Decoding schemes for foliated sparse quantum error-correcting codes

A. Bolt,[1] D. Poulin,[2] and T. M. Stace[1,*]

[1]*ARC Centre for Engineered Quantum System, Department of Physics, University of Queensland, Brisbane, QLD 4072, Australia*

[2]*Département de Physique, Université de Sherbrooke, Québec, J1K 2R1, Canada*

(Received 5 March 2018; published 3 December 2018)

Foliated quantum codes are a resource for fault-tolerant measurement-based quantum error correction for quantum repeaters and for quantum computation. They represent a general approach to integrating a range of possible quantum error correcting codes into larger fault-tolerant networks. Here, we present an efficient heuristic decoding scheme for foliated quantum codes, based on message passing between primal and dual code "sheets." We test this decoder on two different families of sparse quantum error correcting code: turbo codes and bicycle codes, and show reasonably high numerical performance thresholds. We also present a construction schedule for building such code states.

DOI: 10.1103/PhysRevA.98.062302

I. INTRODUCTION

Quantum information processing (QIP) requires that the computational process must be performed with high fidelity. In a noisy environment this will require quantum error correction (QEC) [1,2]. Depending on the computational model, this noise manifests in different ways. A conceptually and technologically important step in the project to build quantum computers was the observation by Raussendorf *et al.* [3,4] that highly entangled *cluster states*, are universal resource states with which to perform a quantum computation. In cluster-state-based computation, the computation is driven forward by a series of measurements.

Subsequently, Raussendorf *et al.* [5–8] proposed a method of *fault-tolerant* quantum computation utilising cluster states. In this scheme, a 3D cluster state lattice is constructed, which can be viewed as a *foliation* of Kitaev's surface code [9,10]. Alternating *sheets* within this foliated structure correspond to primal or dual surface codes. Measurements on the *bulk* qubits generate correlations between corresponding logical qubits on the boundary of the lattice. In these schemes, errors are partially revealed through parity check operators, which can be determined from the outcomes of single-qubit measurements.

Raussendorf's 3D measurement-based computation scheme has proved important for the practical development of quantum computers [11–14], due to its high fault-tolerant computational error thresholds $\lesssim 1\%$ [10]. Furthermore, the robustness to erasure errors [15–17] makes these schemes attractive for various architectures, including optical networks [18]. This high threshold is a result of the underlying surface code, which itself has a high computational error threshold $\sim 10\%$ [9,10,15,16].

Fault-tolerant, measurement-based quantum computation is achieved, in part, by "braiding" defects within the foliated cluster, to generate a subgroup of the Clifford group. By virtue of their topologically protected nature, braiding operations can be made extremely robust, and so can be used to distill magic states. Together, these resources allow for universal quantum computation [7,8,19].

In an earlier paper, we showed that all Calderbank-Steane-Shor (CSS) codes can be *clusterized* using a larger cluster state resource [20]. These cluster state codes can be *foliated* as a generalization of Raussendorf's 3D lattice [20]. In that work we also demonstrated the performance of a turbo code with a heuristic decoder that we have developed.

In this paper, we present a detailed description of the decoding algorithm, and we apply the decoder to two classes of foliated CSS codes: serial turbo codes [21–24], and bicycle codes. In contrast to the surface code, these code families have finite rate. This allows for a much lower overhead of physical qubits to encoded qubits as the size of the code is increased, as compared to surface codes. In both cases, the decoder on the complete foliated cluster state uses a soft-input–soft-output (SISO) decoder within each sheet of the code as a subroutine, followed by an exchange of marginal information between neighboring primal and dual code sheets. Iterating these steps yields an error pattern consistent with the error syndrome.

We present Monte Carlo simulations of the error-correcting performance using this decoder, assuming independently distributed Pauli X and Z errors. We analyze the code performance in terms of both the Bit Error Rate (BER) and Word Error Rate (WER). Our numerical results, simulating uncorrelated Pauli noise errors, indicate that the codes exhibit reasonably high (pseudo-)thresholds in the order of a few percent.

In Sec. II, we review the clusterization of CSS codes. Section III reviews the foliation of clusterized codes and presents a general decoding approach. In Sec. IV, we review the construction of decoding trellises for convolutional codes, which are a resource for the convolutional decoding. In Sec. V, we develop decoding trellises for clusterized convolutional codes within a larger foliated code. In Sec. VI, we present a decoding algorithm for foliated convolutional codes. This is a subroutine for the foliated turbo decoder. In Sec. VII, we present the decoding scheme for foliated

*stace@physics.uq.edu.au

 ©2018 American Physical Society

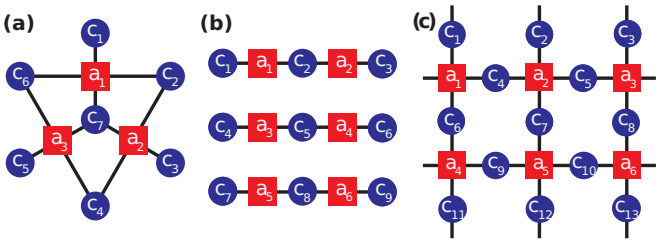

FIG. 1. Examples of progenitor clusters for clusterized CSS codes. (a) Clusterized Steane code. (b) Clusterized Shor code. (c) Clusterized surface code. Code qubits (blue circles) are connected by cluster bonds (black lines) to ancilla qubits (red squares). An X-basis measurement of ancilla a_k projects neighboring code qubits onto an eigenstate of $\otimes_{\mathcal{N}_{a_k}} Z \in \mathcal{S}_Z$.

turbo codes and display our numerical results of simulated trials. Section VIII presents a decoding algorithm for foliated bicycle codes, and presents numerical results. In Sec. IX, we analyze the construction of clusterized convolutional and turbo codes from an architectural viewpoint. We investigate fault-tolerant constructions of foliated turbo codes.

II. CLUSTERIZED CODES

We begin by reiterating some definitions, in order to set notation for what follows. A *cluster state* is defined on a collection of qubits located at the vertices of a graph [3,4,25]. A qubit at vertex v carries with it an associated cluster stabilizer $K_v = X_v(\otimes_{\mathcal{N}_v} Z) \equiv X_v Z_{\mathcal{N}_v}$, acting on it and its neighbors, $\mathcal{N}_v$. The cluster state is the $+1$ eigenstate of the stabilizers K_v. For example, in Fig. 1(a), there is a cluster stabilizer $X_{a_1} Z_{c_1} Z_{c_2} Z_{c_6} Z_{c_7}$ associated to the ancilla qubit a_1. Operationally, such a state can be produced with single and two qubit gates: each qubit is prepared in a $+1$ eigenstate of X, and then C-PHASE gates are applied to pairs of qubits that share an edge in the graph, e.g., in Fig. 1(a), between qubit c_2 and its graph neighbors a_1 and a_2.

A *stabilizer quantum code* is defined by the code stabilizers, S, which are a set of mutually commuting Hermitian operators, whose simultaneous $+1$ eigenstates define valid code states. A CSS code is one for which the generators for S can be partitioned into a set of generators for X-like stabilizers, $\mathcal{S}_X$, which are products of Pauli X operators acting on subsets of the code qubits, and a set of generators for Z-like stabilizers, $\mathcal{S}_Z$, which are products of Pauli Z operators, i.e., $S = \langle \mathcal{S}_X \cup \mathcal{S}_Z \rangle$.

An $[[n, k, d]]$ CSS code can be generated from a larger *progenitor* cluster state [20]. The progenitor cluster is the cluster state associated with the Tanner graph of $\mathcal{S}_Z$ [26], i.e., a bipartite graph $G = (V, E)$ whose vertices V are labeled as code qubits c, or ancilla qubits a. Each ancilla qubit a is associated to a stabilizer $Z_{\mathcal{N}_a} \in \mathcal{S}_Z$, so that $|V| = n + |\mathcal{S}_Z|$. E contains the graph edge (c, a) if $[H_Z]_{c,a} = 1$, where H_Z is the parity check matrix. The logical X codestate of the CSS codes is obtained by measuring the ancilla qubits of the progenitor cluster state in the X basis [20].

Examples of clusterized CSS codes are shown in Fig. 1. These are the Steane 7 qubit code, 9 qubit Shor code, and a 13 qubit surface code with Z stabilizers generated by

$$\begin{aligned}
\mathcal{S}_Z^{\text{Steane}} &= \left\{ Z_{c_1} Z_{c_2} Z_{c_6} Z_{c_7},\ Z_{c_2} Z_{c_3} Z_{c_4} Z_{c_7},\ Z_{c_4} Z_{c_5} Z_{c_6} Z_{c_7} \right\}, \\
\mathcal{S}_Z^{\text{Shor}} &= \left\{ \begin{matrix} Z_{c_1} Z_{c_2},\ Z_{c_2} Z_{c_3},\ Z_{c_4} Z_{c_5}, \\ Z_{c_5} Z_{c_6},\ Z_{c_7} Z_{c_8},\ Z_{c_8} Z_{c_9} \end{matrix} \right\}, \\
\mathcal{S}_Z^{\text{Surf.}} &= \left\{ \begin{matrix} Z_{c_1} Z_{c_4} Z_{c_6},\ Z_{c_2} Z_{c_4} Z_{c_5} Z_{c_7},\ Z_{c_3} Z_{c_5} Z_{c_8}, \\ Z_{c_6} Z_{c_9} Z_{c_{11}},\ Z_{c_7} Z_{c_9} Z_{c_{10}} Z_{c_{12}},\ Z_{c_8} Z_{c_{10}} Z_{c_{13}} \end{matrix} \right\}.
\end{aligned} \tag{1}$$

For each of the Z-like stabilizers of a code, an ancillary qubit (red squares) is built into a cluster fragment with the associated code qubits (blue circles) in the stabilizer. For instance, in the figure, each ancillary qubit, a_i, is associated with the ith stabilizer element of $\mathcal{S}_Z$. This construction holds for all CSS codes [20].

CSS codes detect X and Z errors independently. Each error type may be corrected independently, although this disregards potentially useful correlations, if such exist. We will assume independent X and Z Pauli noise, and in what follows, we describe the process for detecting and correcting Z errors using $\mathcal{S}_X$ syndrome information. The dual problem, using $\mathcal{S}_Z$ syndrome information to correct X errors proceeds in exact analogy.

We note that for every CSS code, there is a *dual* CSS code. The dual code is generated by exchanging X and Z operators in the stabilizers and logical operators. That is, an X-like stabilizer in the primal code transforms to a Z-like stabilizer in the dual code, and vice versa. Following the prescription above, a dual CSS code also has a progenitor cluster state, i.e., the dual code can also be clusterized in the same way. If the code is self-dual (e.g., the Steane code), then the corresponding primal and dual cluster states are identical.

III. FOLIATED CODES

In this section, we review the general construction of foliated codes as an extension of Raussendorf's 3D cluster state construction [5–8,20]. Section III A covers the generation of foliated codes from the cluster state resources in Sec. II and Sec. III B outlines a heuristic decoding approach which is suitable for general CSS codes. Specific implementations for convolutional, turbo and bicycle codes, which are all examples of low-density parity check (LDPC) codes, are covered in later sections.

A. Foliated code construction

The general foliated construction consists of alternating sheets of primal and dual clusterized codes as defined in Sec. II, which are "stacked" together [20]. The stacking of code sheets simply amounts to the introduction of additional cluster bonds (i.e., C-PHASE gates) between corresponding code qubits on neighboring sheets. This is depicted in Fig. 2 for a specific code example [the Steane code of Fig. 1(a)].

Suppose a CSS code has an X-like stabilizer, $\hat{s}$, with support on code qubits c_{h_j}, indicated by the support vector $\vec{h} = \{h_1, h_2, ...\}$, i.e., $\hat{s} = X_{c_{h_1}} \otimes X_{c_{h_2}} ... \equiv X_{c_{\vec{h}}}$. In the foliated construction, there is a corresponding parity check operator centered on sheet m, given by

$$\hat{P}_{c_{\vec{h},m}} \equiv X_{a_{\vec{h},m-1}} X_{c_{\vec{h},m}} X_{a_{\vec{h},m+1}}, \tag{2}$$

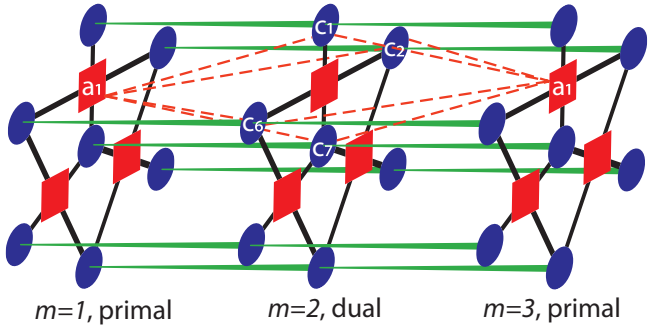

FIG. 2. A foliated Steane code with $L = 3$ sheets. Code qubits (blue circles) share C-PHASE cluster bonds (thick lines) with ancilla qubits (red squares) in the same sheet, and with code qubits in adjacent sheets (green streaked lines). The Steane code is self-dual, so that primal and dual cluster sheets are identical. In this example, the end faces are indexed by $m = 1$ and $m = L = 3$. Bulk qubits include all qubits in the $m = 2$ sheet, and the ancialla qubits in the end faces. The product of cluster stabilizers centered on the labeled qubits (connected by dashed red lines) produces the parity check operator $\hat{P}_{1,m=2}$ in Eq. (3).

where $a_{\vec{h},m\pm1}$ is the ancilla qubit associated with the dual code stabilizer $Z_{c_{\vec{h},m\pm1}}$ acting on sheet $m \pm 1$.

Given that each code sheet is derived from an underlying CSS code with logical operators $X_{\mathcal{L}}$ and $Z_{\mathcal{L}}$, we can define corresponding logical operators within each code sheet, $X_{\mathcal{L},m}$ and $Z_{\mathcal{L},m}$. After the code is foliated, the logical operators in each code sheet commute with the parity check operators [20], i.e., $[\hat{P}_{c_{\vec{h},m}}, Z_{\mathcal{L},m}] = 0$.

A reason for considering this construction is the observation that the foliated code cluster state provides a resource for error tolerant entanglement sharing. This generalizes one of the major insights of Raussendorf *et al.* [5], in which is was shown that after foliating L surface code sheets, the resulting three-dimensional cluster state (defined on a cubic lattice) served as a resource for fault tolerancy creating an entangled Bell pair of surface code logical qubits between the first and last sheet (labelled by the index $m = 1$ and L). Starting from the three-dimensional foliated surface code cluster, this long-range entanglement is generated by measuring each of the *bulk physical qubits* (i.e., all ancilla qubits, and all code qubits in sheets $m = 2$ to $L - 1$) in the X basis. Formally, this is shown by noting that after the bulk qubit measurements, the remaining physical qubits (which are confined to sheets 1 and L) are stabilized by the operators $X_{\mathcal{L},1} \otimes X_{\mathcal{L},L}$ and $Z_{\mathcal{L},1} \otimes Z_{\mathcal{L},L}$ [5], up to Pauli frame corrections that depend on the specific measurement outcomes on the bulk qubits. The underlying surface code makes the protocol described therein robust against Pauli errors on the bulk qubits.

As discussed in Ref. [20], this property is respected for *any* foliated CSS code. That is, measurements on the bulk qubits project the logical qubits encoded within the end sheets into an entangled logical state. This is verified by checking that $X_{\mathcal{L},1} \otimes X_{\mathcal{L},L}$ and $Z_{\mathcal{L},1} \otimes Z_{\mathcal{L},L}$ are in the stabilizer group of the cluster state after bulk measurements are completed.

For an underlying $[[n, k, d]]$ code, there are weight-d undetectable error chains on the foliated cluster, as in Refs. [5–8,10]. Since the structure of the code in the direction of foliation is a simple repetition, it follows that the foliated cluster inherits the distance of the underlying code.

Figure 2 shows an example of the cluster state for a foliated Steane code. Alternating sheets of the primal Steane code cluster state [shown in Fig. 1(a)] and its self-dual are stacked together, with additional cluster bonds (green streaked lines) extending between corresponding code qubits in each sheet; operationally, these correspond to C-PHASE gates between code qubits. The Steane code is self-dual, so primal and dual sheets are identical.

An example of a parity check operator centered on sheet $m = 2$ is

$$\hat{P}_{1,m=2} = X_{a_1,1} X_{c_1,2} X_{c_2,2} X_{c_6,2} X_{c_7,2}. X_{a_1,3}, \tag{3}$$

which is depicted in Fig. 2. For this example, there are two other parity check operators centered on sheet $m = 2$, associated to the other ancilla qubits therein. The logical operators for the Steane code pictured in Fig. 1 a are $Z_{\mathcal{L}}^{\text{Steane}} = Z_{c_1} Z_{c_2} Z_{c_3}$ and $X_{\mathcal{L}}^{\text{Steane}} = X_{c_1} X_{c_2} X_{c_3}$. By inspection, the parity check operator $\hat{P}_{1,m}$ commutes with $Z_{\mathcal{L},m}^{\text{Steane}}$ on sheet m.

Figure 2 also illustrates a minimal example of entanglement sharing between the end sheets of the foliated construction, for $L = 3$ code sheets. After the foliated cluster state is formed, X measurements on the bulk qubits (all qubits in the dual sheet shown, and the ancilla qubits in the end primal sheets) leave the remaining physical qubits (the code qubits in the primal end sheets) stabilized by the operators $X_{\mathcal{L},1}^{\text{Steane}} \otimes X_{\mathcal{L},3}^{\text{Steane}}$ and $Z_{\mathcal{L},1}^{\text{Steane}} \otimes Z_{\mathcal{L},3}^{\text{Steane}}$. Additional examples of the Shor code and surface code are presented in Ref. [20].

B. Decoding approach

Errors in the foliated cluster are detected by parity check operators: a Z error will flip one or more parity checks, giving a nontrivial error syndrome for the foliated cluster. Importantly, the parity check measurement outcomes can be inferred from products of single-qubit X measurement outcomes. The syndrome information then becomes input into a decoder.

While generic CSS codes may not be efficiently decoded, many exact or heuristic decoders are known for specific code constructions [10,17,23,27]. For the purpose of this paper, we assume that whichever code is chosen, a practically useful decoder is known. In what follows, this decoder forms a subroutine that is called repeatedly in the decoding of the larger, foliated code.

We note here that this is qualitatively different from the foliated surface code [4,10], for which the decoder does *not* call the surface code decoder as a subroutine. Instead, the surface code decoders are *generalized* to the foliated version. For instance, the minimum weight perfect matching decoder for the surface code can be modified to the foliated case [5,10], by replacing stabilizer defects in the 2D Kitaev lattice [9], with parity check defects in the 3D Raussendorf lattice [4,10], but still using perfect matching on the syndrome. However, this is a peculiarity of the Raussendorf construction as a graph product code [28], constructed from repeated graph products of a repetition code.

Here, we propose a general purpose, a heuristic method of decoding foliated codes, which is based upon the decoding of

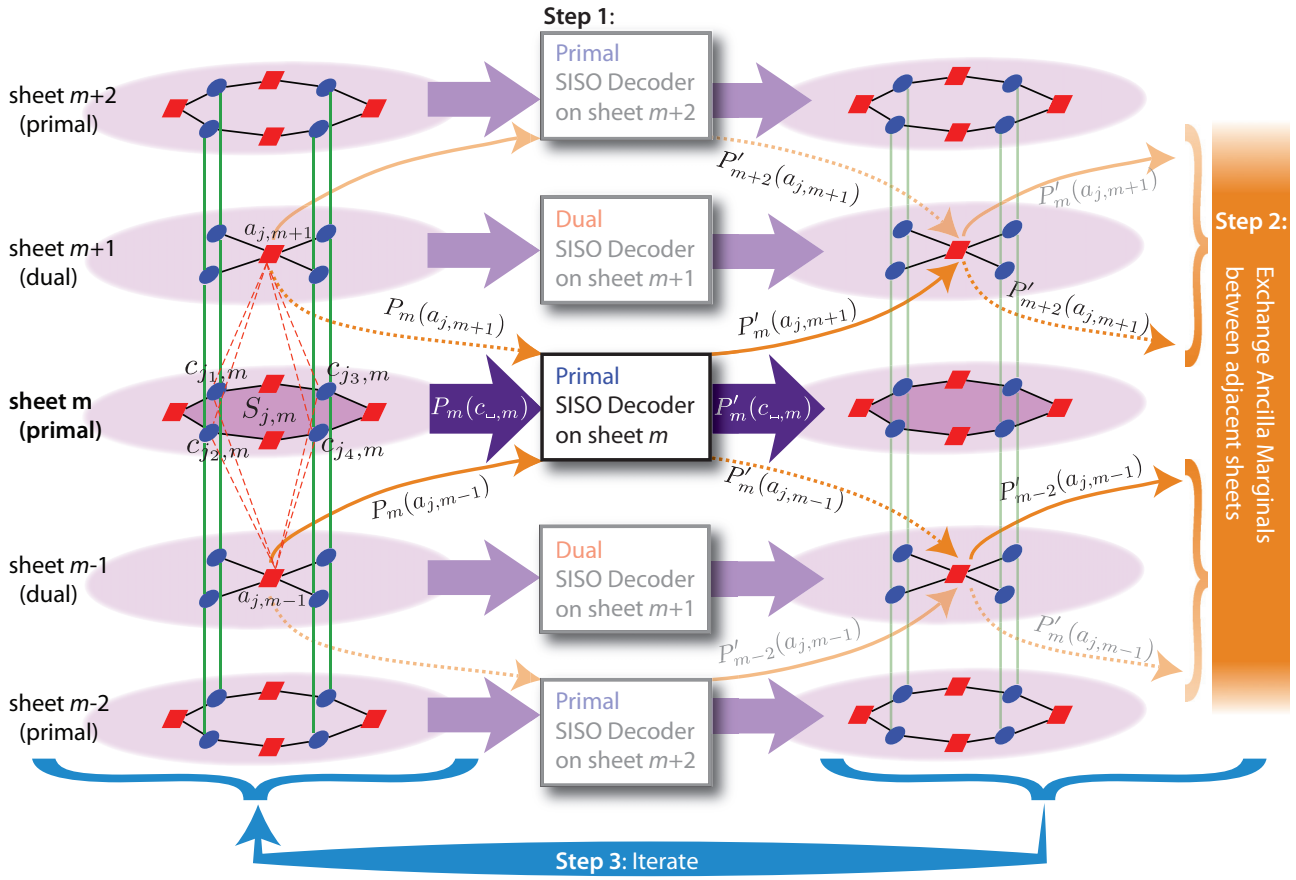

FIG. 3. A schematic flow of *beliefs* (i.e., marginal probabilities) in the heuristic decoder for a generic foliated cluster code. Code qubits are shown as circles, ancilla qubits as rectangles within each code sheet. Cluster bonds are shown as black (green) lines within (between) code sheets (as in Fig. 2). The decoding sheet consists of code qubits $c_{\text{␣},m}$ in sheet m, and the ancilla in the adjacent sheets, $a_{\text{␣},m\pm1}$. At the left, one stabilizer, $S_{j,m}$, with support on code qubits $c_{j_{1,\ldots,4},m}$ in sheet m is shown (shaded), and forms a parity check operator with the associated ancilla qubits $a_{j,m\pm1}$ (connected by dashed red lines). Step 1: on each decoding sheet m, the soft-input–soft-output (SISO) decoder receives syndrome data $\vec{S}_m$ and input error marginals $P_m(c_{\text{␣}}, m)$ on the code qubits (thick purple arrows) and adjacent ancilla qubits $P_{m\pm1}(a_{\text{␣}}, m)$ (thin orange arrows) to compute updated error marginals P'_m for code and ancilla qubits. This step can be processed in parallel across all sheets, since there are no cross-dependencies. Note that after step 1 there are *two* marginals associated to each adjacent ancilla: e.g., $P'_m(a_{j,m+1})$ (solid orange arrow emerging from the decoder) and $P'_{m+2}(a_{j,m+1})$ (dotted orange arrow emerging from the decoder), which are computed from decoders on sheets m and $m+2$, respectively. If these disagree, $P'_m(a_{j,m+1}) \neq P'_{m+2}(a_{j,m+1})$, the values are exchanged in step 2. After marginal exchange, the process is iterated (step 3), until marginals converge. After marginals on the ancilla have converged sufficiently, maximum likelihood decoding is performed within each code sheet (step 4, not shown).

the underlying CSS codes in the presence of faulty syndrome extraction. If the decoder for the underlying code is efficient then it will be efficient for the full foliated process. The foliated decoding algorithm is a heuristic method based on belief propagation (BP) that may work well in many cases [22,29]. The overall foliated decoder calls a soft-input–soft-output[1] (SISO) decoder for each of the underlying primal and dual CSS codes, as a subroutine. The SISO decoder calculates the probability of a Pauli error, σ, on qubit $q \in \{a_k, c_j\}$, $P(\sigma_q|\vec{S}_{\text{CSS}})$, given a physical error model and syndrome data for the CSS code, $\vec{S}_{\text{CSS}}$, which may itself be unreliable, due to errors on the ancilla qubits. Using such a decoder it is possible to assign a probability of failure to a parity check to account for errors on ancilla qubits, $P(\sigma_a) = \sum_{\vec{S}_{\text{CSS}}} P(\sigma_a|\vec{S}_{\text{CSS}})P(\vec{S}_{\text{CSS}})$.

For the foliated case, consider a parity check operator given by Eq. (2). A nontrivial syndrome can arise because of errors on code qubits $\vec{h}$ within code sheet m, or due to errors on the corresponding ancilla qubits $a_{\vec{h}}$ in adjacent dual sheets $m \pm 1$.

In the case where dual-sheet ancilla qubits are error-free, the decoding problem reduces to a series of independent CSS decoders using perfect syndrome extraction. However, errors on the ancillas mean that the in-sheet syndrome is unreliable. To account for the dual-sheet ancilla errors, we embed the CSS decoder in a *belief propagation* (BP) routine, as iterated in the following scheme. Steps 1, 2, and 3 and illustrated in Fig. 3.

[1]"Soft" values indicate that the decoder accepts and returns probabilities for errors, as opposed to "hard" values, which are binary allocations ("error" OR "no error") at each qubit. For example, perfect matching on the surface code is a hard decoder.

(0) A *decoding sheet* centered on sheet m is defined by a set of qubits q, which contain the code qubits $c_{j,m}$ within the sheet and the ancilla qubits $a_{k,m\pm1}$ on neighboring sheets. The input to the decoder is the syndrome data $\vec{S}_m$ on sheet m, and a prior probability distribution, $P_m^{\rm pr}(a_{k,m\pm1})$, for the marginals on the ancilla qubits associated to the decoding sheet m.

(1) The SISO decoder on sheet m computes marginal error probabilities for the code qubits in the sheet, $P'_m(c_{j,m})$, assuming marginals on the ancilla in sheet $m \pm 1$, and updated marginals on the ancilla qubits $P'_m(a_{k,m})$. This step can be parallelized across all sheets.

(2) The assumed error model for ancilla qubits is updated using the results from step 1. $P_m(a_{k,m\pm1}) \to P'_m(a_{k,m\pm1}) = P_{m\pm2}(a_{k,m\pm1}|S_{m\pm2})$, where $P_{m\pm2}(a_{k,m\pm1})$ is the probability distribution found from a neighboring decoding process. We refer to this update rule as an *exchange of marginals*.

(3) Using the updated ancilla marginals from step 2, we iterate back to step 1 until ancilla marginals converge sufficiently, i.e., $P_m(a_{k,m\pm1}) \approx P'_m(a_{k,m\pm1})$, in which case we proceed to step 4.

(4) Use the converged marginals computed on the code and ancilla qubits, $P_m(c_{j,m})$ and $P_m(a_{k,m\pm1})$, to calculate an error correction chain (parametrized by error correction binary support vector, $\vec{e}$) with (approximately) maximum likelihood $P_m(\vec{e}|\vec{S}_m, P_m(c_{j,m}), P_m(a_{k,m\pm1}))$, within each code sheet. This could employ a hard decoder.

In Secs. IV–VIII, we describe specific soft decoding implementations for foliated convolutional, turbo, and LDPC bicycle codes. For convolutional and turbo codes, we first introduce a trellis decoding framework, which is then adapted for use as a single-layer decoder. This is then combined with the BP process above to generate a full decoding scheme. For bicycle codes, we use belief propagation directly on the Tanner graph representation of the foliated code.

IV. CONVOLUTIONAL TRELLIS CONSTRUCTION

In this section, we review the construction of trellises as a tool for SISO decoding of convolutional codes [22,23,30,31]. Section V modifies this construction for use with single sheets of the foliated code, which is itself a subroutine in the full decoding of foliated convolutional codes.

Generators and stabilizers of convolutional codes are translations of some "seed" generators or stabilizers, which act over a sequence of *frames*. Each frame labels a contiguous block of n qubits. We assume here that the code has τ frames, labeled by a frame index, $t = 1, \ldots, \tau$. For a classical rate $\frac{k}{n}$ code, the generator matrix (for logical Z operators) has the form

$$\begin{array}{l}\text{frame: } \ldots \quad t-1 \quad t \quad t+1 \quad t+2 \quad \ldots \\ \mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \ldots & G^{(1)} & G^{(2)} & \ldots & G^{(\nu_g)} & \ldots & & \\ & \ldots & G^{(1)} & G^{(2)} & \ldots & G^{(\nu_g)} & \ldots & \\ & & \ldots & G^{(1)} & G^{(2)} & \ldots & G^{(\nu_g)} & \ldots \end{bmatrix}_{k\tau\times n\tau},\end{array} \tag{4}$$

where $G^{(i)}$ are binary-valued $k \times n$ submatrices. We use the notation A^{T} to denote the transpose of the matrix A; bold face matrices indicate generators acting on the entire set of physical qubits. All other elements in G are zero. Each $G^{(i)}$ acts on a single frame of n physical bits, encoding k logical bits. The code is built from translations of the submatrices $[G^{(1)}, \ldots, G^{(\nu_g)}]$. Each component, $G^{(j)}$, acts on a single frame of the code. The value of ν_g is the *codeword memory* length, which counts the number of frames over which parity check operators have support.

Later, we will discuss a specific example of a convolutional code that illustrates this construction. For concreteness, we preempt that example by reference to Fig. 7(a), each row of which depicts a convolutional code with frames consisting of $n = 3$ qubits (blue circles), and with parity check operators (red squares) extending over $\nu_g = 3$ frames. Note that in this example, each parity check operator has support on 6 of the qubits (indicated by thick black lines) within the $\nu_g = 3$ contiguous frames.

Similarly, we define the parity check generator matrix

$$\mathbf{H} = \begin{bmatrix} \ldots & H^{(1)} & \ldots & H^{(\nu_h)} & \ldots & & \\ & \ldots & H^{(1)} & \ldots & H^{(\nu_h)} & \ldots & \\ & & \ldots & H^{(1)} & \ldots & H^{(\nu_h)} & \ldots \end{bmatrix}_{n_z\tau\times n\tau}, \tag{5}$$

where $n_z = |\mathcal{S}_Z|/\tau$ is the number of Z-like stabilizers per frame. The value of ν_h is the *parity check memory* length. Typically, ν_h and ν_g are of similar size.

Codeword generators are expressed in the form $\hat{g} = Z^{\otimes\vec{g}}$, where $\vec{g} \in \mathbb{Z}_2^{n\tau}$ is in the row space of $\mathbf{G}_Z$. We have introduced the notation $Z^{\otimes\vec{v}} = Z_1^{v_1} \otimes Z_2^{v_2} \ldots$ with $v_j \in \mathbb{Z}_2$. Similarly, the stabilizer generators formed from $\mathbf{H}_Z$ are $\hat{h} = Z^{\otimes\vec{h}}$, where $\vec{h} \in \mathbb{Z}_2^{n\tau}$ is in the row space of $\mathbf{H}_Z$.

The commutation relationships between generator and stabilizer matrices manifest as orthogonality conditions, i.e.,

$$\begin{bmatrix} \mathbf{G}_Z^{\mathsf{T}} \\ \mathbf{H}_Z \\ \mathbf{ISF}_Z \end{bmatrix} \begin{bmatrix} \mathbf{G}_X & \mathbf{ISF}_X^{\mathsf{T}} & \mathbf{H}_X^{\mathsf{T}} \end{bmatrix} = \mathbb{I}_{n\tau\times n\tau}, \tag{6}$$

where we have introduced the *inverse syndrome formers* (ISFs). The ISF is useful determining an initial, valid decoding pattern, known as a *pure error*. Each row of $\mathbf{ISF}_Z$ corresponds to an operator that commutes with all X-like stabilizer generators, $\hat{g} \in \text{RowSpace}(\mathbf{G}_X)$, and with all but one of the parity check generators $\in \text{RowSpace}(\mathbf{H}_X^{\mathsf{T}})$; there are as many ISF generators as there are parity check generators. The rows of $\mathbf{G}_Z^{\mathsf{T}}$ and $\mathbf{H}_Z$ form an orthonormal set, but do not fully span $\mathbb{Z}_2^{n\tau}$. The $\mathbf{ISF}_Z$ submatrix can be computed by finding an orthonormal completion of the rowspace of $\mathbf{G}_Z^{\mathsf{T}}$ and $\mathbf{H}_Z$, e.g., using Gram-Schmidt. The matrix $\mathbf{ISF}_Z$ is not unique: any orthogonal completion of the basis that satisfies $\mathbf{ISF}_Z \cdot$

$\mathbf{H}_X^\mathsf{T} = \mathbb{I}$ will suffice.[2] Similarly, $\mathbf{ISF}_X^\mathsf{T}$ is generated from an orthonormal completion of the column space of $\mathbf{G}_X$ and $\mathbf{H}_X^\mathsf{T}$.

Suppose some (unknown) pattern $\vec{\varepsilon} \in \mathbb{Z}_2^{n\tau}$ of Z errors gives rise to a syndrome that is revealed by the X-like parity checks, i.e., $\vec{S} = \mathbf{H}_X\vec{\varepsilon} \in \mathbb{Z}_2^{n_x\tau}$. We use $\mathbf{ISF}_Z$ to generate a pure error, $\hat{E}^0 = Z^{\otimes\vec{e}^0} \equiv Z^{\otimes(\mathbf{ISF}_Z^\mathsf{T}\cdot\vec{S})}$, based only on the syndrome data. The binary-valued support vector $\vec{e}^0 \in \mathbb{Z}_2^{n\tau}$ corresponds to a possible error correction pattern that satisfies the syndrome $\vec{S}$, i.e., $\vec{S} = \mathbf{H}_X\vec{\varepsilon} = \mathbf{H}_X\vec{e}^0$. Therefore $\hat{E}^0$ is a *valid* decoding pattern, but it is unlikely to be the most probable decoding, and so is unlikely to robustly correct the original error. However, it defines a reference decoding from which the set of *all* valid decodings, $\mathcal{E}$, can be enumerated through

$$\begin{aligned}\mathcal{E} &= \{\hat{E}^0\hat{h}\hat{g}|\hat{h} \in \mathcal{S}_Z, \hat{g} \text{ is a code word}\},\\ &= \left\{Z^{\otimes(\vec{e}^0+\vec{h}+\vec{g})}\middle|\vec{h} \in \text{RowSpace}(\mathbf{H}_Z),\right.\\ &\quad \left.\vec{g} \in \text{RowSpace}\left(\mathbf{G}_Z^\mathsf{T}\right)\right\},\end{aligned} \tag{7}$$

where we take linear combinations of rows of $\mathbf{H}_Z$ and $\mathbf{G}_Z^\mathsf{T}$ over $\mathbb{Z}_2$. In what follows, we suppress the subscripts X and Z.

A valid decoding of the syndrome data may be written as $\hat{E}^0Z^{\otimes\vec{p}}$, where $\vec{p} \equiv \vec{h} + \vec{g} \in \text{RowSpace}(\mathbf{H}) \cup \text{RowSpace}(\mathbf{G}^\mathsf{T}) \subset \mathbb{Z}_2^{\tau n}$. That is, we define

$$\vec{p} = \sum_{i=1}^{k}\sum_{j=1}^{\tau} l_{i,j}^{(g)}\mathbf{G}_{i+j}^\mathsf{T} + \sum_{i=1}^{n_z}\sum_{j=1}^{\tau} l_{i,j}^{(h)}\mathbf{H}_{i+j}, \tag{8}$$

where $\mathbf{A}_i$ refers to the i^{th} row of $\mathbf{A}$.

Relative to the (easily found) pure error support vector, $\vec{e}^0$, the vector $\vec{p}$ parametrizes all possible valid decoding through the coefficients $l_{i,j}^{(g)}$ and $l_{i,j}^{(g)}$. A good decoder will return optimal values of $l_{i,j}^{(g)}$ and $l_{i,j}^{(h)}$, corresponding to an element, $\hat{E} = Z^{\vec{p}+\vec{e}^0} \in \mathcal{E}$, that has a high likelihood of correcting the original error. For low error rates, this amounts finding a $\vec{p}$ that minimizes the Hamming weight of $\vec{p} + \vec{e}^0$. For readers unfamiliar with this general construction, we provide a short example based on the seven-qubit Steane code in Appendix A.

A. Seed generators of convolutional codes

Finding an optimal $\vec{p}$ by enumerating over all $2^{(k+n_z)\tau}$ possible binary values of $l_{i,j}^{(g)}$ and $l_{i,j}^{(h)}$ becomes computationally intractable as the size of the code is increased. However, the repeated structure of convolutional codes, in blocks of length νn, allows for a simplification, using a decoding trellis. The trellis decoder reduces the search space to $\sim 2^{k\nu_g+n_z\nu_h}\tau$, so that the decoding is linear in the code size τ, albeit with potentially large prefactor depending on the total memory lengths ν_g and ν_h.

As the number of encoded bits increases, the size of the generator matrix increases. We therefore use a more compact representation using transfer functions, which are polynomials in the *delay* operator, denoted by D. We interpret D as a discrete generator of frame shifts, so that D^q represents a "delay" of q frames. We also define the inverse shift generator, $\tilde{D}$, which acts like a reverse translation such that $D^a\tilde{D}^b = \delta_{ab}$. Details of the construction are given in Appendix B.

Using this delay notation, we define the *seed matrix* of a rate $\frac{k}{n}$ convolutional code in terms of the submatrices $G^{(i)}$, in Eq. (4),

$$\begin{aligned}G^\mathsf{T}(D) &\equiv G^{(1)} + DG^{(2)} + \cdots + D^{\nu_g-1}G^{(\nu_g)},\\ &\equiv \begin{bmatrix} g_{11}(D) & g_{12}(D) & \dots & g_{1n}(D)\\ \vdots & & & \vdots\\ g_{k1}(D) & g_{k2}(D) & \dots & g_{kn}(D)\end{bmatrix}_{k\times n},\end{aligned} \tag{9}$$

where $g_{ij}(D)$ is a polynomial in D, defined by

$$g_{ij}(D) = \sum_{q=1}^{\nu_g} D^{q-1}G_{ij}^{(q)}. \tag{10}$$

The utility of the delay operator notation is that (1) it enables us to write $G^\mathsf{T}(D)$ in terms of a matrix that is independent of the number of frames τ and (2) it sets the degree of the polynomial entries.

Similarly,

$$\begin{aligned}H(D) &\equiv H^{(1)} + DH^{(2)} + \cdots + D^{\nu_h-1}H^{(\nu_h)},\\ &\equiv \begin{bmatrix} h_{11}(D) & h_{12}(D) & \dots & h_{1n}(D)\\ \vdots & & & \vdots\\ h_{z1}(D) & h_{z2}(D) & \dots & h_{zn}(D)\end{bmatrix}_{n_z\times n},\end{aligned}$$

where $h_{ij}(D) = \sum_{q=1}^{\nu_h} D^{q-1}H_{ij}^{(q)}$.

In this delay notation, Eq. (6) becomes

$$\begin{aligned}&D^u\begin{bmatrix} G_Z^\mathsf{T}(D)\\ H_Z(D)\\ \text{ISF}_Z(D)\end{bmatrix}\left[G_X(\tilde{D}) \quad \text{ISF}_X^\mathsf{T}(\tilde{D}) \quad H_X^\mathsf{T}(\tilde{D})\right]\tilde{D}^v\\ &= \mathbb{I}_{n\times n}\delta_{uv}.\end{aligned} \tag{11}$$

Since convolutional codes are partitioned into frames, we write $\vec{p} = (\vec{p}_1, \vec{p}_2, \dots, \vec{p}_t, \dots, \vec{p}_\tau)$, where $\vec{p}_j \in \mathbb{Z}_2^n$. From Eq. (8), we have $\vec{p} = \sum_i^\tau \vec{p}_t D^{i-1}$, and

$$\begin{aligned}\vec{p}_t =& \sum_{i=1}^{k}\sum_{j=1}^{\nu_g+1} l_{i,t-j+1}^{(g)}G_i^\mathsf{T}(D)\tilde{D}^{j-1}\\ &+ \sum_{i=1}^{n_z}\sum_{j=1}^{\nu_h+1} l_{i,t-j+1}^{(h)}H_i(D)\tilde{D}^{j-1},\\ \equiv& U_p(\alpha_t, \vec{l}_t),\end{aligned} \tag{12}$$

where $\vec{l}_t \equiv \{\vec{l}_t^{(g)}; \vec{l}_t^{(h)}\} \equiv \{l_{1,t}^{(g)}, \dots, l_{k,t}^{(g)}; l_{1,t}^{(h)}, \dots, l_{n_z,t}^{(h)}\}$ and

$$\alpha_t \equiv \left\{\vec{l}_{t-1}^{(g)}, \vec{l}_{t-2}^{(g)}, \dots, \vec{l}_{t-\nu_g}^{(g)}; \vec{l}_{t-1}^{(h)}, \vec{l}_{t-2}^{(h)}, \dots, \vec{l}_{t-\nu_h}^{(h)}\right\}. \tag{13}$$

The *memory state* α_t is a list of length $\nu_g + \nu_h$ that records values of $\vec{l}_q$ for $q = t - \nu_{g,h} + 1$ up to $q = t - 1$. It becomes useful when we discuss a SISO decoder in Sec. VI, which optimizes the choice of l's.

The purpose of these manipulations is that $\vec{p}_t$ depends only on prior $\vec{l}_t$'s going back a number of frames depending on the memory length $\nu_{g,h}$. Importantly, it does not grow with τ, so

[2]That is, $\mathbf{ISF}_Z^\mathsf{T}$ is a pseudoinverse of $\mathbf{H}_X$, and vice versa.

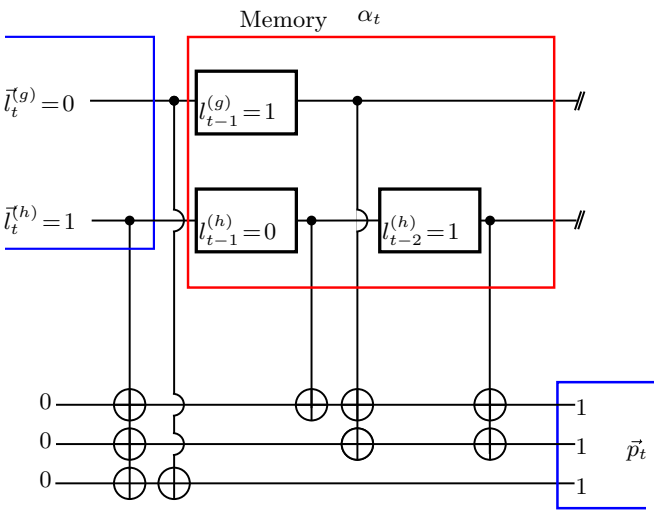

FIG. 4. A schematic circuit for computing $\vec{p}_t$ in Eq. (12) for a single frame, t, based on the $\frac{k}{n}=\frac{1}{3}$ convolutional code defined in Eq. (14). The circuit receives values of $\vec{l}_t$ for frame t, and returns a $\vec{p}_t$, which depends on the recent history of $\vec{l}$, which is stored in memory elements (black boxes). For an input of $\vec{l}_t=\{0;1\}$ and a memory state $\alpha_t=\{1;01\}$, the circuit would return $p_t=(111)$.

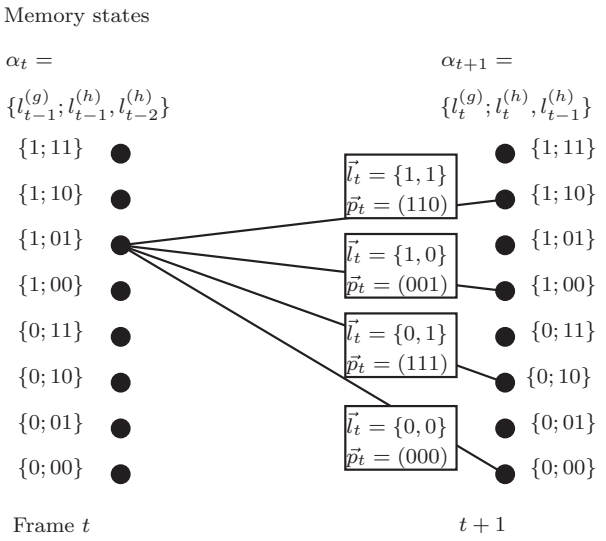

FIG. 5. The structure of transitions between memory states corresponding to the example illustrated in Fig. 4 using the convolutional code defined in Eq. (14). The recovery operation is determined by $\vec{p}_t=U_p(a_t,\vec{l}_t)=\{111\}$. Enumerating over all possible logical inputs $\vec{l}_t$, starting memory states a_t, and frames t, a complete trellis is constructed, shown in Fig. 6.

that the search space for optimising over l's does not grow exponentially with the size of the code.

From Eq. (13), it is simple to see that the memory state α_{t+1} depends on $\vec{l}_t$, and components of α_t. As a result, there are consistency conditions that relate α_{t+1} to α_t. These conditions are represented by a *trellis*, which depicts all valid *transitions* from α_t to α_{t+1}.

We now illustrate this construction for the example of a $\frac{k}{n}=\frac{1}{3}$ convolutional code. Figure 4 shows a (classical) circuit diagram relating $\vec{p}_t$ to $\vec{l}_t$, for the self-dual rate convolutional code defined by

$$\begin{aligned} G_Z^{\mathsf{T}}(D) &= [\quad D \qquad D \qquad 1], \\ H_Z(D) &= [1+D+D^2 \quad 1+D^2 \quad 1], \\ \mathrm{ISF}_Z(D) &= [\quad D \qquad D \qquad 0]. \end{aligned} \tag{14}$$

Reading off the highest powers of D in $G_Z^{\mathsf{T}}(D)$ and $H_Z(D)$, respectively, we see that $\nu_g=1$ and $\nu_h=2$, so that the memory state $\alpha_t=\{l_{t-1}^{(g)};l_{t-1}^{(h)},l_{t-2}^{(h)}\}$ is a list with three entries. In the equivalent circuit depicted in Fig. 4, the memory is represented by black boxes.

Figure 5 illustrates allowed transitions from state $\alpha_t=\{1;01\}$ to a new state α_{t+1} at frame t for the convolutional code defined in Eq. (14). Given that this code example is sufficiently simple, we show all possible choices for $\vec{l}_t$ for this transition, i.e., $\vec{l}_t=\{0;0\}$, $\vec{l}_t=\{0;1\}$, $\vec{l}_t=\{1;0\}$, and $\vec{l}_t=\{1;1\}$. We also show the corresponding codeword block frame $\vec{p}_t$, respectively, (000), (111), (001), and (110) for each possible choice of $\vec{l}_t$. Expanding Fig. 5 over several frames yields the trellis, shown in Fig. 6.

With the foregoing machinery in place, error correction works as follows. A set of errors, $\vec{\varepsilon}$, produces a syndrome $\vec{S}$. From $\vec{S}$, we use the ISF to calculate $\vec{e}^0$. Any path, $\vec{p}=(\vec{p}_1,\vec{p}_2,\ldots,\vec{p}_t,\ldots,\vec{p}_\tau)$, through the trellis that begins in the trivial state $\vec{p}_1=(0\ldots0;0\ldots0)$ and ends in the state $\vec{p}_\tau=(0\ldots0;0\ldots0)$ state corresponds to a valid decoding. The path $\vec{p}_{\min}$ that minimizes the Hamming weight of $\vec{p}_{\min}+\vec{e}^0$ is the most likely decoding solution. If no detectable errors occur so that $\vec{e}^0$ is trivial (i.e., if $\vec{e}^0=\{00\ldots\}$), then the lowest weight path through the trellis has $\vec{p}_{\min}=(000\ldots)$,

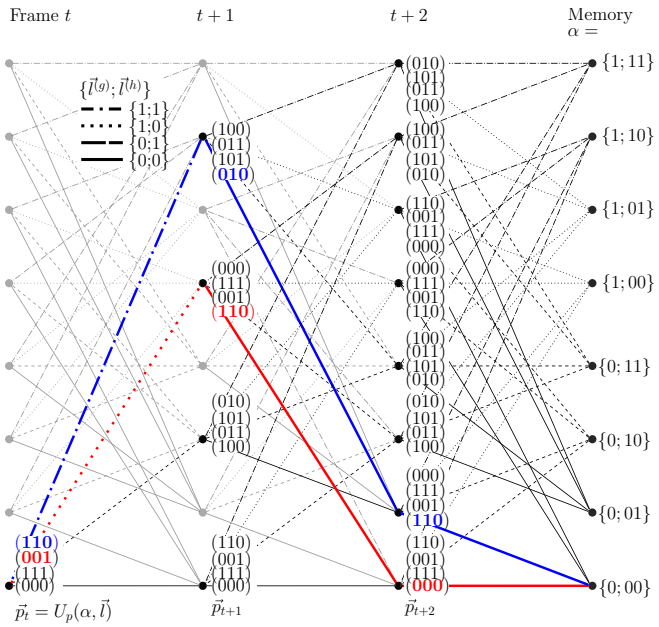

FIG. 6. The decoding trellis for the convolutional code in Eq. (14). Each transition corresponds to a different product of $\hat{g}$ and $\hat{h}$ at a given time step. The memory state records the values of previous transitions. Each path corresponds to a certain product of $\hat{g}$ and $\hat{h}$ terms. The blue and red paths are the most likely paths for the syndromes in Eqs. (15) and (16), respectively.

and $\mathrm{wt}(\vec{e}^{\,0} + \vec{p}_{\min}) = 0$. For nontrivial $\vec{e}^{\,0}$, finding this path is the core of the decoding problem.

Given a trellis, such as in Fig. 6, and priors for the qubit error marginals, a trellis decoding algorithm will return a suitable choice for $\vec{p}_{\min}$. Several algorithms exist to identify (optimally or suboptimally) the most likely trellis path. These include the Viterbi algorithm [32] the Bahl, Cocke, Jelinek, Raviv (BCJR) algorithm [33], and the Benedetto algorithm [34].

The last of these algorithms is a maximum *a posteriori* (MAP) SISO decoder, which is capable of dealing with parallel transitions, i.e., those for which several transitions between states α_t and α_{t+1} exist. Single-layer foliated codes necessarily have parallel transitions in their trellis descriptions, as will be shown in Sec. V. For this reason, we will use the Benedetto algorithm as a SISO decoder for convolutional codes. This is discussed in detail in Sec. VI.

1. Examples

Before concluding this section, we give two illustrative examples using the trellis shown in Fig. 6. Because the examples are sufficiently simple, we find the lowest weight path through the trellis by inspection. In practice, we use a modified version of Benedetto's trellis algorithm [34] to determine the most likely path through the trellis (see Sec. VI).

In the first example, the error pattern, $\vec{\varepsilon}$, is a single error occurring on the first bit in frame $t+1$, as indicated in the following equation:

frame:	…	$t-2$	$t-1$	t	$t+1$	$t+2$	…
$\vec{\varepsilon} =$	…	000	000	000	100	000	…
$\vec{S} =$	…	0	1	1	1	0	…
$\vec{e}^{\,0} =$	…	000	000	110	110	110	…,
$\vec{p}_B =$	…	000	000	110	010	110	…
$\vec{e}_{\min} =$		$\vec{e}^{\,0} + \vec{p}_B$					
$=$	…	000	000	000	100	000	….

(15)

Given $\vec{\varepsilon}$, we find $\vec{S}$ and $\vec{e}^{\,0}$, also shown in Eq. (15) (for this and the following example, we work through the calculation of $\vec{S}$ and $\vec{e}^{\,0}$ from $\vec{\varepsilon}$ in Appendix D). Using $\vec{e}^{\,0}$, we find (by inspection for this example) the path $\vec{p}_B$ through the trellis that minimizes $\mathrm{wt}(\vec{e}^{\,0} + \vec{p}_B) \equiv \mathrm{wt}(\vec{e}_{\min}) = 1$. This path is shown in blue in Fig. 6; the blue highlighted triples of binary values in the figure correspond, frame by frame, to the triples in $\vec{p}_B$. This path determines the optimal error correction procedure, which is to apply the correction operator $Z^{\otimes \vec{e}_{\min}}$. In this example, $\vec{e}_{\min} = \vec{\varepsilon}$, so the error correction would succeed since $\vec{e}_{\min}$ and $\vec{\varepsilon}$ are logically equivalent, i.e., the correction $\vec{e}_{\min}$ would return the system to the correct codeword.

In the second example, the error pattern, $\vec{\varepsilon}$, consists of two errors occurring on the first and second qubits in frame $t+1$. The most likely path through the trellis $\vec{p}_R$ is shown in red in Fig. 6. We have

frame:	…	$t-2$	$t-1$	t	$t+1$	$t+2$	…
$\vec{\varepsilon} =$	…	000	000	000	110	000	…
$\vec{S} =$	…	0	0	1	0	0	…
$\vec{e}^{\,0} =$	…	000	000	000	110	000	….
$\vec{p}_R =$	…	000	000	001	110	000	…
$\vec{e}_{\min} =$		$\vec{e}^{\,0} + \vec{p}_R$					
$=$	…	000	000	001	000	000	….

(16)

Again, the red highlighted triples of binary values in Fig. 6 correspond, frame-by-frame, to the triples in $\vec{p}_R$ in Eq. (16). The lowest weight recovery operation is a single error on the third qubit in frame t. In this example, the product of the recovery operation and physical errors is $\vec{e}_{\min} + \vec{\varepsilon} = \ldots 001\,110 \ldots$, which is a nontrivial codeword of G_Z^{T} in Eq. (14), that is, the decoder fails in this example.

This second example is illustrative: because this is $d = 3$ code, adjacent errors are not expected to be corrected. However, if errors are sufficiently far apart (determined by the code memory length), then they behave as if they were independent, and so the convolutional code can decode many more errors than $d/2$ if they are sparsely distributed.

V. FOLIATED CONVOLUTIONAL DECODING

In this section, we build on the trellis construction in Sec. IV to operate on independent sheets of a foliated convolutional code, accounting for additional ancilla. Decoding can be performed using the algorithm in Sec. VI. This represents a subroutine in the full foliated decoding algorithm which exchanges marginals between sheets. This is reviewed in Sec. V B.

A. Trellises for single decoding sheets

Consider a foliated code based on a rate $r = k/n$ CSS convolutional code with $n_z = |\mathcal{S}_Z|/\tau$ Z-like stabilizers and $n_x = |\mathcal{S}_X|/\tau$ X-like stabilizers per frame, so that there are $n = k + n_x + n_z$ physical qubits per frame. Foliated parity check operators are defined in Eq. (2). Figure 7(a) shows an example of a foliated rate 1/3 convolutional code, and a specific parity check operator, $\hat{P} = X_{a_1} X_{c_1} \ldots X_{c_6} X_{a_2}$, is indicated by the labeled vertices $a_1, c_1, \ldots, c_6$ and a_2, where the ancilla qubits are on the adjacent code sheets $m \pm 1$. Measuring all such operators associated to code sheet m yields the syndrome $\vec{S}_m$ for that sheet.

For the purposes of decoding, we introduce *decoding sheets*. A decoding sheet m is identified with the corresponding code sheet m: it refers to the same code qubits, but includes virtual ancilla qubits associated to the neighboring sheets.

The shaded elipse in Fig. 7(b) illustrates the formation of a single decoding sheet for a foliated convolutional code; this decoding sheet, m, is identified with the corresponding code sheet, m, in Fig. 7(a). Independent decoding of each decoding sheet can be performed by creating a virtual code associated with a single sheet of the foliated structure. This code accounts for the ancillary qubits a_1 and a_2 in adjacent sheets by introducing virtual ancilla a_1'' and a_2'. There are $2n_x$

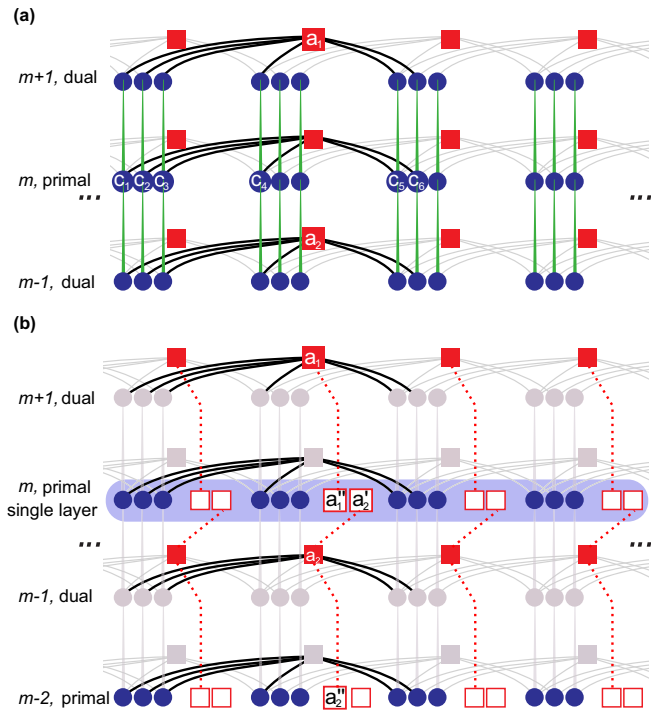

FIG. 7. (a) The foliated rate $\frac{1}{3}$ convolutional code defined by Eq. (14). The product of cluster stabilizers centered on the numbered qubits generates a parity check operator $\hat{P} = K_{c_1} K_{c_2} \ldots K_{a_2} = X_{c_1} X_{c_2} \ldots X_{a_2}$. All other parity check operators are translations of this seed. (b) The corresponding primal lattice. Dual elements have been grayed out. Because there is only $n_x = 1$ X-like stablizer per frame, we introduce $2n_x = 2$ virtual qubits, labeled a_i' and a_i'', corresponding to ancilla qubits $a_{1,2}$ on neighboring sheets. This results in a rate $\frac{k}{n+2n_x} = \frac{1}{5}$ for sheet m of the foliated structure. The marginals on virtual qubit a_1'', computed within sheet m, and virtual qubit a_1' in sheet $m+2$ (which ultimately refer to the same physical qubit a_1) are exchanged iteratively, so that the foliated decoder converges.

virtual ancilla associated to the ancilla on the adjacent code sheets, so the virtual code has rate $\frac{k}{n+2n_x}$.

We note that a physical ancilla qubit $a_{j,m}$ is represented virtually in two different decoding sheets $m \pm 1$ (as $a'_{j,m}$ and $a''_{j,m}$). This yields a consistency condition on ancilla marginals between neighboring sheets, which we discuss later.

Within a decoding sheet [see Fig. 7(b)], we begin by finding an initial recovery operation, $\hat{E}^0$, which satisfies a received syndrome, $\vec{S}$. As with the previous section this can be achieved by using ISFs. Setting the final $2n_x$ qubits in a frame to correspond to virtual ancilla qubits, the seed generator is given by

$$\overline{G}_Z^{\mathsf{T}}(D) = \begin{bmatrix} G_Z^T(D) & \mathbf{0}_{k\times n_x} & \mathbf{0}_{k\times n_x} \end{bmatrix}, \tag{17}$$

where G_Z^T refers to the generator matrix of the base convolutional code, as exemplified in Eq. (14). Similarly, the seed parity check operators and ISF are given by

$$\overline{P}(D) = \begin{bmatrix} H_X(D) & \mathbb{I}_{n_x\times n_x} & \mathbb{I}_{n_x\times n_x} \end{bmatrix},$$
$$\overline{\mathrm{ISF}}_Z(D) = \begin{bmatrix} \mathrm{ISF}_Z(D) & \mathbf{0}_{n_x\times n_x} & \mathbf{0}_{n_x\times n_x} \end{bmatrix},$$
$$\overline{H}_Z(D) = \begin{bmatrix} H_Z(D) & \mathbf{0}_{n_z\times n_x} & \mathbf{0}_{n_z\times n_x} \end{bmatrix}.$$

Additional pairs of gauge operators are generated from the degrees of freedom introduced by the extra ancilla qubits. The seed generators, stabilizers, ISFs, and gauges J_Z and J_X satisfy the orthogonality relations

$$D^i \begin{bmatrix} \overline{G}_Z^{\mathsf{T}}(D) \\ \overline{H}_Z(D) \\ \overline{\mathrm{ISF}}_Z(D) \\ \overline{J}_Z(D) \end{bmatrix} \begin{bmatrix} \overline{G}_X^{\mathsf{T}}(\widetilde{D}) & \overline{\mathrm{ISF}}_X(\widetilde{D}) & \overline{P}(\widetilde{D}) & \overline{J}_X(\widetilde{D}) \end{bmatrix} = \mathbb{I}\delta_{i0}, \tag{18}$$

which implicitly defines $\overline{J}_{X,Z}$. A valid choice of $\overline{J}_X$ is

$$\overline{J}_X = \left[\begin{array}{c|cc} \mathbf{0}_{n_x\times n} & \mathbb{I}_{n_x\times n_x} & \mathbf{0}_{n_x\times n_x} \\ \mathbf{0}_{n_x\times n} & \mathbf{0}_{n_x\times n_x} & \mathbb{I}_{n_x\times n_x} \end{array}\right], \tag{19}$$

which orthogonal to G_Z, H_Z and ISF_Z.

Generally, $\overline{J}_Z$ depends on the details of the code. Each $Z^{\otimes \vec{j}} \in J_Z$ is a set of operators which commute with G_X, P_X, and ISF_X, but anticommute with a single $X^{\otimes a} \in J_X$.

As an example, foliating the code in Eq. (14), we have

$$\overline{J}_Z(D) = \left[\begin{array}{ccc|cc} 1+D & 1 & 1 & D & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right]. \tag{20}$$

Because J_Z commutes with the stabilizers, they correspond to undetectable error patterns within the sheet. As an aside, using the Raussendorf lattice as an example, these would correspond to error chains that pass through a sheet in the "time"-like direction of foliation: they leave no syndrome data within the sheet (but would be detected by other sheets in the 3D lattice).

The set of valid recovery operations is given by

$$\mathcal{E} = \big\{ Z^{\otimes \vec{e}^0 + \vec{h} + \vec{g} + \vec{J}} | \vec{h} \in \mathrm{RowSpace}(\mathbf{H}_Z), \; \vec{g} \in \mathrm{RowSpace}\big(\mathbf{G}_Z^{\mathsf{T}}\big), \; \vec{J} \in \mathrm{RowSpace}(\mathbf{J}_Z) \big\},$$

where

$$\mathbf{J}_Z = \begin{bmatrix} \ldots & J^{(1)} & \ldots & J^{(\nu_j)} & \ldots & & \\ & \ldots & J^{(1)} & \ldots & J^{(\nu_j)} & \ldots & \\ & & \ldots & J^{(1)} & \ldots & J^{(\nu_j)} & \ldots \end{bmatrix}_{2\tau\times n\tau}, \tag{21}$$

by analogy with Eq. (4). In the example in Eq. (20),

$$J^{(1)} = \left[\begin{array}{ccc|cc} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right]$$
$$\text{and} \quad J^{(2)} = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]. \tag{22}$$

Given $\mathbf{H}_Z$, $\mathbf{G}_Z^{\mathsf{T}}$, and $\mathbf{J}_Z$, a trellis may be constructed, closely mirroring the trellis construction in Sec. IV. The main difference is that the decoding trellis must be modified to account for $\overline{J}_Z$ terms, which represent the virtual ancilla in each sheet. This gives rise to larger memories and more allowed transitions. The $\vec{l}$ terms are modified to include $l_t^{(J)}$ components.

Once the trellis is formed, a soft-input–hard-output decoder will take as input marginal error probabilities on the

qubits, and return $\vec{h}$, $\vec{g}$, and $\vec{J}$, which correspond to a specific, maximum likelihood error pattern consistent with the syndrome data on sheet m. More suited to our goal is a soft-input–soft-output (SISO) decoder which returns updated *a posteriori* probabilities for qubit error marginals. We discuss such a decoder in Sec. VI.

B. Ancilla marginal exchange for consistency between decoding sheets

When decoding is performed on each convolutional code sheet, a given ancilla qubit, $a_{k,m}$, is associated to corresponding virtual ancilla, $a''_{k,m}$ and $a'_{k,m}$, from each of the neighboring decoding sheets $m+1$ and $m-1$. For consistency, we require the SISO marginal *a posteriori* error probabilities on $a'_{k,m}$ and $a''_{k,m}$ to match, i.e., $P(a''_{k,m}) = P(a'_{k,m})$.

Independent SISO decoding of adjacent decoding sheets does not automatically respect this constraint, so we perform sequential rounds of independent decoding, and iteratively exchange marginals, $P(a''_{k,m}) \leftrightarrow P(a'_{k,m})$ between each round, until the marginals are satisfactorily converged.

This approach is similar to belief propagation on a modified Tanner graph, with a message passing schedule, which gives priority to intrasheet messages over that of intersheet messages [26,35–37]. It is unlikely that the process we have proposed is optimal, but good numerical decoding results are still possible as shown in Sec. VII for turbo codes.

In practical settings it may be desirable to terminate the iterative message passing after a fixed number of rounds. In this case, the marginals on the ancilla from adjacent sheet decoders may not have converged. One heuristic resolution to this is to average the inconsistent marginals on each ancilla, $P(a_{k,m}) := (P(a''_{k,m}) + P(a'_{k,m}))/2$, and then make a hard decoding choice, by choosing the most likely error configuration at each ancilla. A final decoding round is performed on each sheet. This ensures that the decoding falls within the code space.

VI. TRELLIS SISO ALGORITHM

So far, we have concentrated on constructing trellises for convolutional codes. As already mentioned, we then use the trellis to find optimal error configurations (i.e., a hard decoding) or assign error marginals (i.e., soft decoding) that are consistent with the syndrome data.

In this section, we present a modified MAP SISO trellis decoder, following the approach of Benedetto *et al.* [34] on classical convolutional codes. The modified algorithm compares codewords and stabilizers against a trial error pattern, $\vec{e}^{\,0}$, determined from the syndrome data, and calculates marginal error probabilities consistent with it.

The algorithm consists of three stages: (1) a forward pass, which passes through the trellis from left to right, (2) a backward pass, which passes through the trellis from right to left, and (3) a local update, which determines marginals using information from forward and backward passes.

In the rest of this section, we detail the core of the SISO algorithm. As in prior sections, the trellis memory state at frame t is α_t. The initial error pattern $\vec{e}_t^{\,0}$ is calculated from the syndrome $\vec{S}$ and the ISF. The physical operations associated with a transition through the trellis are denoted $\vec{p}_t$.

A. Forward pass algorithm

The forward pass algorithm assigns a probability for each memory state, $A(\alpha_t|\vec{S}_{i\leqslant t})$, as we pass through the trellis starting from the first frame, $t=1$ (i.e., the leftmost frame in the presentation of Fig. 6) and traversing to $t=\tau$ (right). Here, $\vec{S}_{i\leqslant t}$ is a shorthand notation which indicates that only syndrome information up to frame t is used to calculate likelihoods. At each frame, there are $2^{(kv_g+n_z v_h)}$ memory states to store. Note that for decoding sheets in the foliated construction a memory term v_J must be incorporated for the J_Z gauges.

We assign initial probabilities $A(\alpha_1=\vec{0})=1$ and $A(\alpha_1\neq\vec{0})=0$ to memory states at frame $t=1$. The forward pass algorithm then computes probabilities for subsequent memory states as

$$A(\alpha_{t+1}|\vec{S}_{i\leqslant t}) = \sum_{\vec{l}_t} A(\alpha_t|\vec{S}_{i\leqslant t-1})\Pr\big(\vec{p}_t+\vec{e}_t^{\,0}\big)\Pr(\vec{l}_t), \quad (23)$$

where $\vec{p}_t = U_p(\alpha_t,\vec{l}_t)$ is given by Eq. (12), and we recall from Eq. (13) that α_t on the RHS of Eq. (23) is determined by α_{t+1} and $\vec{l}_t$. $\Pr(\vec{p}_t+\vec{e}_t^{\,0})$ is the *a priori* probability of the error pattern $\vec{p}_t+\vec{e}_t^{\,0}$, which depends on the details of the prior error mode; for our purposes, this is just an i.i.d. error process on each of the physical qubits, so that $\Pr(\vec{p}_t+\vec{e}_t^{\,0})$ is given by the binomial formula. $\Pr(\vec{l}_t)$ is the *a priori* probability of undetectable error processes [generated by Eq. (8)]. For convolutional codes, we take $\Pr(\vec{l}_t)$ to be a constant (so that it factors out of A), however, it will become important when we consider turbo codes, in which physical errors in the inner code affect logical priors in the outer code.

B. Backward pass algorithm

The backward pass algorithm is identical to the forward pass, but working now from the last frame, $t=\tau$ back to the first, $t=1$. The update rule is given similarly,

$$B(\alpha_t|\vec{S}_{i\geqslant t1}) = \sum_{\vec{l}_t} B(\alpha_{t+1}|\vec{S}_{i\geqslant t+1})\Pr\big(\vec{p}_t+\vec{e}_t^{\,0}\big)\Pr(\vec{l}_t). \quad (24)$$

The initial conditions are set as $B(\alpha_\tau=\vec{0})=1$ and $B(\alpha_\tau\neq\vec{0})=0$, where τ is the final frame.

C. Local update algorithm

The local update calculates the likelihoods of physical error patterns, $\vec{e}_{p,t}=\vec{p}_t+\vec{e}_t^{\,0}\in\mathbb{Z}_2^n$, at frame t, given a valid error configuration $\vec{e}_t^{\,0}$ (which is itself derived directly from the syndrome data). That is, we calculate the marginals $P(\vec{e}_{p,t}|\vec{S})$, and the marginals over logical bits $P(\vec{e}_{l,t}|\vec{S})$, where $\vec{e}_l\in\mathbb{Z}_2^{k+n_z}$ is a specification of the logical states at frame t. These marginals depend in turn on the marginal beliefs of memory states computed in the forward, A, and backward, B, pass algorithms. As inputs, the decoder uses an initial prior distribution, $\Pr(\vec{e}_{p,t})$, for the error configuration $\vec{e}_{p,t}$, and the logical error patterns, $\Pr(\vec{l}=\vec{e}_{l,t})$, as well as the syndrome,

and computes the marginals

$$P\left(\vec{e}_{p,t}|\vec{e}_t^{\,0}\right) = \mathcal{N}_p \sum_{\substack{\vec{l}_t,\,\alpha_t\,:\\ \vec{p}_t = U_p(\alpha_t,\vec{l}_t)}} A(\alpha_t)B(\alpha_{t+1})\Pr\left(\vec{p}_t + \vec{e}_t^{\,0}\right)\Pr(\vec{l}_t), \tag{25}$$

$$P\left(\vec{e}_{l,t}|\vec{e}_t^{\,0}\right) = \mathcal{N}_l \sum_{\substack{\alpha_t\,:\\ \vec{p}_t = U_p(\alpha_t,\,e_{l,t})}} A(\alpha_t)B(\alpha_{t+1}) \times \Pr\left(\vec{p}_t + \vec{e}_t^{\,0}\right)\Pr(\vec{e}_{l,t}), \tag{26}$$

where $\mathcal{N}_p$ and $\mathcal{N}_l$ are normalization constants chosen to ensure that $\sum_{\vec{e}_{p,t}} P(\vec{e}_p|\vec{e}_t^{\,0}) = 1$ and $\sum_{\vec{e}_{l,t}} P(\vec{e}_l|\vec{e}_t^{\,0}) = 1$. Again, we recall from Eq. (13) that α_t on the RHS of Eq. (23) is determined by α_{t+1} and $\vec{l}_t$.

Equations (23)–(26) are the ingredients for the *sum-product* belief propagation algorithm: the forward and backward pass algorithms each run independently, and then the local update calculates marginals for each of the 2^n possible error configuration over each of the τ frames. Storing this information requires memory $\sim\tau 2^n$. As an aside, the *max-sum* algorithm, which has some practical performance benefits, approximates the *sum-product* algorithm, by summing over logarithms of marginals [33].

VII. FOLIATED TURBO CODES

Our main motivation for studying turbo codes is to demonstrate the foliated construction and BP decoder in an extensible, finite-rate code family. Practically, these and other finite-rate codes may have applications in fault-tolerant quantum repeater networks [38–41], where local nodes create optimal clusterized codes to reduce resource overheads or error tolerance [42], however, we do not address these applications here.

A. Turbo code construction

A turbo code is essentially a concatenation of two convolutional codes, albeit with an *interleaver* between them. When convolutional codes fail, they tend to produce bursts of errors on logical bits. Turbo codes address this by concatenating encoded (qu)bits from the *inner* convolutional code into widely separated logical (qu)bits in the *outer* code. The interleaver is simply a permutation Π on the inner logical qubits, and serves to transform a local burst of errors from the inner decoder into widely dispersed (and thus approximately independent) errors that the outer decoder is likely to correct.

For the numerical results we present in this section, we choose Π to be a completely random permutation on the inner code. This is a conventional choice for benchmarking turbo codes, however, a completely random permutation leads to highly delocalized encodings. This may be undesirable in the quantum setting, and the optimal choice of interleavers was discussed in Refs. [43,44]. In the context of constructing clusterized codes, the interleaver choice will affect the weight of stabilizers. In Sec. IX, we return to this issue, and show that by choosing a structured interleaver, we reduce the weight of stabilizers substantially. This reduces the weight of correlated errors that build up during the systematic construction of the cluster state resource.

Turbo codes are generated using underlying convolutional codes. We use two different convolutional codes, one with $d = 3$, which we refer to as the C3 family of codes, and one with $d = 5$, which we refer to as the C5 family [45]. When embedded as clusterized codes the distance of these codes are reduced to 2 and 3, respectively. This represents the effective code distance d_{eff} of a single clusterized code sheet, which forms part of the larger foliated structure. While the effective distances are diminished for codes acting within a single sheet, the distance of the foliated convolutional codes remain 3 and 5, respectively. This is because some error patterns, which are undetectable within a single sheet decoder are detectable by neighboring layers, as discussed in Sec. V.

At the end of this section, we present numerical results about the decoding performance of two families of foliated turbo codes, T9 and T25 codes. T9 codes are generated from the concatenation and interleaving of two C3 codes; similarly, the T25 code is formed from the concatenation and interleaving of two C5 codes. The distances of these turbo codes are $d_{T9} = 9$ and $d_{T25} = 25$, respectively.

B. Turbo code decoders

We now develop a decoding method for foliated turbo codes based on the trellis construction methods outlined in Sec. V and the SISO decoder in Sec. VI. The trellis construction and SISO decoder allow for the decoding of foliated convolutional codes by iteratively decoding individual sheets followed by a series of marginal exchanges on ancilla qubits.

Turbo codes consist of an interleaved concatenation of two convolutional codes. The decoding approach is shown schematically in Fig. 8. Using an *a priori* distribution of qubit error states $\Pr(q_I)$ a soft-input–soft-output (SISO) decoder is implemented and marginal values for qubits $P(q_I)$ and logical qubits $P(l_I)$ are calculated for each decoding sheet in the foliated code. Marginals are then exchanged between the shared ancilla qubits in neighboring layers and used as prior values for a new round of trellis decoding. This process is applied iteratively.

Within a sheet decoder (i.e., red boxes in Fig. 8), the logical marginals $P(l_I)$ are deinterleaved (Π^{-1}) and used as priors for the outer decoder $P(q_O)$. The same process of ancilla marginal exchange is performed and the decoding process is iterated. Finally, the qubit marginals from the outer decoder $P(q_I)$ are interleaved (Π) and used as logical priors $P(l_I)$ for the inner code.

C. Numerical results for turbo codes

As noted earlier, X errors on the foliated cluster commute with parity check measurements. Thus, for our simulations, we assume a phenomenological error model in which uncorrelated Z errors are distributed independently across the cluster with probability p. The decoder performance is quantified in terms of both word error rate (WER), which is the probability of one or more logical errors across all k encoded qubits, and the bit error rate (BER), which is the probability of an error in any of the encoded qubits. These are defined formally in Eqs. (28) and (30).

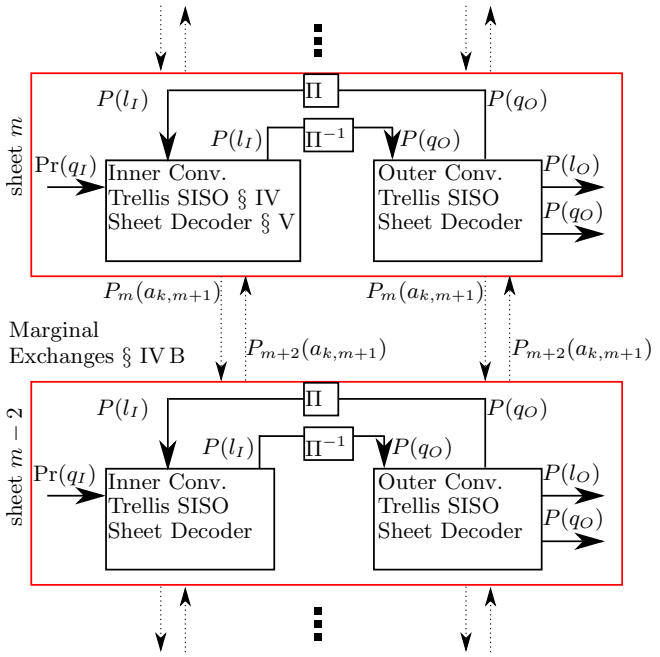

FIG. 8. The decoding process for foliated turbo codes. An inner sheet convolutional decoder is run for each sheet in the foliated code. Ancillas marginals are exchanged between neighboring primal (dual) layers, i.e., next-nearest neighbors, shown as dotted arrows. The process is then iterated. The logical marginals $P(l_I)$ are then deinterleaved (Π^{-1}) and used by the outer sheet convolutional decoders. Ancilla marginals are exchanged, and as before the process is iterated. The outer sheet decoders pass physical marginals $P(q_O)$ to corresponding inner decoders via the interleaver Π for further decoding rounds. This completes the turbo feedback loop. The outer sheet decoders determine the final marginal outputs, $P(l_O)$ and $P(q_O)$, after the iterative feedback and marginal exchanges are completed.

One common approach to numerically evaluating code performance curves is to sample error patterns, $\vec{\varepsilon}$, use the decoder to find a recovery operation $\vec{e}_{\text{rec}}$, and then test for success or failure of the decoder with respect to the specific error sample. The decoder is successful if $\vec{\varepsilon}_l \equiv \mathbf{G} \cdot (\vec{\varepsilon} + \vec{e}_{\text{rec}}) = \vec{0}$, and unsuccessful if not. If the decoder fails on any logical qubit, this constitutes a word error; the hamming weight of $\vec{\varepsilon}_l$ counts the number of logical bit errors.

One approach to generating code performance curves is to fix the error rate per physical qubit, p, then generate N_{trials} error configurations at that error rate. The decoder will fail on some number of those trials, and then the WER is a function of the error rate, given by

$$\text{WER}(p) = \left.\frac{\#\text{Word Failures}}{N_{\text{trials}}}\right|_p. \tag{27}$$

The error rate is then incremented, $p \to p'$, and new trials are run for the new value of p. Similarly, we define the BER (at a given error rate, p, per physical qubit) to be

$$\text{BER}(p) = \left.\frac{\#\text{Bit Failures}}{k\, N_{\text{trials}}}\right|_p. \tag{28}$$

In the numerical results reported here, we employ binomial sampling, in which we sample over a fixed number of errors, $j = 0, 1, 2, \ldots$, and compute the failure probability for each j,

$$P_{\text{Word}}(j) = \left.\frac{\#\text{Word Failures}}{N_{\text{trials}}}\right|_{\text{fixed } j}, \tag{29}$$

for a suitable range of values of j. We then use the binomial formula to relate $\text{WER}(p)$ to $P_{\text{Word}}(j)$:

$$\text{WER}(p) = \sum_{j=0}^{n} P_{\text{Word}}(j) \binom{n}{j} p^j (1-p)^{n-j}. \tag{30}$$

In practice, the upper limit of the sum can be truncated to much less than n. Similarly, we define

$$P_{\text{Bit}}(j) = \left.\frac{\#\text{Bit Failures}}{k\, N_{\text{trials}}}\right|_{\text{fixed } j}, \tag{31}$$

so that the BER is given by

$$\text{BER}(p) = \sum_{j=0}^{n} P_{\text{Bit}}(j) \binom{n}{j} p^j (1-p)^{n-j}. \tag{32}$$

In what follows we present numerical results for code performance as a function of the code size $k = nr$, and for several different foliation depths, where we vary the number of code sheets, L. We note that the case $L = 1$ is a special case: it corresponds to decoding a single clusterized code sheet, in which errors may also occur on the ancilla qubits (i.e., the red squares in Fig. 1). This is equivalent to decoding the base (i.e., unclusterized) code, but including faulty syndrome measurements.

As an aside, we validate this binomial sampling method by comparing the numerical results of Eq. (29) with numerical results generated by the conventional sampling approach, Eq. (30). Figure 10 contains a series of WER trials generated using the conventional sampling with for the case of $k = 40$ and $L = 6$ (second panel, LHS), shown as points with error bars, using $N_{\text{trials}} = 10^5$ for the smallest value of p. These points are in close agreement with the data generated through binomial sampling (solid lines).

1. Numerical results for T9 turbo codes

Figure 9 shows the performance of the $[n, k = n/16, 9]$ T9 self-dual foliated turbo code. We see that for a given foliation depth L, the performance *degrades* with the size of the code k. This indicates that the foliated T9 code is a poorly performing code.

This is explained by considering the effective distance of the clusterized, and then foliated T9 code. The T9 code is generated using two constituent rate $r = \frac{1}{3}$ C3 codes. By construction these codes have a distance of 3, however, when clusterized into a single code sheet (which is equivalent to accounting for noisy stabilizer measurements that generate faulty syndrome data) the distance is reduced to 2. In this setting, the seed generator and stabilizer are

$$G = [D \quad D \quad 1 \quad | \quad 0 \quad 0], \tag{33}$$

$$H = [1 + D + D^2 \quad 1 + D^2 \quad 1 \quad | \quad 1 \quad 1], \tag{34}$$

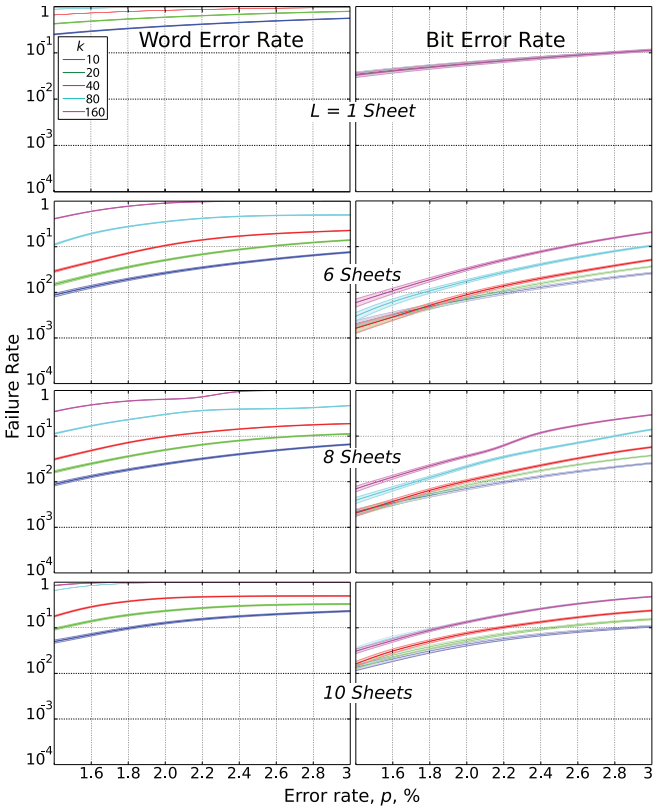

FIG. 9. Numerical performance results for the foliated T9 $[n, k = n/16, 9]$ turbo code, for different numbers of foliated layers, $L = 1, 6, 8, 10$ (rows), as a function of the error rate per qubit, p. Different colors correspond to different code sizes, $k = nr = 10$ (lowest curve), 20, 40, 80, and 160 (highest curve) logical qubits; shading indicates $\pm 1\sigma$. Word error rate (left column) counts any error(s) across all k logical qubits. Bit error rate (right column) counts the failure rate per logical qubit. The T9 code does not exhibit threshold behavior: its performance degrades as the code size grows.

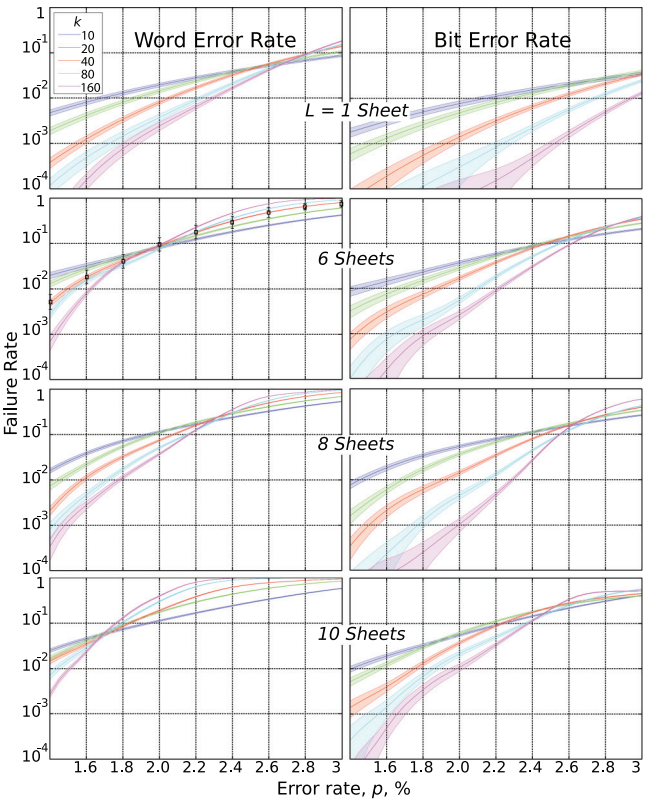

FIG. 10. Numerical performance results for the foliated T25 $[n, k = n/16, 25]$ turbo code, for different numbers of foliated layers, L (rows), as a function of the error rate per qubit, p. Different colors correspond to different code sizes with the number of logical qubits indicated as $k = nr = 10$ (shallowest curves), 20, 40, 80, and 160 (steepest curves); shading indicates $\pm 1\sigma$. The word error rate (left column) counts any error(s) across all k logical qubits. Bit error rate (right column) counts the failure rate per logical qubit. The T25 code exhibits thresholdlike behavior, in that for small error rates, the performance of the code improves with code size. In order to verify the binomial sampling process, a series of trials (shown as squares with error bars black for $k = 40$, $L = 6$) were performed at each value of p, and then evaluating Eq. (30).

respectively. [Here, we use the notation that qubits the left of the vertical bar are code qubits in the C3 code, and those to the right of the vertical bar are ancilla qubits associated to code stabilizer measurements; these correspond to quintuplets of qubits in each frame of the shaded code sheet in Fig. 7(b).]

For the C3 code, an example of an undetectable weight 2 error pattern error pattern in a single code sheet is $\vec{e} = [001|10]$, which has support on one of the ancilla. Since the distance of the clusterized code sheet is 2, it is reduced to an error *detecting* code within the sheet. We note that if this specific error pattern were isolated within a larger foliated structure, it *would* be detected by parity checks in the adjacent sheets, which have support on the affected ancilla qubit.

Given the reduced distance of the clusterized code, we do not necessarily expect this code to perform well in the foliated regime. This is borne out in the numerical results: for a given foliation depth, L; the T9 code has no threshold.

Though this negative result is unsurprising given the foregoing discussion on the clusterized code distance, we show this as an example of a poorly-performing foliated code. The simplest way to rectify this issue is to increase the underlying code distance, which we do in the next example.

2. *Numerical results for T25 turbo codes*

Figure 10 shows the performance of the $[n, k = n/16, 25]$ T25, self-dual foliated turbo code. For each L, there is a threshold error rate around $p \sim 2\%$, below which the code performance improves with code length (up to at least 160 encoded logical qubits per code sheet, encoded into 4160 physical qubits per sheet), consistent with (pseudo-)threshold behavior seen in turbo codes [23]. As L increases, the threshold decreases, more pronouncedly for the WER than the BER. The range of k and L that we can simulate is limited by computational time, so we cannot explore the asymptotic performance for large L. Nevertheless, numerics indicate that foliated turbo codes perform quite well for moderate depth foliations.

We note that the foliated construction transforms a clusterized code into a fault tolerant resource state, but with a reduced threshold. This is seen in Fig. 10, in which the threshold is seen to reduce with the number of sheets in

the foliation. A similar effect is also seen in Raussendorf's foliated surface-code construction, in which the fault-tolerant threshold $\lesssim 1\%$ is smaller than the $\sim 11\%$ threshold (assuming perfect stabilizer measurements) for the surface code on which it is based.

One important difference between the surface code and a turbo code family is that the code distance is fixed in the latter, whereas it grows in the former. As a result, we do not expect this threshold behavior to survive for large foliation depths: this would be analogous to the degradation in performance of Raussendorf's construction if the transverse code size were held fixed, while the foliation depth were increased. In the numerical results shown, the code distance is fixed at $d = 25$, and so we expect that for foliation depths $L \gg d$, the threshold will disappear; consequently, the code is more correctly described as having a *pseudothreshold*. Nevertheless, there may be applications where this is sufficient for practical purposes.

VIII. FOLIATED BICYCLE CODES

A. Construction

Bicycle code are a class of finite-rate LDPC codes. They are self-dual CSS codes generated by sparse circulant matrices. A circulant matrix is formed by a seed row vector which is rotated by one element in each successive row in the matrix. For a binary sparse cyclical matrix $C = m \times m$ with row weight w, a bicycle code can be defined by $H_X = H_Z = [C|C^{\mathsf{T}}]$. By construction H_X and H_Z are orthogonal:

$$H_X H_Z^{\mathsf{T}} = [C|C^{\mathsf{T}}][C|C^{\mathsf{T}}]^{\mathsf{T}} = CC^{\mathsf{T}} + C^{\mathsf{T}}C = \mathbf{0}. \tag{35}$$

To create the generator matrix k rows are removed from H. This generates a $[[2m, k, d \approx 2w]]$ quantum code. Here, the distance is only approximately $2w$ and will depend on the rows removed from C and the construction of C itself.

We can separate the code into two Tanner graph representations, corresponding to X and Z stabilizers. Since bicycle codes are self-dual the Tanner graphs will be identical in both cases.

In the foliated setting, stabilizers are parity check operators of the form given in Eq. (2). The code can be separated into two Tanner graphs, one which contains qubits within the primal sublattice, and one which contains on qubits in the dual sublattice. For example, the primal Tanner graph contains the code qubits in odd sheets $2m + 1$ and ancilla qubits in even sheets $2m$. Primal parity checks are parity check operators, which are centered on the sheets $2m + 1$.

B. Decoding

Bicycle decoding is typically performed using belief propagation on the Tanner graph representation of the code [36,46–48]. A variable node corresponds to a given physical qubit; and records the likelihood of all possible errors on that qubit. A factor node corresponds to a given stabilizer, which constrains the possible error states of the connected variable (qubit) nodes.

A factor graph $G = (V, E)$ is a bipartite graph defined by the set, $V = A \cup I$, of variable nodes $I = \{q_1, q_2, \ldots, q_n\}$, and factor nodes, $A = \{a_1, a_2, \ldots, a_{(n-k)/2}\}$, and edges, E, between variable and factor nodes, $E = \{(q, a)|a \in A, q \in N(a)\}$, where $N(a)$ is the set of all variables, which appear in constraint a.

The belief propagation algorithm calculates marginal distributions for the possible error states of each qubit by using repeated message passing. A *belief*, $b_i(\varepsilon_j)$, represents the probability that qubit q_i has suffered error ε_j; the index j enumerates over possible errors in the error model. This belief is calculated from *messages*, $m_{a\to q}(\varepsilon_j) \in [0, 1]$, in which factor nodes, a, report a marginal probability that node q has suffered error ε_j,

$$b_i(\varepsilon_j) = \frac{1}{\mathcal{N}_i} \prod_{a \in N(q_i)} m_{a\to q_i}(\varepsilon_j), \tag{36}$$

where $\mathcal{N}_i$ is a normalization condition to ensure $\sum_{\varepsilon_j} b_i(\varepsilon_j) = 1$ at each q_i.

To calculate $b_i(\varepsilon_j)$, we also pass messages, $m_{q\to a}(\varepsilon_j) \in [0, 1]$ from qubit nodes to check nodes, reporting the likelihood that q is subject to error ε_j. The values of messages in both directions are determined by iterating over the following consistency conditions:

$$m_{a\to q}(\varepsilon_j) = \sum_{\vec{e}_p : p \in N(a)\setminus q} f_a(\vec{e}_p|\varepsilon_j \text{ on } q) \prod_{r \in N(a)\setminus q} m_{r\to a}(e_{p,r}), \tag{37}$$

$$m_{q\to a}(\varepsilon_j) = \prod_{b \in N(q)\setminus a} m_{b\to q}(\varepsilon_j). \tag{38}$$

Here we sum over all possible configurations of errors $\vec{e}_p$ over the neighbors of a (excluding the target qubit q), and $f_a(\vec{e}_p|\varepsilon_j \text{ on } q) \in [0, 1]$ are constraint functions that return the *a priori* likelihood of the error configuration $\vec{e}_p$ given that qubit q is subject to error ε_j, and $e_{p,r}$ is the restriction of the error configuration $\vec{e}_p$ to qubit r. The function f serves two purposes: it vanishes on error configurations that are inconsistent with syndrome data, and otherwise returns the likelihood of a valid error configuration. For our numerical simulations, we will assume independently distributed errors, which implicitly defines f. We note in passing that f may be tailored to correlated error models if necessary.

To begin the iterative message passing, we initialize the messages on the RHS of Eq. (37) using the *a priori* error model

$$m_{q\to a}(\varepsilon_j) = \Pr(\varepsilon_j). \tag{39}$$

Belief propagation is exact on tree graphs, allowing for the factorization of complete probability distribution into marginals over elements, and Eqs. (37) and (38) naturally terminate at the leaves of the tree [35]. For loopy graphs, the iterative message passing does not terminate in a fixed number of steps; rather we test for convergence of the messages to a fixed point. In some cases, particularly at higher error rates, the message passing may not converge; in this case we simply register a decoding failure on the subset of logical qubits that are affected. Further, the presence of many short cycles may lead to poor performance of the decoder. In the foliated bicycle code, there are numerous short, intersheet, graph cycles, however, in the numerical results we present

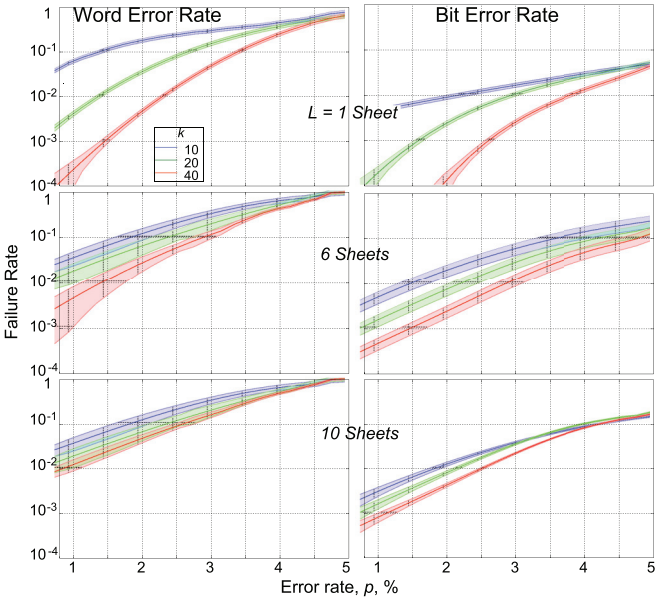

FIG. 11. Numerical performance results for a foliated $[n, k = n/16, d \approx 26]$ bicycle code, for different numbers of foliated layers, $L = 1, 6, 10$ (rows), as a function of the error rate per qubit, p. Different colors correspond to different code sizes, $k = nr = 10$ (shallowest curves), 20, and 40 (steepest curves), shading indicates $\pm 1\sigma$. Word error rate (left column) counts any error(s) across all k logical qubits. Bit error rate (right column) counts the failure rate per logical qubit.

in the next section, we see empirically that the decoder is effective nonetheless.

Finally, we note that the message passing algorithm described here can be applied directly to the full foliated code. Here, the marginal exchange between sheets happens concurrently with intrasheet message passing, i.e., Eq. (38) performs both intersheet marginal exchange and intrasheet message passing.

C. Numerical results for bicycle codes

We analyze the performance of the codes as a function of the code size $k = nr$, and the number of foliated layers L. Figure 11 shows the performance of a $d \leqslant 26$,[3] $r = 1/16$, bicycle code, based on Monte Carlo simulations of errors. We use the same binomial sampling schedule as described in Section VII C.

For each L, there is a threshold error rate around $p \sim 4.5\%$, below which the code performance improves with code length (up to at least 40 encoded logical qubits per code sheet). As with the Turbo codes, as L increases, the performance increase gained from larger codes is diminished. This result shows that LDPC bicycle codes are potentially promising codes for foliating.

[3]This distance bound is established by a heuristic minimisation of the length of the logical operators.

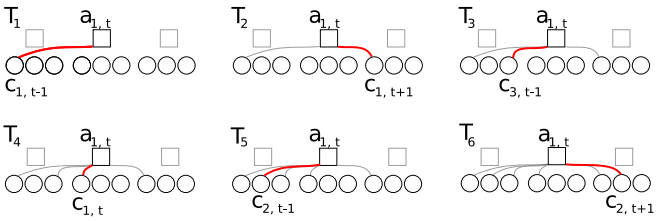

FIG. 12. A suitable time ordering for implementing C-PHASE gates between ancilla qubits and code qubits for the C3 convolutional code defined in Eq. (14), using the transpose interleaver defined in Eq. (41). Translations of these gates generate the full clusterized code. Thick, red lines indicate C-PHASE to implement at the corresponding time T_j; fine grey lines indicate previously implemented gates. The distance between frames t and t', and frames t' and t'' are $\frac{\tau}{3}$.

IX. CLUSTERIZED CODE ARCHITECTURE

In this section, we analyze the gate based implementation of cluster state resources for clusterized convolutional and turbo codes. The faulty implementation of cluster bonds (C-PHASE gates) causes correlated errors to arise during the construction of the cluster state resource. It is important to model these types of errors and ensure that the decoding process is fault-tolerant.

A. Schedule for cluster-state construction

Cluster state construction requires the preparation of resource $|+\rangle$ qubits and the implementation of C-PHASE gates, $\Lambda(a, b)$, between pairs of qubits. The gates must be implemented over a series of time steps so that during any given time step, T_j, any qubit is addressed by at most one phase gate. To generate clusterized codes phase, gates are implemented between ancilla qubits and code qubits according to the Tanner graph of the $\mathcal{S}_Z$ stabilizers. The number of ancilla qubits is $|\mathcal{S}_Z|$. The minimum number of time steps required to implement pairwise C-PHASE gates between each ancilla and the code qubits is proportional to the weight of the largest stabilizer.

The bonds in a clusterized convolutional code can be characterized by a series of qubit pair operations $\Lambda_{T_m}(a_{i,j}, c_{i',j'})$, where $a_{i,j}$ refers to the ith ancilla qubit in frame j, $c_{i',j'}$ refers to the i'th code qubit in frame j' and T_m is a "time" index in the cluster construction schedule.

As an example consider the C3 code defined in Eq. (14). The parity check operators are weight 6, and the corresponding ancilla qubit are represented by square vertices in the figure. As a result, this clusterized code may be implemented in six time steps, $T_1, \ldots, T_6$. A possible schedule for implementing C-PHASE gates is shown in Fig. 12. Thick, red lines indicate C-PHASE to implement at the corresponding time T_j; fine grey lines indicate previously implemented gates. Implied, but not shown are simultaneous translations of these gates across all frames, indexed by $\ldots, t-1, t, t+1, \ldots$.

For the case of Turbo codes, the number of cluster bonds between an ancilla qubit for an outer parity check depends on the weight of the outer convolutional parity check, the weight of the inner generator, and the choice of interleaver.

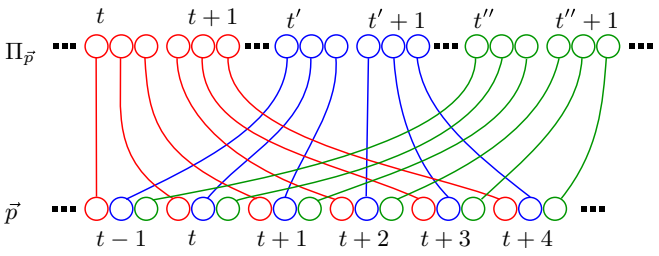

FIG. 13. The transpose interleaver for a rate $\frac{1}{3}$ code. The interleaver takes a sequence of frames containing bits $\vec{p}$ and returns a sequence $\Pi_{\vec{p}}$, containing the first bits in every frame, followed by the second bits in every frame and then the third bits in every frame. If $\vec{p}$ contains τ frames, then $t' = t + \tau/3$ and $t'' = t + 2\tau/3$.

The outer parity check is formed by encoding using the inner code generator. In the case of a random interleaver and a large code, each bit within the convolutional parity check is distant from each other bit. As a result the total weight for the outer parity check will be $\mathrm{wt}(H_{\mathrm{outer}}) \times \mathrm{wt}(G_{\mathrm{inner}})$.

As we discussed in Sec. VII A, the choice of interleaver has an effect on the weight of the code stabilizers. In the T9 and T25 clusterized turbo codes, a completely random interleaver will generate stabilizers with weight up to 18[4] and 98[5], respectively. In what follows, we describe more structured interleavers that reduce these weights to 10 and 26, respectively.[6]

An interleaver that achieves these lower weight stabilizers is one which permutes the order of bits $\vec{p}$ such that in a block of f frames, the first bit within each frame is mapped to a single contiguous block $\vec{b}_t$ by the permutation; the second bit within each frame is mapped to another, well spaced block, $b_{t'}$, and so on. This is illustrated in Fig. 13. The input sequence over τ frames is

$$\begin{aligned}\vec{p} &= \left(\left(p_1^1, p_1^2, \ldots, p_1^n\right), \left(p_2^1, \ldots, p_2^n\right), \ldots, \left(p_\tau^1, \ldots, p_\tau^n\right)\right) \\ &\equiv (\vec{p}_1, \vec{p}_2, \ldots, \vec{p}_\tau) \end{aligned} \tag{40}$$

where $\vec{p}_t = (p_t^1, p_t^2, \ldots, p_t^n)$ is the input vector over frame t, with n physical (qu)bits. The interleaver, Π, applies a permutation on $\vec{p}$ such that

$$\begin{aligned}\Pi_{\vec{p}} &= \left(\left(p_1^1, p_2^1, \ldots, p_\tau^1\right), \left(p_1^2, \ldots, p_\tau^2\right), \ldots, \left(p_1^n, \ldots p_\tau^n\right)\right) \\ &= (\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n). \end{aligned} \tag{41}$$

We call this interleaver a *transpose* interleaver.[7] This interleaver does not disperse bits as widely throughout the bitstream as a completely random interleaver, however, it does generate inner parity check stabilizers which have significantly lower rate than $\mathrm{wt}(H_{\mathrm{outer}}) \times \mathrm{wt}(G_{\mathrm{inner}})$.

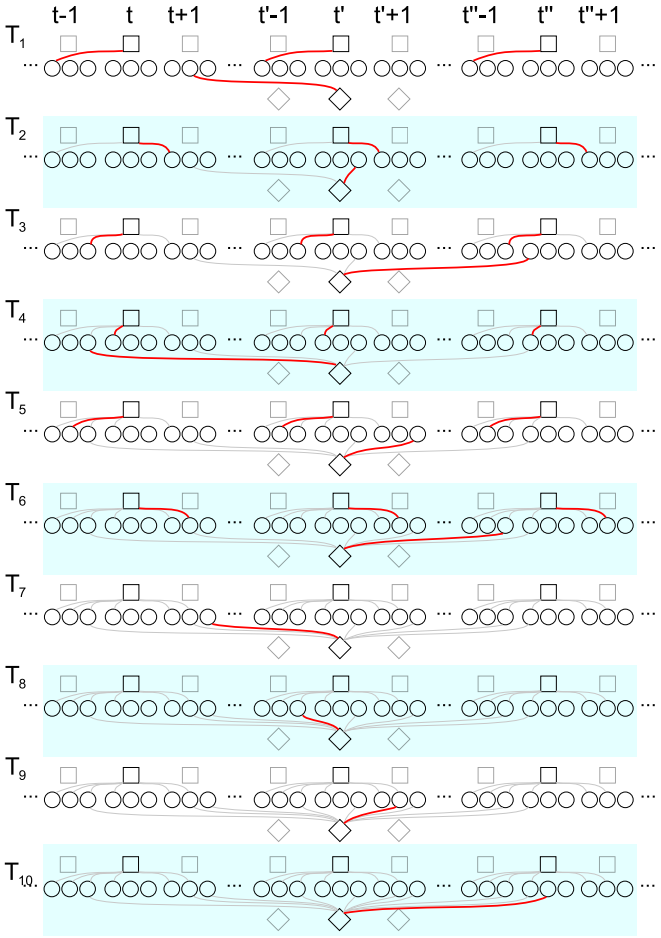

FIG. 14. A suitable time ordering for implementing C-PHASE gates between ancilla qubits and code qubits in a clusterized T9 turbo code, which is a concatenation of two C3 convolutional codes. The gates for three inner seed stabilizers (square ancillas) and one outer seed stabilizer (diamond ancilla) are shown. All other stabilizers are translations of these. Each qubit is acted on by at most one gate at each time step. Thick, red lines indicate C-PHASE to implement at the corresponding time T_j, fine grey lines indicate previously implemented gates.

Using a transpose interleaver, the cluster construction schedule is shown in Fig. 14 for the T9 code. In this example, there are two classes of parity check operators for each frame: square vertices correspond to weight 6 stabilizers, and the diamond vertices correspond to weight 10 stabilizers. As a result, this clusterized code may be implemented in ten time steps, $T_1, \ldots, T_{10}$.

Also, using a transpose interleaver, the T25 code can be implemented in 26 time steps, and has maximum weight 26 stabilizers. We present the cluster-state construction schedule and interleaver details in Appendix E.

To generate the cluster resource for a foliated code, each sheet can be generated independently using a suitable schedule for the corresponding primal and dual clusterized codes. Additional two time-steps are then required to connect the

[4] $\mathrm{wt}(G_{\mathrm{inner}}) \times \mathrm{wt}(H_{\mathrm{outer}}) = 3 \times 6$.

[5] $\mathrm{wt}(G_{\mathrm{inner}}) \times \mathrm{wt}(H_{\mathrm{outer}}) = 7 \times 14$.

[6] Note that we have not done threshold simulations for these interleavers; the numerical results presented in Fig. 10 may depend on the choice of an interleaver.

[7] If we write $\vec{p}$ as a $\tau \times n$ matrix where the $\vec{p}_t$ are row vectors, then $\Pi_{\vec{p}} = \vec{p}^{\mathsf{T}}$ is the $n \times \tau$ matrix transpose.

code qubits of neighboring sheets to build the fully foliated network.

B. Error propagation during cluster-state construction schedule

During the construction of cluster states errors may accumulate on individual qubits, or be caused by faulty gate implementation between qubits. We simplify the analysis of these errors by restricting our analysis to X and Z Pauli errors. A Z error commutes with C-PHASE gates, however, an X error does not, and will propagate a Z error to the neighboring qubit.

Consider the case where an X error occurs on an ancilla qubit, a_k, at some time, T_ε in the construction schedule. Subsequent C-PHASE gates will generate Z errors on all code qubits $c_i \in N(a_k)$ subject to gates $\Lambda_{T_k}(c_i, a_k)$ where $T_k > T_\varepsilon$. Note that $\bigotimes_{c_i \in N(a_k)} Z_{c_i}$ is a stabilizer, so the this error pattern is equivalent Z errors on qubits c_i subject to gates $\Lambda_{T_{k'}}(c_i, a_k)$, where $T_{k'} \leqslant T_\varepsilon$. As a result, the maximum number of Z errors arising during the cluster construction is equal to half the weight of the stabilizer. For this reason codes with low weight parity checks are desirable.

The choice of time ordering for gates affects the types of correlated error patterns that arise during construction. Depending on the choice of code, some time orderings may be more favorable than others, producing error patterns which are more likely to be corrected. One criterion that should be met is that any single physical error during the cluster construction should lead to a correctable (i.e., decodable) correlated error pattern after the Λ gates have been made. If the number of correlated errors is less than half the code distance, $w_{\max} < d/2$, then this condition is always met. On the other hand, if the number of errors that are propagated is larger than $d/2$, then we must verify explicitly that the resulting error pattern is correctable.

As an example, the schedule for constructing the $d = 9$ T9 code, which is shown in Fig. 14 using the transpose interleaver. This code has stabilizers of weight 10 associated to the ancillae indicated by diamonds. As a result, an X error midway through the cluster construction could result in a pattern of up to $w_{\max} = 5$ correlated Z errors on code qubits adjacent to the diamond ancillae. In this case, even though $w_{\max} > d/2$, we have checked that the decoder correctly corrects all such errors arising in the schedule in Fig. 14.

Similarly, for the $d = 25$ T25 code, a maximum stabilizer weight of 26 can be achieved using the transpose interleaver. A physical error during cluster construction could cause correlated error of weight $w_{\max} = 13$. Again, in this case, even though $w_{\max} > d/2$, we have checked that the schedule for cluster construction (listed in Appendix E) produces an error pattern, which is correctly decoded.

X. CONCLUSION

In conclusion, we have shown how to clusterize arbitrary CSS codes. We have shown how to foliate clusterized codes, generalizing Raussendorf's 3D foliation of the surface code. We have described a generic approach to decoding errors that arise on the foliated cluster using an underlying soft decoder for the CSS code as a subroutine in a BP decoder, and applied it to error correction by means of a foliated turbo code. We have also shown how decoding can be performed in the case of foliated bicycle codes. These construction may have applications where codes with finite rate are useful, such as long-range quantum repeater networks.

We have exemplified the foliated construction with several code families, namely the T9 and T25 codes, and a the LDPC Bicycle code. We believe this is the first example of a finite rate generalization of a sparse cluster-state code with pseudothreshold behavior (i.e., up to moderately large code sizes). The T25 and the Bicycle code both exhibit thresholdlike behavior, even with moderate levels of foliation.

An important direction for future work is the analysis of error models that take into account the cluster state construction schedule in Sec. IX B, the correlated error patterns that arise during construction. Another direction that will be important for repeater application is the tolerance of the foliated construction and decoding process to erasure errors. This is likely to depend on the percolation threshold in the corresponding tanner graph, as discussed in Ref. [16]. Finally, developing code deformation protocols for performing gates within the general foliated architecture is an important direction for future research.

ACKNOWLEDGMENTS

This work was funded by ARC Future Fellowship FT140100952 and the ARC Centre of Excellence for Engineered Quantum Systems CE110001013, NSERC and the Canadian Institute for Advanced Research (CIFAR). We thank Sean Barrett, Andrew Doherty, Guillaume Duclos-Cianci, Naomi Nickerson, Terry Rudolph, Clemens Mueller, and Stephen Bartlett for helpful conversations.

APPENDIX A: EXAMPLE OF INVERSE SYNDROME FORMERS

We provide an example of the construction of an inverse syndrome former (ISF) and the corresponding pure errors, using the example of the seven-qubit Steane code. asad

The Steane code is self-dual, so that the generator **G**, parity check, **H** and **ISF** matrices are the same for the X- and Z-like operators, so we will drop the Pauli labels. The parity check matrix **H** for the Steane code is given by the binary support vectors corresponding to the stabilizers defined in $\mathcal{S}_Z^{\text{Steane}}$ in Eq. (1), and $\mathbf{G}^\mathsf{T} = \{1, 1, 1, 1, 1, 1, 1\}$. The **ISF** is chosen to satisfy Eq. (6). We group **G**, **H** and one possible choice of **ISF** as submatrices in a composite, square matrix

$$\left[\begin{array}{c} \mathbf{G}^\mathsf{T} \\ \hline \mathbf{H} \\ \hline \mathbf{ISF} \end{array}\right] = \left[\begin{array}{ccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}\right]. \tag{A1}$$

It is straightforward to check that

$$\begin{bmatrix} \mathbf{G}^\mathsf{T} \\ \mathbf{H} \\ \mathbf{ISF} \end{bmatrix} . [\mathbf{G} \quad \mathbf{ISF}^\mathsf{T} \quad \mathbf{H}^\mathsf{T}] = \mathbb{I}_{7\times 7}, \tag{A2}$$

consistent with Eq. (6), i.e., $\mathbf{ISF}^\mathsf{T}$ is a pseudoinverse to $\mathbf{H}$.

A given error chain $\vec{\varepsilon}$ yields a syndrome $\vec{S} = \mathbf{H}\cdot\vec{\varepsilon} \in \mathbb{Z}_2^3$, from which we can compute a pure error $\vec{e}^{\,0} = \mathbf{ISF}^\mathsf{T}\cdot\vec{S}$. Since $\mathbf{H}\cdot\mathbf{ISF}^\mathsf{T} = \mathbb{I}$, the pure error satisfies $\mathbf{H}\cdot\vec{e}^{\,0} = \mathbf{H}\cdot\mathbf{ISF}^\mathsf{T}\cdot\vec{S} = \vec{S}$, i.e., it has the same syndrome as the actual error. Equivalently, $\vec{\varepsilon} + \vec{e}^{\,0}$ is a logical operator on the code space.

APPENDIX B: TRANSFER FUNCTION NOTATION

Transfer functions are a convenient method of expressing code families that have a regular structure but an arbitrary length. Convolutional codes are a family of codes that are often represented by transfer functions. In this Appendix, we show the relationship between the full matrix expressions for the generators, $\mathbf{G}$, and the seed generators, G.

A $k\times 1$ vector of logical bits is expressed by vector $L^\mathsf{T} = [l_1, l_2, \ldots, l_k]$. The generator matrix $\mathbf{G}$ can be represented using a finite dimensional *seed* generator with delay operations. We introduce $\underline{\underline{D}}$ and $\underline{\underline{\widetilde{D}}}$, which are $f\times fn$ and $fn\times f$ matrices defined by $\underline{\underline{D}} = [D^0\mathbb{I}_f \mid D^1\mathbb{I}_f \mid \ldots \mid D^{k-1}\mathbb{I}_f]$ and $\underline{\underline{\widetilde{D}}}^\mathsf{T} = [\widetilde{D}^0\mathbb{I}_f \mid \widetilde{D}^1\mathbb{I}_f \mid \ldots \mid \widetilde{D}^{k-1}\mathbb{I}_f]$ for a rate $\frac{1}{f}$ code. The operators D and $\widetilde{D}$ satisfy $\widetilde{D}^i \times D^j = \delta_{ij}$. Here, D is the usual delay operator and $\widetilde{D}^i$ is a mnemonic for the inverse of D^i. Then we have

$$\underline{\underline{\widetilde{D}}} \times \underline{\underline{D}} = \left[\begin{array}{c|c|c} \widetilde{D}^0 D^0\mathbb{I}_f & \ldots & \widetilde{D}^0 D^{k-1}\mathbb{I}_f \\ \widetilde{D}^1 D^0\mathbb{I}_f & \ldots & \widetilde{D}^1 D^{k-1}\mathbb{I}_f \\ \vdots & & \vdots \\ \widetilde{D}^{k-1} D^0\mathbb{I}_f & \ldots & \widetilde{D}^{k-1} D^{k-1}\mathbb{I}_f \end{array}\right] \mathbb{I}_{fk}. \tag{B1}$$

We write $\mathbf{G}L = \underline{\underline{\widetilde{D}}}\,\underline{\underline{D}}\mathbf{G}L$ from which we can derive finite size seed generator.

As an illustration consider a rate $\frac{1}{3}$ code with generator matrix

$$\mathbf{G}^\mathsf{T} = \begin{bmatrix} 111 & 100 & 110 & & & \ldots \\ & 111 & 100 & 110 & & \\ & & 111 & 100 & 110 & \end{bmatrix}. \tag{B2}$$

We can express the encoding as

$$\mathbf{G}L = \underline{\underline{\widetilde{D}}}[\mathbb{I}_f \mid \mathbb{I}_f D \mid \ldots] \times \mathbf{G} \times \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_k \end{bmatrix}, \tag{B3}$$

$$\begin{aligned} &= \underline{\underline{\widetilde{D}}} \begin{bmatrix} 1+D+D^2 & D+D^2+D^2 & D^2+D^3+D^4 & \ldots \\ 1+D^2 & D+D^3 & D^2+D^4 & \\ 1 & D & D^2 & \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_k \end{bmatrix} \\ &= \underline{\underline{\widetilde{D}}} \begin{bmatrix} (1+D+D^2)(l_1 + l_2 D + \ldots + l_k D^{k-1}) \\ (1+D^2)(l_1 + l_2 D + \ldots + l_k D^{k-1}) \\ (1)(l_1 + l_2 D + \ldots + l_k D^{k-1}) \end{bmatrix}, \\ &= \underline{\underline{\widetilde{D}}} \begin{bmatrix} 1+D+D^2 \\ 1+D^2 \\ 1 \end{bmatrix} \sum_{i=0}^{k-1} D^i l_{i+1}, \\ &= \underline{\underline{\widetilde{D}}} G \sum_{i=0}^{k-1} D^i l_{i+1}, \end{aligned} \tag{B4}$$

where $\underline{\underline{\widetilde{D}}}$ is an $fn\times f$ matrix, with $f = 3$. Here, we have $D^0 \equiv 1$. In the last line, we define the seed generator G, which is a 3×1 matrix defined in terms of delay operators.

To output the physical qubits from this seed generator, a string of logical input bits L is multiplied by $\Delta(D) = [1\ D\ D^2 \ldots]$. In our working example, we have

$$\sum_{i=0}^{k-1} D^i l_{i+1} \equiv \Delta(D) \times L. \tag{B5}$$

The code qubits can be determined by multiplying this expression by G. For the more general case of a rate $\frac{b}{f}$, code takes b logical inputs and produces f physical outputs at each frame. For an encoding operation, we have

$$\begin{aligned} c &= GL, \\ &= \underline{\underline{\widetilde{D}}}\,\underline{\underline{D}} GL, \\ &= \underline{\underline{\widetilde{D}}} G \Delta L, \end{aligned} \tag{B6}$$

where $[c] = fn \times 1$, $[\underline{\widetilde{D}}] = fn \times f$, $[\underline{\underline{D}}] = f \times fn$, $[G] = f \times b$, and $[\Delta] = b \times \overline{bn}$.

APPENDIX C: TRANSFER FUNCTION MANIPULATION

Now that we have established the implementation of transfer functions as a description of convolutional codes, consider the problem of determining the ISF. The standard approach takes a pseudoinverse of the seed generator matrix. For a rate $\frac{b}{f}$ code, the generator matrix has size $nb \times nf$, where n is the number of frames. We have the property

$$\mathbf{G}^{-1} \times \mathbf{G} = \mathbb{I}_{nb \times nb}. \tag{C1}$$

To express this in terms of transfer function notation, we have

$$\begin{aligned}\mathbb{I}_{nb\times nb} &= \underline{\underline{\widetilde{D}}}\mathbf{G}^{-1}\mathbf{G}\underline{\underline{D}},\\ &= \underline{\underline{\widetilde{D}}}\mathbf{G}^{-1}G(D)\\ &\quad\times\begin{bmatrix}1 & D & \dots & D^{k-1} & 0 & 0 & \dots & 0\\ 0 & 0 & \dots & 0 & 1 & D & \dots & D^{k-1}\\ & & & & \vdots & & & \end{bmatrix},\\ &= \Delta^{\mathsf{T}}(\widetilde{D})G^{-1}(\widetilde{D})G(D)\Delta(D).\end{aligned} \tag{C2}$$

We can use the identity

$$\widetilde{D}^j D^i = \delta^{ij} = \widetilde{D}^0 D^{i-j} \tag{C3}$$

to express functions of $\widetilde{D}$ in terms of D. Note that the order of operations must be performed so that all D follow $\widetilde{D}$ terms. From Eq. (C2), we have

$$\begin{aligned}\mathbb{I}_{nb} &= \widetilde{D}^0\Delta^{\mathsf{T}}(D^{-1})G^{-1}(D^{-1})G(D)\Delta(D),\\ &= \widetilde{D}^0\Delta D\Delta^{\mathsf{T}}(D^{-1})G^{-1}(D^{-1})G(D)\Delta(D)\Delta^{\mathsf{T}}(\widetilde{D}),\\ &= \widetilde{D}^0 G^{-1}(D^{-1})G(D),\\ &= G^{-1}(\widetilde{D})G(D).\end{aligned} \tag{C4}$$

The pseudoinverse of $G(D)$ is $G(D^{-1})$. To calculate this, we make a small alteration to $G(\widetilde{D})$ by substituting the terms D^i for $\widetilde{D}^{-1}$.

We now work through an example to demonstrate this approach. Consider the case of a rate $\frac{2}{3}$ convolutional code where we wish to find its parity check matrix. One of the generators is taken from our working example and the second generator input is $[D, D, 1]$. We have

$$[G^{\mathsf{T}}|I] = \left[\begin{array}{ccc|ccc} D & 1 & 1 & 1 & 0 & 0\\ 1+D+D^2 & 1+D^2 & 1 & 0 & 1 & 0\\ & & & 0 & 0 & 1\end{array}\right], \tag{C5}$$

which has a pseudoinverse of

$$[I|G^{-1}] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1+D & 1+D & 1+D+D^2\\ 0 & 1 & 0 & D & D & 1+D^2\\ & & & 1+D & D & D^2\end{array}\right]. \tag{C6}$$

To satisfy the condition $G^{-1}G = \mathbb{I}$, we note that the product of the first column of G^{-1} and the first row of G must be 1. The same is true for the second column and second row. The product of the third column of A and the generators must be zero. Since $HG = 0$, and the number of parity checks is $n - k = 1$, this means the third column must be equivalent to H^{T}. Expressed in transfer function form this gives us

$$H(D^{-1}) = [1 + D + D^2 \quad 1 + D^2 \quad D^2]. \tag{C7}$$

It follows that

$$\begin{aligned}H(D) &= [1 + D^{-1} + D^{-2} \quad 1 + D^{-2} \quad D^{-2}],\\ &= [1 + D + D^2 \quad 1 + D^2 \quad 1] \times D^{-2}.\end{aligned} \tag{C8}$$

We recognize that the parity check is equivalent to the first seed generator shifted by 2 frames. If we compensate for the shift, then the two are equivalent ($H_X = H_Z$) since this convolutional code is self-dual. Performing an inverse on Eq. (C6), we should reclaim the original seed generators as well as the inverse syndrome former:

$$\left[G_0^{\mathsf{T}}(D^{-1}) \mid I\right] = \left[\begin{array}{ccc|ccc} 1+D & 1+D & 1+D+D^2 & 1 & 0 & 0\\ D & D & 1+D^2 & 0 & 1 & 0\\ 1+D & D & D^2 & 0 & 0 & 1\end{array}\right], \tag{C9}$$

$$\left[I \;\middle|\; \begin{matrix}G(D)\\ \text{ISF}\end{matrix}\right] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & D & D & 1\\ 0 & 1 & 0 & 1+D+D^2 & 1+D^2 & 1\\ 0 & 0 & 1 & D & 1+D & 0\end{array}\right]. \tag{C10}$$

The inverse syndrome former ISF is given by

$$\text{ISF}(D) = [D \quad 1 + D \quad 0]. \tag{C11}$$

The product of $\text{ISF}(D)$ and $H(D)$ is exactly 1. We can express this as

$$\text{ISF}(D^i) \times H(D^j) = \delta_{ij}. \tag{C12}$$

For an arbitrary syndrome, we can use the ISF to generate an error pattern which satisfies the syndrome.

APPENDIX D: SYNDROME AND ERROR PATTERN CALCULATIONS

Here we show how to calculate the syndrome $\vec{S}$ and initial error pattern $\vec{e}^{\,0}$ in the examples given in Sec. IV. The results appearing in Eq. (15) are calculated using H, $\vec{\varepsilon}$, and the ISF. The complete list of stabilizers is given by translations of the seed stabilizer by D^j. The index j gives the jth row of the parity check matrix. We have, for all $j \in \{1, \dots, \tau\}$,

$$\begin{aligned}\mathbf{H}_j &= [1 + D + D^2 \quad 1 + D^2 \quad 1]D^j,\\ \mathbf{ISF} &= [\quad D \quad\quad D \quad\quad 0].\end{aligned}$$

For reference, we also write $\mathbf{H}$ in the less compact but more direct notation of Eq. (5), using the colors to match terms above with locations of 1's in the first row below; subsequent

rows are translations of the top row

$$\mathbf{H}=\begin{bmatrix} \dots & 000 & 111 & 100 & 110 & 000 & \dots & \\ & \dots & 000 & 111 & 100 & 110 & 000 & \dots \\ & & \dots & 000 & 111 & 100 & 110 & 000 & \dots \end{bmatrix}_{n_z\tau\times n\tau},$$

For the blue path in the example of Eq. (15), the error pattern, expressed in delay notation is

$$\vec{\varepsilon}=[D \quad 0 \quad 0]D^t, \tag{D1}$$

The jth element of the syndrome is then given by

$$\begin{aligned} S_j &= \mathbf{H}_j\cdot\vec{\varepsilon}^{\mathsf{T}}(\widetilde{D}),\\ &= D^j(1+D+D^2)\widetilde{D}^{t+1},\\ &= \delta_{j,t+1}+\delta_{j+1,t+1}+\delta_{j+2,t+1},\\ &= \delta_{j,t+1}+\delta_{j,t}+\delta_{j,t-1}, \end{aligned}$$

which corresponds to the syndrome, $\vec{S}$, listed in Eq. (15), with 1's in the syndrome at frames $t-1, t$ and $t+1$, and zero everywhere else.

In delay notation,

$$\vec{S}=\sum_j S_j D^j = D^{t+1}+D^t+D^{t-1}, \tag{D2}$$

and then using the ISF and the syndrome we calculate the initial error pattern

$$\begin{aligned} \vec{e}^{\,0} &= \mathbf{ISF}\cdot\vec{S},\\ &= [D^t+D^{t+1}+D^{t+2} \qquad D^t+D^{t+1}+D^{t+2} \qquad 0], \end{aligned}$$

which is expressed as the string of bits $\dots 110110110\dots$ in Eq. (15), with the first triplet belonging to frame t.

The example using the red path, as shown in Eq. (16), uses the same code. As such the terms for H and ISF are the same as the previous example. The error pattern used in this example is

$$\vec{\varepsilon}=[D \quad D \quad 0]D^t.$$

This generates the jth element of the syndrome

$$\begin{aligned} S_j &= D^j(1+D+D^2)\widetilde{D}^{t+1}+D^j(1+D^2)\widetilde{D}^{t+1},\\ &= D^{j+1}\widetilde{D}^{t+1},\\ &= \delta_{j+1,t+1},\\ &= \delta_{j,t}. \end{aligned}$$

In delay notation, $\vec{S}=D^t$, and then using the ISF and the syndrome we calculate the initial error pattern This agrees with the syndrome in Eq. (16).

The initial error pattern is given by

$$\begin{aligned} \vec{e}^{\,0} &= \mathbf{ISF}\cdot\vec{S},\\ &= [D^{t+1} \quad D^{t+1} \quad 0], \end{aligned}$$

which is expressed as the string of bits 110 in frame $t+1$ of in Eq. (16).

APPENDIX E: SCHEDULE FOR CONSTRUCTING THE T25 CLUSTER CODE

The C5 code is a self-dual rate $r=\frac{1}{3}$ code with stabilizers generated by

$$H=\begin{bmatrix} 1+D+D^2 & 1+D^2+D^3 & 1+D^2+D^3 \\ +D^3+D^4 & +D^5 & +D^4+D^5 \end{bmatrix}. \tag{E1}$$

The weight of this stabilizer is 14, and as such a clusterized code can be constructed in 14 time steps. Figure 15 depicts the form of the cluster state used to construct the clusterized C5 code. One possible scheduling for gate operations is recorded below using $\Lambda(a_{i,j}, c_{i',j'}, T_m)$ for gate operations, where i refers to the ith ancilla qubit within a frame and j refers to the jth frame of the code and T_m is a time index. The terms a and c denote ancilla and code qubits respectively. The C5 code is a rate $r=\frac{1}{3}$ code and as such there are 3 code qubits and 1 ancilla qubit per frame. A possible scheduling for gate operations is given by

$$\begin{aligned} &\text{Order C3} = \\ &\begin{array}{ll} \Lambda_{T_1}(a_{1,t}, c_{1,t}), & \Lambda_{T_2}(a_{1,t}, c_{3,t+4}),\\ \Lambda_{T_3}(a_{1,t}, c_{1,t+1}, & \Lambda_{T_4}(a_{1,t}, c_{3,t+3}),\\ \Lambda_{T_5}(a_{1,t}, c_{3,t+2}), & \Lambda_{T_6}(a_{1,t}, c_{1,t+3}),\\ \Lambda_{T_7}(a_{1,t}, c_{1,t+2}, & \Lambda_{T_8}(a_{1,t}, c_{2,t}),\\ \Lambda_{T_9}(a_{1,t}, c_{2,t+3}), & \Lambda_{T_{10}}(a_{1,t}, c_{3,t+5}),\\ \Lambda_{T_{11}}(a_{1,t}, c_{3,t}), & \Lambda_{T_{12}}(a_{1,t}, c_{2,t+2}),\\ \Lambda_{T_{13}}(a_{1,t}, c_{2,t+5}), & \Lambda_{T_{14}}(a_{1,t}, c_{1,t+4}). \end{array} \end{aligned} \tag{E2}$$

Encoding these stabilizers with another C5 convolutional code produces a turbo code whose outer stabilizers are weight $7\times 14=98$, where 7 is the weight of the generators, G_{C5}. The weight of these stabilizers is very high.

To reduce the effective weight of these outer stabilizers it is possible to make a choice of interleaver such that the weight of these stabilizers is greatly reduced by taking operator products with inner stabilizers (whose form is identical to the scheduling outlined for the C3 code above). One such choice of interleaver is to generate independent inner encodings for

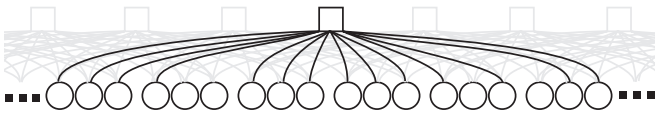

FIG. 15. The cluster state resource for the $[n, n/3, 5]$ C5 code. The ancilla (square) shares cluster bonds (lines) with the code qubits (circles). A single stabilizer of weight 14 is illustrated in black, and all other stabilizers are a translation of this seed. Construction of this cluster state requires assigning a schedule for each bond in the stabilizer. The same scheduling can be used in parallel with the other stabilizers, such that the total number of time steps required to generate the entire cluster is 14.

set of qubits produced by the outer encoder, as determined by their position within a frame. For example encode all of the qubits, which are in the first position in every frame, then encode all the qubits which are in the second position etc. This produces an inner encoding where logical qubits are more closely correlated than a random interleaver. The benefit is that an inner stabilizer can be generated with a weight of only 26, as compared to 98.

One possible scheduling for a weight 26 stabilizer produced by this choice of interleaver is given by

$$\begin{aligned}&\text{Order T25(1)} =\\ &\Lambda_{T_1}(A_{1,t}, c_{2,t}), \quad \Lambda_{T_4}(A_{1,t}, c_{2,t+5}),\\ &\Lambda_{T_7}(A_{1,t}, c_{2,t+1}), \quad \Lambda_{T_{10}}(A_{1,t}, c_{2,t+3}),\\ &\Lambda_{T_{13}}(A_{1,t}, c_{3,t+2}), \quad \Lambda_{T_{16}}(A_{1,t}, c_{3,t+5}),\\ &\Lambda_{T_{19}}(A_{1,t}, c_{3,t}), \quad \Lambda_{T_{20}}(A_{1,t}, c_{3,t+3}),\\ &\Lambda_{T_{23}}(A_{1,t}, c_{2,t+2}).\end{aligned} \tag{E3}$$

$$\begin{aligned}&\text{Order T25(2)} =\\ &\Lambda_{T_2}(A_{1,t}, c_{2,t'}), \quad \Lambda_{T_5}(A_{1,t}, c_{1,t'+5}),\\ &\Lambda_{T_8}(A_{1,t}, c_{3,t'+1}), \quad \Lambda_{T_{11}}(A_{1,t}, c_{2,t'+3}),\\ &\Lambda_{T_{14}}(A_{1,t}, c_{2,t'+5}), \quad \Lambda_{T_{17}}(A_{1,t}, c_{3,t'}),\\ &\Lambda_{T_{21}}(A_{1,t}, c_{1,t'+4}), \quad \Lambda_{T_{24}}(A_{1,t}, c_{2,t'+2}).\end{aligned} \tag{E4}$$

$$\begin{aligned}&\text{Order T25(3)} =\\ &\Lambda_{T_3}(A_{1,t}, c_{2,t''}), \quad \Lambda_{T_6}(A_{1,t}, c_{2,t''+6}),\\ &\Lambda_{T_9}(A_{1,t}, c_{3,t''+1}), \quad \Lambda_{T_{12}}(A_{1,t}, c_{3,t''+2}),\\ &\Lambda_{T_{15}}(A_{1,t}, c_{3,t''+6}), \quad \Lambda_{T_{18}}(A_{1,t}, c_{3,t''}),\\ &\Lambda_{T_{22}}(A_{1,t}, c_{2,t''+3}), \quad \Lambda_{T_{25}}(A_{1,t}, c_{1,t''+2}),\\ &\Lambda_{T_{26}}(A_{1,t}, c_{1,t''+3}).\end{aligned} \tag{E5}$$

The scheduling has been split into three components, (1), (2), and (3), referring to each of the outputs bitstreams from the outer encoder. The frame indices t, t' and t'' refer to frames which are removed from each other, such that each code qubit in the scheduling is uniquely identified by the scheme. Finally, we have substituted a with A to refer to the outer ancilla qubits.

[1] P. Shor, Phys. Rev. A **52**, R2493 (1995).
[2] A. M. Steane, Phys. Rev. Lett. **77**, 793 (1996).
[3] R. Raussendorf and H. J. Briegel, Phys. Rev. Lett. **86**, 5188 (2001).
[4] R. Raussendorf, D. E. Browne, and H. J. Briegel, Phys. Rev. A **68**, 022312 (2003).
[5] R. Raussendorf, S. Bravyi, and J. Harrington, Phys. Rev. A **71**, 062313 (2005).
[6] R. Raussendorf, J. Harrington, and K. Goyal, Ann. Phys. **321**, 2242 (2006).
[7] R. Raussendorf and J. Harrington, Phys. Rev. Lett. **98**, 190504 (2007).
[8] R. Raussendorf, J. Harrington, and K. Goyal, New J. Phys. **9**, 199 (2007).
[9] A. Kitaev, Ann. Phys. **303**, 2 (1997).
[10] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, J. Math. Phys. **43**, 4452 (2002).
[11] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Phys. Rev. A **86**, 032324 (2012).
[12] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and John M. Martinis, Nature **508**, 500 (2014).
[13] D. Ristè, S. Poletto, M. Z. Huang, A. Bruno, V. Vesterinen, O. P. Saira, and L. DiCarlo, Nat. Commun. **6**, 6983 (2015).
[14] J. M. Gambetta, J. M. Chow, and M. Steffen, npj Quantum Information **3**, 2 (2017).
[15] T. M. Stace and S. D. Barrett, Phys. Rev. A **81**, 022317 (2010).
[16] S. D. Barrett and T. M. Stace, Phys. Rev. Lett. **105**, 200502 (2010).
[17] G. Duclos-Cianci and D. Poulin, Phys. Rev. Lett. **104**, 050504 (2010).
[18] T. Rudolph, APL Photonics **2**, 030901 (2017).
[19] S. Bravyi and A. Kitaev, Phys. Rev. A **71**, 022316 (2005).
[20] A. Bolt, G. Duclos-Cianci, D. Poulin, and T. M. Stace, Phys. Rev. Lett. **117**, 070501 (2016).
[21] C. Berrou, N. Ecole, A. Glavieux, and P. Thitmajshima, in *Communications. ICC' 93 Geneva. Technical Program, Conference Record, IEEE International Conference on (vol. 2)* (IEEE, 1993).
[22] R. McEliece, D. MacKay, and J. Cheng, IEEE J. Sel. Areas Commun. **16**, 140 (1998).
[23] D. Poulin and H. Tillich, J.-P. Ollivier, IEEE Trans. Inf. Theory **55**, 2776 (2009).
[24] H. Utby, Master's thesis, University of Bergen, 2006.
[25] H. Briegel, D. Browne, W. Dur, R. Raussendorf, and V. d. Nest, Nat. Phys. **5**, 19 (2009).
[26] R. Tanner, IEEE Trans. Inform. Theory **27**, 533 (1981).
[27] H. Ollivier and J.-P. Tillich, Phys. Rev. Lett. **91**, 177902 (2003).
[28] J.-P. Tillich and G. Zemor, in Information Theory, IEEE International Symposium on (2009).
[29] D. Poulin and Y. Chung, Quantum Info. Comput. **8**, 987 (2008).

[30] D. Forney, Ph.D. thesis, Massachusetts Institute of Technology, 1965.
[31] H. R. Sadjadpour, Proc. SPIE Digital Wireless Commun. II **4045**, 73 (2000).
[32] A. J. Viterbi, IEEE Trans. Inf. Theory **13**, 260 (1967).
[33] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, IEEE Trans. Inf. Theory **20**, 284 (1974).
[34] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, TDA Progress Report. **42**, 63 (1996).
[35] D. MacKay and R. Neal, Electronic Lett. **33** (1997).
[36] D. Mackay, G. Mitchison, and P. McFadden, IEEE Trans. Inf. Theory **50**, 2315 (2004).
[37] J. Pearl, in Proceedings of the 7th Conference of the Cognitive Science Society, University of California (1985).
[38] Y. Li, S. D. Barrett, T. M. Stace, and S. C. Benjamin, Phys. Rev. Lett **105**, 250502 (2010).
[39] L.-M. Duan, M. Lukin, J. Cirac, and P. Zoller, Nature (London) **414**, 413 (2001).
[40] S. C. Benjamin, J. Eisert, and T. M. Stace, New J. Phys. **7**, 194 (2005).
[41] S. D. Barrett, P. P. Rohde, and T. M. Stace, New J. Phys. **12**, 093032 (2010).
[42] T. Satoh, K. Ishizaki, S. Nagayama, and R. Van Meter, Phys. Rev. A **93**, 032302 (2016).
[43] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and G. Nebe, IEEE J. Sel. Areas Commun. **19**, 831 (2001).
[44] S. Vafi and T. Wysocki, in Proceedings of the 6th Australian Communications Theory Workshop (2005).
[45] D. Forney, M. Grassl, and S. Guha, IEEE Trans. Inf. Theory **53**, 865 (2007).
[46] B. J. Frey and D. J. MacKay, in *Neural Information Processing Systems*, edited by M. J. Kearns, S. A. Solla, and D. A. Cohn (IEEE, New York, 1998), Vol. 11, pp. 479–485.
[47] M. Hagiwara and H. Imai, *IEEE International Symposium on Information Theory* (IEEE, New York, 2017).
[48] D. Mackay, IEEE Trans. Inf. Theory **45**, 399 (1999).