Optimal quantum subsystem codes in 2-dimensions

Theodore J. Yoder
IBM T.J. Watson Research Center

Given any two classical codes with parameters $[n_1, k, d_1]$ and $[n_2, k, d_2]$, we show how to construct a quantum subsystem code in 2-dimensions with parameters $[\![N, K, D]\!]$ satisfying $N \leq 2n_1n_2$, $K = k$, and $D = \min(d_1, d_2)$. These quantum codes are in the class of generalized Bacon-Shor codes introduced by Bravyi. We note that constructions of good classical codes can be used to construct quantum codes that saturate Bravyi's bound $KD = O(N)$ on the code parameters of 2-dimensional subsystem codes. One of these good constructions uses classical expander codes. This construction has the additional advantage of a linear time quantum decoder based on the classical Sipser-Spielman flip decoder. Finally, while the subsystem codes we create do not have asymptotic thresholds, we show how they can be gauge-fixed to certain hypergraph product codes that do.

I. INTRODUCTION

One of the perhaps more surprising facts to come out of quantum information theory is the close relation between classical and quantum error-correcting codes. Exemplary of this relation is the Calderbank-Shor-Steane (CSS) construction [1, 2], which maps two classical codes (the first's dual contained in the second) to a quantum code. Important concepts in classical coding have analogous quantum concepts. For instance, a good family of classical $[n, k, d]$ or quantum $[\![n, k, d]\!]$ codes is one that asymptotically achieves constant rate k/n and constant relative distance d/n. Using the CSS construction, one can draw on what is known classically to prove the existence of asymptotically good families of quantum codes [1] and even construct them [3, 4].

Because the classical codes input to the CSS construction must be related, it is sometimes difficult to use the CSS construction directly to make quantum codes with desirable properties. For example, the low-density parity check (LDPC) property, which can be defined for classical [5] or quantum [6, 7] codes alike, demands that every parity or stabilizer check involves a constant number of bits or qubits and every bit or qubit is involved in a constant number of checks. It is pointed out in [6] that one needs to use bad (i.e. not good) classical LDPC codes to make quantum LDPC codes via the CSS construction, and that bad classical LDPC codes are uncommon, both because they are not worth studying if one is solely motivated by classical applications, but also because, asymptotically, most classical LDPC codes are actually good.

To easily create LDPC quantum codes, another method of converting classical codes to quantum ones has been developed. The hypergraph product [7] converts *any* two classical codes to a quantum code. Notably, if the constituent classical codes are LDPC, so is the quantum code. The popular surface code is a special case, the hypergraph product of two classical repetition codes.

Yet, due to anticipated hardware limitations, it is common to place even more practical constraints on quantum codes beyond the LDPC condition. A popular demand is that parity checks are geometrically local in 2-dimensions so that it is unnecessary to interact qubits that are physically far apart in the plane. Bounds are known on the parameters $[\![N, K, D]\!]$ of 2-dimensional quantum codes of stabilizer subspace [8] and subsystem [9] varieties. The subspace bound $KD^2 = O(N)$ is saturated constructively by the surface code [10, 11] and its relatives. The subsystem bound $KD = O(N)$ is known to be tight [9], but explicit constructions have heretofore been lacking.

Here, we establish another relation between classical and quantum codes. We show how to create an $[\![N, K, D]\!]$ quantum subsystem code that is local in 2-dimensions from any two classical codes with parameters $[n_1, k, d_1]$ and $[n_2, k, d_2]$ and prove that $N \leq 2n_1n_2$, $K = k$, and $D = \min(d_1, d_2)$. The quantum code belongs to the class of generalized Bacon-Shor codes introduced by Bravyi [9], a class we therefore refer to simply as Bravyi-Bacon-Shor codes. One can recover the traditional Bacon-Shor code [12, 13] from our construction by starting with two classical repetition codes.

Bravyi-Bacon-Shor codes created this way have two important properties related to the constituent classical codes. First, if the classical codes are good, then the Bravyi-Bacon-Shor codes saturate the 2-dimensional subsystem code bound $KD = O(N)$. Second, decoders for the classical codes can be used to decode the quantum code. If the classical codes are LDPC and the decoders are linear time (in the size of the classical code), then the quantum decoding, including both data and measurement errors, is also linear time (in the size of the quantum code). Handling measurement errors in the quantum setting requires that the classical decoders handle errors in calculations of the parity checks. Though this is not a standard model in classical error-correction, the Sipser-Spielman flip decoder for expander codes [14] does apply to this situation [15].

Finally, we show how to gauge-fix Bravyi-Bacon-Shor codes. This is the process of moving encoded data from a quantum subsystem code into a related subspace code. For instance, the Bacon-Shor code can be gauge-fixed to the surface code. Thus, as a generalization of Bacon-Shor codes, Bravyi-Bacon-Shor codes should gauge-fix to a generalization of the surface code. This is indeed the case. We show that a Bravyi-Bacon-Shor code can be

gauge-fixed into certain hypergraph product codes – the hypergraph product of a classical repetition code and (either) one of the classical codes used to build the Bravyi-Bacon-Shor code. This reveals Bravyi-Bacon-Shor codes as a sort of *subsystem* hypergraph product code.

In Section II we review the codes we will be discussing and establish notation. In Section III, we provide our construction of Bravyi-Bacon-Shor codes from classical codes and show how to decode them. In Section IV, we gauge-fix Bravyi-Bacon-Shor codes to hypergraph product codes. Section V concludes.

II. CODE BACKGROUND

In this section, we review the codes that play a major role in the paper. These are (a) classical codes, including transpose and LDPC codes, (b,c) two versions of Bravyi-Bacon-Shor codes, and (d) hypergraph product codes.

A. Classical codes and their transposes

In this paper, we use "classical code" to mean a classical linear code. A linear code $\mathcal{C}$ is a subset of the set of length-n bit strings $\mathcal{C} \subseteq \mathbb{F}_2^n$ and can be defined by a parity check matrix $H \in \mathbb{F}_2^{m\times n}$ by setting $\mathcal{C} = \ker(H)$. This means that $w \in \mathcal{C}$ if and only if $Hw = 0$.

The number of encoded bits $k = \dim \mathcal{C}$ is related to the rank of H by the rank-nullity theorem

$$k = n - \text{rank}(H). \tag{1}$$

Gaussian elimination can be used to find a basis for the kernel of H. This basis can be arranged as the rows of a generating matrix $G \in \mathbb{F}_2^{k\times n}$ satisfying $\text{rank}(G) = k$ and $HG^T = 0$. Of course, any $G' = QG$ for full-rank matrix $Q \in \mathbb{F}_2^{k\times k}$ is an equally valid generating matrix.

The distance of the code is the minimum (Hamming) weight of a nonzero vector in $\mathcal{C}$. That is,

$$d = \min\{|\vec{w}| > 0 : \vec{w} \in \mathcal{C}\}. \tag{2}$$

Code parameters of $\mathcal{C}$ are collected in the tuple notation $[n, k, d]$.

Although not part of traditional classical coding theory, the "transpose" of a classical code will be important for defining hypergraph product codes in Section II D. The transpose of $\mathcal{C}$ is another linear code $\mathcal{C}^T$ that has H^T as its parity check matrix. Let us say that $H \in \mathbb{F}_2^{n^T\times n}$, where T modifying a scalar (like n) is to be treated as a superscript (not the transpose). Thus, $\mathcal{C}^T$ is another linear code with parameters $[n^T, k^T, d^T]$. Codewords in $\mathcal{C}^T$ represent redundancy (linear dependencies) between parity checks, the rows of H. Indeed, by the rank-nullity theorem and the fact that the column rank and row rank of a matrix are equal,

$$n - k = n^T - k^T. \tag{3}$$

If H were full rank (i.e. no check redundancy), $n^T = n-k$ and so $k^T = 0$.

The $[n, 1, n]$ repetition code $\mathcal{C}_R$ will be used at several points in this paper. Its parity check matrix (without redundancy) and its generating matrix can be written as

$$H_R = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 & 0 \\ & \ddots & & \ddots & & \ddots & \\ 0 & 0 & 0 & \dots & 0 & 1 & 1 \end{pmatrix} \in \mathbb{F}_2^{(n-1)\times n}, \tag{4}$$

$$G_R = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix} \in \mathbb{F}_2^{1\times n}. \tag{5}$$

When we use the repetition code, its length n will be context-appropriate (e.g. so that matrix multiplications can work).

Finally, let us briefly define classical LDPC codes.

Definition 1 (classical LDPC [5])**.** A classical code $\mathcal{C}$ is (b, c)-LDPC if there is a matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ such that $\ker(H) = \mathcal{C}$, every column contains at most b 1s, and every row contains at most c 1s. We call H an LDPC set of parity checks.

For example, the repetition code is $(2, 2)$-LDPC with H_R being an LDPC set of parity checks for the code.

B. Bravyi-Bacon-Shor codes

Bravyi-Bacon-Shor (BBS) codes are defined entirely by a binary matrix $A \in \mathbb{F}_2^{n_1\times n_2}$ and consist of qubits placed on sites (i, j) of a $n_1 \times n_2$ square lattice L for which $A_{ij} = 1$. If $|A|$ is the number of 1s in A, there are $N = |A|$ qubits in the code. Let us take a moment to establish notation for Pauli operators on this lattice.

A Pauli X or Z acting on the qubit at site (i, j) in the lattice is written X_{ij} or Z_{ij}. A Pauli operator acting on multiple qubits is specified by its support.

$$\text{For } S \in \mathbb{F}_2^{n_1\times n_2}, \quad X(S) = \prod_{ij} (X_{ij})^{S_{ij}}. \tag{6}$$

Of course, S should be such that $S_{ij} = 1$ implies $A_{ij} = 1$, because qubits only exist at those sites. We say $S \subseteq A$ if this is true. We also use the notation $S \cap A$ to indicate the pointwise product of binary matrices S and A: $(S \cap A)_{ij} = S_{ij}A_{ij}$ for all i, j. It is always the case that $S \cap A \subseteq A$.

Conveniently, multiplication and commutation of Paulis are equivalent to addition and inner products of the support matrices,

$$X(S_1)X(S_2) = X(S_1 + S_2), \tag{7}$$

$$[X(S_1), Z(S_2)] = (-1)^{\text{tr}\left(S_1^T S_2\right)} I \tag{8}$$

where $[P, Q] = PQP^\dagger Q^\dagger$ is the group commutator and I the identity operator.

From A we can also define two classical codes corresponding to its column-space and row-space:

$$\mathcal{C}_1 = \mathrm{col}(A), \tag{9}$$
$$\mathcal{C}_2 = \mathrm{row}(A). \tag{10}$$

These accordingly have generating matrices G_1 and G_2, parity check matrices H_1 and H_2, and code parameters $[n_1, k, d_1]$ and $[n_2, k, d_2]$. Both $\mathcal{C}_1$ and $\mathcal{C}_2$ encode the same number of bits $k = \mathrm{rank}(A) = \mathrm{rank}(G_1) = \mathrm{rank}(G_2)$ because of the well-known equivalence of matrix row and column rank.

BBS codes are subsystem codes and, as such, are described by a gauge group of Pauli operators. This gauge group can be divided into X-type operators and Z-type ones, and so in this sense BBS codes are CSS subsystem codes. The gauge group is generated by XX interactions between any two qubits sharing a column of lattice L and ZZ interactions between any two qubits sharing a row. We can write the entire gauge groups of X- and Z-type like

$$\mathcal{G}_X^{(\mathrm{bbs})} = \{X(S) : G_R S = 0, S \subseteq A\}, \tag{11}$$
$$\mathcal{G}_Z^{(\mathrm{bbs})} = \left\{Z(S) : S G_R^T = 0, S \subseteq A\right\}, \tag{12}$$

recalling that $G_R = (1, 1, \ldots, 1)$ is the generating matrix of the repetition code. Therefore, $G_R S = 0$ implies that columns of S have even weight and $S G_R^T = 0$ implies its rows have even weight.

Bare logical operators of a subsystem code commute with all its gauge operators. In the case of BBS codes, to commute with all Z-type gauge operators, a bare logical X-type operator must be supported on entire rows of the lattice. Likewise, to commute with all X-type gauge operators, a bare logical Z-type operator must be supported on entire columns. Therefore,

$$\mathcal{L}_X^{(\mathrm{bbs})} = \left\{X(S \cap A) : S H_R^T = 0\right\}, \tag{13}$$
$$\mathcal{L}_Z^{(\mathrm{bbs})} = \{Z(S \cap A) : H_R S = 0\}. \tag{14}$$

An example BBS code is shown in Fig. 1 with the gauge operators highlighted in part (a) and the logical operators in part (b).

When performing error-correction with a subsystem code, a complete generating set of gauge operators is measured. However, since not all gauge operators commute, the only reliable information gathered from this process is the eigenvalues of the stabilizers, the elements of the gauge group that do in fact commute with all gauge operators. In other words, the stabilizer is the intersection of the group of bare logical operators with the gauge group.

$$\mathcal{S}_X^{(\mathrm{bbs})} = \mathcal{L}_X^{(\mathrm{bbs})} \cap \mathcal{G}_X^{(\mathrm{bbs})} \tag{15}$$
$$= \left\{X(S \cap A) : S H_R^T = 0, G_1 S = 0\right\}, \tag{16}$$
$$\mathcal{S}_Z^{(\mathrm{bbs})} = \mathcal{L}_Z^{(\mathrm{bbs})} \cap \mathcal{G}_Z^{(\mathrm{bbs})} \tag{17}$$
$$= \left\{Z(S \cap A) : H_R S = 0, S G_2^T = 0\right\}. \tag{18}$$

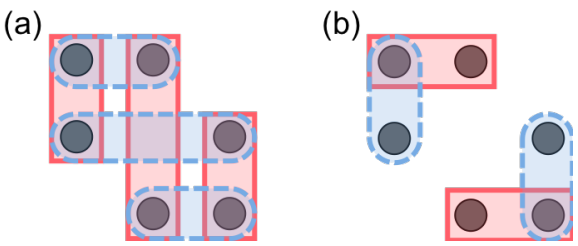

FIG. 1. A $[\![6, 2, 2]\!]$ Bravyi-Bacon-Shor code corresponding to $A = \left(\begin{smallmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right)$ [9]. In (a), we encircle the supports of X-type (red, square, solid) and Z-type (blue, rounded, dashed) gauge operators. In (b), we show the supports of X- and Z-type logical operators for the two encoded qubits. We do not show them, but there are just two stabilizers, $X^{\otimes 6}$ and $Z^{\otimes 6}$.

Here $G_1 S = 0$ demands that each column of S is a parity check of code $\mathcal{C}_1$ and thus intersects columns of A, which are codewords of $\mathcal{C}_1$, at an even number of places. Thus, $S \cap A$ has an even number of 1s in each column and this implies $X(S \cap A)$ is in $\mathcal{G}_X^{(\mathrm{bbs})}$. Similar reasoning holds for the Z-type stabilizers.

The number of encoded qubits K can be determined by counting the number of bare logical operators that are inequivalent under multiplication by stabilizers. That is, we would like the size of the quotient group $\mathcal{L}_X^{(\mathrm{bbs})}/\mathcal{S}_X^{(\mathrm{bbs})}$.

$$|\mathcal{L}_X^{(\mathrm{bbs})}/\mathcal{S}_X^{(\mathrm{bbs})}| = \frac{|\mathcal{L}_X^{(\mathrm{bbs})}|}{|\mathcal{S}_X^{(\mathrm{bbs})}|} = \frac{2^{n_1}}{|\ker(G_1)|} = 2^k. \tag{19}$$

Likewise, $|\mathcal{L}_Z^{(\mathrm{bbs})}/\mathcal{S}_Z^{(\mathrm{bbs})}| = 2^k$. This implies $K = k = \mathrm{rank}(A)$ encoded qubits.

Dressed logical operators are bare logical operators multiplied by any number of gauge operators.

$$\hat{\mathcal{L}}_X^{(\mathrm{bbs})} = \mathcal{G}_X^{(\mathrm{bbs})} \mathcal{L}_X^{(\mathrm{bbs})}, \tag{20}$$
$$\hat{\mathcal{L}}_Z^{(\mathrm{bbs})} = \mathcal{G}_Z^{(\mathrm{bbs})} \mathcal{L}_Z^{(\mathrm{bbs})}. \tag{21}$$

Equivalently, dressed logical operators are exactly those Pauli operators that commute with all stabilizers. The distance D of the BBS code is the minimum nonzero weight of a dressed logical operator.

To calculate D, imagine first taking $X(S \cap A) \in \mathcal{L}_X^{(\mathrm{bbs})}$ and reducing its weight by multiplying by gauge operators from $\mathcal{G}_X^{(\mathrm{bbs})}$, which are two-qubit X operators within columns. Clearly then, each column of $S \cap A$ can at best be reduced to contain either zero or one 1 depending on the parity of the number of 1s in that column. We calculate the parity of a column by taking its dot product with $G_R = (1, 1, \ldots, 1)$. Thus,

$$\min_{g \in \mathcal{G}_X^{(\mathrm{bbs})}} |gX(S \cap A)| = |G_R(S \cap A)|. \tag{22}$$

Note that $S H_R^T = 0$ if and only if rows of S are codewords of the classical repetition code, i.e. all 1s or all 0s.

Accordingly, for some $\vec{r} \in \mathbb{F}_2^{n_1}$, $S \cap A = \text{diag}(\vec{r})A$, where $\text{diag}(\vec{r})$ is the square, diagonal matrix with $\vec{r}$ along the diagonal. Thus, $|G_R(S \cap A)| = |\vec{r}A|$, and

$$\begin{aligned} D_X &= \min\left\{|q| > 0 : q \in \hat{\mathcal{L}}_X^{(\text{bbs})}\right\} && (23)\\ &= \min\{|\vec{r}A| > 0 : \vec{r} \in \mathbb{F}_2^{n_1}\} && (24)\\ &= \min\{|\vec{x}| > 0 : \vec{x} \in \text{row}(A)\} && (25)\\ &= d_2, && (26)\end{aligned}$$

by definition of the code distance of $\mathcal{C}_2 = \text{row}(A)$. Likewise,

$$\begin{aligned} D_Z &= \min\left\{|q| > 0 : q \in \hat{\mathcal{L}}_Z^{(\text{bbs})}\right\} && (27)\\ &= \min\{|\vec{x}| > 0 : \vec{x} \in \text{col}(A)\} && (28)\\ &= d_1. && (29)\end{aligned}$$

The overall code distance of the BBS code is $D = \min(D_Z, D_X) = \min(d_1, d_2)$.

The discussion so far has reproduced Bravyi's theorem

Theorem 2 (Bravyi [9])**.** *The Bravyi-Bacon-Shor code constructed from $A \in \mathbb{F}_2^{n_1 \times n_2}$, denoted BBS($A$), is an $[\![N, K, D]\!]$ quantum subsystem code with gauge group generated by 2-qubit operators and*

$$\begin{aligned} N &= |A|, && (30)\\ K &= \textit{rank}(A), && (31)\\ D &= \min\{|\vec{y}| > 0 : \vec{y} \in \textit{row}(A) \cup \textit{col}(A)\}. && (32)\end{aligned}$$

Assuming without loss of generality that no row or column of A is all 0s (if there is such a row or column, then it can be removed without changing the code), it is worth noting the bounds

$$D \min(n_1, n_2) \leq \min(D_X n_1, D_Z n_2) \leq |A| \leq n_1 n_2. \quad (33)$$

The second inequality is based off the fact that each row (column) of A needs to contain at least D_X (D_Z) qubits.

C. Augmented Bravyi-Bacon-Shor codes

In this subsection, we discuss geometric locality of the BBS codes. In particular, we review the modification that makes them local in 2-dimensions.

Definition 3 (quantum LDPC codes)**.** A subsystem code with gauge group $\mathcal{G}$ is (β, γ)-LDPC if, there is a subset $\mathcal{G}_{\text{ldpc}} \subseteq \mathcal{G}$ such that

- $\mathcal{G}_{\text{ldpc}}$ generates $\mathcal{G}$, i.e. $\mathcal{G} = \langle \mathcal{G}_{\text{ldpc}} \rangle$.
- each qubit is in the support of at most β of the $g \in \mathcal{G}_{\text{ldpc}}$.
- the support of each $g \in \mathcal{G}_{\text{ldpc}}$ contains at most γ qubits.

We refer to $\mathcal{G}_{\text{ldpc}}$ as an LDPC generating set.

Every BBS code is $(4, 2)$-LDPC. An LDPC generating set $\mathcal{G}_{\text{ldpc}}$ contains just the two-qubit gauge operators between consecutive qubits in a row or column.

Definition 4 (quantum geometric locality)**.** An infinite family of (β, γ)-LDPC subsystem codes is local in M-dimensions if there is a constant ρ such that all codes in the family have an LDPC generating set $\mathcal{G}_{Md}$ and the qubits of the code can be arranged on vertices of an M-dimensional (hyper)cubic lattice in such a way that no two qubits in the support of the same $g \in \mathcal{G}_{Md}$ are more than (Manhattan) distance ρ apart.

To attempt to show that a family of BBS codes is local in 2-dimensions, one might try $\mathcal{G}_{2d} = \mathcal{G}_{\text{ldpc}}$ from above. While this is of course an LDPC generating set, it is not necessarily true that elements of $\mathcal{G}_{2d}$ are supported in constant-sized regions of the 2-dimensional lattice. The difficulty is that A may contain two consecutive 1s in the same row or column that are separated by many 0s (potentially a number of 0s that grows with code size) and thus consecutive qubits are far apart.

To remedy this, Bravyi [9] introduces two more qubits at every site (i, j) such that $A_{ij} = 0$. One qubit participates in the two-qubit gauge operators of row i and the other in the gauge operators of column j. Hence, we now say that there are three types of qubits making up the code – type 0 qubits reside at sites where $A_{ij} = 1$, whereas type 1 and type 2 qubits reside at sites where $A_{ij} = 0$. These qubit types can be used to define two lattices – L_1 consists of qubits of type 0 and type 1 and L_2 consists of qubits of type 0 and type 2. It is important to note that the lattices share the type 0 qubits, i.e. the lattices are identified at the sites where $A_{ij} = 1$.

To distinguish Paulis acting on qubits in lattices L_1 or L_2, we use superscripts, e.g. $X_{ij}^{(L_1)}$ or $X_{ij}^{(L_2)}$ for single-qubit Paulis and $X^{(L_1)}(S)$ or $X^{(L_2)}(S)$ for Paulis acting on multiple qubits specified by support S. Of course, due to the identification of qubits between L_1 and L_2, a particular (say, X-type) Pauli P does not have unique supports S_1, S_2 such that $P = X^{(L_1)}(S_1)X^{(L_2)}(S_2)$. Indeed, letting $\mathbb{1}$ be the matrix of all 1s,

$$X^{(L_1)}(S_1)X^{(L_2)}(S_2) = X^{(L_1)}(T_1)X^{(L_2)}(T_2) \quad (34)$$

if and only if $S_1 \cap (\mathbb{1} - A) = T_1 \cap (\mathbb{1} - A)$, $S_2 \cap (\mathbb{1} - A) = T_2 \cap (\mathbb{1} - A)$, and $(S_1 \cap A) + (S_2 \cap A) = (T_1 \cap A) + (T_2 \cap A)$.

Using this notation, the augmented Bravyi-Bacon-Shor code (aBBS) has gauge groups

$$\mathcal{G}_X^{(\text{abbs})} = \{X^{(L_1)}(S)X^{(L_2)}(T) : G_R S = 0, T \subseteq \mathbb{1} - A\}, \quad (35)$$

$$\mathcal{G}_Z^{(\text{abbs})} = \{Z^{(L_1)}(S)Z^{(L_2)}(T) : T G_R^T = 0, S \subseteq \mathbb{1} - A\}. \quad (36)$$

Intuition for this gauge group arises by developing a generating set local in 2-dimensions. This generating set $\mathcal{G}_{2d}$ can be chosen to be the set of all two-qubit gauge operators on *neighboring* qubits in the lattices as well

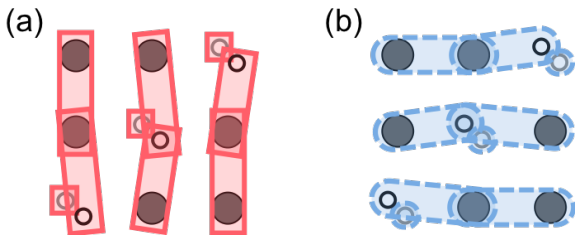

FIG. 2. Generating sets of (a) X-type and (b) Z-type gauge operators for the augmented version of the code from Fig. 1. Type 0 qubits are shown as large, filled circles, while type 1 and 2 qubits are small and unfilled. The generating sets consist entirely of two-qubit (dark) and single-qubit (light) operators.

as all one-qubit gauge operators. That is, with $[t] = \{1, 2, \ldots, t\}$, we have

$$\mathcal{G}_{2d} = \{X_{ij}^{(L_1)} X_{i+1,j}^{(L_1)} : i \in [n_1 - 1], j \in [n_2]\} \quad (37)$$
$$\cup \{Z_{ij}^{(L_2)} Z_{i,j+1}^{(L_2)} : i \in [n_1], j \in [n_2 - 1]\} \quad (38)$$
$$\cup \{X_{ij}^{(L_2)} : A_{ij} = 0\} \quad (39)$$
$$\cup \{Z_{ij}^{(L_1)} : A_{ij} = 0\}. \quad (40)$$

This set, which has $4n_1n_2 - (n_1 + n_2) - 2|A|$ independent generators, is clearly local in 2-dimensions: it consists of two-qubit X operators between qubits sharing a column in lattice 1, two-qubit Z operators between qubits sharing a row in lattice 2, and single-qubit operators on type 1 and type 2 qubits. An example of $\mathcal{G}_{2d}$ for a $[\![12, 2, 2]\!]$ aBBS code is shown in Fig. 2.

Bare logical operators and stabilizers of an aBBS code are derived similarly to those of a BBS code. Rather than go through those arguments again, we just record the results here.

$$\mathcal{L}_X^{(\text{abbs})} = \{X^{(L_2)}(S) : SH_R^T = 0\}, \quad (41)$$
$$\mathcal{L}_Z^{(\text{abbs})} = \{Z^{(L_1)}(S) : H_R S = 0\}, \quad (42)$$
$$\mathcal{S}_X^{(\text{abbs})} = \{X^{(L_2)}(S) : SH_R^T = 0, G_1 S = 0\}, \quad (43)$$
$$\mathcal{S}_Z^{(\text{abbs})} = \{Z^{(L_1)}(S) : H_R S = 0, SG_2^T = 0\}. \quad (44)$$

Code parameters K and D are also unchanged. Collecting this into a theorem, we have:

Theorem 5 (Bravyi [9]). *The augmented Bravyi-Bacon-Shor code constructed from $A \in \mathbb{F}_2^{n_1 \times n_2}$, denoted aBBS($A$), is an $[\![N, K, D]\!]$ quantum subsystem code that is local in 2-dimensions, has a gauge group generated by 1- or 2-qubit operators, and*

$$N = 2n_1n_2 - |A|, \quad (45)$$
$$K = \mathit{rank}(A), \quad (46)$$
$$D = \min\{|\vec{y}| > 0 : \vec{y} \in \mathit{row}(A) \cup \mathit{col}(A)\}. \quad (47)$$

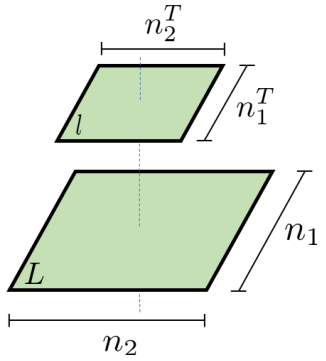

FIG. 3. The two lattices of qubits that make up a hypergraph product code.

D. Hypergraph Product Codes

Introduced in [7], the hypergraph product takes two classical parity check matrices $H_1 \in \mathbb{F}_2^{n_1^T \times n_1}$ and $H_2 \in \mathbb{F}_2^{n_2^T \times n_2}$ and produces a quantum code. The code ultimately is of CSS type (though the traditional CSS construction is not used to obtain it) and so its stabilizer group can be separated into stabilizers of Pauli X-type and those of Pauli Z-type.

Our description of the hypergraph product is a little unconventional but is in line with how we described BBS and aBBS codes, making it easier to relate the two later. It is essentially a description in terms of the "reshaped" matrices used at some points by Campbell [16].

In our notation, qubits of the hypergraph product code are placed on the vertices of two square lattices (see Fig. 3). The first lattice L is $n_1 \times n_2$. The second lattice l is $n_1^T \times n_2^T$. A Pauli X or Z acting on the qubit at site (i, j) in lattice L is denoted $X_{ij}^{(L)}$ or $Z_{ij}^{(L)}$ and similarly for Paulis acting in lattice l. A Pauli operator acting on multiple qubits is specified by its support, e.g. $X^{(L)}(S)$ or $Z^{(L)}(S)$, just as for the BBS and aBBS codes.

On the classical side, we define generating matrices G_1 and G_2 for the classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$ corresponding to H_1 and H_2. We define their code parameters as $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$. Similarly, let F_1 and F_2 be generating matrices for codes $\mathcal{C}_1^T$ and $\mathcal{C}_2^T$ with code parameters $[n_1^T, k_1^T, d_1^T]$ and $[n_2^T, k_2^T, d_2^T]$.

Using this notation, the hypergraph product of H_1 and H_2 is a quantum code $\text{HGP}(H_1, H_2)$ defined by the following sets of stabilizers, divided into X-type and Z-type,

$$\mathcal{S}_X^{(\text{hgp})} = \{X^{(L)}(S)X^{(l)}(T) : SH_2^T = H_1^T T, \quad (48)$$
$$G_1 S = 0, TF_2^T = 0\},$$
$$\mathcal{S}_Z^{(\text{hgp})} = \{Z^{(L)}(S)Z^{(l)}(T) : H_1 S = TH_2, \quad (49)$$
$$SG_2^T = 0, F_1 T = 0\}.$$

To show these stabilizers commute, let $M = X^{(L)}(S)X^{(l)}(T) \in \mathcal{S}_X^{(\text{hgp})}$ and $M' = Z^{(L)}(S')Z^{(l)}(T') \in \mathcal{S}_Z^{(\text{hgp})}$. By Eq. (8), we need to show $\text{tr}(S^T S') +$

$\mathrm{tr}(T^T T') = 0$. Notice that $G_1 S = 0$ demands that columns of S are parity checks for $\mathcal{C}_1$. In other words, there exists A such that $S = H_1^T A$. Likewise, because $T F_2^T = 0$, rows of T are parity checks for $\mathcal{C}_2^T$, or, equivalently, there exists B such that $T = B H_2^T$. Finally, the same reasoning holds for S' and T', showing the existence of A' and B' such that $S' = A' H_2$ and $T' = H_1 B'$. Since $S H_2^T = H_1^T T$ and $H_1 S' = T' H_2$, we have $H_2 A^T H_1 = H_2 B^T H_1$ and $H_1 A' H_2 = H_1 B' H_2$. Putting it all together we have

$$\mathrm{tr}(S^T S') = \mathrm{tr}(A^T H_1 A' H_2) \tag{50}$$
$$= \mathrm{tr}(B^T H_1 B' H_2) = \mathrm{tr}(T^T T'), \tag{51}$$

completing the proof.

In Appendix A, we connect this description of the hypergraph product code with the original definition, and derive other relevant properties. We note here that logical operators for $\mathrm{HGP}(H_1, H_2)$ are

$$\mathcal{L}_X^{(\mathrm{hgp})} = \{X^{(L)}(S) X^{(l)}(T) : S H_2^T = H_1^T T\}, \tag{52}$$
$$\mathcal{L}_Z^{(\mathrm{hgp})} = \{Z^{(L)}(S) Z^{(l)}(T) : H_1 S = T H_2\}, \tag{53}$$

and it has code parameters $[\![N, K, D]\!]$ [7]:

$$N = n_1 n_2 + n_1^T n_2^T, \tag{54}$$
$$K = k_1 k_2 + k_1^T k_2^T, \tag{55}$$
$$D = \min(d_1, d_2, d_1^T, d_2^T). \tag{56}$$

Keep in mind that if one of the classical codes or their transposes encodes no logical bits (i.e. is the empty set), its distance is by definition infinite. Lastly, if H_1 is a (b_1, c_1)-LDPC set of parity checks and H_2 is a (b_2, c_2)-LDPC set of parity checks, then $\mathrm{HGP}(H_1, H_2)$ is (β, γ)-LDPC for [7]

$$\beta = \max(b_1 + b_2, c_1 + c_2), \tag{57}$$
$$\gamma = \max(c_1 + b_2, b_1 + c_2). \tag{58}$$

III. CONSTRUCTING AND DECODING OPTIMAL 2-DIMENSIONAL SUBSYSTEM CODES

In this section, we show how to make BBS codes from two classical codes. We note that using good classical codes leads to optimal scaling of the quantum code parameters and show how classical decoders are used to decode the quantum codes.

A. Bravyi-Bacon-Shor codes from classical codes

In Section II B we noted that an $[\![N, K, D]\!]$ BBS code specified by matrix $A \in \mathbb{F}_2^{n_1 \times n_2}$ defines two classical codes $\mathcal{C}_1 = \mathrm{col}(A)$ and $\mathcal{C}_2 = \mathrm{row}(A)$, and that, if those classical codes have parameters $[n_1, k, d_1]$ and $[n_2, k, d_2]$, we have code parameter relations $K = k$ and $D = \min(d_1, d_2)$. The goal now is to explore the converse: given two classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$, how should we construct a BBS code with the same relations in code parameters?

Suppose that the classical codes have generating matrices $G_1 \in \mathbb{F}_2^{k \times n_1}$ and $G_2 \in \mathbb{F}_2^{k \times n_2}$. We then construct the code $\mathrm{BBS}(A)$ with

$$A = G_1^T Q G_2 \in \mathbb{F}_2^{n_1 \times n_2}, \tag{59}$$

where Q is any full-rank $k \times k$ matrix representing the non-uniqueness of the generating matrices. Adjusting Q can change the number of physical qubits in the code.

Now notice that

$$\mathrm{col}(A) = \{G_1^T Q G_2 \vec{x} : \vec{x} \in \mathbb{F}_2^{n_2}\} \tag{60}$$
$$= \{G_1^T Q \vec{y} : \vec{y} \in \mathbb{F}_2^{k}\} \tag{61}$$
$$= \{G_1^T \vec{z} : \vec{z} \in \mathbb{F}_2^{k}\} \tag{62}$$
$$= \mathrm{col}(G_1^T) = \mathrm{row}(G_1) = \mathcal{C}_1. \tag{63}$$

The second equality relies on G_2 being full-rank and the third on Q being full-rank. Likewise, similar reasoning shows that $\mathrm{row}(A) = \mathcal{C}_2$.

Therefore, we have the following theorem.

Theorem 6. *For all full-rank $Q \in \mathbb{F}_2^{k \times k}$ and every two classical codes $\mathcal{C}_1$, $\mathcal{C}_2$ with parameters $[n_1, k, d_1]$, $[n_2, k, d_2]$ and generating matrices $G_1 \in \mathbb{F}_2^{k \times n_1}$, $G_2 \in \mathbb{F}_2^{k \times n_2}$, let $A = G_1^T Q G_2$. Then BBS(A) is an $[\![N, K, D]\!]$ quantum subsystem code and aBBS(A) an $[\![N_{2d}, K, D]\!]$ subsystem code local in 2-dimensions with*

$$\min(n_1 d_2, d_1 n_2) \le N \le n_1 n_2, \tag{64}$$
$$n_1 n_2 \le N_{2d} \le 2 n_1 n_2 - \min(n_1 d_2, d_1 n_2), \tag{65}$$
$$K = k, \tag{66}$$
$$D = \min(d_1, d_2). \tag{67}$$

The lower bound on N and upper bound on N_{2d} are provided by Eq. (33).

Before discussing the theorem's implications, let us briefly present some examples, starting with the Bacon-Shor code.

Example 1. *Let $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}_R$ be the $[n, 1, n]$ repetition code (see Eq. (4)). Then $A = G_R^T G_R = \mathbb{1}$ (the all 1s matrix) represents a Bravyi-Bacon-Shor with a qubit at every lattice site, X-type (Z-type) gauge operators between pairs of qubits in the same column (row), and X-type (Z-type) stabilizers that span pairs of rows (columns). That is, we have reconstructed the Bacon-Shor code [12].*

Example 2. *The $[7, 4, 3]$ Hamming code is generated by*

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

Let $A = G^T Q G$ for full-rank 4×4 matrix Q. Taking $Q = I$ gives a $[\![25, 4, 3]\!]$ Bravyi-Bacon-Shor code:

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

Alternatively, taking $Q = \left(\begin{smallmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{smallmatrix}\right)$ minimizes the number of qubits, giving a $[\![21, 4, 3]\!]$ Bravyi-Bacon-Shor code.

In [9], Bravyi shows that for any family of $[\![N, K, D]\!]$ quantum subsystem codes local in 2-dimensions $KD = O(N)$. He then provides a nonconstructive argument that families of aBBS codes exist that saturate this bound. Theorem 6 elucidates this existence proof by connecting it to the classical case. If we have a family of good $[n, k, d]$ classical codes – i.e. there are constants α, β such that for all n, $k \geq \alpha n$ and $d \geq \beta n$ – then the aBBS code family created from Theorem 6 satisfies $KD = \alpha\beta n^2 \geq \alpha\beta N_{2d}/2$. So the aBBS codes created this way saturate Bravyi's bound.

Moreover, Theorem 6 provides the means to elevate the nonconstructive existence proof to an explicit constructive proof. One only needs an explicit construction of good classical codes. Such constructions exist, e.g. expander codes. We review these classical codes in detail in Appendix B 1.

Finally, we should point out that although Theorem 6 produces BBS and aBBS codes for which $K, D = O(\sqrt{N})$, it can be used to trade off K and D. Bravyi and Terhal [17] have shown that $D = O(\sqrt{N})$ for subsystem codes in 2-dimensions. So, assume that we would like a code family with $K = \alpha N^{1-a}$ and $D = \beta N^a$ for some constants $a \leq 1/2$, α, and β. To construct this code family, use Theorem 6 and a good family of classical codes to make a quantum code family with $K = \alpha N^a$, $D = \beta N^a$, and $O(N^{2a})$ physical qubits. Take N^{1-2a} copies of this family to make the desired family with $K = \alpha N^{1-2a} N^a = \alpha N^{1-a}$, $D = \beta N^a$, and $N^{1-2a} O(N^{2a}) = O(N)$ physical qubits.

If, for whatever reason, a family with parameters $KD = o(N)$ is desired (i.e. KD scales strictly less than N), then one can take a family with $K'D = \Theta(N)$ and ignore a fraction $1 - K/K'$ of the encoded qubits. See Fig. 4 for a summary of the last two paragraphs.

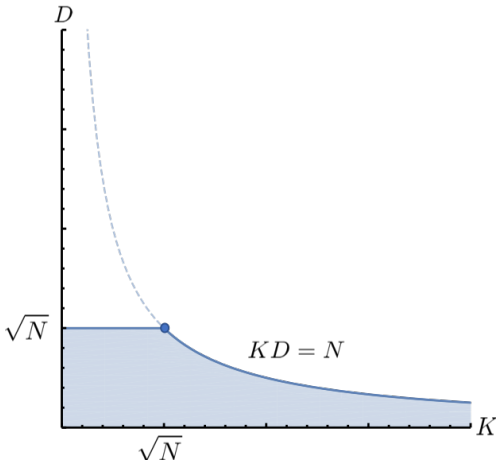

FIG. 4. The region of possible [9, 17] 2-dimensional subsystem $[\![N, K, D]\!]$ code families. We can construct families at all these points by appealing to constructions of good classical codes and using Theorem 6.

B. Decoding BBS codes

Correcting errors on a quantum subsystem code involves (1) measuring a generating set of gauge operators, (2) reconstructing the values of the stabilizers from the results, and (3) applying a Pauli correction. An advantage of the BBS or aBBS codes is that the generating set of gauge operators includes only 2-qubit operators (see Eq. 37) despite the stabilizers being high-weight. In this section, we identify another convenient feature – the correction in the third step can be calculated by decoders for the corresponding classical codes.

Let us begin by making some assumptions about the error model and codes. While not essential to the main conclusions, these assumptions simplify the discussion. Regarding the error model, we assume that each qubit suffers an X error with probability q and, independently, a Z error with the same probability q. Two-qubit Pauli measurements (e.g. of the gauge operators) are assumed to fail with probability q'. Regarding the codes, we assume that A is an $n \times n$ symmetric matrix, so there is just one $[n, k, d]$ classical code $\mathcal{C} = \text{row}(A) = \text{col}(A)$ under consideration. We let H and G be the parity check and generating matrices of this code.

What is essential to our conclusions here is the existence of a decoding algorithm $\mathcal{D}$ for the classical code $\mathcal{C}$ of the following form. Decoder $\mathcal{D}$ takes as input faulty parity check information from a faulty codeword $\vec{i} = H(\vec{w} + \vec{e}) + \vec{f}$, where $\vec{w} \in \mathcal{C}$, $\vec{e}$ represents data errors, and $\vec{f}$ represents errors in the "measurement" of the parity checks (though a more appropriate classical terminology might be the "calculation" of the checks). The decoder's task is then to find a recovery $\vec{e}' = \mathcal{D}(\vec{i})$ that is close to $\vec{e}$, and update the classical state from $\vec{w}+\vec{e}$ to $\vec{w} + \vec{e} + \vec{e}'$. This process is repeated some number of rounds, alternating the application of random noise $\vec{e}$ and $\vec{f}$ with decoding. Afterwards, we imagine "ideal" decoding with $\vec{f} = \vec{0}$ is performed, and if $\vec{w}$ is not the final state, then the error-correction has failed. Failure (after the given number of rounds) occurs with some probability $\bar{p}$, which is a function of the probability distribution of errors $\vec{e}$ and $\vec{f}$.

Generally, in classical error-correction, measurement of

the parity checks is considered perfect, and so decoders of the required capability are seldom created or used. However, expander codes do have a suitable decoder, the flip decoder, which is discussed in Appendix B 2.

We discuss the decoding of (symmetric, $A = A^T$) BBS codes in detail, then in the next section briefly discuss the case of (symmetric) aBBS codes, which is similar. Our reasoning hinges on associating the stabilizers of the BBS code with parity checks of the classical code $\mathcal{C}$ and the dressed logical operators of the BBS code with the codewords of $\mathcal{C}$.

To realize these associations, we rewrite $\mathcal{S}_X^{(\mathrm{bbs})}$ from Eq. (15). Let $X(S \cap A) \in \mathcal{S}_X^{(\mathrm{bbs})}$. Since $SH_R^T = 0$, rows of S are codewords of $\mathcal{C}_R$, either all 1s or all 0s. Because $GS = 0$, columns of S are parity checks of $\mathcal{C}$. Therefore, $S \cap A = \mathrm{diag}(\vec{r})A$ for some $\vec{r} \in \mathrm{row}(H)$. We have

$$\mathcal{S}_X^{(\mathrm{bbs})} = \{X\left(\mathrm{diag}(\vec{r})A\right) : \vec{r} \in \mathrm{row}(H)\}. \tag{68}$$

Similarly, rewrite $\mathcal{S}_Z^{(\mathrm{bbs})}$ as

$$\mathcal{S}_Z^{(\mathrm{bbs})} = \{Z\left(A\,\mathrm{diag}(\vec{c})\right) : \vec{c} \in \mathrm{row}(H)\}. \tag{69}$$

Thus, the parity checks of the classical code indicate which sets of rows or columns constitute a stabilizer.

Dressed logical operators $\hat{\mathcal{L}}_X^{(\mathrm{bbs})}$ are exactly those X-type Paulis that commute with all the Z-type stabilizers. But because Z-type stabilizers are supported on entire columns of A, they are only sensitive to whether an even or odd number of Pauli X errors occurred within a column. Indeed, X errors within a column are equivalent up to gauge operators. Say that a column is odd if it contains an odd number of X errors. An X-type operator commutes with all the Z-type stabilizers if and only if it consists of odd columns corresponding to a codeword $\vec{w} \in \mathcal{C}$, i.e. column i is odd if and only if $\vec{w}_i = 1$. In other words, the even or oddness of a column corresponds to the 0 or 1 state of an effective classical bit of the code $\mathcal{C}$. Symmetry of A dictates that the same correspondence holds for Z-type dressed logical operators $\hat{\mathcal{L}}_Z^{(\mathrm{bbs})}$ and Z errors in rows.

The upshot of the previous paragraph is that to decode a BBS(A) code, we may collect X- or Z-type stabilizer information $\vec{\sigma}$, run the classical decoder $\vec{e}' = \mathcal{D}(\vec{\sigma})$, and apply a Z- or X-type Pauli correction to a single qubit in each row or column indicated by $\vec{e}'$. We call this the decoder induced by $\mathcal{D}$, or simply the *induced decoder* for BBS(A). To evaluate how well the induced decoder works, we just need to map the quantum errors to the effective classical errors that the decoder $\mathcal{D}$ sees.

The probability that an odd number of X errors occurs within column i containing c_i qubits is

$$p_i = \sum_{\substack{l=1 \\ l \text{ odd}}}^{c_i} \binom{c_i}{l} q^l (1-q)^{r_i - l} = \frac{1}{2}\left(1 - (1-2q)^{c_i}\right). \tag{70}$$

By symmetry, this situation is the same for Z errors in the rows. So p_i is the probability that bit i has flipped in the classical code.

Similarly, stabilizers of the Bravyi-Bacon-Shor code are the product of several two-qubit gauge operators. For instance, there is an Z-type stabilizer $Z(A\,\mathrm{diag}(\vec{h}_j))$ for row $\vec{h}_j$ of H, and it is made of $c'_j = |A\,\mathrm{diag}(\vec{h}_j)|/2 \leq n|\vec{h}_j|/2$ two-qubit gauge measurements. The probability this stabilizer measurement is incorrect depends only on whether an even or odd number of its constituent gauge measurements are incorrect:

$$p'_j = \sum_{\substack{l=1 \\ l \text{ odd}}}^{c'_j} \binom{c'_j}{l} q^l (1-q')^{c'_j - l} = \frac{1}{2}\left(1 - (1-2q')^{c'_j}\right). \tag{71}$$

By symmetry, this situation is identical for the X-type stabilizers. Thus, p'_j is the probability that the parity check calculation for parity check j is incorrect.

These relations between quantum and classical errors give us the following lemma.

Lemma 7. *Say that using decoder $\mathcal{D}$ on the classical error model in which data errors have probabilities p_i and parity check errors have probabilities p'_j results in a logical error rate of $\bar{p}(p_i, p'_j)$. The induced decoder with respect to $\mathcal{D}$ on an error model in which qubits fail with independent X or Z errors with probability q and two-qubit Pauli measurements fail with probability q' has a logical error rate*

$$\bar{q}(q, q') \leq 2\bar{p}(p_i, p'_j) \tag{72}$$

where p_i and p'_j are given by Eqs. (70) and (71).

The factor of two in Eq. (72) results from the X and Z errors being decoded separately. Independent X, Z noise is of course not critical to the lemma. For depolarizing noise for example, in which Pauli X, Y, or Z errors occur with equal probability $q/3$, the logical error rate is at most $\bar{q}(2q/3, q')$ since $2q/3$ is the probability of a Z or X error. On the other hand, the induced decoder does discount the correlations in X and Z noise, so is not expected to be optimal in this case.

Lemma 7 indicates two ways to improve the decoding of BBS codes, even before tailoring to the noise. The first, more obvious way, is to find better decoders for the constituent classical codes. This is of course subject to the constraint that these classical decoders can tolerate measurement noise, which we noted previously is non-standard but attainable for expander codes for example.

The second way to improve decoding is by reducing the values of c_i (the number of qubits in row or column i) and c'_j (the number of gauge-operators making up stabilizer j). This correlates roughly with minimizing $|A|$, the number of qubits in the BBS code, which can be done without change in the code parameters by appropriate choice of Q in Theorem 6. For some quantification of how this helps, note that for small q and q', $p_i \approx c_i q$ and $p'_j \approx c'_j q'$. In this regime, Lemma 7 implies that if the classical code has a "useful" (e.g. order 10^{-a} for some moderately large

a) logical error rate for $p_i < p$ and $p'_j < p'$, then the quantum code has a useful (i.e. order 10^{-a}) logical error rate for $q < p/(\max_i c_i)$ and $q' < p'/(\max_j c'_j)$. The best we could hope for is that $\max_i c_i$ and $\max_j c'_j$ are on order of the code distance d, and this then sets the scale for useful error-correction with BBS codes.

Finally, let us discuss the time complexity of an induced decoder. This can be broken down into two parts: (1) the time it takes to acquire the stabilizer values that are input to $\mathcal{D}$ and (2) the time it takes to run $\mathcal{D}$ twice, once for X-stabilizers, once for Z. A particular stabilizer corresponding to a weight-w parity check is the sum of $O(wn)$ two-qubit measurements and therefore takes $O(wn)$ time to compute. If m stabilizer values are needed as input to the classical decoders, and the classical decoders run in time at most t, then induced decoding takes time $O(mwn + t)$. Using BBS codes constructed from classical expander codes as an example, the flip decoder $\mathcal{D}$ (see Appendix B 2) requires just $m = O(n)$ bits of input from weight $w = O(1)$ checks and runs in time $t = O(n)$. Thus, induced decoding takes time $O(n^2 + n) = O(N)$, i.e. linear in the size of the quantum code.

C. Decoding aBBS codes

In this subsection, we briefly discuss the decoding of (symmetric, $A = A^T$) aBBS codes assuming we can only measure operators in $\mathcal{G}_{2d}$, Eq. (37), i.e. two-qubit operators on neighboring qubits and some single-qubit measurements. We still advocate using the induced decoder of the previous section, but it is now more difficult to collect the stabilizer values from this restricted set of gauge operator measurements.

Similar to how we derived Eqs. (68), (69), we can rewrite the stabilizers of the aBBS codes to correspond to classical parity checks (recall, $\mathbb{1}$ is the matrix of all 1s):

$$\mathcal{S}_X^{(\text{aBBS})} = \left\{X^{(L_2)}\left(\text{diag}(\vec{r})\mathbb{1}\right) : \vec{r} \in \text{row}(H)\right\}, \tag{73}$$

$$\mathcal{S}_Z^{(\text{aBBS})} = \left\{Z^{(L_1)}\left(\mathbb{1}\,\text{diag}(\vec{c})\right) : \vec{c} \in \text{row}(H)\right\}. \tag{74}$$

This leads to similar conclusions about errors on the effective bits of the classical code $\mathcal{C}$. With the recognition that $c_i = n$ for all i, Eq. (70) still represents the probability of error for an effective classical bit.

As one may expect, because we have restricted what gauge operators may be measured to those in $\mathcal{G}_{2d}$, aBBS decoding also differs from BBS decoding in how eigenvalues of the stabilizers are calculated. If $\vec{h}_j$ is a row of H and $S = Z^{(L_1)}(\mathbb{1}\,\text{diag}(\vec{h}_j))$ is the corresponding stabilizer, then we should let c'_j be the minimal number of elements of $\mathcal{G}_{2d}$ whose product is S. Since S may include rows that are $O(n)$ distance apart, c'_j may be as a large as $O(n^2)$. With this redefinition of c'_j however, Eq. (71) again represents the probability of error for a parity check. Lemma 7 holds given these changes to c_i and c'_j.

Now we discuss the runtime. Because c'_j can be so large, we may be worried that it takes more time to decode, because ostensibly stabilizers corresponding to even just constant-weight parity checks may be the sum of as many as $O(n^2)$ elements of $\mathcal{G}_{2d}$ (and note that $|\mathcal{G}_{2d}| = O(n^2)$). However, a simple application of dynamic programming solves this. Suppose that we measure all two-body Z-gauge operators and get values $m_{ij} \in \{0, 1\}$ corresponding to positions (i, j) in the lattice. We can sweep across the lattice calculating the cumulative values across rows

$$M_{ij} = \sum_{l=1}^{j} m_{il} \tag{75}$$

using just $O(n^2) = O(N)$ time. Z-type stabilizers corresponding to constant-weight parity checks are once again the sum of $O(n)$ of the M_{ij} as well as $O(n)$ single-qubit measurements. Symmetry dictates the same is true for X-type stabilizers. Therefore, for example, the induced decoder with respect to the flip decoder for aBBS codes constructed from classical expander codes can still be implemented in linear time.

IV. GAUGE-FIXING

In this section, we show that an aBBS code can be gauge-fixed to the corresponding BBS code and to certain hypergraph product codes. We begin, however, by defining gauge-fixing in general.

A. Definition

One way to think about subsystem codes is that in addition to the logical qubits encoded in the code, there are additional encoded qubits, the gauge qubits, which we do not care about protecting. In fact, the logical operators for these gauge qubits may be very low weight – they are the gauge operators that we measure to perform error-correction.

The existence of gauge qubits, however, leads us to imagine a family of related codes in which some or all of the gauge qubits are fixed to some stabilizer state $|\psi_g\rangle$. In these related codes, called gauge-fixings, we have removed some or all of the gauge degrees of freedom by removing operators from the gauge group that do not stabilize $|\psi_g\rangle$. Generally, this makes error-correction more difficult – a generating set for the new gauge group may necessarily contain higher weight operators – but by reducing the size of the group of dressed logical operators, the environment has fewer ways to introduce logical errors to the data. This may even result in asymptotic error-correction thresholds in the gauge-fixed codes where none

existed in the original subsystem code. A well-known example is the gauge-fixing of the Bacon-Shor code to the surface code [18].

To talk about gauge-fixing in general, let us consider a subsystem code with gauge group $\mathcal{G}$. Other important groups are

1. The bare logical operators $\mathcal{L}(\mathcal{G})$: the set of all Paulis that commute with all elements of $\mathcal{G}$, also known in group theory as the centralizer of $\mathcal{G}$.
2. The stabilizers $\mathcal{S}(\mathcal{G})$: the intersection of $\mathcal{L}(\mathcal{G})$ with $\mathcal{G}$, also known as the center of $\mathcal{G}$.
3. The dressed logical operators $\hat{\mathcal{L}}(\mathcal{G}) = \mathcal{G}\ \mathcal{L}(\mathcal{G})$: the centralizer of $\mathcal{S}(\mathcal{G})$.

The number of encoded qubits we denote $K(\mathcal{G})$ ($4^{K(\mathcal{G})}$ is the size of $\mathcal{L}(\mathcal{G})$ after modding out stabilizers and constant factors) and the code distance (the weight of the lowest weight element of $\hat{\mathcal{L}}(\mathcal{G})$) we denote $D(\mathcal{G})$. A subsystem code has no gauge qubits, i.e. is a subspace stabilizer code, if and only if $\mathcal{S}(\mathcal{G}) = \mathcal{G}$.

Definition 8. We say that $\mathcal{G}'$ is a gauge-fixing of $\mathcal{G}$ if

1. $\mathcal{S}(\mathcal{G}) \leq \mathcal{S}(\mathcal{G}') \leq \mathcal{G}' \leq \mathcal{G}$
2. $K(\mathcal{G}) = K(\mathcal{G}')$

Generalizing the language slightly, we also say that a code $\mathcal{Q}'$ is a gauge-fixing of a code $\mathcal{Q}$ if their gauge groups are related appropriately.

By the definition, a subsystem code and its gauge-fixing have the same total number of physical qubits and logical qubits. We can also say something about their code distances.

Lemma 9. *If $\mathcal{G}'$ is a gauge-fixing of $\mathcal{G}$, then $\hat{\mathcal{L}}(\mathcal{G}') \leq \hat{\mathcal{L}}(\mathcal{G})$ and $D(\mathcal{G}') \geq D(\mathcal{G})$.*

We prove this fact in Appendix C.

A concept more general than gauge-fixing is gauge-switching. If both $\mathcal{G}'$ and $\mathcal{G}''$ are gauge-fixings of $\mathcal{G}$, then one can move encoded logical information from $\mathcal{G}'$ to $\mathcal{G}''$ (or vice-versa) while keeping it protected with the stabilizers $\mathcal{S}(\mathcal{G}') \cap \mathcal{S}(\mathcal{G}'') \geq \mathcal{S}(\mathcal{G})$ and with code distance at least $D(\mathcal{G})$. Measuring the gauge group $\mathcal{G}''$, applying a correction based on the values of $\mathcal{S}(\mathcal{G})$ using a decoder for $\mathcal{G}$, and finally projecting onto the $+1$-eigenspaces of elements of $\mathcal{S}(\mathcal{G}'') - \mathcal{S}(\mathcal{G})$ using the appropriate elements of $\mathcal{G}$ achieves this information transfer.

B. Gauge-fixing an aBBS code to a BBS code

To warm up to Definition 8, we show that a BBS code specified by binary matrix A is a gauge-fixing of the aBBS code specified by the same matrix. Of course, these codes do not have the same number of physical qubits, so to make the previous sentence precise we include ancilla qubits to the BBS code. This will be a common occurrence in our gauge-fixing theorems, and so we take a moment to discuss it.

Given a quantum code $\mathcal{Q}$, we will consider appending three types of ancillas: (1) qubits in the $|+\rangle$ state, (2) qubits in the $|0\rangle$ state, and (3) bare gauge qubits denoted $|\perp\rangle$. The code including ancillas is written $\mathcal{Q}|+^{m_+}\rangle|0^{m_0}\rangle|\perp^{m_g}\rangle$ with the number of each type of ancilla indicated. Appending ancillas in this way extends the code's gauge group. Ancillas $|+\rangle$ indicate the inclusion of Paulis X_i into the gauge group for each ancilla index i. Likewise, $|0\rangle$ ancillas indicate inclusion of Z_i. Bare gauge qubits $|\perp\rangle$ indicate inclusion of both X_i and Z_i.

Now we can formally state the relation between BBS(A) and aBBS(A).

Theorem 10. *For all binary matrices $A \in \mathbb{F}_2^{n_1 \times n_2}$, $\mathcal{Q}' = BBS(A)\big|+^{n_1 n_2 - |A|}\big\rangle\big|0^{n_1 n_2 - |A|}\big\rangle$ is a gauge-fixing of $\mathcal{Q} = aBBS(A)$.*

Proof. We place both codes on the lattices L_1 and L_2 defined in Section II C for the aBBS codes (recall, two $n_1 \times n_2$ lattices that share qubits wherever $A_{ij} = 1$). The gauge group of $\mathcal{Q}$ is defined in Eqs. (35), (36). For $\mathcal{Q}'$, however, we should rewrite the gauge group to fit on these two lattices and to include the ancillas. As one may suspect from their quantity, the $|+\rangle$ ancillas are the type 2 qubits (recall, those in L_2 but not in L_1) and $|0\rangle$ ancillas are the type 1 qubits (those in L_1 but not L_2).

$$\mathcal{G}_X^{(\mathcal{Q}')} = \big\{X^{(L_1)}(S)X^{(L_2)}(T) : G_R S = 0, S \subseteq A \quad T \subseteq \mathbb{1} - A\big\}, \tag{76}$$

$$\mathcal{G}_Z^{(\mathcal{Q}')} = \big\{Z^{(L_1)}(S)Z^{(L_2)}(T) : T G_R^T = 0, T \subseteq A \quad S \subseteq \mathbb{1} - A\big\}. \tag{77}$$

Stabilizers of $\mathcal{Q}'$ include not just the stabilizers of BBS(A), but also single-qubit Pauli Xs on type 2 qubits and single-qubit Pauli Zs on the type 1 qubits. So we have

$$\mathcal{S}_X^{(\mathcal{Q}')} = \big\{X^{(L_2)}(S+T) : G_1 S = 0, S H_R^T = 0, T \subseteq \mathbb{1} - A\big\}, \tag{78}$$

$$\mathcal{S}_Z^{(\mathcal{Q}')} = \big\{Z^{(L_1)}(S+T) : S G_2^T = 0, H_R S = 0, T \subseteq \mathbb{1} - A\big\}. \tag{79}$$

Now it is clear that

$$\mathcal{G}_X^{(\mathcal{Q}')} \leq \mathcal{G}_X^{(\text{aBBS})}, \quad \mathcal{G}_Z^{(\mathcal{Q}')} \leq \mathcal{G}_Z^{(\text{aBBS})}, \tag{80}$$

$$\mathcal{S}_X^{(\text{aBBS})} \leq \mathcal{S}_X^{(\mathcal{Q}')}, \quad \mathcal{S}_Z^{(\text{aBBS})} \leq \mathcal{S}_Z^{(\mathcal{Q}')}. \tag{81}$$

This takes care of part (1) of Definition 8.

Adding ancillas does not change the number of logical qubits in $\mathcal{Q}'$, and so both $\mathcal{Q}'$ and $\mathcal{Q}$ have rank(A) logical qubits, showing part (2) of Definition 8 holds. □

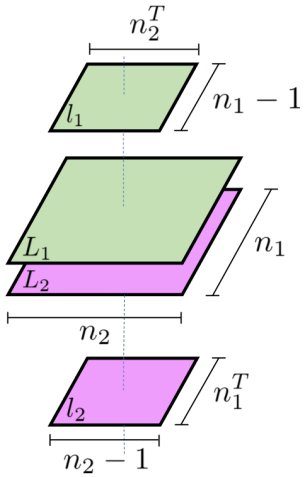

FIG. 5. Gauge-fixing an aBBS code to HGP codes takes place on four lattices of qubits. Each code involved is supported on two lattices – aBBS(A) is supported on L_1 and L_2, HGP(H_R, H_2) on L_1 and l_1, and HGP(H_1, H_R) on L_2 and l_2.

C. Gauge-fixing an aBBS code to hypergraph product codes

In this section, we show that certain hypergraph product codes are gauge-fixings of an aBBS code. Informally, our main result is that for all $A \in \mathbb{F}_2^{n_1 \times n_2}$ both HGP(H_R, H_2) and HGP(H_1, H_R) are gauge-fixings of aBBS(A), where we only require that the rows of $H_1 \in \mathbb{F}_2^{n_1^T \times n_1}$ and $H_2 \in \mathbb{F}_2^{n_2^T \times n_2}$ span $\ker(A^T)$ and $\ker(A)$, respectively.

Just like the case of a BBS code in the last section, to formalize this gauge-fixing we need to define all three of these codes on the same set of physical qubits. Four lattices of qubits are involved, which we label L_1, L_2, l_1, and l_2. The code aBBS(A) is supported on the $n_1 \times n_2$ lattices L_1 and L_2. Recall that qubits in L_1 and L_2 are identified at the positions where $A_{ij} = 1$, so there are $2n_1n_2 - |A|$ total qubits in $L_1 \cup L_2$. The code HGP(H_R, H_2) is supported on lattices L_1 and l_1, thus making l_1 a $(n_1-1)\times n_2^T$ lattice. Similarly, the code HGP(H_1, H_R) is supported on lattices L_2 and l_2, and so l_2 is a $n_1^T \times (n_2 - 1)$ lattice. A schematic of this qubit arrangement is shown in Fig. 5.

Theorem 11. *Let $A \in \mathbb{F}_2^{n_1 \times n_2}$ and $H_1 \in \mathbb{F}_2^{n_1^T \times n_1}$, $H_2 \in \mathbb{F}_2^{n_2^T \times n_2}$ be such that row(H_1) = $\ker(A^T)$, row(H_2) = $\ker(A)$. Then the codes*

$$\mathcal{Q}' = HGP(H_R, H_2)\left|+^{n_1n_2-|A|}\right\rangle\left|\perp^{n_1^T(n_2-1)}\right\rangle, \quad (82)$$

$$\mathcal{Q}'' = HGP(H_1, H_R)\left|0^{n_1n_2-|A|}\right\rangle\left|\perp^{(n_1-1)n_2^T}\right\rangle \quad (83)$$

are gauge-fixings of

$$\mathcal{Q} = aBBS(A)\left|\perp^{n_1^T(n_2-1)+(n_1-1)n_2^T}\right\rangle. \quad (84)$$

Proof. We just show that $\mathcal{Q}'$ is a gauge-fixing of $\mathcal{Q}$. Showing the same for $\mathcal{Q}''$ is analogous. To show this, we do not need lattice l_2 and so omit it when we write down Pauli operators. Indeed, without l_2, code $\mathcal{Q}'$ is a subspace code – its gauge group is its stabilizer group.

Parity check matrices H_1 and H_2 define classical codes $\mathcal{C}_1$ = col(A) and $\mathcal{C}_2$ = row(A), each encoding k = rank(A) bits. These codes have some generating matrices G_1 and G_2 that we will use. Code $\mathcal{C}_2^T$ has a generating matrix F_2. By the discussions in Section II, we have that both $\mathcal{Q}'$ and $\mathcal{Q}$ encode k qubits, thus verifying part (2) of the gauge-fixing definition, Definition 8.

Now, let us write down the stabilizers of $\mathcal{Q}'$ and the gauge group and stabilizers of $\mathcal{Q}$. These follow from the appropriate equations in Section II, but with additions due to the ancillas: $|+\rangle$ ancillas in $L_2 - L_1$ for $\mathcal{Q}'$ and $|\perp\rangle$ ancillas in l_1 for $\mathcal{Q}$.

By Eq. (48), $\mathcal{S}_X^{(\mathcal{Q}')} = \{X^{(L_1)}(S_1)X^{(L_2)}(S_2)X^{(l_1)}(T) : S_1H_2^T = H_R^TT,\ G_RS_1 = 0,\ TF_2^T = 0,\ S_2 \subseteq \mathbb{1} - A\}$, (85)

Eq. (49), $\mathcal{S}_Z^{(\mathcal{Q}')} = \{Z^{(L_1)}(S)Z^{(l_1)}(T) : H_RS = TH_2,\ SG_2^T = 0\}$, (86)

Eq. (35), $\mathcal{G}_X^{(\mathcal{Q})} = \{X^{(L_1)}(S_1)X^{(L_2)}(S_2)X^{(l_1)}(T) : G_RS_1 = 0,\ S_2 \subseteq \mathbb{1} - A\}$, (87)

Eq. (36), $\mathcal{G}_Z^{(\mathcal{Q})} = \{Z^{(L_1)}(S_1)Z^{(L_2)}(S_2)Z^{(l_1)}(T) : S_2G_R^T = 0,\ S_1 \subseteq \mathbb{1} - A\}$, (88)

Eq. (43), $\mathcal{S}_X^{(\mathcal{Q})} = \{X^{(L_2)}(S) : SH_R^T = 0,\ G_1S = 0\}$, (89)

Eq. (44), $\mathcal{S}_Z^{(\mathcal{Q})} = \{Z^{(L_1)}(S) : H_RS = 0,\ SG_2^T = 0\}$. (90)

To show part (1) of Definition 8, we have four inclusions to prove: (a) $\mathcal{S}_X^{(\mathcal{Q}')} \subseteq \mathcal{G}_X^{(\mathcal{Q})}$, (b) $\mathcal{S}_Z^{(\mathcal{Q})} \subseteq \mathcal{S}_Z^{(\mathcal{Q}')}$, (c) $\mathcal{S}_Z^{(\mathcal{Q}')} \subseteq \mathcal{G}_Z^{(\mathcal{Q})}$, (d) $\mathcal{S}_X^{(\mathcal{Q})} \subseteq \mathcal{S}_X^{(\mathcal{Q}')}$.

Both inclusions (a) and (b) are obvious, so we focus on (c) and (d). For (c), let $M = Z^{(L_1)}(S)Z^{(l_1)}(T) \in \mathcal{S}_Z^{(\mathcal{Q}')}$. Set $S_1 = S \cap (\mathbb{1} - A)$ and $S_2 = S \cap A$, so that $M = Z^{(L_1)}(S_1)Z^{(L_2)}(S_2)Z^{(l_1)}(T)$. Now $SG_2^T = 0$ implies that rows of S are parity checks of code $\mathcal{C}_2$. Since rows of A are codewords of $\mathcal{C}_2$, each row of $S_2 = S \cap A$ contains

an even number of 1s. Thus, $S_2 = S_2 G_R^T = 0$, and so $M \in \mathcal{G}_Z^{(\mathcal{Q})}$.

For (d), let $M = X^{(L_2)}(S) \in \mathcal{S}_X^{(\mathcal{Q})}$. Set $S_1 = S \cap A$ and $S_2 = S \cap (\mathbb{1} - A)$. Since $G_1 S = 0$, columns of S are parity checks of $\mathcal{C}_1$. Columns of A are codewords of $\mathcal{C}_1$, and so each column of S_1 contains an even number of 1s, or $G_R S_1 = 0$. Similarly, $S H_R^T = 0$ implies that rows of S are codewords of $\mathcal{C}_R$, i.e. all 1s or all 0s. Therefore, $\mathrm{row}(S_1) \subseteq \mathrm{row}(A) = \mathcal{C}_2$ and $S_1 H_2^T = 0$. This shows $M = X^{(L_1)}(S_1) X^{(L_2)}(S_2) X^{(l_1)}(0) \in \mathcal{S}_X^{(\mathcal{Q}')}$. □

A special case of Theorem 11 is the gauge-fixing of the Bacon-Shor code $\mathrm{BBS}(\mathbb{1}) = \mathrm{aBBS}(\mathbb{1})$ (see Example 1) to the surface code $\mathrm{HGP}(H_R, H_R)$ (see Example 3 in the Appendix).

Let us conclude this section by briefly discussing the code $\mathrm{HGP}(H_1, H_R)$ that we just showed is a gauge-fixing of $\mathrm{aBBS}(A)$. In particular, we would like to argue that it has an asymptotic threshold when H_1 is chosen appropriately. Kovalev and Pryadko [19] have shown that any $[\![N, K, D]\!]$ quantum code family that is (β, γ)-LDPC for constants β and γ and has distance scaling at least logarithmically in code size, i.e. $D = \Omega(\log N)$, possesses an asymptotic threshold. Say that H_1 is a full-rank, (b, c)-LDPC set of parity checks for code $\mathcal{C}_1$ with parameters $[n, k, d]$ and H_R represents the length n repetition code. Then, $\mathrm{HGP}(H_1, H_R)$ is (γ, γ)-LDPC for $\gamma = \max(b, c)+2$ and has parameters $[\![N, k, d]\!]$ with $N \leq 2n^2$. Clearly then, if $\mathcal{C}_1$ is an LDPC code family with d scaling at least logarithmically in n, i.e. $d = \Omega(\log n)$, then by [19] the quantum code family $\mathrm{HGP}(H_1, H_R)$ has an asymptotic threshold.

V. DISCUSSION

We have presented another connection between classical and quantum error-correction and discussed one of its consequences, the construction of Bravyi-Bacon-Shor subsystem codes that are local in 2-dimensions and have optimal parameters. We also showed a somewhat surprising connection between Bravyi-Bacon-Shor codes and the hypergraph product codes via the process of gauge-fixing.

One of the consequences of our results is the ability to gauge-switch between several hypergraph product codes. For example, one can switch between $\mathrm{HGP}(H_R, H_2)$ and $\mathrm{HGP}(H_1, H_R)$ for any H_1 and H_2 or between $\mathrm{HGP}(H_1, H_R)$ and $\mathrm{HGP}(H_1', H_R)$ where H_1 and H_1' are different parity check matrices for the same classical code. In the process, encoded data is protected by the underlying augmented Bravyi-Bacon-Shor code (see Theorem 11), which has the same code distance as the hypergraph product codes in question although it lacks an asymptotic threshold. Nonetheless, generalizing this gauge-switching idea to more hypergraph product codes would be an interesting extension of our work here.

ACKNOWLEDGEMENTS

The author gratefully acknowledges helpful discussions with Sergey Bravyi, Ken Brown, Chris Chamberland, and Andrew Cross. Partial support for this project was generously provided by the IBM Research Frontiers Institute.

Appendix A: Hypergraph product codes

In this appendix, we review the original presentation of hypergraph product codes [7] and verify that our description in Section II D is equivalent. We also review the derivation of the hypergraph product code parameters. Mainly, our arguments are similar to those in [7] and [16].

Recall that the input to the construction is two parity check matrices $H_1 \in \mathbb{F}_2^{n_1^T \times n_1}$ and $H_2 \in \mathbb{F}_2^{n_2^T \times n_2}$. These have corresponding full-rank generating matrices $G_1 \in \mathbb{F}_2^{k_1 \times n_1}$ and $G_2 \in \mathbb{F}_2^{k_2 \times n_2}$ for the classical codes $\mathcal{C}_1$ and $\mathcal{C}_2$. Additionally, there are full-rank generating matrices $F_1 \in \mathbb{F}_2^{k_1^T \times n_1^T}$ and $F_2 \in \mathbb{F}_2^{k_2^T \times n_2^T}$ for the transpose classical codes $\mathcal{C}_1^T$ and $\mathcal{C}_2^T$.

In the original description, the supports of Pauli operators are specified by vectors from $\mathbb{F}_2^N$ with $N = n_1 n_2 + n_1^T n_2^T$. Generating sets of X- and Z-type stabilizers are presented as rows of matrices:

$$S_X = \begin{pmatrix} H_1 \otimes I_{n_2} & I_{n_1^T} \otimes H_2^T \end{pmatrix}, \tag{A1}$$

$$S_Z = \begin{pmatrix} I_{n_1} \otimes H_2 & H_1^T \otimes I_{n_2^T} \end{pmatrix}, \tag{A2}$$

where I_n is the $n \times n$ identity matrix. That is, if $X^{\vec{v}} = \prod_{i=1}^N X_i^{\vec{v}_i}$ and we wanted to write out the entire sets of Pauli stabilizers, we would have

$$\mathcal{S}_X = \left\{ X^{\vec{v}} : \vec{v} \in \mathrm{row}(S_X) \right\}, \tag{A3}$$

$$\mathcal{S}_Z = \left\{ Z^{\vec{u}} : \vec{u} \in \mathrm{row}(S_Z) \right\}. \tag{A4}$$

It is easy to see that these stabilizers commute, because $S_X S_Z^T = 0$. Moreover, from the generating sets in Eqs. (A1), (A2), we note that using classical LDPC parity checks H_1 and H_2 lead to a quantum LDPC code with the appropriate parameters from Eqs. (57), (58).

We can calculate the number of encoded qubits by finding the number of independent stabilizer generators $\mathrm{rank}(S_X) + \mathrm{rank}(S_Z)$ and subtracting that from N. Basic linear algebra says

$$\mathrm{rank}(S_X) = \mathrm{rank}(S_X^T) = n_1^T n_2 - \dim(\ker(S_X^T)). \tag{A5}$$

Since

$$S_X^T = \begin{pmatrix} H_1 \otimes I_{n_2} \\ I_{n_1^T} \otimes H_2^T \end{pmatrix} \tag{A6}$$

has kernel

$$\ker(S_X^T) = \{ x \otimes y : x \in \mathcal{C}_1, y \in \mathcal{C}_2^T \}, \tag{A7}$$

we see that $\dim(\ker(S_X^T)) = \dim(\mathcal{C}_1)\dim(\mathcal{C}_2^T) = k_1 k_2^T$. A similar argument holds for S_Z. Thus, we have

$$\text{rank}(S_X) = n_1^T n_2 - k_1 k_2^T, \tag{A8}$$
$$\text{rank}(S_Z) = n_1 n_2^T - k_1^T k_2. \tag{A9}$$

Accordingly, the hypergraph product code encodes

$$K = N - (n_1^T n_2 - k_1 k_2^T) - (n_1 n_2^T - k_1^T k_2) \tag{A10}$$
$$= (n_1 - n_1^T)(n_2 - n_2^T) + k_1 k_2^T + k_1^T k_2 \tag{A11}$$
$$= (k_1 - k_1^T)(k_2 - k_2^T) + k_1 k_2^T + k_1^T k_2 \tag{A12}$$
$$= k_1 k_2 + k_1^T k_2^T \tag{A13}$$

qubits. For the third equality, we used Eq. (3). This verifies Eq. (55).

Let us create a generating set of logical operators for these qubits. We notice that

$$L_X = \begin{pmatrix} H_1 \otimes I_{n_2} & I_{n_1^T} \otimes H_2^T \\ I_{n_1} \otimes G_2 & 0 \\ 0 & F_1 \otimes I_{n_2^T} \end{pmatrix}, \tag{A14}$$

$$L_Z = \begin{pmatrix} I_{n_1} \otimes H_2 & H_1^T \otimes I_{n_2^T} \\ G_1 \otimes I_{n_2} & 0 \\ 0 & I_{n_1^T} \otimes F_2 \end{pmatrix} \tag{A15}$$

do in fact provide sets of logical operators because $S_Z L_X^T = 0$ and $S_X L_Z^T = 0$ demonstrate the appropriate commutation.

To show that these are indeed complete sets of logical operators, we can calculate the rank of $C = L_X L_Z^T$, which encodes how the X- and Z-type logical operators commute. There should be K independent, anti-commuting pairs of logical operators, so the rank of C should be K. Since

$$C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & G_1^T \otimes G_2 & 0 \\ 0 & 0 & F_1 \otimes F_2^T \end{pmatrix}, \tag{A16}$$

we do have

$$\text{rank}(C) = \text{rank}(G_1)\text{rank}(G_2) + \text{rank}(F_1)\text{rank}(F_2) \tag{A17}$$
$$= k_1 k_2 + k_1^T k_2^T = K. \tag{A18}$$

Now consider "reshaping" [16] the vectors that represent Paulis into matrices. Let $\hat{e}_i$ be the unit vector $(\hat{e}_i)_j = \delta_{ij}$. A vector $\vec{s} \in \mathbb{F}_2^{n_1 n_2}$ can be decomposed as

$$\vec{s} = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} S_{ij} \hat{e}_i \otimes \hat{e}_j \tag{A19}$$

where S is the matrix corresponding to $\vec{s}$ and the support of Pauli $X^{\vec{s}}$ once we have placed it on the $n_1 \times n_2$ lattice L. We previously wrote this Pauli as $X^{(L)}(S)$. Likewise, vectors $\vec{t} \in \mathbb{F}_2^{n_1^T n_2^T}$ are reshaped to represent Paulis on the $n_1^T \times n_2^T$ lattice l.

Linear transformations of $\vec{s}$ correspond to matrix multiplications on S. By Eq. (A19),

$$(U \otimes V)\vec{s} \longmapsto USV^T. \tag{A20}$$

Likewise with transformations on $\vec{t}$.

At this point we can justify our presentation of the stabilizers and logical operators, Eqs. (48, 49) and (52, 53) in the main text. We can characterize elements of $\text{row}(S_X)$ by the fact that they commute with all rows of L_Z.

$$\begin{pmatrix} \vec{s} \\ \vec{t} \end{pmatrix} \in \text{row}(S_X) \text{ iff } L_Z \begin{pmatrix} \vec{s} \\ \vec{t} \end{pmatrix} = \vec{0}. \tag{A21}$$

Reshaping the linear equations on the right using Eq. (A20) gives the equations

$$SH_2^T = H_1^T T, \quad G_1 S = 0, \quad T F_2^T = 0, \tag{A22}$$

which are exactly the conditions on S and T in $\mathcal{S}_X^{(\text{hgp})}$, Eq. (48).

Similarly, elements of $\text{row}(S_Z)$ are characterized by commutation with rows of L_X, elements of $\text{row}(L_X)$ by commutation with rows of S_Z, and elements of $\text{row}(L_Z)$ by commutation with rows of S_X. After reshaping the appropriate linear equations, one can confirm Eqs. (49, 52, 53).

Finally, we prove that the hypergraph product code has the claimed distance $D = \min(d_1, d_2, d_1^T, d_2^T)$, involving the distances of all the codes $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_1^T$, and $\mathcal{C}_2^T$. If any of these codes encode no bits, its code distance is defined to be infinite.

We begin by bounding the weight of nontrivial X-type logical operators, those elements of $\mathcal{L}_X^{(\text{hgp})} - \mathcal{S}_X^{(\text{hgp})}$. If $M = X^{(L)}(S)X^{(l)}(T)$, then $SH_2^T = H_1^T T$ and there is an $M' \in \mathcal{L}_Z^{(\text{hgp})} - \mathcal{S}_Z^{(\text{hgp})}$ that anticommutes with M. In fact, given the basis in L_Z, we know something about the form of M' – it corresponds either to a row of $G_1 \otimes I_{n_2}$ (case (1)) or to a row of $I_{n_1^T} \otimes F_2$ (case (2)).

In case (1), we can take $M' = X^{(L)}(S')$ where S' is an outer product $S' = \vec{c}\,\hat{e}_j^T$ for some $\vec{c} \in \mathcal{C}_1$ and some j. As M and M' anticommute,

$$1 = \text{tr}\left(S^T S'\right) = \hat{e}_j^T S^T \vec{c} \tag{A23}$$

and clearly $S^T\vec{c} \neq \vec{0}$. Now, $H_2 S^T \vec{c} = T^T H_1 \vec{c} = 0$ and thus $S^T\vec{c}$ is a nonzero vector in $\ker(H_2) = \mathcal{C}_2$. Therefore, $|M| \geq |S| = |S^T| \geq |S^T\vec{c}| \geq d_2$.

In case (2), the argument is analogous. Take $M' = X^{(l)}(T')$ where T' is the outer product $T' = \hat{e}_i \vec{b}^T$ for some $\vec{b} \in \mathcal{C}_2^T$ and some i. As M and M' anticommute,

$$1 = \text{tr}\left(T^T T'\right) = \vec{b}^T T^T \hat{e}_i \tag{A24}$$

and clearly $\vec{b}^T T^T \neq \vec{0}$. Also, $\vec{b}^T T^T H_1 = \vec{b}^T H_2 S^T = \vec{0}^T$ and so $T\vec{b}$ is a nonzero vector in $\ker(H_1^T) = \mathcal{C}_1^T$. Thus, $|M| \geq |T| \geq |T\vec{b}| \geq d_1^T$.

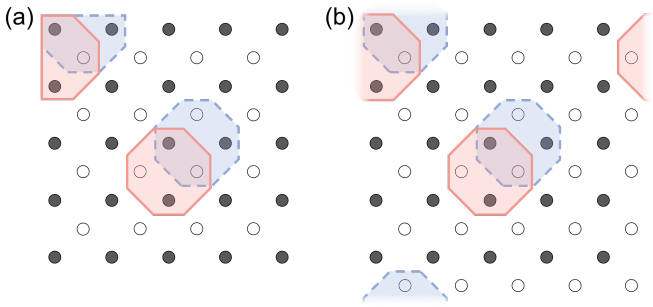

FIG. 6. The surface code (a) with boundary and (b) on the torus drawn on the L (filled qubits) and l (unfilled qubits) lattices of the hypergraph product. Some example X- (red, solid) and Z-type (blue, dashed) stabilizers are shown. These example stabilizers correspond to select rows of the matrices S_X and S_Z of the appropriate hypergraph products.

From these two cases, we conclude $|M| \geq \min(d_1^T, d_2)$. If we go through the analogous argument for non-trivial Z-type logical operators, we would find their weight bounded below by $\min(d_1, d_2^T)$. Thus, the code distance of the hypergraph product code is $D \geq \min(d_1, d_2, d_1^T, d_2^T)$. By looking at L_X and L_Z, however, we see that there are indeed logical operators saturating this inequality, and so $D = \min(d_1, d_2, d_1^T, d_2^T)$ as claimed in Eq. (56).

We conclude this appendix by reviewing the surface code as a special case of the hypergraph product. In fact, there are two versions of the surface code that can be made: the one with boundary [10] and the one on a torus [11].

Example 3. *The surface code with boundary [10] is an $[\![n^2+(n-1)^2, 1, n]\!]$ code. These parameters match those of $HGP(H_R, H_R)$. Indeed, we draw some of the stabilizers indicated by rows of S_X and S_Z in Fig. 6(a), in which one can recognize the surface code.*

Example 4. *The surface code on a torus [11] is a $[\![2n^2, 2, n]\!]$ code, matching the parameters of $HGP(H'_R, H'_R)$ for*

$$H'_R = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 & 0 \\ & \ddots & & \ddots & & \ddots & \\ 0 & 0 & 0 & \dots & 0 & 1 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}. \tag{A25}$$

This is an over-complete parity check matrix for the $[n, 1, n]$ classical repetition code – the sum of all rows is $\vec{0}$. Notice the transpose code is also the $[n, 1, n]$ repetition code. We draw some of the stabilizers corresponding to rows of S_X and S_Z in Fig. 6(b) in which one can recognize the surface code on the torus.

Appendix B: Expander codes

Constructions of good families of classical LDPC codes based on expander graphs are known. In this section, we review the segment of expander theory that is needed to prove the goodness of these codes, and therefore the goodness of Bravyi-Bacon-Shor codes constructed from them. All of this section is classical and we expect to do no more than inform any uninitiated readers of what is already known.

1. Construction

The objects used to construct good classical LDPC codes are called lossless-expanders [20], though we will refer to them simply as expanders. Mathematically, these expanders are undirected, bipartite graphs, which we will represent by a tuple (L, R, E) of left nodes, right nodes, and edges. A node v has a degree, the number of edges incident to it, which we denote $\deg v$. Given a set of nodes $V \subseteq L \cup R$, we can talk about its set of neighbors

$$\Gamma(V) = \{u : \exists v \in V \text{ s.t. } (u, v) \in E\}. \tag{B1}$$

Expanders attempt to maximize the size of $\Gamma(S)$ for all $S \subseteq L$ sufficiently small.

Definition 12 (Expanders)**.** A $(n, m, b, \delta, \epsilon)$ expander is a bipartite graph (L, R, E) satisfying

Size: $|L| = n$, $|R| = m$,

Degree: $\forall v \in L$, $\deg(v) = b$, $\forall w \in R$, $\deg(w) = c = nb/m$,

Expansion: $\forall S \subseteq L$ s.t. $|S| \leq (1-\delta)n$, $(1-\epsilon)b|S| \leq |\Gamma(S)| \leq b|S|$.

In particular, expanders with smaller δ and ϵ are better than those with larger values. The expansion property is trivial if $\delta > 1-2/n$ for instance. Moreover, if $\delta \leq 1-2/n$ and $\epsilon = 0$, only the graph with $m = nb$ right-nodes and n connected components suffices to meet the definition. Finally, $b = 1$ or $c = nb/m = 1$ lead to similar trivialities. Thus, we take $\delta \leq 1-2/n$, $\epsilon > 0$, and $b, c > 1$ throughout.

From the definition, one can prove other facts about expanders. One very useful fact for us concerns the size of the set of "unique" neighbors,

$$\Gamma_1(V) = \{u \in \Gamma(V) : |\Gamma(\{u\}) \cap S| = 1\}. \tag{B2}$$

Elements of $\Gamma_1(V)$ are the elements of $\Gamma(V)$ that have just one neighbor in V.

Lemma 13. *Suppose the bipartite graph (L, R, E) is an $(n, m, b, \delta, \epsilon)$ expander and $S \subseteq L$ satisfies $|S| \leq (1-\delta)n$. Then*

$$\Gamma_1(S) \geq (1-2\epsilon)b|S|. \tag{B3}$$

Proof. The number of edges leaving S is $b|S|$. This is the same as the number of edges entering S from $\Gamma(S)$. The nodes in $\Gamma_1(S) \subseteq \Gamma(S)$ have exactly 1 such edge, while those in $\Gamma_{\geq 2}(S) = \Gamma(S) - \Gamma_1(S)$ have at least 2 such edges. Thus,

$$\begin{aligned} b|S| &\geq 2|\Gamma_{\geq 2}(S)| + |\Gamma_1(S)| && \text{(B4)} \\ &= |\Gamma_{\geq 2}(S)| + |\Gamma(S)| && \text{(B5)} \\ &= 2|\Gamma(S)| - |\Gamma_1(S)| && \text{(B6)} \\ &\geq 2(1-\epsilon)b|S| - |\Gamma_1(S)|, && \text{(B7)} \end{aligned}$$

where the last inequality uses the expansion property. □

To create a classical code from an expander, we will use (the simplest version of) Tanner's construction [21]. This prescribes that we view the left nodes L as a set of code bits and each right node as specifying a parity check on the bits that are its neighbors. More precisely, define the incidence matrix $\Lambda \in \mathbb{F}_2^{|L|\times|R|}$ of a bipartite graph $G = (L, R, E)$ as

$$\Lambda_{uv} = \begin{cases} 0, & (u,v) \notin E \\ 1, & (u,v) \in E \end{cases}. \tag{B8}$$

Then, $H = \Lambda^T$ takes the role of a parity check matrix to define the Tanner code of G, $\mathcal{C}_G = \ker(H)$.

If G is an expander, we call $\mathcal{C}_G$ an expander code. In this case, we can place useful bounds on its code parameters.

Lemma 14. *Suppose $G = (L, R, E)$ is an $(n, m, b, \delta, \epsilon)$ expander with $\epsilon < 1/2$. Then $\mathcal{C}_G$ is a $[n, k, d]$ code with $k \geq n - m$ and $d \geq 2(1-\epsilon)\lfloor(1-\delta)n\rfloor$.*

Proof. The parity check matrix H of $\mathcal{C}_G$ has m rows, and thus its kernel is at least $n - m$ dimensional. So, $k \geq n - m$.

Let $\vec{s} \in \mathbb{F}_2^n$ be a bit string and $S = \{v : \vec{s}_v = 1\} \subseteq L$ be its support. We show that if $|\vec{s}| = |S| < 2(1-\epsilon)\lfloor(1-\delta)n\rfloor$, then there must be a parity check unsatisfied by $\vec{s}$, and so $\vec{s}$ is not a codeword. To do this, it is sufficient to show that $\Gamma_1(S)$ is not empty – any $w \in \Gamma_1(S)$ cannot be a satisfied check as only a single bit in the check is 1.

Suppose first that $|S| \leq (1-\delta)n$. Then by Lemma 13, we have $|\Gamma_1(S)| \geq (1-2\epsilon)b|S| > 0$, using the assumption $\epsilon < 1/2$.

Now suppose $(1-\delta)n < |S| < 2(1-\epsilon)\Delta$ where $\Delta = \lfloor(1-\delta)n\rfloor$. Let $T \subseteq S$ satisfy $|T| = \Delta$. So,

$$\Gamma_1(T) \geq (1-2\epsilon)b\Delta \tag{B9}$$

by Lemma 13. At the same time $|S - T| = |S| - |T| < (1-2\epsilon)\Delta < \Delta$ implies

$$|\Gamma(S-T)| < (1-2\epsilon)b\Delta, \tag{B10}$$

because nodes in $S - T$ are degree b. A check w is in $\Gamma_1(S)$ if $w \in \Gamma_1(T)$ and $w \notin \Gamma(S-T)$. Since $|\Gamma_1(T)| > |\Gamma(S-T)|$ by Eqs. (B9), (B10), we have $|\Gamma_1(S)| > 0$. □

It is worth noting when a family of expander codes $[n, k, d]$ is good, i.e. $k = \Theta(n)$ and $d = \Theta(n)$. Using Lemma 14, it is sufficient that ϵ, δ are constant (independent of n) and that $m/n = b/c$ is constant. It is typical to construct families in which b (the degree of nodes on the left) and c (the degree of nodes on the right) are both constant individually. This makes the code a low-density parity check code and also enables the efficient decoder discussed in the next section.

Lemma 14 assumes $\epsilon < 1/2$ which means it is only sufficient for analyzing expander codes constructed from expanders with sufficiently large expansion. For a long time, although expanders of arbitrarily large size with $\epsilon < 1/2$ were known to exist by counting, it was not known how to construct them. However, the zig-zag construction [20] eventually solved this problem. For our purposes, a suitable distillation of their result is the following.

Theorem 15 (Hoory, Linial, Wigderson [22], Thm. 10.4)**.** *For every $\epsilon > 0$ and $\alpha \in (0, 1)$, there exist constants γ, σ and an explicit family of $(n, m, b, \delta, \epsilon)$ expanders with $m = \alpha n$,*

$$b \leq \left(\frac{1}{\epsilon\alpha}\right)^{\gamma}, \tag{B11}$$

$$\delta \leq 1 - \sigma\,(\epsilon\alpha)^{\gamma+1}. \tag{B12}$$

Using Lemma 14, the corresponding expander codes have parameters $[n, k, d]$ with

$$k \geq (1-\alpha)n, \tag{B13}$$

$$d \geq 2(1-\epsilon)\lfloor\sigma(\epsilon\alpha)^{\gamma+1}n\rfloor. \tag{B14}$$

Theorem 15 is a theoretically important result – it provides a construction of a good family of classical codes, and moreover the parity checks involve only constant numbers of bits. However, the constants involved may not be the most practical, and random instances of bipartite graphs, like those analyzed in the Appendix of [14] or in Theorem 8.7 of [23], may be less cumbersome to work with.

2. Decoding

Sipser and Spielman [14] analyzed a decoder for classical expander codes that operates in greedy fashion by flipping any bits that overall reduce the number of unsatisfied parity checks. We will refer to this as the flip decoder. They show that for expanders with sufficiently large expansion ($\epsilon < 1/4$) the flip decoder corrects any number of errors within a constant fraction of the code distance and does so in time proportional to the code size, i.e. in linear time. Later Spielman [15] analyzed the flip decoder in the scenario that the parity checks are noisy in addition to the bits. It is this latter scenario that is most relevant to the quantum case where we may only

noisily measure parity checks and not the data qubits themselves. We provide a somewhat generalized presentation of Spielman's analysis here.

Let $\hat{e}_i$ denote the vector with elements $(\hat{e}_i)_j = \delta_{ij}$. Here it represents a flip of the i^{th} bit. The flip decoder is defined as follows.

Definition 16 (Sipser-Spielman Flip Decoder [14, 15])**.** Given an expander code $\mathcal{C}$ with parity check matrix $H \in \mathbb{F}_2^{m\times n}$ and a vector indicating unsatisfied checks $\vec{u} \in \mathbb{F}_2^m$, return a set of corrections $\vec{e}' \in \mathbb{F}_2^n$ by doing the following.

(1) Initialize $\vec{e}' = 0^n$ and $\vec{u}' = \vec{u}$.

(2) Repeat

(a) Find $i \in \{1, 2 \ldots, n\}$ such that $|\vec{u}'| > |H\hat{e}_i - \vec{u}'|$. If none exists, return $\vec{e}'$.

(b) Let $\vec{e}' \leftarrow \vec{e}' + \hat{e}_i$ and $\vec{u}' \leftarrow H\hat{e}_i - \vec{u}'$.

Steps (2a) and (2b) constitute a decoding "round".

Since the number of unsatisfied checks $|\vec{u}'|$ decreases each round and there are $O(n)$ checks in a $[n, k, d]$ expander code, it is somewhat reasonable to believe that this decoder takes linear time.

Lemma 17 (Sipser and Spielman [14])**.** *Let $\mathcal{C}$ be an $[n, k, d]$ expander code based on an $(n, m, b, \delta, \epsilon)$ expander graph with b and m/n constant. The flip decoder for $\mathcal{C}$ runs in time $O(n)$.*

Proof. Proving this simply requires a suitable data structure. We assume that the adjacency matrix of the expander (or equivalently the check matrix of the code) is given in a sparse matrix representation, so it takes constant time to obtain a list of neighbors of a bit or check in the expander graph.

Recall $\vec{u} \in \mathbb{F}_2^m$ is given as the value of the m parity checks. At the beginning of the decoding, we calculate for each bit i the number v_i of unsatisfied checks that it is involved in. This takes $O(bn) = O(n)$ total time. We construct $b + 1$ linked lists, one for each possible value of v_i, and place each i in the corresponding list. That is, for each $i \in \{1, 2, \ldots, n\}$, we store $\{i, v_i, p_i, n_i\}$, where $p_i, n_i \in \{1, 2, \ldots, n\}$ point to the previous and next elements in the linked list (or are null if i is at the head or tail). Variables $h_v \in \{1, 2, \ldots, n\}$ for every $v \in \{0, 1, \ldots, b\}$ point to the linked list heads (or null if the list is empty). The initial set up of p_i, n_i, and h_v values takes $O(n)$ time. It is also important to note that removing from and attaching to the front of linked lists take $O(1)$ time.

The main body of the flip decoding algorithm is the iteration in Step (2) of Definition 16. Since the number of unsatisfied clauses strictly decreases during each round, there are at most $O(m) = O(n)$ rounds. Moreover, each round can be made to take constant time, as we now show.

Every round the algorithm begins by finding the non-null h_v with largest v. This takes $O(b)$ time. If $0 \le v \le b/2$, then there is no bit to flip to reduce the number of unsatisfied clauses and the algorithm returns. If $v > b/2$, then flip bit h_v. This causes $b = O(1)$ checks j to flip and we update the values u_j accordingly. Within each of the flipped checks are $c = O(1)$ bits i which now participate in either one more or one fewer unsatisfied check. The values v_i should be updated accordingly and the linked list element $\{i, v_i, p_i, n_i\}$ removed from its current linked list and inserted at the head of list h_{v_i}, which takes $O(1)$ time. Thus, the entire round takes $O(1)$ time. □

Presently, we concern ourselves with how well the decoder corrects errors. The main result is that the number of errors on the data can be reduced to a constant fraction of the number of errors on the checks.

Theorem 18 (Spielman [15])**.** *Let $\mathcal{C}$ be an expander code constructed from a $(n, m, b, \delta, \epsilon)$ expander with $\epsilon < \frac{1}{4} - \frac{r}{b}$ for $1 \le r < b/4$. Given input $\vec{u} = H(\vec{s}_0 + \vec{e}) + \vec{f}$ for $\vec{s}_0 \in \mathcal{C}$ and provided*

$$|\vec{e}| + \frac{2}{b}|\vec{f}| \le (1 - 2\epsilon)\lfloor(1 - \delta)n\rfloor, \tag{B15}$$

the noisy flip decoder returns $\vec{e}'$ such that $|\vec{e}' - \vec{e}| < |\vec{f}|/r$.

Proof. Let $E = \{i : \vec{e}_i + \vec{e}'_i = 1\}$ be the set of corrupted message bits and $U = \{j : \vec{u}'_j = 1\}$ be the set of unsatisfied checks at any point during execution of the algorithm. Let $S = \Gamma(E) - U$ be the satisfied checks in the neighborhood of E. Provided $|E| = |\vec{e}' - \vec{e}| \le (1 - \delta)n$, the expansion property implies

$$|U| + |S| \ge |\Gamma(E)| \ge (1 - \epsilon)b|E|. \tag{B16}$$

This gives a lower bound on $|U|$ and $|S|$.

We can get an upper bound on these by a counting argument. Imagine we add m additional nodes to the left side of the bipartite expander graph and connect these new nodes pairwise to the corresponding m check nodes on the right side. These new nodes represent the presence (if set to 1) or absence (if set to 0) of an error on the check bit. So, of these new nodes, $|\vec{f}|$ are set to 1, those in the set $F = \{j + n : \vec{f}_j = 1\}$. Now every check in U is connected to at least one node in $E \cup F$ and every check in S is connected to at least two nodes in $E \cup F$. Since there are $b|E| + |\vec{f}|$ edges leaving $E \cup F$, we have

$$b|E| + |\vec{f}| \ge |U| + 2|S|. \tag{B17}$$

Combine Eqs. (B16), (B17) to get

$$(1 - \epsilon)b|E| - |U| \le |S| \le \frac{1}{2}(b|E| + |\vec{f}| - |U|), \tag{B18}$$

or, removing $|S|$ entirely and using $\epsilon < \frac{1}{4} - \frac{r}{b}$,

$$\left(\frac{1}{2}b + 2r\right)|E| < (1 - 2\epsilon)b|E| \le |\vec{f}| + |U|. \tag{B19}$$

Thus, if $|\vec{f}|/r \leq |E| \leq (1-\delta)n$, then

$$|U| > \frac{1}{2}b|E| + |\vec{f}|. \tag{B20}$$

If $u_x = |\Gamma(x) \cap U|$ for $x \in E$ is the number of unsatisfied checks that x participates in, then clearly

$$|\vec{f}| + \sum_{x \in E} u_x \geq |U| > \frac{1}{2}b|E| + |\vec{f}|, \tag{B21}$$

or, simply,

$$\frac{1}{|E|}\sum_{x \in E} u_x > \frac{1}{2}b, \tag{B22}$$

implying that there exists $y \in E$ such that $u_y > b/2$. Thus, there is always a bit to flip in step (2a) provided $|\vec{f}|/r \leq |E| \leq (1-\delta)n$.

We complete the proof by showing that $|E| \leq (1-\delta)n$ always holds and therefore the flip algorithm only finishes if $|E| = |\vec{e}' - \vec{e}| < |\vec{f}|/r$.

The noisy flip algorithm flips one bit at a time and $|E| < \lfloor (1-\delta)n \rfloor$ at the beginning of the algorithm, so if $|E| > (1-\delta)n$ at some time, then there is a prior time at which $|E| = \lfloor (1-\delta)n \rfloor$. Then, we can apply Eq. (B19) to find

$$|U| \geq (1-2\epsilon)b\lfloor (1-\delta)n \rfloor - |\vec{f}|. \tag{B23}$$

Let U_0 denote U at the very start of the algorithm (i.e. when $\vec{e}' = 0^n$ and $|E| = |\vec{e}|$). By Eq. (B17), we see

$$|U_0| \leq b|\vec{e}| + |\vec{f}|. \tag{B24}$$

Moreover, the intermediate rounds of the algorithm always decrease the size of U. So, $|U_0| > |U|$ and hence

$$b|\vec{e}| + 2|\vec{f}| > (1-2\epsilon)b\lfloor (1-\delta)n \rfloor. \tag{B25}$$

However, this is in contradiction with Eq. (B15). □

We briefly remark that although $(1-2\epsilon)\lfloor (1-\delta)n \rfloor < d/2$ by Lemma 14, it is not much less than the lower bound on $d/2$ from that lemma. The difference is the factor $(1-2\epsilon)/(1-\epsilon)$, which is constant and near unity when ϵ is constant and small. Also, since $|\vec{e}| + |\vec{f}| \geq |\vec{e}| + \frac{2}{b}|\vec{f}|$, the assumption

$$|\vec{e}| + |\vec{f}| \leq (1-2\epsilon)\lfloor (1-\delta)n \rfloor \tag{B26}$$

is a weaker replacement for Eq. (B15), but one that makes the total number of errors $|\vec{e}| + |\vec{f}|$ more prominent.

This theorem implies that errors can be kept at a manageable level over time. A simple model of data storage is one in which we periodically error correct based on noisy readout of the parity checks, and noise on the data occurs in between these corrections. Suppose at most $|\vec{e}|$ errors occur on the data between corrections and during correction at most $|\vec{f}|$ parity checks are misread. Then, after correction, Theorem 18 guarantees at most $|\vec{f}|/r$ errors remaining on the data. These errors combine with the $|\vec{e}|$ data errors in the next step. Thus, a steady state is achieved – following any correction the data has at most $|\vec{f}|/r$ errors provided that

$$|\vec{e}| + \frac{1}{r}|\vec{f}| + \frac{2}{b}|\vec{f}| \leq (1-2\epsilon)\lfloor (1-\delta)n \rfloor. \tag{B27}$$

It is sufficient (though weaker) for

$$|\vec{e}| + 2|\vec{f}| \leq (1-2\epsilon)\lfloor (1-\delta)n \rfloor. \tag{B28}$$

In this classical scenario, assuming constant error rates, $|\vec{e}|$ and $|\vec{f}|$ both scale linearly with n and so this condition is realistically achievable, even asymptotically.

Appendix C: Proof of Lemma 9

A subsystem code's distance is the minimum weight of a dressed logical operator. Thus, to show $D(\mathcal{G}') \geq D(\mathcal{G})$, we just need to show $\hat{\mathcal{L}}(\mathcal{G}') \leq \hat{\mathcal{L}}(\mathcal{G})$. As $\mathcal{G}'$ is a gauge-fixing of $\mathcal{G}$, we have that $\mathcal{S}(\mathcal{G}) \leq \mathcal{S}(\mathcal{G}') \leq \mathcal{G}' \leq \mathcal{G}$ and $K(\mathcal{G}) = K(\mathcal{G}')$.

Notice first that $\mathcal{L}(\mathcal{G}) \leq \mathcal{L}(\mathcal{G}')$ because anything that commutes with all elements of $\mathcal{G}$ also commutes with all elements of $\mathcal{G}' \leq \mathcal{G}$. Second, elements of $\mathcal{S}(\mathcal{G}') - \mathcal{S}(\mathcal{G})$ are not in $\mathcal{L}(\mathcal{G})$, and so the quotient groups $\mathcal{L}(\mathcal{G})/\mathcal{S}(\mathcal{G})$ and $\mathcal{L}(\mathcal{G})/\mathcal{S}(\mathcal{G}')$ are isomorphic. Thus, combine these two observations to get

$$\mathcal{L}(\mathcal{G})/\mathcal{S}(\mathcal{G}) = \mathcal{L}(\mathcal{G})/\mathcal{S}(\mathcal{G}') \leq \mathcal{L}(\mathcal{G}')/\mathcal{S}(\mathcal{G}'). \tag{C1}$$

However, $K(\mathcal{G}) = K(\mathcal{G}')$ dictates that $|\mathcal{L}(\mathcal{G})/\mathcal{S}(\mathcal{G})| = |\mathcal{L}(\mathcal{G}')/\mathcal{G}'|$ so

$$\mathcal{L}(\mathcal{G})/\mathcal{S}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')/\mathcal{S}(\mathcal{G}'). \tag{C2}$$

Because $\mathcal{S}(\mathcal{G}) \leq \mathcal{G}$ and $\mathcal{S}(\mathcal{G}') \leq \mathcal{G}'$,

$$\hat{\mathcal{L}}(\mathcal{G}) = \mathcal{G}\ \ \mathcal{L}(\mathcal{G}) = \mathcal{G}(\mathcal{L}(\mathcal{G})/\mathcal{S}(\mathcal{G})), \tag{C3}$$

$$\hat{\mathcal{L}}(\mathcal{G}') = \mathcal{G}'\ \ \mathcal{L}(\mathcal{G}') = \mathcal{G}'(\mathcal{L}(\mathcal{G}')/\mathcal{S}(\mathcal{G}')). \tag{C4}$$

Using Eq. (C2) and the fact that $\mathcal{G}' \leq \mathcal{G}$, we get $\hat{\mathcal{L}}(\mathcal{G}') \leq \hat{\mathcal{L}}(\mathcal{G})$, completing the proof.

[1] A Robert Calderbank and Peter W Shor, "Good quantum error-correcting codes exist," Physical Review A **54**, 1098 (1996).

[2] Andrew M Steane, "Error correcting codes in quantum theory," Physical Review Letters **77**, 793 (1996).

[3] Alexei Ashikhmin, Simon Litsyn, and Michael A Tsfasman, "Asymptotically good quantum codes," Physical Review A **63**, 032311 (2001).

[4] Hao Chen, "Some good quantum error-correcting codes from algebraic-geometric codes," IEEE Transactions on Information Theory **47**, 2059–2061 (2001).

[5] Robert Gallager, "Low-density parity-check codes," IRE Transactions on information theory **8**, 21–28 (1962).

[6] David JC MacKay, Graeme Mitchison, and Paul L McFadden, "Sparse-graph codes for quantum error correction," IEEE Transactions on Information Theory **50**, 2315–2330 (2004).

[7] Jean-Pierre Tillich and Gilles Zémor, "Quantum ldpc codes with positive rate and minimum distance proportional to the square root of the blocklength," IEEE Transactions on Information Theory **60**, 1193–1202 (2014).

[8] Sergey Bravyi, David Poulin, and Barbara Terhal, "Tradeoffs for reliable quantum information storage in 2d systems," Physical review letters **104**, 050503 (2010).

[9] Sergey Bravyi, "Subsystem codes with spatially local generators," Physical Review A **83**, 012320 (2011).

[10] Sergey Bravyi and A Yu Kitaev, "Quantum codes on a lattice with boundary," arXiv preprint quant-ph/9811052 (1998).

[11] A Yu Kitaev, "Fault-tolerant quantum computation by anyons," Annals of Physics **303**, 2–30 (2003).

[12] Dave Bacon, "Operator quantum error-correcting subsystems for self-correcting quantum memories," Physical Review A **73**, 012340 (2006).

[13] Panos Aliferis and Andrew W Cross, "Subsystem fault tolerance with the bacon-shor code," Physical review letters **98**, 220502 (2007).

[14] Michael Sipser and Daniel A Spielman, "Expander codes," IEEE transactions on Information Theory **42**, 1710–1722 (1996).

[15] Daniel A Spielman, "Linear-time encodable and decodable error-correcting codes," IEEE Transactions on Information Theory **42**, 1723–1731 (1996).

[16] Earl T Campbell, "A theory of single-shot error correction for adversarial noise," arXiv preprint arXiv:1805.09271 (2018).

[17] Sergey Bravyi and Barbara Terhal, "A no-go theorem for a two-dimensional self-correcting quantum memory based on stabilizer codes," New Journal of Physics **11**, 043029 (2009).

[18] Muyuan Li, Daniel Miller, Michael Newman, Yukai Wu, and Kenneth R Brown, "2-d compass codes," arXiv preprint arXiv:1809.01193 (2018).

[19] Alexey A Kovalev and Leonid P Pryadko, "Fault tolerance of quantum low-density parity check codes with sublinear distance scaling," Physical Review A **87**, 020304 (2013).

[20] Michael Capalbo, Omer Reingold, Salil Vadhan, and Avi Wigderson, "Randomness conductors and constant-degree lossless expanders," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing* (ACM, 2002) pp. 659–668.

[21] R Tanner, "A recursive approach to low complexity codes," IEEE Transactions on information theory **27**, 533–547 (1981).

[22] Shlomo Hoory, Nathan Linial, and Avi Wigderson, "Expander graphs and their applications," Bulletin of the American Mathematical Society **43**, 439–561 (2006).

[23] Tom Richardson and Ruediger Urbanke, *Modern coding theory* (Cambridge university press, 2008).