the qudit toric code to the operator $Z(C)$.

We have seen in this subsection that the code-space of the toric code is acted on by operators of form $Z(C)$, where trivial cycles *C* act trivially on the code-space and non-trivial cycles *C* perform logical operations on the code-space. In fact, the vertex operators are prescribed such that the syndromes of operators $Z(C)$ for 1-chains *C* which are not cycles will introduce syndromes equal to the boundary of the 1-chain, $\delta_1[C]$. We see an example of a 1-chain with its corresponding boundary written in green in figure A5(c). Once more it is easily checked that chains homologous to *C* will generate the same syndrome, by simple calculation,

Table A1. A table showing the relationship between *X*- and *Z*-type operators on the qudit toric code and their respective chain in homology theory.

Toric Code Property	Lattice	Homological Description
Plaquette (*Z*) stabilizer subgroup	Primal	Set of homologically trivial 1-cycles
Vertex (*X*) stabilizer subgroup	Dual	Set of homologically trivial 1-cycles
Z^k error configuration	Primal	1-chain
Vertex syndrome configuration	Primal	Boundary of *Z*-error 1-chain
X^k error configuration	Dual	1-chain
Plaquette syndrome configuration	Dual	Boundary of *X*-error 1-chain
$\bar{Z}^k$ logical operator	Primal	Homologically non-trivial 1-cycle
$\bar{X}^k$ logical operator	Dual	Homologically non-trivial 1-cycle

we find the boundary of $C' = C + \partial A$ where ∂A is the boundary of a 2-chain:

$$\delta_0 \left[C' \right] = \delta_0 \left[C + \partial A \right] = \delta_0 \left[C \right] + \delta_0 \left[\partial A \right] = \delta_0 \left[C \right]. \tag{A.10}$$

Finally, we remark that all the homological properties of vertex stabilizers, logical $\bar{X}$ operators and *X*-type error chains are the same if we move to a dual lattice. On the dual lattice, the roles of plaquettes and vertex operators are interchanged, the same homology mapping captures the relationship between *X*-errors, logical $\bar{X}$ operators, plaquette syndromes. We summarise the correspondences between the toric code properties and homology concepts in table A1.

Appendix B. Hashing bound threshold

The hashing bound is an important quantity from quantum Shannon theory [65]. It is often described in relation to the capacity of a communication channel. For instance, consider the Pauli noise channel: a channel with Kraus operators $\sqrt{p_j}\sigma_j$ where σ_j are the (qubit or qudit) Pauli operators (including the identity) and $\sum_j p_j = 1$. Then the hashing bound represents a lower bound on the capacity of this channel [28, 65]. This bound is given by the rate *R* achievable by using a random coding protocol, given by

$$R = 1 - H\left(p_j\right), \tag{B.1}$$

where *H* is the base-2 entropy defined as

$$H\left(p\right) = -\sum_j p_j \log_2\left(p_j\right). \tag{B.2}$$

We shall call the values of p_j at which *R* reaches zero the *hashing bound threshold*, denoted here as p_{th}^{H}. Note that some authors call the hashing bound threshold simply the hashing bound.

For one parameter noise families, the hashing bound threshold is given by a single value of that parameter. For example, for the qubit independent noise model, where *X* and *Z* errors occur independently with probability *p*, i.e. $p_x = p_z = p\left(1 - p\right)$, $p_y = p^2$ and $p_1 = \left(1 - p\right)^2$, the
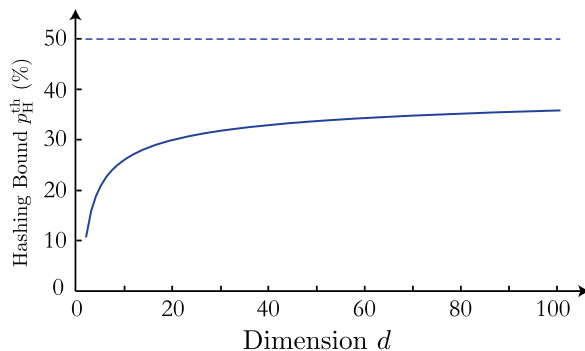
Figure B1. The hashing bound threshold for the independent error model (defined in the main text) for parameter *p* as a function of qudit dimension *d*.

hashing bound threshold is $p_{th}^H = 0.110028\%$ (to 6 d.p.). The closeness between this value and the optimal threshold for the qubit toric code under the independent noise model was noted by Dennis *et al* [22]. The same correspondence between the hashing bound and the optimal threshold has been established for surface codes with different lattice structures [67, 68].

Dennis *et al* also showed that the optimal decoder for the qubit toric code can be mapped to the random-bond Ising model (RBIM) with the optimal threshold corresponding to a phase transition point known as the Nishimori point. The generalization of this mapping to the qudit toric code of their argument is straightforward and leads to a model known as the Potts gauge glass (PGG).

Further work by Nishimori and collaborators [29, 66] implied that the similarity between the optimal and the hashing bound thresholds applies to more general statistical–mechanical models. In particular they showed that the Nishimori point for the RBIM and PGG could be estimated via a duality argument. The value of the Nishimori point (and thus the optimal decoder threshold) they derive is identical to the hashing bound threshold for the independent noise model.

A similar close relationship between the hashing bound threshold and optimal threshold for different noise models has also been observed. For example, the optimal threshold of the qubit toric code for depolarizing noise is estimated to be $p_{th}^{opt} = 18.9\%$ [30], and this, again, is very close to the hashing bound for that noise model $p_{th}^H = 18.93\%$.

Given the evidence that the hashing bound threshold is close to the optimal decoder threshold for qudit codes under the independent noise model, it represents a natural point of comparison for the thresholds in our study. We plot in figure B1 the hashing bound thresholds for this error model as a function of dimension *d*. In the limit of $d \to \infty$ the hashing bound threshold $p_{th}^H \to 50\%$.

References

[1] Shor P W 1995 *Phys. Rev.* A **52** R2493
[2] Calderbank A R and Shor P W 1996 *Phys. Rev.* A **54** 1098
[3] Steane A 1996 *Proc. R. Soc.* A **452** 2551

[4] Kitaev A Y 2003 *Ann. Phys. (Amsterdam)* **303** 2
[5] Bravyi S and Kitaev A 2005 *Phys. Rev.* A **71** 022316
[6] Raussendorf R, Harrington J and Goyal K 2007 *New J. Phys.* **9** 199
[7] Fowler A G and Goyal K 2009 *Quantum Inf. Comput.* **9** 0721
[8] Raussendorf R and Harrington J 2007 *Phys. Rev. Lett.* **98** 190504
[9] Wang D S, Fowler A G and Hollenberg L C L 2011 *Phys. Rev.* A **83** 020302
[10] Anwar H, Campbell E T and Browne D E 2012 *New J. Phys.* **14** 063006
[11] Campbell E T, Anwar H and Browne D E 2012 *Phys. Rev.* X **2** 041021
[12] Duclos-Cianci G and Poulin D 2013 *Phys. Rev.* A **87** 062338
[13] Bravyi S B and Kitaev A Y 1998 arXiv:quant-ph/9811052
[14] Freedman M H and Meyer D A 2011 *Found. Compt. Math.* **1** 325
[15] Bullock S S and Brennen G K 2007 *J. Phys. A: Math. Theor.* **40** 3481
[16] Andriyanova I, Maurice D and Tillich J-P 2012 arXiv:1202.3338
[17] Viyuela Q, Rivas A and Martin-Delgado M A 2012 *New J. Phys.* **14** 033044
[18] Schulz M D, Dusuel S, Orús R, Vidal J and Schmidt K P 2012 *New J. Phys.* **14** 025005
[19] Beigi S, Shor P and Whalen D 2011 *Commun. Math. Phys.* **306** 663
[20] Kitaev A and Kong L 2012 *Commun. Math. Phys.* **313** 351
[21] Brown B J, Al Shimary A and Pachos J K 2014 *Phys. Rev. Lett.* **112** 120503
[22] Dennis E, Kitaev A, Landahl A and Preskill J 2002 *J. Math. Phys.* **43** 4452
[23] Nishimori H 2001 *Statistical Physics of Spin Glasses and Information Processing: An Introduction* (Oxford: Oxford University Press)
[24] Honecker A, Picco M and Pujol P 2001 *Phys. Rev. Lett.* **87** 047201
[25] Merz F and Chalker J T 2002 *Phys. Rev.* B **65** 054425
[26] Ohzeki M 2009 *Phys. Rev.* E **79** 021129
[27] de Queiroz S L A 2009 *Phys. Rev.* B **79** 174408
[28] Bennett C H, DiVincenzo D P, Smolin J A and Wootters W K 1996 *Phys. Rev.* A **54** 3824
[29] Takeda K and Nishimori H 2005 *J. Phys. Soc. Japan* **74** 115
[30] Bombin H, Andrist R S, Ohzeki M, Katzgraber H G and Martin-Delgado M A 2012 *Phys. Rev.* X **2** 021004
[31] Edmonds J 1965 *Can. J. Math.* **17** 449
[32] Wang C, Harrington J and Preskill J 2003 *Ann. Phys.* **303** 31
[33] Duclos-Cianci G and Poulin D 2010 *Phys. Rev. Lett.* **104** 050504
[34] Duclos-Cianci G and Poulin D 2010 *IEEE ITW* **1** 1–5
[35] Bravyi S and Haah J 2011 arXiv:1112.3252
[36] Duclos-Cianci G and Poulin D 2014 *Quantum Inf. Comput.* **14** 0721–40
[37] Sarvepalli P and Raussendorf R 2012 *Phys. Rev.* A **85** 022317
[38] Bombin H, Duclos-Cianci G and Poulin D 2012 *New J. Phys.* **14** 073048
[39] Gottesman D 1997 *PhD Thesis* arXiv:quant-ph/9705052v1
[40] Nielsen M A and Chuang I L 2000 *Quantum Computation and Quantum Information* (Cambridge: Cambridge University Press)
[41] Gottesman D 1999 *Chaos Solitons Fractals* **10** 1749
[42] Kolmogorov V 2009 *Math. Prog. Comput.* **1** 43
[43] Karp R M 1972 *Complexity Computer Computations* (New York: Plenum Press) p 85
[44] Haah J 2011 *Phys. Rev.* A **83** 042330
[45] Bravyi S and Haah J 2013 *Phys. Rev. Lett.* **111** 200501
[46] Harrington J W 2004 *PhD Thesis* CaltechTHESIS
[47] Dennis E 2005 *PhD Thesis* arXiv:quant-ph/0503169
[48] Moreira J C and Farrell P G 2006 *Essentials of Error-Control Coding* (New York: Wiley)
[49] Proakis J and Salehi M 2000 *Digital Communications* (New York: McGraw-Hill)
[50] Wootton J R 2013 arXiv:1310.2393

[51] Brell C G, Burton S, Dauphinais G, Flammia S T and Poulin D 2013 arXiv:1311.0019
[52] Wootton J R, Burri J, Iblisdir S and Loss D 2014 *Phys. Rev.* X **4** 011051
[53] Stace T M and Barrett S D 2010 *Phys. Rev.* A **81** 022317
[54] Broadbent S R and Hammersley J M 1957 *Math. Proc. Camb. Phil. Soc.* **53** 629
[55] Grimmett G 1989 *Percolation* (Berlin: Springer)
[56] Stauffer D and Aharony A 1994 *Introduction To Percolation Theory* (Boca Raton: CRC Press)
[57] Malarz K and Galam S 2005 *Phys. Rev.* E **71** 016125
[58] Majewski M and Malarz K 2007 *Acta Phys. Pol.* **B38** 2191
[59] Gardner M 1997 *The Last Recreations* (New York: Springer) p 162
[60] Duclos-Cianci G 2013 private communication
[61] Bravyi S and Vargo A 2013 *Phys. Rev.* A **88** 062308
[62] Watson F H E and Barrett S D 2013 arXiv:1312.5213
[63] Nakahara M 2003 *Geometry, Topology and Physics* (London: Taylor and Francis)
[64] Henle M 1994 *A Combinatorial Introduction to Topology* (New York: Dover)
[65] Wilde M M 2013 *Quantum Information Theory* (Cambridge: Cambridge University Press)
[66] Nishimori H and Nemoto K 2002 *J. Phys. Soc. Japan* **71** 1198
[67] Röthlisberger B, Wootton J R, Heath R M, Pachos J K and Loss D 2012 *Phys. Rev.* A **85** 022313
[68] Keisuke F and Yuuki T 2012 *Phys. Rev.* A **86** 020303(R)