Decoding Merged Color-Surface Codes and Finding Fault-Tolerant Clifford Circuits Using Solvers for Satisfiability Modulo Theories

Noah Shutty[1,2,*] and Christopher Chamberland[1,3,†]

[1] *AWS Center for Quantum Computing, Pasadena, California 91125, USA*
[2] *Stanford Institute for Theoretical Physics, Stanford University, Stanford, California 94305, USA*
[3] *IQIM, California Institute of Technology, Pasadena, California 91125, USA*

(Received 7 February 2022; revised 14 April 2022; accepted 16 May 2022; published 28 July 2022)

Universal fault-tolerant quantum computers will require the use of efficient protocols to implement encoded operations necessary in the execution of algorithms. In this work, we show how SMT solvers can be used to automate the construction of Clifford circuits with certain fault-tolerance properties and we apply our techniques to a fault-tolerant magic-state-preparation protocol. Part of the protocol requires converting magic states encoded in the color code to magic states encoded in the surface code. Since the teleportation step involves decoding a color code merged with a surface code, we develop a decoding algorithm that is applicable to such codes.

DOI: 10.1103/PhysRevApplied.18.014072

I. INTRODUCTION

Many problems in quantum computing require the construction of Clifford circuits with some desired properties. For instance, in topological quantum error correction, multiqubit gates used to measure the stabilizers of the code must be implemented in a particular order to prevent small errors from propagating to large errors that reduce the effective distance of the code [1–4]. In many cases, fault-tolerant circuits for syndrome extraction require the use of extra ancilla qubits, known as *flag qubits*, the role of which is to detect or prevent the propagation of errors arising from a small number of faults to large data-qubit errors [3–16]. For instance, in Refs. [17,18], it has been shown that flag qubits can be used to fault-tolerantly prepare high-fidelity magic states without the use of top-down magic-state-distillation (MSD) protocols, by which we refer to protocols that use encoded gates to implement all the Clifford operations of the distillation circuits. As such, the Clifford circuits are typically not fault tolerant. We refer to protocols that prepare magic states using fault-tolerant Clifford circuits as bottom-up protocols. In many cases, fault-tolerant circuits used in bottom-up protocols are constructed from either first principles or brute-force numerical methods.

In this paper, we show in Sec. II how to formulate the desired properties (or constraints) of Clifford circuits in a format compatible with satisfiability modulo theories (SMT). These problems can then be solved using SMT solvers such as Z3 [19]. In Sec. III, we apply these techniques to construct fault-tolerant circuits for preparing $|H\rangle$-type magic states encoded in the color code [20–23], where the physical qubits are constrained to live on a two-dimensional (2D) lattice, interacting via nearest neighbors with low-degree connectivity. Such constructions have the potential to be suitable for many quantum computing hardware architectures currently under development.

Currently, the leading approach to protect logical information afflicted by noise during a quantum computation is to use a 2D topological quantum error-correcting code [24], such as the surface code [25,26] or the color code. Such codes are then combined with MSD and lattice surgery to perform universal fault-tolerant quantum computation. In particular, the surface code has several advantages over the color code [27]. For instance, the surface code has a much higher noise threshold than the color code and can achieve desired logical failure rates using fewer qubits for physical error rates expected in near-term architectures. Variations of the surface code, such as the XZZX code [28], may provide some advantages over the surface code in settings where the underlying noise model exhibits some bias. However, the surface code still provides lower overhead costs to achieve a given logical error rate for most studied realistic noise models [29]. Since the methods of Sec. III are used to prepare magic states encoded in the

*noahshutty@gmail.com
†mathematicschris@gmail.com

color code, in Sec. IV we show how magic states encoded in the color code can be converted to magic states encoded in the surface code. The schemes involve a teleportation protocol that is implemented using lattice-surgery methods. In particular, using gauge fixing to perform an $X \otimes X$ logical Pauli measurement, the color codes and surface code are merged into one code. However, known decoders for surface codes and color codes are not suitable for correcting errors of the merged code. As such, we conclude this paper by presenting a decoding algorithm that can be used to decode the merged code and is hopefully of value for successfully converting states encoded in the color code to states encoded in the surface code.

II. QUANTUM CIRCUIT DESIGN AS AN SMT DECISION PROBLEM

Quantum computers will require high-level quantum algorithms to be compiled to low-level gate implementations that are efficient, fault-tolerant, and compatible with the hardware constraints. This compilation to a physically implementable circuit is a topic of intense scientific research, with significant effort invested in reducing the gate count and depth required to implement algorithms.

Many of the core primitives in a quantum computation are, or can be viewed as, the implementation of Clifford circuits [30]. Unlike a general n-qubit unitary operation, which is specified by an exponential number of real values, an arbitrary n-qubit Clifford circuit can be specified by just $O(n^2)$ bits. In addition, these "simple" circuits can be efficiently simulated using a classical computer. Despite their mathematical simplicity, the compilation of Clifford circuits is sometimes performed by a skilled researcher, possibly aided by software that verifies that the circuit has the desired computational and fault-tolerance properties. This approach is time-consuming, unpredictable, and may not be as flexible as desired.

In this paper, we document an alternative approach to hand designing Clifford circuits in which the constraints on a quantum circuit are formulated as an *SMT decision problem*, which we define in Sec. II A 1. This problem can then be solved by an off-the-shelf SMT solver such as Z3 [19]. Despite SMT decision problems having exponential or worse time complexity for hard instances, "automated-reasoning" software libraries such as Z3 have been heavily optimized and refined through decades of research and are now widely applied in formal software verification and electronic design automation, among many other domains. They can scale to solve problems containing thousands of variables in diverse domains through careful tuning of problem encoding and solver techniques [31–33]. In Sec. II A we show how arbitrary computations from the Clifford group can be represented by bit matrices and how to solve for circuits implementing these operations using a limited gate set. We then explain how faults in the circuit can be symbolically propagated through to the end of the circuit, allowing constraints to be added to the SMT problem. In Secs. II B 4 and II B 5, we show how such constraints can be used to design circuits with guaranteed fault-tolerance properties. In Sec. II C, we describe techniques for constructing the SMT problems iteratively, which enable more scalable solutions that in turn can be used to solve more difficult circuit-design problems of practical interest.

A. Notation and definitions

1. SMT decision problems

Boolean formulas consist of expressions such as the following:

$$F = (X_1 \vee X_2) \wedge (X_2 \vee X_3) \wedge (\neg X_2 \vee \neg X_1). \tag{1}$$

These expressions involve some Boolean (i.e., binary) variables [the X_i terms in Eq. (1)] along with logical operators such as $\wedge$, $\vee$, $\oplus$, and $\neg$. A Boolean formula such as F is *satisfiable* if there is a way to assign 0 or 1 to each of the X_i such that F evaluates to 1. We call such an assignment of bits to the X_i a *satisfying assignment*. Equation (1) has the satisfying assignment $X_1 = X_3 = 0, X_2 = 1$.

SMT [31] extend the notion of a Boolean formula to an *SMT formula* such as the following:

$$F_{\text{SMT}} = (X_1 + (X_1 \oplus X_2) \wedge (X_1 + X_2 + (X_3 \neq 0) + X_4 \leq 1)) \leq 3. \tag{2}$$

These SMT formulas support variables and clauses over larger non-Boolean domains, such as the integers. They also support operators such as integer arithmetic ($+$, $-$, $\times$, $\div$) and comparison ($=$, $\neq$, $\leq$, $\geq$) along with the Boolean operators above. The *type* of an SMT expression is determined by the topmost operator in the parse tree; e.g., the formula in Eq. (2) will evaluate to a Boolean due to the comparison operator.

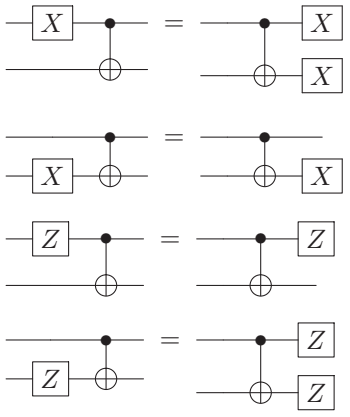
An *SMT decision problem* is an SMT formula such as F_{SMT} which is of Boolean type (i.e., evaluates to a Boolean $\in \{0, 1\}$). An *SMT solver*, such as Z3 [19], is a software program that uses heuristic strategies to find either a satisfying assignment of values to all of the variables X_i, such that $F_{\text{SMT}}(\{X_i\})$ evaluates to 1, or a formal proof that no such assignment exists.

SMT solvers exhibit good performance for a wide range of problems from program verification to network engineering [31–33]. This performance improves each year, as measured by competitions [34,35]. SMT solvers have only recently been applied to quantum circuit synthesis, gate scheduling, and qubit routing [36–39]. This work uses the bit-matrix representation of Clifford operations to efficiently encode whole-circuit-design problems subject

to fault-tolerance constraints into SMT decision problems. As we explain in the subsequent sections, this key technique enables synthesis of large fault-tolerant circuits from scratch to implement nontrivial Clifford operations while maintaining compatibility with 2D hardware.

2. Clifford group

The Clifford group on n qubits, G [modulo a global phase $U(1)$], is isomorphic to the binary symplectic group $\mathrm{Sp}(2n, \mathbb{F}_2)$, the elements of which may be considered matrices in $\mathbb{F}_2^{2n\times 2n}$, preserving the symplectic inner product. In the context of quantum computation, these matrices can be thought of as acting on a *bit-vector representation* $\mathbf{x} \in \mathbb{F}_2^{2n}$ of a Pauli-group stabilizer $\prod_{i=1}^{n} X_i^{x_i} Z_i^{x_{n+i}}$ of a quantum state, modulo the unimportant global phase. For example, the controlled-NOT (CNOT) gate acts on the basis $\{X_1, X_2, Z_1, Z_2\}$ of the vector space over $\mathbb{F}_2$ of the Pauli group (modulo phase) on two qubits, as follows:



(3)

(4)

(5)

(6)

Therefore, the binary matrix representing the CNOT gate is

$$\overline{\mathrm{CNOT}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7}$$

We call this matrix the *bit-matrix representation* of the CNOT gate and denote it by $\overline{\mathrm{CNOT}}$. Given a Pauli operator with bit vector $\mathbf{e}$ acting on the input state to a Clifford circuit C, we can propagate the operator through the circuit as $\mathbf{e}' = \overline{C}\mathbf{e}$. That is, we left multiply the bit vector by the bit-matrix representation of the Clifford operation implemented by the circuit.

It is helpful to define the *reduced bit matrices* $\overline{\mathrm{CNOT}}|_X$ and $\overline{\mathrm{CNOT}}|_Z$, which are

$$\overline{\mathrm{CNOT}}|_X = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \overline{\mathrm{CNOT}}|_Z^T.$$

These matrices characterize the action of the Clifford operation, when one Pauli type (X or Z) is disregarded.

3. Product-sum relation

Given a Clifford circuit C consisting of a series of g gates with bit matrices $G_1, \ldots, G_g$, we now describe how we can find the bit-matrix representation $\overline{C}$ of the entire circuit C. Naively, we could multiply all of the gates in time order, finding $C = \prod_{i=1}^{g} \overline{G_i}$. This requires g matrix multiplications, where there are g gates in the circuit. In our approach to designing quantum circuits with SMT solvers, the bit matrices of the gates (i.e., the G_i) are *symbolic matrices*, meaning that their entries may be SMT formulas of some abstract variables rather than Boolean constants $\in \{0, 1\}$. It turns out that when the bit matrices $G_1, \ldots, G_g$ are symbolic, the naive approach of multiplying all gates in time order results in unwieldy formulas with an extremely large *formula size*, as we explain below. So, we do not use this naive approach and we later explain the *product-sum relation*, which allows us to keep the formula size small when we design large quantum circuits.

Informally, we can define the formula size of a formula F as

$$\begin{aligned} \mathrm{size}(F) = {} & |\{\text{variable occurrences in } F\}| + \\ & |\{\text{logical connectives in } F\}|. \end{aligned} \tag{8}$$

The exact definition of formula size is not as important as the empirical fact that formulas with larger size require more memory and processing time to manipulate when constructing and solving the SMT decision problem with an SMT solver. It is therefore of fundamental importance to keep the formula size as small as possible to be able to design large quantum circuits. Note that we can multiply and add symbolic bit matrices modulo 2 using the SMT operators $\wedge, \oplus$. For example, we can multiply and add symbolic 2×2 matrices as follows:

$$\begin{aligned} & \begin{pmatrix} x_{00}^{(0)} & x_{01}^{(0)} \\ x_{10}^{(0)} & x_{11}^{(0)} \end{pmatrix} \begin{pmatrix} x_{00}^{(1)} & x_{01}^{(1)} \\ x_{10}^{(1)} & x_{11}^{(1)} \end{pmatrix} \\ & \quad = \begin{pmatrix} (x_{00}^{(0)} \wedge x_{00}^{(1)} \oplus x_{01}^{(0)} \wedge x_{10}^{(1)}) & (x_{00}^{(0)} \wedge x_{01}^{(1)} \oplus x_{01}^{(0)} \wedge x_{11}^{(1)}) \\ (x_{10}^{(0)} \wedge x_{00}^{(1)} \oplus x_{11}^{(0)} \wedge x_{10}^{(1)}) & (x_{10}^{(0)} \wedge x_{01}^{(1)} \oplus x_{11}^{(0)} \wedge x_{11}^{(1)}) \end{pmatrix} \\ & \begin{pmatrix} x_{00}^{(0)} & x_{01}^{(0)} \\ x_{10}^{(0)} & x_{11}^{(0)} \end{pmatrix} \oplus \begin{pmatrix} x_{00}^{(1)} & x_{01}^{(1)} \\ x_{10}^{(1)} & x_{11}^{(1)} \end{pmatrix} \\ & \quad = \begin{pmatrix} x_{00}^{(0)} \oplus x_{00}^{(1)} & x_{01}^{(0)} + x_{01}^{(1)} \\ x_{10}^{(0)} \oplus x_{10}^{(1)} & x_{11}^{(0)} \oplus x_{11}^{(1)} \end{pmatrix} \end{aligned} \tag{9}$$

The 2×2 symbolic matrices on the left side above have entries with formula size 1. Their product on the right has entries with size 7, while their sum has entries with size 3. As illustrated by this example, the addition of symbolic matrices results in less of a formula-size increase than multiplication.

We now introduce a technique that allows us to construct the bit matrix for C with only N matrix multiplications, where N is the number of time steps of C. The technique works by replacing many of the symbolic matrix multiplications with additions. Specifically, we add instead of multiplying certain matrices corresponding to gates that act simultaneously on disjoint sets of qubits. As we have just explained above, symbolic matrix addition incurs a smaller size increase than multiplication. This technique therefore decreases the resulting formula sizes and improves solver performance, since $N \ll g$.

Given a bit-matrix representation $\overline{G} \in \mathbb{F}_2^{2n\times 2n}$ of a Clifford gate G acting on n qubits, which acts trivially on the ℓth qubit, it can be shown that

$$\overline{G}_{ij} = \delta_{ij} \forall i,j \in (\{\ell, \ell+n\} \times [2n]) \cup ([2n] \times \{\ell, \ell+n\}), \tag{10}$$

where $[n] := \{1,\dots,n\}$. In other words, the matrix $\overline{G}$ must leave invariant all possible Pauli operators on the ℓ^{th} qubit. We now define the following notation for a bit matrix $\overline{G}$:

$$\Delta\overline{G} := \overline{G} \oplus I_{2n\times 2n}. \tag{11}$$

From the previous observation, we can see that the matrix $\Delta\overline{G}$ is supported only in the combinatorial rectangle with rows and columns indexed in the set S of qubits supporting the gate corresponding to G. Therefore the product of two gates G_1G_2 simplifies when acting on disjoint supports:

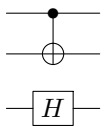
$$\begin{aligned}\overline{G}_1\overline{G}_2 &= (I \oplus (\overline{G}_1 \oplus I))(I \oplus (\overline{G}_2 \oplus I)) \\ &= I \oplus (\overline{G}_1 \oplus I) \oplus (\overline{G}_2 \oplus I) \oplus \underbrace{(\overline{G}_1 \oplus I)(\overline{G}_2 \oplus I)}_{=0} \\ &= I \oplus \Delta\overline{G}_1 \oplus \Delta\overline{G}_2. \end{aligned} \tag{12}$$

In general, for m simultaneous Clifford gates $G_1,\dots,G_m$ acting on pairwise disjoint sets of qubits, we can compute the composite bit matrix of all these gates as

$$\prod_{i=1}^{m} \overline{G}_i = I \oplus \bigoplus_{i=1}^{m} \Delta\overline{G}_i. \tag{13}$$

We refer to Eq. (13) as the *product-sum relation*.

We now give a small example of the product-sum relation for the circuit on three qubits shown in Eq. (14):



(14)

We refer to the CNOT gate on qubits 1 and 2 as $\text{CNOT}_{1,2}$ and the H (Hadamard) gate on qubit 3 as H_3. The bit matrices for these gates are as follows:

$$\overline{\text{CNOT}_{1,2}} = \begin{pmatrix} 1&0&0&0&0&0\\ 1&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&1&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix}$$

$$\overline{H_3} = \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&0&0&0&1\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&1&0&0&0 \end{pmatrix}. \tag{15}$$

These follow from the elementary propagation rules in Eqs. (3)–(6) as well as the relations $HX = ZH$ and $XH = HZ$. From these, we may easily check that

$$\begin{aligned}&(\overline{H_3} \oplus I)(\overline{\text{CNOT}_{1,2}} \oplus I) \\ &= \begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&1&0&0&1\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&1&0&0&1 \end{pmatrix} \begin{pmatrix} 0&0&0&0&0&0\\ 1&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0 \end{pmatrix} \\ &= 0_{6\times 6}, \end{aligned}$$

as required in Eq. (12). We can also verify that adding $I \oplus \Delta\overline{\text{CNOT}_{1,2}} \oplus \Delta\overline{H_3} = \overline{\text{CNOT}_{1,2}}\overline{H_3}$ as per the product-sum relation given in Eq. (13).

The product-sum relation is used in Sec. II B 2 to reduce the number of costly symbolic bit-matrix multiplications in the construction of the SMT decision problem from scaling with g (the number of gates) to scaling with N (the number of time steps). For shallow quantum circuits on many qubits, $N \ll g$, resulting in substantial savings on formula size. The product of the N symbolic matrices of dimensions $2n \times 2n$ can then be computed as a symbolic matrix the entries of which have formula size $O(Nn^3)$.

B. Solving for Clifford circuits

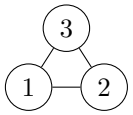
The reader may notice many familiar notions, which in this section are reformalized as SMT formulas to enable the precise characterization of the Clifford-circuit-design problem as an SMT decision problem.

1. Gate-time encoding

To encode a Clifford-circuit-design problem as an SMT decision problem (as defined in Sec. II A 1), we require a format for encoding an arbitrary circuit supported by the hardware in terms of some Boolean variables $\{X_i\}$. This encoding should be efficient and easy to implement in the

SMT-solver software. We use the *gate-time encoding* of a circuit, which we define as follows. Suppose that our quantum computer has n qubits and that it supports w distinct fundamental gate operations $\{G_1, \ldots, G_w\}$. Finally, suppose that we wish to encode a circuit with at most N time steps. Then, we encode the circuit by wN symbolic Boolean variables indexed as X_{ij}, where $i \in [w]$ and $j \in [N]$. By convention, the gate G_i is applied at time step j if and only if $X_{ij} = 1$. The Boolean values X_{ij} then specify an arbitrary-depth N circuit C_X consisting of G_i gates.

For example, consider the layout of three qubits labeled $1, 2, 3$ on the planar graph below. We can imagine that this corresponds to a physical device on a 2D surface, where the qubits have nearest-neighbor interactions shown by the graph edges. More specifically, for any pair of qubits connected by an edge, we can implement any CNOT gate on that pair. Suppose also that we can implement any single-qubit Pauli X, Y, Z, Hadamard H, or phase S gate. Then, we would have $w = 6 + 3 + 3 + 3 + 3 + 3 = 21$ distinct gates $G_1, \ldots, G_w$, and our gate set would be as follows:



$$\begin{aligned}\{G_1, \ldots, G_w\} = \{ & \mathrm{CNOT}_{1,3}, \mathrm{CNOT}_{3,1}, \\ & \mathrm{CNOT}_{1,2}, \mathrm{CNOT}_{2,1}, \\ & \mathrm{CNOT}_{2,3}, \mathrm{CNOT}_{3,2}, \\ & X_1, X_2, X_3, Z_1, Z_2, Z_3, \\ & Y_1, Y_2, Y_3, H_1, H_2, H_3, \\ & S_1, S_2, S_3\}\end{aligned}$$

The product-sum relation then gives a symbolic expression for the bit matrix of the Clifford operation $\overline{C}$ performed by the circuit C in terms of Boolean variables X_{ij} and the bit matrices for the individual gates $\overline{G_i}$. This expression is given by

$$\overline{C} = \prod_{j=1}^{N} \left(I \oplus \bigoplus_{i=1}^{w} X_{ij} \Delta \overline{G_i} \right). \tag{16}$$

At high level, our technique to construct an SMT decision problem is to construct a symbolic bit matrix P that is simply the bit matrix $\overline{C}$ of the circuit C but where the entries $\overline{C}_{ij}$ are now formulas involving the variables X_{kl} that determine the circuit C, as well as some auxiliary variables Y_i and the constants $0, 1$. The auxiliary variables are used to avoid exponentially compounding increases in formula size when multiplying the N time-step matrices and is explained in Sec. II B 2.

If we suppose that we have some Clifford operation O the bit matrix of which is $\overline{O}$, we can enforce that the circuit implements the desired Clifford operation simply by requiring that

$$\overline{C} = \overline{O}. \tag{17}$$

In practice, it is easy to compute the bit matrix $\overline{O}$ as long as the operation O is well defined—e.g., we can use a long unoptimized circuit that implements O non-fault-tolerantly to compute the bit matrix $\overline{O}$. We give more details on how $\overline{C}$ is constructed in Sec. II B 2.

Additional requirements on the circuit can be added by conjuncting this core equality with any number of additional constraints. We begin in Sec. II B 3 by describing how some simple requirements related to the validity of the gate scheduling can be added to the SMT decision problem. We explain how similar techniques are used to enforce geometric locality and enable joint codesign of hardware layout with low-level gate scheduling. In Sec. II B 4, we explain how faults can be symbolically propagated through the symbolic bit matrix and show an example application of this technique for fault-tolerant surface-code-syndrome extraction. We generalize this technique in Sec. II B 5 to design v-flag fault-tolerant circuits to implement desired Clifford operations.

2. Symbolic bit-matrix construction

For each $i \in [w]$, we find the bit-matrix representation $\overline{G_i} \in \mathbb{F}^{2n \times 2n}$ of the ith gate. We then use the product-sum relation to express the symbolic bit matrix for the circuit specified by X_{ij}, using Eq. (16).

It is helpful for Sec. II B 4 to define more generally a sequence of symbolic bit matrices for the partial circuits consisting of the last $N - k$ steps of the circuit C:

$$\overline{C}^{(k)} := \prod_{j=k+1}^{N} \left(I \oplus \bigoplus_{i=1}^{w} X_{ij} \Delta \overline{G_i} \right). \tag{18}$$

Clearly, we have that $\overline{C} = \overline{C}^{(0)}$ and, furthermore, all of the matrix multiplications over $\mathbb{F}_2$ can be implemented with any SMT solver using the fundamental operations of multiplication (i.e., logical AND) and addition mod 2 (i.e., exclusive-or XOR) on the formulas constituting the symbolic matrix, as explained in Sec. II A 3.

We next explain two optimizations that we find dramatically reduce the formula size of the symbolic bit matrices $\overline{C}^{(k)}$.

First, we explain the use of auxiliary variables in symbolic matrix multiplication. Suppose that we are given three symbolic matrices X, Y, Z the dimensions of which are all $n \times n$ and the entries of which all have formula size 1 and we must compute their symbolic product XYZ. Using the multiplication of two symbolic matrices as a primitive, we could compute their symbolic product matrix as follows (naive approach):

(1) Compute the symbolic matrix YZ, the entries of which have formula size $\Theta(n)$.
(2) Compute the symbolic matrix $X(YZ)$, the entries of which have formula size $\Theta(n^2)$.

More generally, the formula size of the entries of the product of N matrices using the naive approach is $\Theta(n^{N-1})$. Rather than computing XYZ directly, we can introduce the auxiliary symbolic matrix $W = (w_{ij})$ and proceed as follows (optimized approach):

(1) Compute the symbolic matrix YZ, the entries of which have formula size $\Theta(n)$.
(2) Add the formula equivalent to $W = YZ$ to the SMT problem formula, increasing the formula size by $\Theta(n^3)$.
(3) Compute the symbolic matrix XW, the entries of which have formula size $\Theta(n)$.

More generally, the formula size of the entries of the product of N matrices using the optimized approach remains $O(n)$ but we incur a cost of $O((N-2)n^3)$ in the problem size. However, this additional cost is negligible compared to the scaling of the naive approach above.

Second, we explain how incremental simplification is used to optimize formula size when manipulating formulas with lots of constants. SMT solvers such as Z3 [19] generally operate in two stages. In the first stage, the formula is constructed to be solved according to the user's wishes. In the second stage, the solver applies a heuristic set of approaches to simplify the formula, derive lemmas, and eventually solve the formula or prove that it is unsatisfiable. Importantly, the formula given by the user in the first stage is left "as is" during the first stage—even simple formulas such as the parity expression

$$0 \oplus 0 \oplus 0 \oplus \ldots \oplus 0 \oplus 1$$

are not reduced or simplified (e.g., in the above example, to the literal value 1) in any way. This leads to a huge slowdown in constructing the SMT problem, as manipulation of these large symbolic formulas has a much larger memory and time overhead than direct manipulation of single bits. The SMT solver's simplification routines may be manually triggered on the partial expressions that build up a formula but this brings its own associated slow-downs, since in many cases the solver spends time performing nontrivial simplifications. Moreover, we observe that a presimplified formula can take longer to solve, as it is generally better to leave decisions about nontrivial simplifications, substitution, etc. to the finely tuned heuristics of the solver at solving time rather than to manually force potentially nontrivial simplifications during problem construction.

To avoid the worst of the slow-downs when manipulating large symbolic bit matrices, we implement the variadic subroutines FastOr, FastAnd, FastXor, which pool all constant arguments and only call the symbolic SMT Boolean functions when needed. An example showing how this reduces the formula size is shown in Table I. We find that this optimization leads to a $\times 100$ speed-up in the construction of the matrices $P^{(k)}$ and is thus extremely important for scaling up to large circuits involving dozens of qubits and time steps. Here is a small example. When evaluating one term in the inner sum in Eq. (16), we must symbolically compute the product $X_{ij}\,\Delta\overline{G}_i$, where we recall that X_{ij} is a Boolean variable and $\Delta\overline{G}_i$ is a bit matrix of (literal) Boolean values. As a toy example, imagine that the matrix is equal to

$$\Delta\overline{G}_i = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{19}$$

Then, performing the multiplication using FastAnd gives the result

$$X_{i,j}\,\Delta\overline{G}_i = \begin{pmatrix} 0 & X_{ij} \\ X_{ij} & 0 \end{pmatrix}, \tag{20}$$

whereas using the logical operators of Z3 directly gives

$$X_{i,j}\,\Delta\overline{G}_i = \begin{pmatrix} \text{“}X_{ij} \wedge 0\text{”} & \text{“}X_{ij} \wedge 1\text{”} \\ \text{“}X_{ij} \wedge 1\text{”} & \text{“}X_{ij} \wedge 0\text{”} \end{pmatrix}. \tag{21}$$

For large matrices with many 0 entries, this simplification has a dramatic effect.

Another simplification we can make pertains to the use of auxiliary variables in bit-matrix construction.

TABLE I. Partial simplification by pooling constants in high-arity functions, here the FastAnd function. In the table, X and Y are two Boolean variables.

Arguments	And(arguments)	FastAnd(arguments)
$X, 0, 1, 1$	"$X \wedge 0 \wedge 1 \wedge 1$"	0
$X, Y, 1, 1$	"$X \wedge Y \wedge 1 \wedge 1$"	$X \wedge Y$
$1, 1, 1, 1$	"$1 \wedge 1 \wedge 1 \wedge 1$"	1

3. Constraints on valid circuits

Valid circuits must generally satisfy some *gate-exclusion relations*. We describe these in terms of a set of SMT assertions $\{f_i\}$ that are conjuncted into the constraint satisfaction problem as $\bigwedge_i f_i$. In a typical setup, there can be at most one gate acting on each qubit at any one time step. We can capture this as, for a time step $t \in [N]$ and a

qubit $q \in [n]$,

$$f^{(t,q)} = \left(\sum_{i\in[w],q\in\text{supp}(G_i)} X_{it} \leq 1 \right), \tag{22}$$

in which the sum is over the integers (i.e., not modulo 2). We then obtain the combined valid circuit constraints as

$$\text{IsValidCircuit}(\{X_{ij}\}) = \bigwedge_{t\in[N],q\in[n]} f^{(t,q)}.$$

In some cases, we may modify the gate-exclusion relations (22). Here are several examples:

(a) We may choose to represent particular gates by products of those in our gate set $G_1, \ldots, G_w$. For example, we may represent the Pauli Y gate on a single qubit as acting with both the X and Z gates simultaneously. This would decrease the number of distinct gates w, making the circuit representation more efficient.

(b) We may limit the number of gates of a particular type that are applied, because, e.g., the noise rate or overhead cost may be especially high for that gate type.

(c) We may enforce that few or no idling locations occur, as idling qubits add additional fault locations to the circuit.

(d) We may limit the number of distinct long-range gates between distant qubits, in case we can only manage to implement a few such gates and wish to use them sparingly.

(e) We may limit the number of distinct qubits with which any one qubit $q \in [n]$ interacts, so as to minimize the degree of the connectivity graph. That is, we may enforce a condition such as

$$\left(\sum_{i\in[w],q\in\text{supp}(G_i)} \underbrace{\bigvee_{t\in[N]} X_{it}}_{\text{gate } i \text{ is used at least once}} \right) \leq d, \tag{23}$$

in which d is the desired maximum degree.

(f) We may bind the gates used in one circuit-design problem with the gates used in a different problem—that is, we may codesign multiple protocols simultaneously, with a global degree or other gate constraint enforced jointly across all the protocols. For example, suppose that we are designing two protocols labeled 1 and 2, which share a common set of qubits $[n]$ and possible gates indexed by $1, \ldots, w$. Suppose that we wish for the protocols to be implementable on the same hardware, which is subject to a maximum degree connectivity of d. Then, we can enforce a condition such as

$$\left(\sum_{i\in[w],q\in\text{supp}(G_i)} \bigvee_{t\in[N]} X_{it}^{(1)} \vee \bigvee_{t\in[N]} X_{it}^{(2)} \right) \leq d, \tag{24}$$

in which the $\{X_{ij}^{(k)}\}_{i,j}$ are the gate-time encoding variables for protocol k.

In all of these cases, we can construct the appropriate gate-exclusion relations f_i, using standard techniques to encode them as SMT decision formulas.

4. Topological fault tolerance using symbolic fault propagation

We now introduce *symbolic fault propagation* and show how it can be applied to design fault-tolerant syndrome-extraction circuits for topological codes such as the surface code. Since arbitrary noise operators can be expressed as linear combinations of Pauli operators, we can model the noise as a distribution on Pauli operators at each space-time location (i.e., idle, gate, state preparation and measurement) in a given circuit. Specifically, we describe a Pauli noise operator (ignoring global phases) as a column vector $\mathbf{e} \in \mathbb{F}_2^{2n}$ in which the first n rows correspond to X errors on the n qubits and the second n rows correspond to Z errors.

Suppose that a noise process occurs at time step k, resulting in some initial Pauli-noise-operator vector $\mathbf{e}$. We *symbolically propagate* the error arising from the fault through the remainder of the circuit by computing the symbolic vector $\mathbf{e}' = \overline{C^{(k)}}\mathbf{e}$ [where the partial-circuit bit matrices are defined in Eq. (18)]. We then impose constraints as needed on the final propagated error to ensure that the scheme is fault tolerant up to the full code distance. The combined fault-propagation constraint is then

$$\bigwedge_{\text{single fault } \mathbf{e}} \neg\text{NotFaultTolerant}(\mathbf{e}'),$$

in which NotFaultTolerant is a Boolean formula NotFaultTolerant$(e'_1, \ldots, e'_{2n})$ on $2n$ variables, which decides whether the propagated error at the single fault location invalidates the desired fault-tolerance properties of the circuits. For example, the gate scheduling chosen for the syndrome-extraction circuit of a topological code must satisfy the property that errors propagate perpendicularly to the appropriate logical operator. An example is provided in Fig. 1.

In a noise model where two-qubit gates are afflicted by two-qubit Pauli errors, we may wish to enforce these constraints only for faults occurring on those two-qubit gates the associated Boolean variables X_{ij} of which are set to 1.

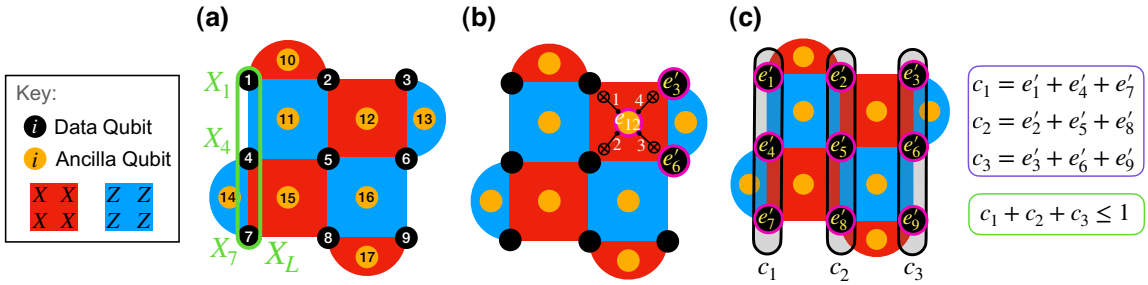


FIG. 1. An illustration of how fault-propagation constraints can be used to find a fault-tolerant gate scheduling for the surface code such that at most $(d-1)/2$ faults does not result in a logical error. (a) A 3×3 surface code with a vertical logical X_L operator. (b) For a particular syndrome-extraction circuit with the CNOT gates and time steps shown for one of the X-type stabilizers, a single fault resulting in an X error $\mathbf{e}$ on qubit 12 after the second time step propagates to a weight-2 X error on the data qubits 3 and 6 that is parallel to the logical X_L operator. (c) Symbolically propagating the error arising from a fault at that space-time location as $\mathbf{e}' = \overline{C}^{(2)}\mathbf{e}$, we can impose the fault-propagation constraints (right) to ensure that the fault does not propagate parallel to a logical operator. By imposing this constraint for each possible fault location for X-type errors, we ensure that no single faults in the syndrome-extraction circuit propagate parallel to a X_L logical operator. Similar constraints can be written to prevent Z-type errors from propagating parallel to a Z_L logical operator.

In this case, we can amend the combined constraints as

$$\bigwedge_{\substack{\text{single fault}\mathbf{e} \\ \text{after time step } k}} \left(\neg\text{NotFaultTolerant}(\mathbf{e}')\right) \vee \text{IsInvalidFault}(\text{X}, \mathbf{e}, \text{k}), \tag{25}$$

in which $\text{IsInvalidFault}(X, \mathbf{e}, k)$ is a Boolean formula that decides whether the error $\mathbf{e}$ could have occurred at that space-time location.

5. Flag fault tolerance

Another interesting application of symbolic fault propagation is in the construction of v-flag circuits used to fault-tolerantly measure a given operator (e.g., a stabilizer of an error-correcting code). Following the definition introduced in Ref. [10], a v-flag circuit for measuring a stabilizer g_i has the property where, for any set of $t \leq v$ faults resulting in the error E such that $\min(\text{wt}(E), \text{wt}(Eg_i)) > t$, at least one of the flag qubits are measured nontrivially. Here, $\text{wt}(P)$ corresponds to the weight of an operator P. More details can be found in Ref. [10]. In what follows, we say that a circuit is *flagged* if at least one of the flag qubits is measured nontrivially. For each qubit i that is to be measured, we let $P_i \in \{X, Y, Z\}$ be the Pauli measurement basis. As the P_i-basis measurement of qubit i must deterministically give a trivial $+1$ measurement outcome in the absence of any faults, the initial stabilizers of the input state must be mapped by the desired Clifford operation C to a set of stabilizer generators, which generate a stabilizer group that includes the stabilizer P_i on the final state.

To symbolically verify whether a propagated error resulting from t faults causes the circuit to flag, it suffices to obtain the symbolic propagated error and to verify whether any flag qubit i gives a nontrivial P_i-basis measurement outcome. For instance, if qubit i is measured in the $P_i = X$ basis, we verify that there is a Z or Y error on that qubit in the symbolic propagated error $\mathbf{e}'$.

We may allow the SMT solver to decide which qubits are to play the role of flag qubits. In this case, to each possible flag qubit i we add a Boolean variable IsFlag_i, where IsFlag_i is set to 1 if and only if qubit i is a flag qubit in the final protocol. Similarly, we may allow the SMT solver to choose the measurement basis P_i for each measured qubit i by adding Boolean variables MeasuredInX_i and MeasuredInZ_i, with the convention that the Y basis is chosen if both are set to 1 and with an added constraint that $\text{MeasuredInX}_i \vee \text{MeasuredInZ}_i = 1$. For convenience, we may still refer to this encoded Pauli variable as P_i. As just mentioned, the P_i-basis Pauli measurement outcome must deterministically give $+1$ when there are no faults, so the desired Clifford operation C must be compatible with the choice of P_i and IsFlag_i. The solution that we use is to make C itself depend on the setting of these variables, so that the circuit does not flag when there are no faults.

For each possible fault in the circuit, we associate a tuple $(k, \mathbf{e}, \mathbf{e}', S)$ where $k \in [N]$ is the time step such that the fault occurs, $\mathbf{e} \in \mathbb{F}_2^{2n}$ is the vector representation of the Pauli noise operator resulting from the fault, $\mathbf{e}' = \overline{C^{(k)}}\mathbf{e}$, and S is the stabilizer in the measurement circuit of which the fault occurs.

We define several functions returning SMT formulas, as follows. The MinWt function returns a formula that evaluates to the minimum integer weight of the propagated error

resulting from a given set of t faults when multiplied by the ℓ distinct stabilizers $\{Q_1, \ldots, Q_\ell\} = \{S_i : i \in [t]\}$ in the measurement circuits of which the t faults occur [40]:

$$\begin{aligned}&\text{MinWt}\left(\left\{\left(k_i, \mathbf{e}_i, \mathbf{e}_i', \overline{S}_i\right) : i \in [t]\right\}\right)\\ &\quad = \min_{x \in \{0,1\}^\ell} \text{wt}\left[\left(\prod_{j=1}^{\ell} \overline{S}_j^{x_j}\right) \sum_{i=1}^{t} \mathbf{e}_i'\right].\end{aligned} \tag{26}$$

Note that the integer min function can be implemented as an SMT formula using the comparison operators and the If-Then-Else operator, which are both supported [19]. The NontrivialOutcome$(i, P_i, \mathbf{e}')$ function returns a formula that evaluates to 1 if the P_i-basis measurement outcome of qubit i gives a nontrivial -1 outcome in the presence of the error $\mathbf{e}'$. As the operation C is guaranteed to give a $+1$ measurement outcome when there are no faults, this is easily computed as

$$\begin{aligned}&\text{NontrivialOutcome}(i, \text{MeasuredInX}_i, \text{MeasuredInZ}_i, \mathbf{e}') =\\ &(e_i' \wedge \text{MeasuredInZ}_i) \oplus (e_{n+i}' \wedge \text{MeasuredInX}_i).\end{aligned} \tag{27}$$

The IsFlagged function returns a formula that evaluates to a Boolean 1 value if and only if there is a flag qubit that gives a nontrivial measurement outcome

$$\begin{aligned}&\text{IsFlagged}\left(\mathbf{e}', \left\{\text{IsFlag}_1, \ldots, \text{IsFlag}_n\right\}\right)\\ &\quad = \bigvee_{i=1}^{n} \text{IsFlag}_i \wedge \text{NontrivialOutcome}(i, P_i, \mathbf{e}'),\end{aligned} \tag{28}$$

where for brevity we abbreviate the variables Measured InX_i, MeasuredInZ_i as simply P_i.

The IsValidFaultSet function returns a formula that evaluates to 1 if and only if the faults occur at a valid origin point:

$$\begin{aligned}&\text{IsValidFaultSet}\left(\left\{\left(k_i, \mathbf{e}_i, \mathbf{e}_i', \overline{S}_i\right) : i \in [t]\right\}\right)\\ &\quad = \bigwedge_{i=1}^{t} \bigvee_{\substack{j=1\\ \text{supp}(G_j) \supseteq \text{supp}\mathbf{e}_i}}^{w} X_{i,j},\end{aligned} \tag{29}$$

where the support of a gate $\text{supp}(G)$ is the set of qubits on which it acts and the support of a noise operator with bit vector $\mathbf{e} \in \mathbb{F}_2^{2n}$ is simply $\text{supp}(\mathbf{e}) = \{i \in [n] : e_i \vee e_{n+i}\}$. Note that $\mathbf{e}$ is not a free variable, as it is known at the time of SMT-decision-problem creation. Therefore, the IsValidFaultSet function returns a small SMT formula, since the logical OR in Eq. (29) is efficiently implemented by the program that constructs the SMT formula, rather than symbolically encoded in the formula itself.

The IsNotTFlagFaultTolerant function returns a formula that evaluates to 1 if and only if the passed error violates the t-flag property of the circuit. That is,

$$\begin{aligned}&\text{IsNotTFlagFaultTolerant}\left(\left\{\left(k_i, \mathbf{e}_i, \mathbf{e}_i', \overline{S}_i\right) : i \in [t]\right\}\right)\\ &\quad = (\text{MinWt} > t) \wedge (\neg\text{IsFlagged}) \wedge \text{IsValidFaultSet},\end{aligned} \tag{30}$$

where we suppress all arguments except IsNotTFlagFault Tolerant for brevity.

To design a v-flag circuit, we construct an SMT problem with fault-tolerance constraints IsVFlag, which returns a formula evaluating to 1 if and only if the circuit is v-flag:

$$\begin{aligned}&\text{IsVFlag}(\{X_{ij}\}, \{P_i\}, \{\text{IsFlag}_i\})\\ &\quad = \bigwedge_{t \le v} \bigwedge_{\{(k_i, \mathbf{e}_i, \mathbf{e}_i', \overline{S}_i) : i \in [t]\}} \neg\text{IsNotTFlagFaultTolerant},\end{aligned} \tag{31}$$

where we again omit the arguments for clarity and it is understood that the conjunction is over all possible sets of $t \le v$ errors that occur at t distinct fault locations.

For Calderbank-Shor-Steane (CSS) code-syndrome measurement circuits built from just CNOT gates, we may only concern ourselves with the propagating error type (e.g., the X-type errors when measuring the X stabilizers). For this purpose, we can consider just the restricted bit matrices $P^{(k)}|_X$, saving a factor of 4 on the size of the symbolic bit matrices $\overline{C}^{(k)}$.

Figure 2 shows a simple example to illustrate the construction of the SMT formula to design a circuit to measure a two-qubit stabilizer with no flag qubits.

C. Iterative solving

A circuit with depth N on n qubits, with w possible gates at each time step, has at most $N(w+q)$ distinct fault locations. As explained in Sec. II B 5, we can construct a SMT formula [Eq. (31)] that evaluates to 1 if and only if the found circuit is v flag. However, this formula will have a size (the number of clauses in the logical AND) that scales with the number of fault locations in the circuit. Specifically, there are up to $\sum_{t=1}^{v} \binom{N(w+q)}{t}$ distinct fault combinations that give constraints in Eq. (31). This large number of possible fault combinations results in a large SMT problem that is difficult to construct and likely not possible to solve directly for circuits with thousands of fault locations.

To circumvent this problem, we make use of an iterative approach as shown in Fig. 3. We first construct the SMT-problem instance F_{SMT} as simply

$$F_{\text{SMT}} = \left(\overline{O} = \overline{C}\right), \tag{32}$$

that is, without any fault-tolerance constraints. We then check the v-flag property for all sets of $t \le v$ errors occurring at t distinct fault locations. These sets of errors

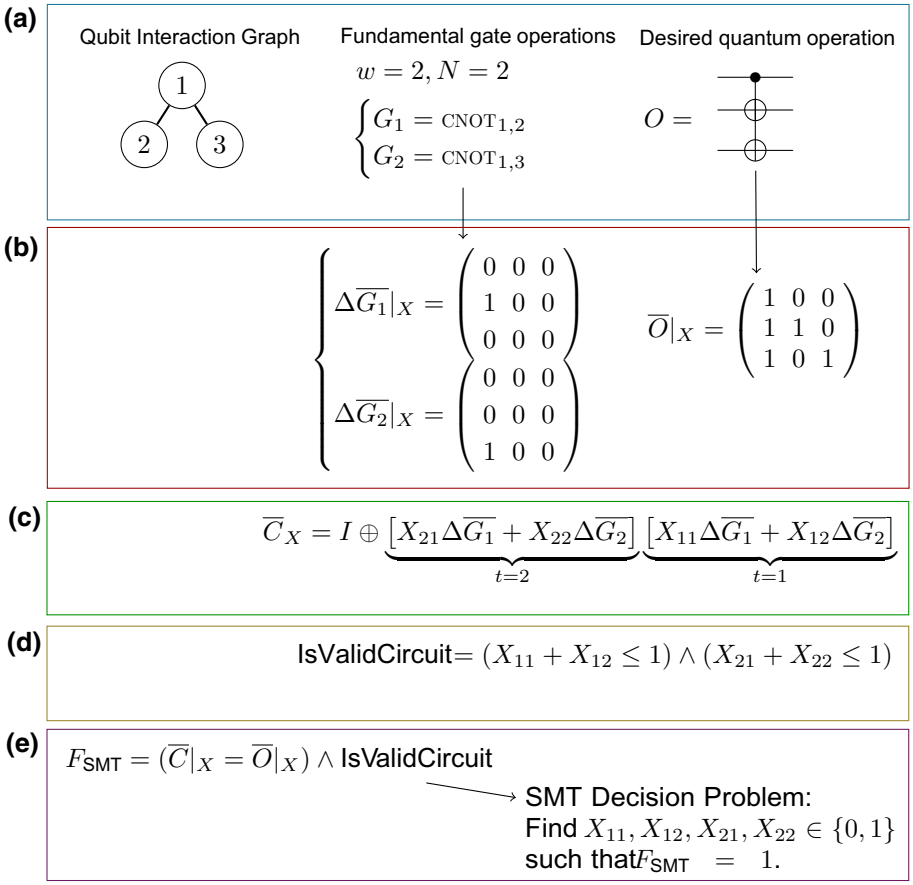


FIG. 2. An example of our framework for encoding a quantum circuit design as an SMT decision problem, applied to a simple set of circuit-design objectives. (a) The design objectives. We start with some hardware layout on three qubits with which we can implement two fundamental quantum gates $\{G_1, G_2\}$. We also have a Clifford circuit describing a quantum operation O. We wish to implement O in $N = 2$ time steps. (b) The reduced-bit-matrix encoding. We first encode the available gates G_1, G_2 and the operation O as bit matrices. Since we only have CNOT gates, both off-diagonal blocks of all involved bit matrices are automatically 0, so it is sufficient to consider just the reduced bit matrices $\overline{G_1}, \overline{G_2}, \overline{O}|_X$. (c) The symbolic bit matrix of the circuit. We use four Boolean variables $X_{11}, X_{12}, X_{21}, X_{22}$ to encode the circuit, where $X_{ti} = 1$ only if gate i is applied at time step t. We evaluate the symbolic reduced bit matrix $\overline{C}|_X$. (d) The circuit validity constraint. We prepare the expression IsValidCircuit, which evaluates to 1 only if each qubit is acted on by at most one gate at each time step. (e) The problem construction. We then write the SMT formula F_{SMT} so that it evaluates to 1 only if the $\{X_{ti}\}$ variables encode a physically implementable circuit that implements the Clifford operation O. We use an off-the-shelf SMT solver such as Z3 [19] to find a solution or a proof that no solutions exist.

correspond precisely to the clauses in Eq. (31). If any set $\{(k_i, \mathbf{e}_i, \mathbf{e}'_i, \overline{S}_i) : i \in [t]\}$ is found that violates the v-flag constraint, that is, such that

$$\text{IsNotTFlagFaultTolerant}\left(\{(k_i, \mathbf{e}_i, \mathbf{e}'_i, \overline{S}_i) : i \in [t]\}\right) = 1,$$

then we let

$$\begin{aligned}&\text{AdditionalConstraint}\\ &\quad = \neg\text{IsNotTFlagFaultTolerant}\left(\{(k_i, \mathbf{e}_i, \mathbf{e}'_i, \overline{S}_i) : i \in [t]\}\right),\end{aligned} \tag{33}$$

and we then update the formula as

$$F_{\text{SMT}} \leftarrow F_{\text{SMT}} \wedge \text{AdditionalConstraint}.$$

We then ask the SMT solver to re-solve F_{SMT}. We repeat this process until either the problem is shown undecidable or there are no fault combinations present in the circuit that violate the v-flag property, as shown in Fig. 3.

We expect two reasons why iterative solving works better than specifying all the constraints at the beginning of the protocol. The first is that many of the constraints are redundant, in a formal sense or a statistical sense. For example, two faults that occur late in the circuit at faraway positions (as measured by the qubit connectivity-graph distance) cannot possibly both flag the same flag qubit—therefore, as long as each of these faults flag *independently*, then the combination of both faults will also flag. Therefore, the constraint ¬IsNotTFlagFaultTolerant for the fault set containing these two faults is redundant in

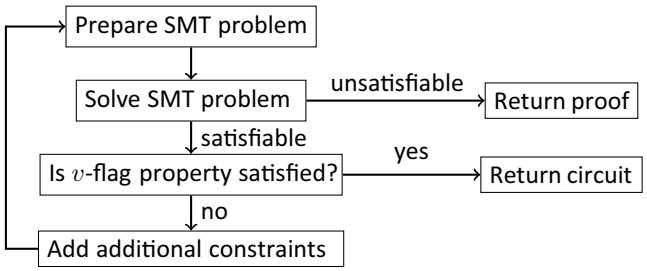


FIG. 3. In iterative v-flag circuit solving, an initial SMT formula is constructed without flag constraints. After a solution is found, the conditions in Eq. (31) are checked. If any violated constraints are found, then some are added to the SMT formula and the problem is re-solved. The process terminates if either a v-flag circuit is found or a problem is proven to be unsatisfiable.

a formal sense with the constraints for each of the faults on their own. Other sets of t faults may have intersecting *light cones* such that the constraint for the fault set is not formally redundant and yet the majority of solutions to the problem do not violate the t-flag constraint ($\neg$IsNotTFlagFaultTolerant) of that fault set.

Interestingly, our numerical simulations show that iterative solving cuts run times to assemble the SMT problem instances dramatically and enables scaling the approach beyond 1-flag to v-flag (with $v \geq 2$), which is simply not possible with the naive approach of constructing the entire SMT problem up front.

III. FAULT-TOLERANT $|H\rangle$-TYPE MAGIC STATE PREPARATION USING SMT SOLVERS

The leading approach to implementing quantum algorithms on a universal fault-tolerant quantum computer is to use MSD [41] in combination with lattice-surgery techniques. Alternative approaches to MSD for achieving universality, such as code switching [21,42–44], have been proposed. Such alternative approaches are not always compatible with the 2D hardware constraints and have been shown to require larger resource overhead costs in their implementation [29,45,46].

T-type magic states [with $|T\rangle = (|0\rangle + e^{i\pi/4}|1\rangle)/\sqrt{2}$] can be used as a resource state to fault-tolerantly implement logical T gates. In Refs. [27,47], $|T\rangle$ states have been prepared by first encoding several $|T\rangle$ states in distance $d = 1$ surface codes using non-fault-tolerant methods and growing the codes to a final distance $d' \gg 1$. Afterward, such states encoded in a distance d' surface code have been injected into a MSD protocol to distill them to a desired target logical failure rate determined by the size of the quantum algorithm being implemented. Since the growing scheme is not fault tolerant, a single fault (in the input $|T\rangle$ state) can result a logical error in the injected magic states prior to the implementation of the MSD protocol. Consequently, many rounds of MSD are required, where each operation is encoded in a large-distance surface code, to generate high-fidelity magic states such that they can be used in an algorithm.

An alternative proposal has been put forth in Refs. [17,18]. In this approach, an encoded $|H\rangle$-type magic state (with $|T\rangle = e^{i\pi/8}HS^\dagger|H\rangle$, where H and S are Hadamard and phase gates) with code distance $d > 1$ is directly prepared using a fault-tolerant protocol, meaning that any errors arising from at most $(d-1)/2$ faults cannot lead to a logical error. The protocols in Refs. [17,18] make use of the color code, which has the convenient property that the logical $\bar{H}$ gate is transversal (along with all other Clifford operations). In this protocol, a physical ($d = 1$) $|H\rangle$ state is grown using non-fault-tolerant methods to a $d > 1$ encoded $|\bar{H}\rangle$ state. Subsequently, the grown state is injected into a bottom-up fault-tolerant magic-state-preparation protocol, where $(d-1)/2$ rounds of transversal logical Hadamard measurements and color-code-syndrome measurements are performed. If any of the syndrome or flag-qubit measurements during the state-preparation protocol are nontrivial, indicating the presence of at least one fault, the protocol is aborted and begins anew. An illustration for the sequence of such operations is shown in Fig. 4(a). In Refs. [17,18], the intermediate code distances $d \in \{3, 5, 7\}$ have been analyzed, and despite additional space-time resource overhead costs due to rejection events, the scheme has been shown to reduce overhead costs compared to previous MSD schemes in order to achieve a desired logical failure rate. The main reason for the overhead reduction is the fault-tolerant nature of the preparation circuits, which can be implemented using physical Clifford operations. If the prepared magic states still require higher fidelities before being used in a quantum algorithm, such states can be injected into a MSD protocol requiring only a small number of distillation rounds.

Although bottom-up protocols have shown promising results in providing low-cost methods for preparing high-fidelity magic states, one of the main challenges stems from finding flag-based circuits with the desired fault-tolerance properties. For the case of preparing $|H\rangle$-type magic states encoded in a distance-d color code, we require v-flag circuits for measuring the logical Hadamard H_L and stabilizers of the color code (the circuits $H_m^{(d)}$ and $ED^{(d)}$) in Fig. 4(a). Further, for many hardware architectures, the qubits in such flag-based circuits are constrained to be laid out on a 2D plane where the qubits can only interact with nearest neighbors. Further, the degree of the interactions must remain low [13].

In Sec. II, we show how SMT solvers can be used to find Clifford circuits with a set of desired properties. One application of the techniques shown in Sec. II B 5 is in finding v-flag circuits for fault-tolerantly measuring stabilizers of an error-correcting code. In this section, we show how the protocols introduced in Secs. II B 5 and II C can be used

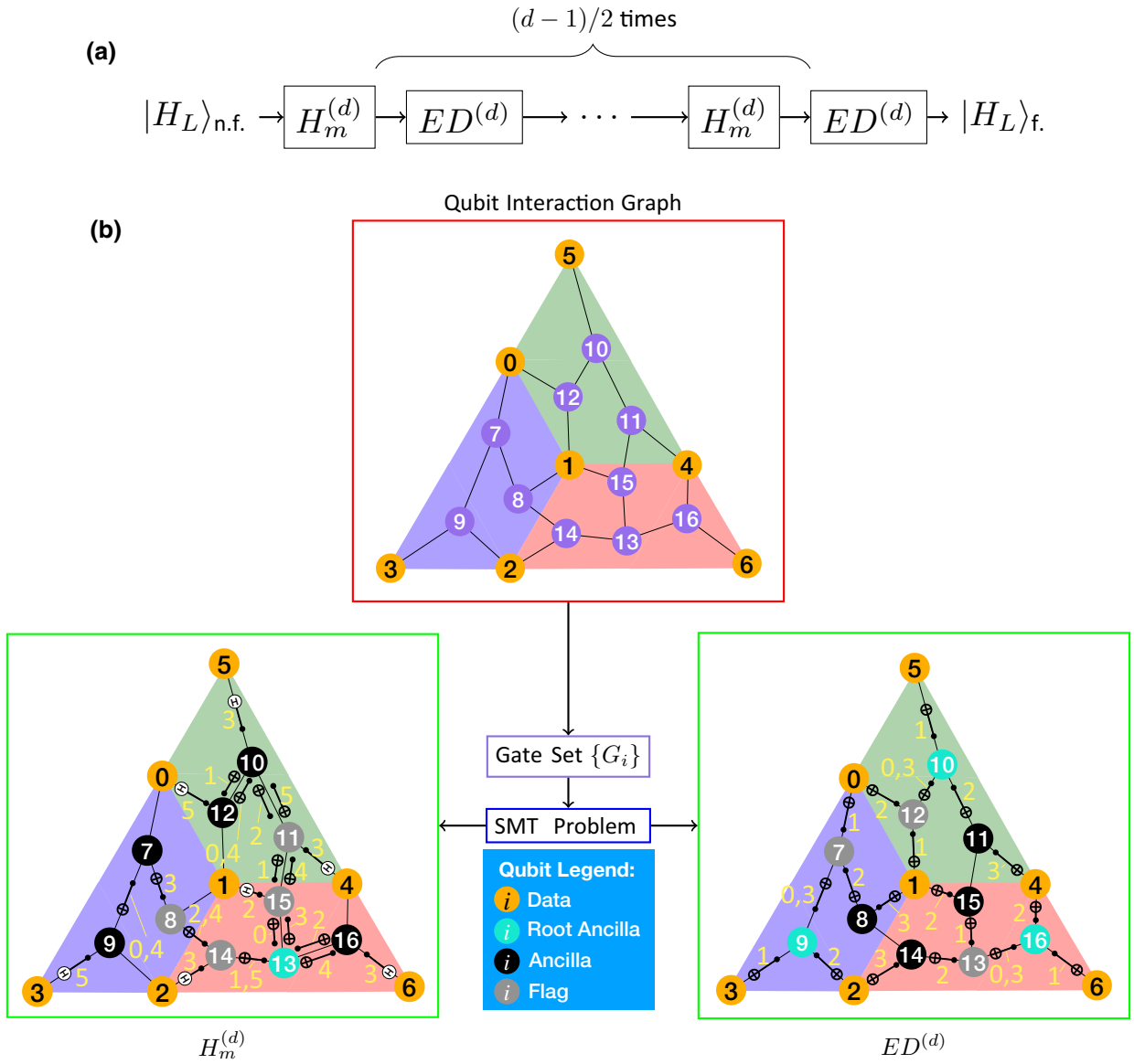


FIG. 4. Codesigning fault-tolerant 1-flag protocols. (a) The circuits $H_m^{(d)}$ are circuits for measuring the logical Hadamard $H_L = H^{\otimes n}$ of a distance-d color code. The circuits $ED^{(d)}$ correspond to one round of stabilizer measurements for a distance-d color code. (b) Given a graph of qubit interactions and an abstract bit-matrix description of several desired quantum computations, the solver produces a protocol for each desired computation that is compatible with the interaction graph, satisfies joint degree and other gate constraints (Sec. II B 3), and is 1-flag (Sec. II B 5).

to construct v-flag $H_m^{(d)}$ and $ED^{(d)}$ circuits with nearest-neighbor and low-degree connectivity constraints imposed by many quantum hardware architectures [13,29].

A. Qubit interaction graph

Recall that we label the n qubits by the integers 1 through n, and denote this set $[n] = \{1, \ldots, n\}$. We now designate a subset $A \subset [n]$ of qubits that are to be prepared in some Pauli eigenstates and measured at the end of the circuit. In our case, for designing $ED^{(d)}$ and $H_m^{(d)}$ circuits, A contains all flag qubits, ancilla qubits, and root-ancilla qubits. The root-ancilla qubit is distinguished from the other ancillas since, for $H_m^{(d)}$ and X-type stabilizer circuits in $ED^{(d)}$, it is initialized in a $+1$ X eigenstate, whereas the other ancillas are initialized in $+1$ Z eigenstates. We refer to these flag, ancilla, and root-ancilla qubits as the A qubits. The data qubits are then $[n] \setminus A$. Since our target hardware is a 2D device where only nearest-neighbor qubits can interact, we create a planar graph $G_{\text{qubits}} = ([n], E)$ in which the vertices correspond to qubits and the edges are between qubits that support two-qubit gates. This graph is called the *qubit interaction graph*. An example of such a qubit interaction graph is shown in Fig. 4(b). In our graph,

we ensure that no two data qubits share an edge, as two-qubit gates are prohibited between data qubits. This graph is designed by hand to use a low degree of connectivity between all qubits and still be connected, such that in the absence of faults it should be possible to implement the desired circuit. We then assemble our set of gates. Because we know that the Clifford operations $ED^{(d)}$ and $H_m^{(d)}$ [48] can be implemented using only preparation and measurement in the Pauli X and Z bases and CNOT gates, our gate set $\{G_1, \ldots, G_w\}$ consists only of CNOT gates. Specifically, for each edge $(u, v) \in E$ where $u, v \in A$, we add $\text{CNOT}_{u,v}$ and $\text{CNOT}_{v,u}$ to our gate set $\{G_i\}$. For edges $(u, v) \in E$ where one of u, v is a data qubit ($u \notin A$ or $v \notin A$), we only add a CNOT from the nondata qubit $\in A$ to the data qubit.

Note that controlled-Hadamard gates are equivalent to CNOT gates up to conjugation by a single-qubit non-Clifford gate. These single-qubit corrections do not propagate errors and can be placed at the beginning and end of the $H_m^{(d)}$ circuit. As such, when deriving fault-tolerant $H_m^{(d)}$ circuits, it suffices to treat all controlled-Hadamard gates as CNOT gates. Lastly, note that if an error-correction scheme was being developed rather than an error-detection scheme, the type of data qubit errors would matter and, as such, the type of two-qubit gate used for measuring the logical Hadamard operator would need to be considered. We leave such considerations to future work on error-correction schemes.

B. Constructing SMT-formula constraints for $ED^{(d)}$ and $H_m^{(d)}$ circuit synthesis

Once the gate set $\{G_i\}$ is specified, we feed it as input to the functions described in Sec. II B. These functions construct SMT formulas that are then used to construct the entire SMT decision problem given below in Eq. (34). The A qubits that are shown in purple in Fig. 4 must each be assigned one of three roles: ancilla qubit, root-ancilla qubit, or flag qubit. The parity of the measurement outcomes of the root ancilla as well as all other ancillas that are not flag qubits is used to obtain the measurement outcome for the operator being measured. Specifically, the measured operator, which is either a code stabilizer for $ED^{(d)}$ or the logical Hadamard for $H_m^{(d)}$, has 0 or more ancilla qubits and exactly one root-ancilla qubit associated with it. The measurement outcome for this measured operator is then encoded as the product of the measurement outcomes across all these ancillas (including the root) that are measured and that interact with the root ancilla. We allow the solver to choose which qubits have the role of flag, ancilla, or root-ancilla qubit by creating Boolean variables $\{\text{IsFlag}_i : i \in [A]\}$ and $\{\text{IsRoot}_i : i \in [A]\}$. We choose the following encoding: for a qubit i, if i is a root-ancilla qubit, then we have $\text{IsRoot}_i = 1, \text{IsFlag}_i = 0$. If i is a nonroot-ancilla qubit, then we have $\text{IsRoot}_i = 0, \text{IsFlag}_i = 0$. If i is a flag qubit, then we have $\text{IsRoot}_i = 0, \text{IsFlag}_i = 1$. We create a formula IsValidRoleAssignment, which evaluates to 1 if and only if there is exactly one root ancilla in each code stabilizer for the syndrome-measurement circuits (or for the entire protocol in the case of the $H_m^{(d)}$ circuit) and that the ancillas, root ancillas, and flag qubits are all distinct.

For an X-type stabilizer measurement and logical H_L measurement, we know that the root ancilla is prepared in a $+1$ eigenstate of the X operator. The nonroot ancillas and flag qubits are prepared in a $+1$ eigenstate of the Z operator. The (root and nonroot) ancillas are measured in the X basis. The flag qubits are measured in the Z basis. For a Z-type stabilizer measurement, the X and Z bases are all swapped, that is, we replace X with Z and Z with X in the preceding description. As such, we do not need to produce a solution for Z-type stabilizer measurements. Here, as in Sec. II B 5, we refer to the measurement bases of each qubit by $\{P_i\}$, with the understanding that this denotes a pair of symbolic Boolean values $(\text{IsMeasuredInXBasis}_i, \text{IsMeasuredInZBasis}_i)$ as explained in Sec. II B 5. In our case, for each qubit, the preparation-and-measurement basis of the qubit is the same and we use P_i to refer to this one basis.

We then set a number of time steps $N^{(H)}$ and $N^{(ED)}$ for each circuit to complete. Then we declare the gate-time encoding variables $\{X_{ij}^{(H)} : i \in [w], j \in [N^{(H)}]\}$ and $\{X_{ij}^{(ED)} : i \in [w], j \in [N^{(ED)}]\}$ for each protocol; these variables are used in separate SMT decision problems (one for each of $ED^{(d)}, H_m^{(d)}$) but we refer to these just as X_{ij} when we are speaking about a generic protocol of the two. We construct the gate-exclusion relations constraint for each circuit as explained in Sec. II B 3. We label these SMT formulas by $\text{GateExclusion}^{(H)}, \text{GateExclusion}^{(ED)}$. Since the construction of these formulas proceeds analogously for both protocols, we refer to these formulas generically as GateExclusion.

Although the qubit interaction graph shown in Fig. 4 already has maximum degree 3 as desired, we can also start with a higher-degree graph and enforce a global degree connectivity constraint with variables $X_{ij}^{(H)}$ and $X_{ij}^{(ED)}$ in Eq. (24), as explained in Sec. II B 3.

We then construct the bit matrix $\overline{O}$ that describes the Clifford operation we would like to implement, as used in Eq. (17). Since we allow the SMT solver to choose which qubits are flag, ancilla, and root-ancilla qubits, we must use a symbolic matrix to represent $\overline{O}$, as explained previously in Sec. II B 5. Recall that we have variables P_i for the preparation-and-measurement basis. For each qubit $i \in A$, we construct a bit vector S_i that describes the initial stabilizer of the input state acting on this qubit. As this depends on P_i, we must make this a symbolic bit vector so that it represents the appropriate initial Pauli stabilizer depending on the role of the qubit. To understand how to construct the symbolic desired bit matrix $\overline{O}$, let us consider its four

quadrants: the upper-left quadrant, corresponding to X-to-X propagation; the lower-right quadrant, corresponding to Z-to-Z propagation; and the off-diagonal quadrants. The off-diagonal quadrants are set to 0. Recall that we call the upper-left quadrant $\overline{O}|_X$ and the lower-right quadrant $\overline{O}|_Z$ reduced bit matrices. Due to the symmetry of CNOT-gate propagation for X- and Z-type Paulis [as shown in Eqs. (3)–(6)], since our circuits are only composed of CNOT gates we have that $\overline{O}|_X = \overline{O}|_Z^T$. By this symmetry, we must therefore only specify one of the two quadrants, so we specify how the $\overline{O}|_X$ quadrant is constructed. Further, we do not have to fully specify $\overline{O}|_X$, as the value of $(\overline{O}|_X)_{j,i}$ does not matter for any i which is a nonroot-ancilla or a flag qubit with $j \neq i$. The reason that $(\overline{O}|_X)_{j,i}$ does not matter for such j and i is the particular input state that we choose. These off-diagonal values $(\overline{O}|_X)_{j,i}$ for nonroot columns i can be set arbitrarily by right multiplying $\overline{O}$ by $\overline{\text{CNOT}_{i,j}}$. These $\text{CNOT}_{i,j}$ gates would have no effect on the input state since the nonroot qubits are initialized in the $|0\rangle$ state. In more formal terms, we only specify the symbolic bit matrix $\overline{O}$ up to right multiplication by an arbitrary Clifford operation that we know stabilizes the incoming state anyway. Therefore, we just set these rows of the symbolic bit vector for this column equal to a wildcard value $*$.

The remaining columns of $\overline{O}|_X$ are associated with the root ancillas of all stabilizers and the data qubits. These columns must be constrained exactly. In particular, the initial X stabilizer on the root ancilla must propagate to an X-type Pauli operator supported on all ancillas (including the root) in its stabilizer, along with the data qubits included in that stabilizer. Therefore for each $i \in A$, if $\text{IsRoot}_i = 1$, we must have the ith column of $\overline{O}_{XX}$ set equal to the vector supported on all the root and nonroot ancillas and data qubits of this stabilizer. This vector can be constructed symbolically using the $\text{IsRoot}_i, \text{IsFlag}_i$ variables to determine whether it is supported on an A qubit. Furthermore, for all the data qubits $i \in [n] \setminus A$, there should be no propagation of X errors from the data qubits onto the A qubits, as the data qubits are always the *target* of CNOT gates. Therefore, for each $i \in [n] \setminus A$, the ith column of $\overline{O}_{XX}$ should be 0 except for a 1 in the ith row.

We have now explained how all of the entries of $\overline{O}$ would be determined, up to right multiplication by arbitrary Clifford stabilizers of the incoming state. We do not have a particular setting of the qubit roles, as these are determined by $\{\text{IsFlag}_i\}$ and $\{\text{IsRoot}_i\}$. We handle this by replacing the constraint given in Eq. (17) with a SMT formula that we label HasDesiredEffect. This is easy to construct by going column by column through the restricted bit-matrix columns of $\overline{C}|_X$. For the ith column, we construct a symbolic bit vector that is determined symbolically by IsRoot_i and IsFlag_i as discussed above, using standard If-Then-Else support from the SMT solver. We then set the nonwildcard rows of this symbolic bit vector equal to the corresponding rows of the symbolic bit matrix of the entire circuit, as computed by the product-sum formula [Eq. (16)].

Finally, we use the techniques of Sec. II B 5 to add v-flag constraints to the SMT problem. Recall that we can produce the SMT formula IsVFlag in its entirety using the function shown in Eq. (31) and providing the appropriate $\{X_{ij}\}$, $\{P_i\}$, and $\{\text{IsRoot}_i\}$ variables for the relevant protocol as inputs. The final SMT decision problem for designing the circuit is then given by

$$F_{\text{SMT}} = \text{IsValidRoleAssignment} \wedge \text{GateExclusion} \wedge \text{HasDesiredEffect} \wedge \text{IsVFlag}. \quad (34)$$

This formula F_{SMT} is then fed into an off-the-shelf SMT solver such as Z3 [19]. For large problem instances, we apply the iterative solving techniques of Sec. II C to speed up the symbolic construction of F_{SMT} and the time taken by the solver to find a solution.

We remark that using the methods described above, the circuit $H_m^{(3)}$ in Fig. 4 has low degree and is obtained systematically. In Fig. 7(a) of Ref. [18], a similar alternative circuit to $H_m^{(3)}$ has been obtained "by hand" and is more highly structured. However, this alternative circuit has much higher degree (see Table II). SMT solvers offer a systematic alternative to hand design when finding optimal low-degree circuits conforming to realistic hardware constraints. Such optimal solutions might have less apparent structure than those that are hand designed. Fortunately, SMT solvers can automatically generate proofs of optimality with respect to protocol depth, degree, etc. For example, we use Z3 to prove that the protocols shown in Fig. 4 have the minimum possible number of time steps for a degree-3 qubit interaction graph given the geometric and other problem constraints.

C. Complexity of finding v-flag circuits using SMT solvers

In the preceding sections, we explain several optimizations for the encoding of a fault-tolerant quantum

TABLE II. A comparison of protocols from Fig. 7 of Ref. [18] with corresponding protocols in this work shown in Fig. 4.

	Figure 7 of Ref. [18]	Figure 4 of this work
Device requirements		
Connectivity degree	6	3
Number of ancilla qubits	9	10
$H_m^{(d)}$ circuit		
Time steps	8	8
Degree used	5	3
$ED^{(d)}$ circuit		
Time steps	7	6
Degree used	3	3

circuit-design problem into an SMT decision formula F_{SMT}. These optimizations reduce the formula size significantly. For example, as explained in Sec. II A 3, the product-sum relation allows us to build formulas with size scaling in the number of time steps N, rather than the (much larger) total number of gates g. For the design of v-flag syndrome-extraction circuits of a distance-d code, these optimizations can ensure that

$$\mathrm{size}\,(F_{\mathrm{SMT}}) = O\,(\mathrm{poly}(d)^v)\,.$$

These formulas therefore have polynomial size whenever v is a constant. The naive brute-force algorithm to solve F_{SMT} has run time $O(2^{\mathrm{size}(F_{\mathrm{SMT}})})$. SMT solvers cannot provide a better run-time guarantee, so our worst-case complexity is still

$$O\left(2^{\mathrm{poly}(d)^v}\right),$$

which is doubly exponential if $v = \Omega(d)$. However, despite the doubly exponential run time in the worst case, we find that the performance is "good enough" for typical problem instances such as small code distances relevant for near-term quantum devices. Our suite of optimizations allows us to solve for stabilizer and $H_m^{(d)}$ measurement circuits for color-code distances up to $d = 7$ with a reasonable run time. As a benchmark, we run our solver for a 90-qubit system of a distance-5 code to codesign the following three circuits:

(1) A 1-flag $H_m^{(d)}$ circuit, with a CNOT depth of 14.
(2) A 1-flag $ED^{(d)}$ circuit, with a CNOT depth of 6.
(3) An additional 1-flag syndrome-extraction circuit for the merged surface-color code described in Sec. IV, with a CNOT depth of 9. Such 1-flag circuits are required due to the weight-6 stabilizers along the boundary of the surface code and color code.

This design problem has 44 data qubits and 46 ancilla qubits and uses distance $d = 5$ codes. In what follows, the numerics are obtained by running the solver on specialized z1d.metal Amazon Web Services (AWS) EC2 instances, which provide an all-core sustained clock frequency of up to 4.0 GHz. Our solver constructs and solves F_{SMT} to find all three circuits in 1 h 14 min, with a degree constraint limiting each qubit to interacting with at most three nearest-neighbor qubits. This time reduces to just 58 min when the degree constraint is loosened to 4.

The heuristics used by the SMT solver (Z3 [19]) are extremely effective compared to the brute-force strategy of guessing every possible circuit. In the $H_m^{(d)}$ subproblem described above, there are 141 possible CNOT gates at each time step, for a total of 1974 possible CNOT gates in the circuit. The number of possible circuits is therefore at least 2^{1974}, which is significantly larger than the number of atoms in the known universe. As such, the heuristics of the solver are hugely beneficial. One possible reason for the effectiveness of SMT solvers on our SMT formulas is that all of our variables are over finite (Boolean) domains and we do not use *quantifiers* such as "∃" and "∀." The problem of deciding whether an SMT formula containing such features is satisfiable is undecidable in general. To illustrate the challenge presented by such formulas, consider the SMT formula $(a^{100} + b^{100} = c^{100}) \wedge (a > 1 \wedge b > 1 \wedge c > 1)$, where the variables a, b, c range over the integers. This formula is unsatisfiable, as implied by Fermat's last theorem [49]. Nonetheless, it would be extremely surprising if contemporary SMT solvers could generate a proof of the unsatisfiability of this formula in any reasonable amount of time.

We add that it is not always necessary to scale the code distance d to obtain a family of schemes that is ultimately fault tolerant. For example, the error-detection scheme for $H_m^{(d)}$ would have an unacceptably high rejection rate for code distances $d \geq 9$ due to the very large number of fault locations. Therefore, in practical settings, $|H\rangle$-type magic states would be prepared using our method for distances $d \leq 7$. Such states would then be teleported to states encoded in the surface code (see Sec. IV below). Afterward, a magic state encoded in the surface code would be grown to a larger code distance using gauge-fixing and lattice-surgery methods. Lastly, a top-down MSD protocol would be used to further increase the fidelity of the state. Therefore, to limit the cost of the top-down protocols, it is crucial that the injected magic states have the highest possible fidelities, warranting the use of sophisticated methodologies such as SMT solvers.

IV. DECODING MERGED SURFACE CODE AND COLOR CODE

A. Motivation

In Sec. III, we provide tools to find circuits that can be used to fault-tolerantly prepare $|H\rangle$-type magic states encoded in the color code. However, a limitation of such protocols is that the final magic states are encoded in the color code rather than the surface code. As previously discussed, the surface code offers much better performance for quantum memory and computation implemented via lattice surgery. In order to ensure that the prepared magic states can be used in a competitive scheme for universal fault-tolerant quantum computation, in this section we consider a protocol for teleporting states encoded in the color code into states encoded in the surface code. The protocol requires the use of lattice-surgery techniques [2,47,50–53], where gauge fixing results in a merged surface or color code, after which the codes are split up again to terminate the two-qubit teleportation step. Our approach is an adaptation of the technique proposed in Ref. [54], although there are several key differences and extensions. First, we do

not require additional data qubits for the code obtained by merging the surface code and color code (which we refer to as the merged code). Second, we explicitly construct a decoder for the merged code (an aspect that has not been addressed in Ref. [54]).

In Ref. [55], the authors describe a decoding algorithm for "hybrid color-toric" (HCT) codes constructed by local modifications to 2D color codes that create localized ball-like toric code regions. These HCT codes and their associated decoding algorithm differ from the merged color-surface code that we consider in the work (see Algorithm 2). In the HCT codes in Ref. [55], there is a correspondence between the vertices of the syndrome lattice of the HCT code and the vertices of the "original" unmodified color-code lattice. This allows their local lift procedure (see Ref. [55], Appendix G) to be described in terms of the standard local lift of the color code introduced by Ref. [56], as it would be applied to the vertices of the original unmodified color code. In our hybrid code, there is no such correspondence, so the HCT local lift is not well defined and we must define the lifting procedure from scratch. Furthermore, the HCT decoder would need to be modified in order to be used for codes with boundaries such as our code; the authors of Ref. [55] suggest but do not explicitly describe how this would be done.

Figure 5 depicts the teleportation circuit for transferring a state encoded in the color code to a state encoded in the surface code. To implement this circuit fault-tolerantly using the hardware constraints mentioned above, lattice surgery must be used to measure the two-qubit $X \otimes X$ operator. After measuring $X \otimes X$, repeated rounds of error correction must be performed on the merged code in order to prevent both timelike and spacelike errors from giving the wrong parity of the measurement outcome. As such, a decoder is required to process the stabilizer syndrome measurements of the merged code during the lattice-surgery protocol.

When performing gauge fixing to merge the color code and surface codes, the X-operator measurements at the boundaries of the two-codes anticommute with some Z stabilizers of the original codes, causing them to merge into elongated stabilizers. The stabilizers of the unmerged codes and the stabilizers of the merged code (including merging operators) are shown in Fig. 6. We also note that the theoretical framework of Ref. [23] can be used

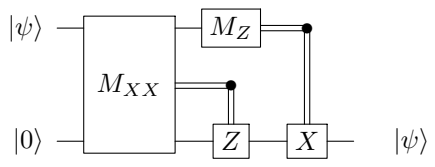


FIG. 5. The quantum circuit for teleporting a logical state $|\psi\rangle$ from the color code to the surface code.

to obtain the semitransparent domain walls (i.e., the stabilizer generators between the surface code and color code), allowing us to perform lattice surgery between the color code and the surface code.

In Sec. IV B, we introduce some notation and provide some definitions. In Sec. IV C, we provide the details of our decoding algorithm for the merged code.

B. Notation and definitions

In what follows, we set $L \geq 3$ to be odd.

A triangular color code (CC) of distance $d = L$ is a CSS code with identical X and Z stabilizers, where the data qubits can be placed at the vertices of a 2D hexagonal lattice. We consider an orientation as shown in Figs. 6(a) and 6(c), right-hand side (for distance 5). We have a 3-coloring of red (R), green (G), and blue (B) tiles corresponding to stabilizer generators of the color code.

The surface code (SC) of distance $d = L$ on a square lattice is a CSS code with differing X- and Z-stabilizer subgroups. The stabilizers for a $d = 5$ surface code are shown on the left-hand side of Figs. 6(a) and 6(c). We assign colors red (R) and blue (B) to the X and Z stabilizers of the surface code, respectively.

For $P \in \{X, Z\}$, we define the CC (SC) P syndrome graph $\mathcal{L}^{(\mathrm{CC},P)*}$ ($\mathcal{L}^{(\mathrm{SC},P)*}$), the vertices of which are associated with stabilizer-measurement outcomes of P-type stabilizers of the color (surface) code. In addition, we add virtual boundary vertices corresponding to virtual stabilizers, the color of which is indicated by its subscript. For the color code, we add three P-type virtual stabilizers to $\mathcal{L}^{(\mathrm{CC},P)*}$:

$$\left\{ v_R^{(\mathrm{CC},P)}, v_G^{(\mathrm{CC},P)}, v_B^{(\mathrm{CC},P)} \right\};$$

these are illustrated as isolated dots in Fig. 6. The support of $v_C^{(\mathrm{CC},P)}$ is the set of qubits of the color code that are not contained in any (real) C-colored P-type stabilizers.

For the surface code, we add one virtual stabilizer to each of $\mathcal{L}^{(\mathrm{SC},X)*}$ and $\mathcal{L}^{(\mathrm{SC},Z)*}$: the X-type red virtual stabilizer $v_R^{(\mathrm{SC},X)}$ is added to $\mathcal{L}^{(\mathrm{SC},X)*}$ and the Z-type blue virtual stabilizer $v_B^{(\mathrm{SC},Z)}$ is added to $\mathcal{L}^{(\mathrm{SC},Z)*}$. The support of $v_R^{(\mathrm{SC},X)}$ is the set of all surface-code qubits that are contained in exactly one (real) X-type stabilizer, while the support of $v_B^{(\mathrm{SC},Z)}$ is the set of all qubits that are contained in exactly one (real) Z-type stabilizer. Finally, we add edges to the syndrome graphs $\mathcal{L}^{(P,\mathrm{CC})*}$, $\mathcal{L}^{(P,\mathrm{SC})*}$ between all pairs of stabilizers (including virtual stabilizers) sharing one or more common qubits.

We now explain how gauge fixing is implemented to combine the SC and the CC into a single merged code. Consider the orientation shown in Fig. 6 for a distance-5 surface code and color code. The rightmost weight-2 blue Z stabilizers are labeled $\alpha_1, \ldots, \alpha_{(d-1)/2}$ from top to bottom. The leftmost weight-4 blue Z stabilizers are labeled $\beta_1, \ldots, \beta_{(d-1)/2}$ from top to bottom. We also denote by

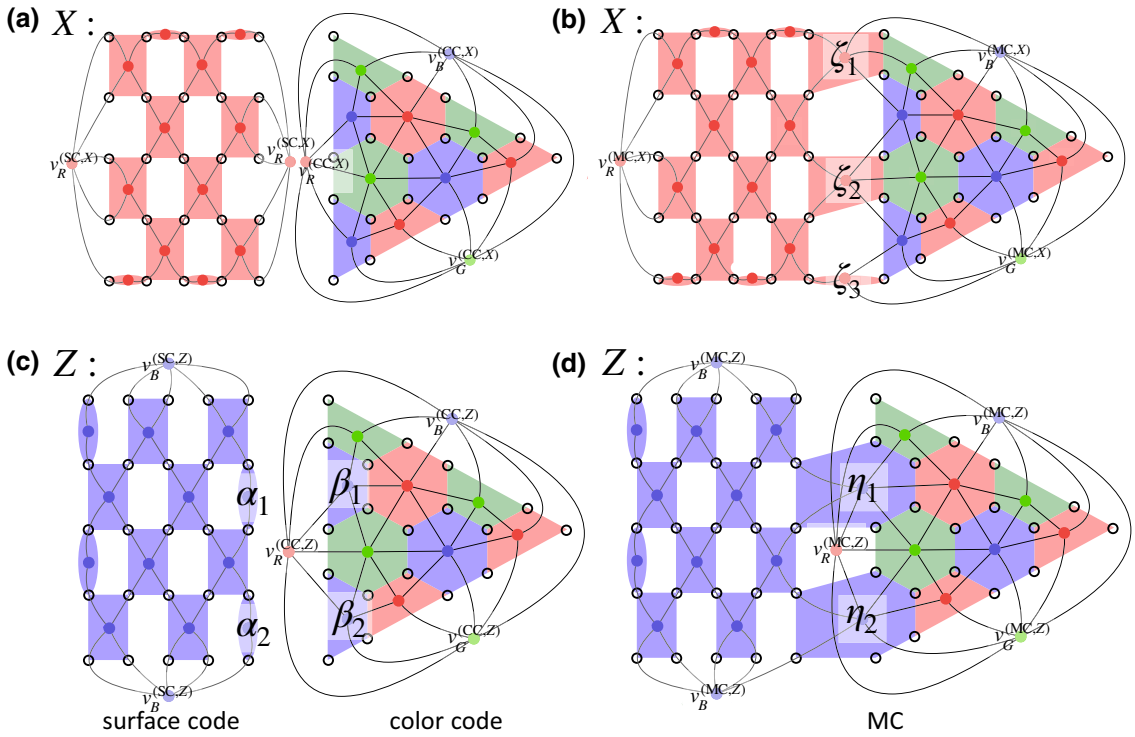


FIG. 6. X and Z syndrome graphs $\mathcal{L}^{(\mathrm{MC},P)*}$ of the merged surface code and color code. Surface-code qubits live on edges, while color-code qubits live on faces. The restriction decoder [3,56] *lifts* one-dimensional paths produced by a toric code decoder to recover 2D face qubits.

$\zeta_1, \ldots, \zeta_{(d+1)/2}$ the red X-type operators that are labeled in Fig. 6(b).

To produce a merged code from a surface code and color code, we gauge fix by measuring the ζ_i operators. After performing the measurement, the ζ_i operators are now red X stabilizers of the resulting merged code. Clearly, ζ_1 anticommutes with both α_1 and β_1 but commutes with $\alpha_1\beta_1$. Therefore upon measuring ζ_1, we no longer have α_1 and β_1 as stabilizers but the operator $\alpha_1\beta_1$ remains a stabilizer. We denote this new stabilizer by η_1. By similar reasoning, we see that all the stabilizers α_i, β_i are removed upon measuring all the merging operators ζ_i, but their products $\eta_i := \alpha_i\beta_i$ are left as new stabilizers of the merged code. These new η_i stabilizers are shown in blue in Fig. 6(c). In what follows, we refer to this new code as the merged code (MC).

For $P \in \{X, Z\}$, we define the MC syndrome graph $\mathcal{L}^{(\mathrm{MC},P)*}$ as follows: the vertices of $\mathcal{L}^{(\mathrm{MC},P)*}$ are associated with the P-type stabilizers of the MC. For the MC, we add the three P-type virtual stabilizers $\{v_R^{(\mathrm{MC},P)}, v_G^{(\mathrm{MC},P)}, v_B^{(\mathrm{MC},P)}\}$ to $\mathcal{L}^{(\mathrm{MC},P)*}$. The six total virtual stabilizers of $\mathcal{L}^{(\mathrm{MC},X)*}$ and $\mathcal{L}^{(\mathrm{MC},Z)*}$ are thus

$$\left\{v_R^{(\mathrm{MC},X)}, v_G^{(\mathrm{MC},X)}, v_B^{(\mathrm{MC},X)}, v_R^{(\mathrm{MC},Z)}, v_G^{(\mathrm{MC},Z)}, v_B^{(\mathrm{MC},Z)}\right\}. \tag{35}$$

All of the MC virtual stabilizers *except for* $v_R^{(\mathrm{MC},X)}$ have the qubit support given by the union of the corresponding original code supports. Specifically,

$$\forall (P,C) \neq (X,R), \mathrm{support}(v_C^{(\mathrm{MC},P)}) = \mathrm{support}(v_C^{(\mathrm{CC},P)}) \cup \mathrm{support}(v_C^{(\mathrm{SC},P)}), \tag{36}$$

with the convention that

$$\mathrm{support}(v_B^{(\mathrm{SC},X)}) = \emptyset$$
$$\mathrm{support}(v_R^{(\mathrm{SC},Z)}) = \emptyset$$
$$\mathrm{support}(v_G^{(\mathrm{SC},X)}) = \emptyset$$
$$\mathrm{support}(v_G^{(\mathrm{SC},Z)}) = \emptyset$$

(as these virtual stabilizers have never been defined for the SC.) The support of the final virtual stabilizer $v_R^{(\mathrm{MC},X)}$ is the set of surface-code qubits that are contained in the support of exactly one red surface-code stabilizer and no red, green, or blue stabilizers in the MC. This corresponds to the leftmost column of qubits in Fig. 6.

Finally, we once again add edges to the syndrome graph $\mathcal{L}^{(\mathrm{MC},P)*}$ between all pairs of stabilizers (including virtual stabilizers) sharing one or more common qubits. The resulting syndrome graph edges are illustrated for $L = 5$

in Fig. 6. We also define a classification of the qubits of the MC as either *face qubits* or *edge qubits*. For a qubit q, its classification is determined by the number t of stabilizers (including virtual stabilizers) that contain q. By inspection, we either have $t = 2$ or $t = 3$. If $t = 2$, we say that q is an edge qubit. Otherwise, $t = 3$ and we say that q is a face qubit. We define the *1-boundary* denoted $\partial_2 q$ of a face qubit q, as the set of the three edges in $\mathcal{L}^{(\mathrm{MC},P)*}$ that connect the three stabilizers of q. Note that we must indeed have all three edges in $\mathcal{L}^{(\mathrm{MC},P)*}$, as any stabilizers sharing a qubit share an edge in $\mathcal{L}^{(\mathrm{MC},P)*}$. We now extend this notion to a set of face qubits. For a set of face qubits $S = \{q_1, \ldots, q_\ell\}$, we define the 1-boundary denoted $\partial_2 S$ as

$$\partial_2 S = \bigoplus_{q \in S} \partial_2 q, \tag{37}$$

in which $\oplus$ denotes the symmetric difference of sets and $\partial_2 q$ is just the 1-boundary of the individual face qubit q. The edges $e \in \mathcal{L}^{(\mathrm{MC},P)*}$ are given a real-valued weight that we denote $\mathrm{wt}(e)$, which is set to either w_1 or w_2, where $w_1, w_2 \in \mathbb{R}$. Intuitively, w_1 sets the weight of edges in the surface code, while w_2 sets the weight of edges in the color code. Specifically, if the stabilizers associated with vertices u and v share any face qubits, we set the weight of (u, v) to w_2, and otherwise we set the weight to w_1. Optimal values for w_1 and w_2 are found numerically by computing the logical error rates of the MC code for a given noise model.

The X distance of MC is $d_X = L$ and the Z distance is $d_Z = 2L$. Note that this is an advantageous configuration for biased-noise models where Z errors are more likely than X errors. Furthermore, the orientation can be swapped such that the X distance is larger in the case of X-biased noise.

For each $P \in \{X, Z\}$, we also define three restricted graphs $\mathcal{L}_{RB}^{(\mathrm{MC},P)*}, \mathcal{L}_{RG}^{(\mathrm{MC},P)*}, \mathcal{L}_{BG}^{(\mathrm{MC},P)*}$ as subgraphs of the full MC syndrome graph $\mathcal{L}^{(\mathrm{MC},P)*}$. These graphs are defined such that:

(1) $\mathcal{L}_{RB}^{(\mathrm{MC},P)*} \cup \mathcal{L}_{RG}^{(\mathrm{MC},P)*} \cup \mathcal{L}_{BG}^{(\mathrm{MC},P)*} = \mathcal{L}^{(\mathrm{MC},P)*}$.
(2) $\mathcal{L}_{C_1C_2}^{(\mathrm{MC},P)*}$ contains all vertices of color $C \in \{C_1, C_2\}$ and edges between these vertices.

The restricted graphs will be used for the decoding algorithm as explained in Sec. IV C.

We now define a few graph-theoretic notions that are used in our MC decoding algorithm. For a graph $G = (V, E)$ with non-negative edge weights $E \xrightarrow{w} \mathbb{R}_{\geq 0}, e \mapsto w(e)$, we recall that $M \subset E$ is a *perfect matching* (PM) if each vertex $v \in V$ has exactly one edge in the set M. More generally, for a subset $A \subset V$, we say that $M \subset E$ is an *A-perfect matching* (A-PM) if each vertex $v \in A$ has exactly one edge in the set M. Note that this permits vertices $v \in V \setminus A$ to have any number of edges in the A-perfect matching. Note also that a V-perfect matching is simply a perfect matching. Finally, we say that $M \subset E$ is a *minimum-weight A-perfect matching* if for all A-perfect matchings M', $\sum_{e \in M} w(e) \leq \sum_{e \in M'} w(e)$. There exists a polynomial-time algorithm [57] for finding a minimum-weight perfect matching of a weighted graph, when one exists. Given a weighted graph G and vertex subset A as above, it is possible to construct a new graph G' such that a minimum-weight perfect matching of G' can be used to recover a minimum-weight A-perfect matching of G. This construction is illustrated for a small example in Fig. 7 and

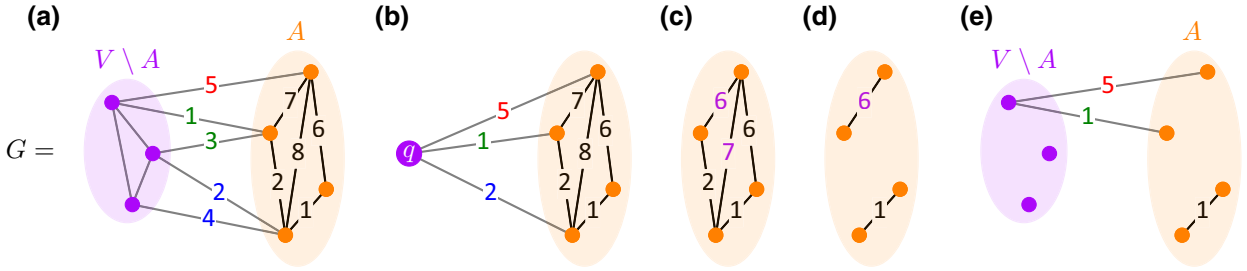


FIG. 7. An illustration of Algorithm 1 on a small graph G. (a) A graph $G = (V, E)$ with non-negative edge weights and a vertex subset $A \subset V$. The weights on edges incident to vertices in $V \setminus A$ are omitted, as these edges can be removed from any A-perfect matching without increasing its total weight. When applying this algorithm to the matching graph of the surface code, all vertices in $V \setminus A$ correspond to virtual boundary vertices (and thus all edges within $V \setminus A$ have zero weight). (b) A new graph is assembled by replacing all vertices in $V \setminus A$ with a single new vertex q. The edges between A and $V \setminus A$ are replaced with some edges from q to vertices in A, as follows. For each vertex $v \in A$ with an edge to $V \setminus A$ in G, we add an edge (q, v), setting the weight of this new edge to the minimum of all edge weights between v and $V \setminus A$ in G. This is analogous to the computation of $q(v)$ in Line 3. (c) For each edge $(v_1, v_2) \in E$ with $v_1, v_2 \in A$, such that $w(v_1, u) + w(v_2, u) < w(v_1, v_2)$, we update the edge weight to $w(v_1, u) + w(v_2, u)$ and set the label function $L(v_1, v_2)$ to 1 (Line 11) (indicated by the purple weight). (d) A minimum-weight perfect matching M' of the graph is computed (in this example, resulting in highlighted edges of weight 6 and 1). (e) A minimum-weight A-perfect matching M is recovered by replacing all edges $e \in M'$ such that $L(e) = 1$ with the two minimum-weight edges from the endpoints of e to $V \setminus A$ (corresponding to $n(v)$ as set in Line 4).

1: Set $w(u_1,u_2)=\infty \Leftrightarrow (u_1,u_2)\notin E$
2: **for** $v\in A$ **do**
3: $q(v)\leftarrow \min_{u\in V\setminus A} w(v,u)$
4: $n(v)\leftarrow \text{argmin}_{u\in V\setminus A} w(v,u)$.
5: **end for**
6: **for** $(u_1,u_2)\in E$ **do**
7: **if** $u_1\in A$ and $u_2\in A$ **then**
8: $L(u_1,u_2)\leftarrow 0$
9: $w'(u_1,u_2)\leftarrow q(u_1)+q(u_2)$
10: **if** $w'(u_1,u_2)\leq w(u_1,u_2)$ **then**
11: $L(u_1,u_2)\leftarrow 1$
12: $w(u_1,u_2)\leftarrow w'(u_1,u_2)$
13: **end if**
14: **end if**
15: **end for**
16: $G'=(V',E')\leftarrow G\setminus(V\setminus A)$
17: **if** $|V'|$ is odd **then**
18: $G'\leftarrow G'\cup\{v_0\}$
19: **for** $u\in A$ **do**
20: $G'\leftarrow G'+(u,v_0)$
21: $w(u,v_0)\leftarrow q(u)$
22: **end for**
23: **end if**
24: Find a min. weight PM $M'\subset E'$
25: $M\leftarrow\emptyset$
26: **for** $(u_1,u_2)\in M'$ **do**
27: **if** $v_0\in\{u_1,u_2\}$ **then**
28: Let $u\in\{u_1,u_2\}\setminus\{v_0\}$
29: $M\leftarrow M\cup(u,n(u))$
30: **else**
31: **if** $L(u_1,u_2)=1$ **then**
32: $M\leftarrow M\cup(u_1,n(u_1))$
33: $M\leftarrow M\cup(u_2,n(u_2))$
34: **else**
35: $M\leftarrow M\cup(u_1,u_2)$
36: **end if**
37: **end if**
38: **end for**

Algorithm 1. Produces a minimum-weight A-perfect matching $M\subset E$ of a weighted graph $G=(V,E)$ with $A\subset V$ and edge weights $w(u,v)$ for $(u,v)\in E$

exploited by Algorithm 1, which uses the minimum-weight perfect-matching algorithm as a subroutine on Line 24.

C. Merged surface-color-code decoding algorithm

A decoding algorithm for triangular color-code families has been provided in Ref. [3]. We now provide a modified version of this decoder to handle the effects of the surface-code boundary, thus making it compatible with the MC code. In this section, we explain how our decoder works in detail. Our full decoding algorithm is then shown in Algorithm 2. For convenience, in this section we refer to specific numbered lines of Algorithm 2.

Let $\mathcal{E}$ be a physical Pauli-error operator on the data qubits with X and Z error syndromes $S_X(\mathcal{E}_Z), S_Z(\mathcal{E}_X)$, respectively. That is, the set $S_P(\mathcal{E}_{P'})$ is a subset of the real vertices of $\mathcal{L}^{(\mathrm{MC},P)*}$. The X and Z syndromes will be decoded independently to produce Z and X correction operators $\mathcal{E}'_Z, \mathcal{E}'_X$, respectively. Hence, in what follows, let $P\in\{X,Z\}$. As in Line 2, fix $P'\in\{X,Z\}$ and $P'\neq P$. Initialize the correction operator $\mathcal{E}'_{P'}=\emptyset$ as in Line 3. We refer to the stabilizers contained in $S_P(\mathcal{E}'_{P'}\mathcal{E}_{P'})$ as the *marked stabilizers*. Note that as the correction operator is empty at the start of the algorithm, the initial set of marked stabilizers is the same as the P-syndrome input to the decoder.

We decode $S_P(\mathcal{E}_{P'})$ in two stages, which we call the *color-code stage* and the *surface-code stage*. The color-code stage corresponds to Line 4 through Line 40, while the surface-code stage corresponds to Line 41 through Line 43.

The color-code stage produces a partial correction $\mathcal{E}'_{P'}$ that only contains color-code qubits. We use this partial correction to update the P-syndrome marked stabilizers $S_P(\mathcal{E}'_{P'}\mathcal{E}_{P'})$. At the end of the color-code stage, no color-code stabilizers are marked except possibly some of the η_i. In contrast, surface-code stabilizers may still be marked. This is because no surface-code qubits are contained in the partial correction $\mathcal{E}'_{P'}$ at this stage and hence any initially marked surface-code stabilizers will remain marked at the end of the color-code stage. To be clear, the partial correction $\mathcal{E}'_{P'}$ at the end of the color-code stage is stored in the software implementing Algorithm 2 and does not need to be actively applied to the physical data qubits.

After the color-code stage, we run the surface-code stage beginning on Line 41. The surface-code stage is simpler than the color-code stage. It makes some final modifications to the partial correction $\mathcal{E}'_{P'}$ so that there are no marked MC stabilizers in the marked set $S_P(\mathcal{E}'_{P'}\mathcal{E}_{P'})$. At this point, the P' correction $\mathcal{E}'_{P'}$ is completed.

We now explain the steps of the color-code stage in detail. The first step is to produce three *colored pairings* $(M_C,\{\Gamma_{u,v,C}\})$ for each of $C\in\{R,G,B\}$. This is done in Line 5 through Line 21. Fix any $C\in\{R,G,B\}$. Let $\{C_1,C_2\}=\{R,G,B\}\setminus\{C\}$. The colored pairing $(M_C,\{\Gamma_{u,v,C}\})$ consists of a set $M_C=\{(u,v)\}$ of pairs of vertices of $\mathcal{L}^{(\mathrm{MC},P)*}_{C_1,C_2}$, along with a path $\Gamma_{u,v,C}$ for each pair $(u,v)\in M_C$. This path $\Gamma_{u,v,C}$ joins u and v through the restricted graph $\mathcal{L}^{(\mathrm{MC},P)*}_{C_1,C_2}$. The path $\Gamma_{u,v,C}$ must be *legal*, as specified by the following conditions:

(1) $\Gamma_{u,v,C}$ is a path from u to v through $\mathcal{L}^{(\mathrm{MC},P)*}_{C_1,C_2}$.
(2) Among the edges of $\Gamma_{u,v,C}$, there is at most one edge that is incident to any virtual vertex.
(3) There is at most one virtual vertex visited by $\Gamma_{u,v,C}$.

(4) If u and v are vertices in the color code (including all the η_i vertices), then $\Gamma_{u,v,C}$ does not visit any vertices in the surface code.

We choose $\Gamma_{u,v,C}$ to be the minimum-weight legal path, which is found by the subroutine MinWeightLegalPath in Line 11. Specifically, MinWeightLegalPath returns the minimum-weight legal path if one exists and returns the placeholder symbol $\perp$ if no path exists satisfying the above conditions. The subroutine MinWeightLegalPath can be easily implemented by modifying Dijkstra's pathfinding algorithm to take into account the above legality conditions. Specifically, during the Dijkstra search, the virtual vertices should be treated as having zero out-edges, and if u and v are color-code vertices as defined above, then all edges to surface-code vertices are ignored.

The colored pairings $(M_C, \{\Gamma_{u,v,C} : (u,v) \in M_C\})$ enable us to recover a set of qubits in the color code that we add to the partial correction $\mathcal{E}'_{P'}$. Specifically, we define a subroutine (Lift), which is applied at some of the vertices visited by the pairing paths. The Lift subroutine has arguments $\text{Lift}(u, \Gamma)$. Here, u is a real (nonvirtual) vertex of the graph $\mathcal{L}^{(\text{MC},P)*}_{C_1,C_2}$ and Γ is a path through (i.e., a subset of edges of) $\mathcal{L}^{(\text{MC},P)*}_{C_1,C_2}$. The Lift subroutine returns a set of qubits, as follows. If u is a real vertex of the surface code, then the Lift subroutine returns the empty set. Otherwise, u is vertex of the color code and $\text{Lift}(u, \Gamma)$ returns a set of face qubits to add to $\mathcal{E}_{P'}$. Specifically, let $\Gamma|_u$ denote the set of edges contained in Γ that are incident to the vertex u. Then we have that

$$\partial_2 \text{Lift}\,(u, \Gamma) = \Gamma|_u. \tag{38}$$

(We remind the reader that ∂_2 denotes the 1-boundary as we define it in Sec. IV B.) The implementation of the Lift subroutine is easily and efficiently implemented using brute-force search or linear-algebra techniques, as described in Ref. [3]. Note that above we do not define the behavior of the Lift subroutine for virtual vertices $v_C^{(\text{MC},P)}$. As we explain, we take care never to apply the Lift subroutine to any such virtual vertices in our decoding algorithm and so this is not an issue. As a cautionary comment, note that above we use Γ as a placeholder for a general subset of edges through the graph $\mathcal{L}^{(\text{MC},P)*}$ and that Γ is *not to be confused* with the specific symbol $\Gamma_{u,v,C}$, which specifically denotes the minimum-weight legal path connecting u and v, as we have just described above. In fact, it is impossible to correctly lift a vertex u with the path $\Gamma = \Gamma_{u,v,C}$, as for this path there would be no subset of face qubits such that Eq. (38) is satisfied. However, in our algorithm, the Lift subroutine is always used in such a way that it has a valid output satisfying Eq. (38).

In our algorithm, we use Lift on Line 31 and Line 39 as follows:

$$\mathcal{E}'_{P'} \leftarrow \mathcal{E}'_{P'} \oplus \text{Lift}\,(u, \Gamma) \tag{39}$$

Above, the parity symbol $\oplus$ denotes the symmetric difference of the two sets $\mathcal{E}'_{P'}$ and $\text{Lift}(u, \Gamma)$. That is, we update the partial correction to be the set of qubits that are contained in exactly one of $\mathcal{E}'_{P'}$ or $\text{Lift}(u, \Gamma)$.

We must apply the Lift subroutine carefully. As shown in Ref. [3], we must avoid lifting at virtual vertices because it decreases the effective distance of the decoder. To avoid lifting at virtual vertices, we preprocess the paths that visit the green virtual stabilizer $v_G^{(\text{MC},P)}$. This preprocessing step takes place in Line 23 through Line 37. To explain this preprocessing step, we define the notion of a *boundary-connected component* (BCC). These components are sequences of colored pairing paths that begin

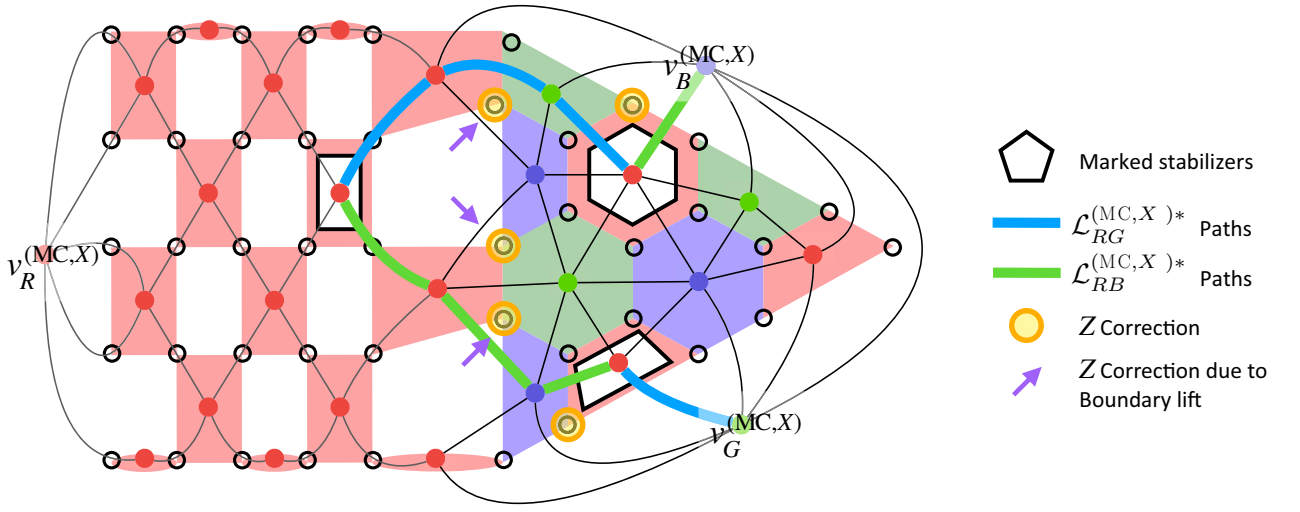


FIG. 8. Lifting at red surface-code vertices in the X syndrome graph is unavoidable when a component connects the blue to green virtual stabilizer and walks through such vertices along the way, as shown in the example.

on the green virtual stabilizer $v_G^{(\mathrm{MC},P)}$. Specifically, a BCC $\theta = (\{v_i\}, \{\chi_i\})$ of length ℓ is a sequence of vertices $v_i \in \mathcal{L}^{(\mathrm{MC},P)*}$ and colors $\chi_i \in \{R, G, B\}$:

$$(v_1, v_2, v_3, \ldots, v_{\ell-1}, v_\ell) \quad (\chi_1, \ldots, \chi_{\ell-1}), \tag{40}$$

for which

(1) $v_1 = v_G^{(\mathrm{MC},P)}$.
(2) $v_\ell = v_C^{(\mathrm{MC},P)} \in \{v_R^{(\mathrm{MC},P)}, v_G^{(\mathrm{MC},P)}, v_B^{(\mathrm{MC},P)}\}$ (including possibly $v_C^{(\mathrm{MC},P)} = v_G^{(\mathrm{MC},P)}$).
(3) For each $i \in 1, \ldots, \ell-1$, we have $(v_i, v_{i+1}) \in M_{\chi_i}$.
(4) All of the $\ell - 2$ pairs (v_i, χ_{i-1}), (v_i, χ_i) for $i \in 2, \ldots, \ell-1$ are distinct.

Additionally, the pairs (v_i, χ_i) are unique across the BCCs. That is, if $\theta' = (\{v_i'\}, \{\chi_i\})$ is any other BCC of length ℓ', and $i \in 2, \ldots, \ell-1$ and $j \in 2, \ldots, \ell'-1$, then we have $\{(v_i, \chi_i), (v_i, \chi_{i-1})\} \cap \{(v_j', \chi_j'), (v_j', \chi_{j-1}')\} = \emptyset$. The BCCs are obtained by doing a depth-first search starting from $v_G^{(\mathrm{MC},P)}$ through the colored multigraph with edges given by $M_R \cup M_G \cup M_B$. In this multigraph, each edge in M_C is given color C. During the depth-first search starting from $v_G^{(\mathrm{MC},P)}$, each edge of color C is removed from the appropriate matching set M_C as soon as it is crossed by the search. The depth-first search is halted when a virtual vertex $v_C^{(\mathrm{MC},P)}$ is reached. This is then set as the final vertex $v_\ell = v_C^{(\mathrm{MC},P)}$ of the BCC. Following this implementation, the BCCs will satisfy the above properties.

We now iterate over each BCC $\theta = (\{v_i\}, \{\chi_i\})$ in Line 23 through Line 37. In Line 24, we declare the path (that is, the set of edges) along the minimum-weight legal paths between subsequent v_i. That is, we set

$$\Gamma_\theta = \cup_{i=1}^{\ell-1} \Gamma_{v_i, v_{i+1}, \chi_i}.$$

We then assign θ a color denoted by $\mathrm{color}(\theta) \in \{R, B\}$ (Line 25). This is done so that Γ_θ never visits the virtual vertex $v_{\mathrm{color}(\theta)}^{(\mathrm{MC},P)}$. We apply the Lift subroutine at each vertex visited by Γ_θ with color equal to $\mathrm{color}(\theta)$. Recall that by definition, the Lift subroutine returns the empty set when passed a surface-code vertex. However, if the color of the BCC is set to red and its path Γ_θ crosses between the color and surface code, then we must do some *variant of* the Lift subroutine to find an appropriate correction in the vicinity of these red surface-code vertices. This is achieved by the subroutine $\mathrm{SCRedLift}(\Gamma_\theta)$. An example of applying SCRedLift is shown for illustration in Fig. 8. Formally, the SCRedLift subroutine finds the ordered sequence of edges

$$e_1, e_2, \ldots, e_{2m-1}, e_{2m} \in \Gamma_\theta$$

that are incident to both surface-code and color-code stabilizer and are ordered by increasing vertical position in the 2D layout. There must be an even number of such edges, since by construction the path Γ_θ must connect between $v_G^{(\mathrm{MC},P)}$ and $v_B^{(\mathrm{MC},P)}$, so Γ_θ must move from the color code to the surface code an even number of times. Now, for two such crossing edges (e_i, e_{i+1}), let $\mathrm{SandwichedQubits}(e_i, e_{i+1})$ denote the set of qubits that lie between e_i and e_{i+1} and are contained in the intersection of the color-code X stabilizers and the ζ_i X stabilizers of the MC code. Then, $\mathrm{SCRedLift}(\Gamma_\theta)$ returns the union of these

1: **for** $P \in \{X, Z\}$ **do**
2: Let $P' \in \{X, Z\}, P' \neq P$
3: $\mathcal{E}'_{P'} \leftarrow \emptyset$ ▷ Initialize empty partial correction
4: Let $A = S_P(\mathcal{E}'_{P'}\mathcal{E}_{P'})$
5: **for** $C \in \{R, G, B\}$ **do**
6: Let $\{C_1, C_2\} = \{R, G, B\} \setminus \{C\}$
7: $A_C \leftarrow A \cap \mathcal{L}_{C_1,C_2}^{(MC,P)*}$
8: $V_C \leftarrow A_C \cup \{v_{C_1}^{(MC,P)}, v_{C_2}^{(MC,P)}\}$
9: $w_C \leftarrow \emptyset$
10: **for** $(u, v) \in V_C \times V_C$ **do**
11: $\Gamma_{u,v,C} \leftarrow \mathrm{MinWeightLegalPath}(u, v, \mathcal{L}_{C_1,C_2}^{(MC,P)*})$
12: **if** $\Gamma_{u,v,C} = \perp$ **then**
13: $w_C(u, v) \leftarrow \infty$
14: **else**
15: $w_C(u, v) \leftarrow \sum_{e \in \Gamma_{u,v,C}} \mathrm{wt}(e)$
16: **end if**
17: **end for**
18: $G_C \leftarrow (V_C, E_C, w_C)$ ▷ Initialize Weighted C-Matching Graph
19: $M_C \leftarrow$ min. weight A_C-PM of G_C using Algorithm 1
20: $\Gamma_C \leftarrow \oplus_{(u,v) \in M_C} \Gamma_{u,v,C}$
21: **end for**
22: $\Gamma_T \leftarrow \Gamma_R \cup \Gamma_G \cup \Gamma_B$
23: **for** BCC $\theta = (\{v_i\}, \{\chi_i\})$ **do**
24: $\Gamma_\theta \leftarrow \cup_{i=1}^{\ell-1} \Gamma_{v_i, v_{i+1}, \chi_i}$
25: **if** $v_\ell \in \{v_G^{(MC,P)}, v_R^{(MC,P)}\}$ **then**
26: Set $\mathrm{color}(\theta) = B$
27: **else**
28: Set $\mathrm{color}(\theta) = R$
29: **end if**
30: **for** $u \in \Gamma_\theta|_{\mathrm{color}(\theta)}$ **do**
31: $\mathcal{E}'_{P'} \leftarrow \mathcal{E}'_{P'} \oplus \mathrm{Lift}(u, \Gamma_\theta)$
32: **end for**
33: **if** $\mathrm{color}(\theta) = R$ **then**
34: $\mathcal{E}'_{P'} \leftarrow \mathcal{E}'_{P'} \oplus \mathrm{SCRedLift}(\Gamma_\theta)$
35: **end if**
36: Set $\Gamma_T \leftarrow \Gamma_T \oplus \Gamma_\theta$
37: **end for**
38: **for** $u \in \Gamma_T|_G$ **do**
39: $\mathcal{E}'_{P'} \leftarrow \mathcal{E}'_{P'} \oplus \mathrm{Lift}(u, \Gamma_T)$
40: **end for**
41: $A \leftarrow S_P(\mathcal{E}'_{P'}\mathcal{E}_{P'})$
42: $\mathcal{E}'_{P'} \leftarrow \mathrm{SurfaceCodeCorrection}(A)$
43: **end for**

Algorithm 2. Produces Z and X corrections $\mathcal{E}'_Z, \mathcal{E}'_X$ given X and Z syndromes $S_X(\mathcal{E}_Z), S_Z(\mathcal{E}_X)$

sandwiched qubits across all the pairs:

$$\text{SCRedLift}(\Gamma_\theta) = \bigcup_{i=1}^{m} \text{SandwichedQubits}(e_{2i-1}, e_{2i}). \tag{41}$$

After applying the appropriate Lift subroutine calls (Line 31) and SCRedLift subroutine calls (Line 34) as appropriate, we can remove the BCC paths Γ_θ from the combined pairing paths (Line 36). After finishing this for all the BCCs θ, we are finished with the preprocessing step.

By carefully processing the BCCs as described and updating the partial correction $\mathcal{E}'_{P'}$ with lifts, we avoid lifting at $v_G^{(\text{MC},P)}$ or any other virtual vertex and we remove all of the paths to $v_G^{(\text{MC},P)}$ from the pairing paths (due to Line 36). The final step of the color-code stage is to iterate through the remaining pairing paths in Line 39, applying the aforementioned lift operation to all of the green vertices visited by the paths. Since we remove all paths to $v_G^{(\text{MC},P)}$ in the preprocessing step, we do not apply Lift at any virtual stabilizers in this final step.

After the color-code stage is concluded on Line 40, we run the surface-code stage on Line 41 through Line 43. First, we update the set of marked vertices $A = S_P(\mathcal{E}'_{P'}\mathcal{E}_{P'})$ based on the partial P'-type correction produced by the previous stage (Line 41). We then obtain a set of qubits SurfaceCodeCorrection(A) using the standard surface-code decoder on Line 42. That is, we set $V_{\text{SC}} = A \cup \{v_R^{(\text{MC},P)}, v_B^{(\text{MC},P)}\}$ and for each pair $(u, v) \in V_{\text{SC}} \times V_{SC}$, such that $u \neq v$, we set $\Gamma_{u,v}$ to be the minimum-weight path that joins u and v through $\mathcal{L}^{(\text{MC},P)*}|_{\text{SC}\cup\{\eta_i\}}$ (or else $\perp$ if none exists) and set $w(u, v) = \sum_{e\in\Gamma_{u,v}} \text{wt}(e)$. We let $G_{\text{SC}} = (V_{\text{SC}}, E_{\text{SC}}, w)$ be a weighted graph. We then find a minimum-weight A-perfect matching M_{SC} of the graph G_{SC} and for each $(u, v) \in M_{\text{SC}}$, we set $\mathcal{E}'_{P'} \leftarrow \mathcal{E}'_{P'} \oplus \Gamma_{u,v}$. This concludes the surface-code stage. After this point, there are no more marked vertices and $\mathcal{E}'_{P'}$ is completed.

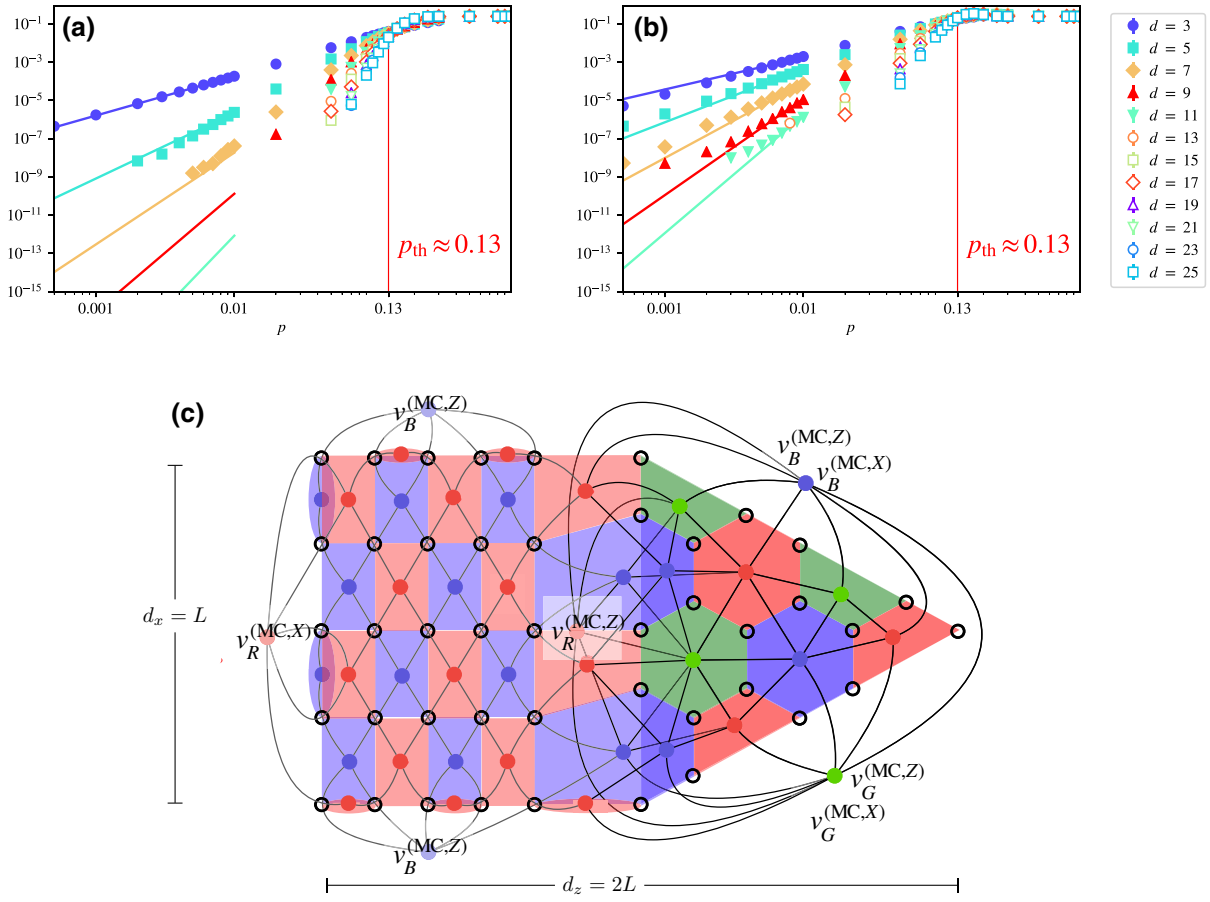


FIG. 9. (a), (b) The (a) $\bar{Z}$ and (b) $\bar{X}$ logical error rates of the merged code for various code distances using the decoder described in Algorithm 2. In obtaining the plots, we use a code-capacity depolarizing noise model. The solid lines are obtained by performing a best fit to the ansatz described in Sec. IV D. (c) The lattice illustrates the d_x and d_z code parameters as a function of the lattice size L. The effective d_x and d_z distances are obtained in Table III using the ansatz described in Eq. (42).

TABLE III. Best-fit parameters for the logical-error-rate curves $p_L^{(X)}$ and $p_L^{(Z)}$.

	$p_L^{(X)}$	$p_L^{(Z)}$
a	0.0964	0.0728
b	0.0108	0.2944
c	0.3441	0.6857

By avoiding all lifts at virtual vertices, our decoder obtains good performance, as shown by our numerics in Sec. IV D. Specifically, we maintain the full effective X distance d_X of the surface-code decoder and the combined effective Z distance $\left(1+\frac{2}{3}\right)d_Z$ of the surface-code decoder along with the color-code decoder ([3]).

D. MC code-capacity simulation results

In this section, we describe Monte Carlo simulation results of the MC code under a code-capacity depolarizing noise model where data qubits are afflicted by X, Y, and Z errors, each occurring with the same probability p. Data for the logical X and Z failure rates of the MC code for various values of L are shown in Fig. 9.

By carefully analyzing the data, we find the threshold for logical X and Z errors to be approximately $p_{\text{th}} \approx 0.13$. However, for many noise-parameter regimes, the logical Z error rate is substantially lower than the logical X error rate. In particular, for both logical X and Z error rates, we perform a best-fit analysis to obtain logical-error-rate curves $p_L^{(X)}$ and $p_L^{(Z)}$ using the following ansatz:

$$p_L = aL^2(bp)^{cL}, \tag{42}$$

where L is shown in Fig. 9 and a, b, and c are parameters obtained by the fit. As such, the parameter c describes the effective d_x and d_z distances, i.e., the minimum-weight X and Z errors that can cause a logical fault. The best-fit parameters are given in Table III. As can be seen from the parameter c in Table III, the effective d_x distance is roughly half the effective d_z distance. The reason is that horizontal Z error chains that can result in a logical Z error must span a length of size $2L$. In contrast, vertical X error chains that can cause a logical X error must span a length of size L.

V. CONCLUSIONS

In this paper, we show in Sec. II how Clifford circuits can be designed where the desired constraints (such as certain fault-tolerance proprieties, the degree of connectivity between the qubits, etc.) can be formulated as an SMT decision problem. We provide several examples of how an SMT formula, which evaluates to a Boolean value, can be formulated for various Clifford circuits.

In Sec. III, we apply our SMT formalism to derive fault-tolerant flag-based circuits to prepare $|H\rangle$-type magic states encoded in the color code. In particular, we discuss how v-flag circuits can be derived with the added constraint that qubits must interact via nearest neighbors with low-degree connectivity constraints. Examples of such circuits for $d = 3$ color codes are provided in Fig. 4. A clear direction of future work would be to obtain such circuits for larger code distances and to optimize the iterative solving techniques described in Sec. II C to reduce the computation time required to find a solution.

Lastly, in Sec. IV, we consider converting states encoded in the color code to states encoded in the surface code. The performance of such conversions is motivated by the fact that surface codes are much better suitable candidates for implementing algorithms via lattice surgery, while at the same time, color codes are particularly well suited for fault-tolerantly preparing magic states.

To convert color codes to surface codes, we provide a decoding algorithm for a code obtained when merging the color code with the surface code via lattice surgery, an integral part of the teleportation step. We then analyze the performance of the merged code for code-capacity noise. A direction of future work would be to extend our decoder to be compatible with lattice-surgery protocols—as has been done, for instance, in Ref. [51]—and analyze the final logical error rates of the prepared magic states under a full circuit-level noise model.

We close with some further suggested directions for designing quantum circuits with SMT solvers. We mention that there is room to optimize our encoding and solver techniques further. For example, using a variant of the Strassen algorithm [58], one could reduce the number of costly symbolic bit-matrix multiplications when constructing F_{SMT}. SMT solvers support a wide variety of strategies that should be fine tuned to obtain the best solver performance. It is also possible that a simpler standalone algorithm could supplant the use of SMT solvers for certain design problems. Lastly, in this work, we only consider the design of deterministic Clifford-like circuits, i.e., the class of unitary circuits that are equivalent to Clifford circuits up to conjugation by local unitaries. It would be interesting if our techniques could be extended beyond this restricted circuit class to nondeterministic and/or non-Clifford circuits. For circuits that are dominated by Clifford gates but that contain a relatively small number of non-Clifford operations, the algorithm of Ref. [59] could be used as a starting point for encoding in an SMT decision problem.

ACKNOWLEDGMENTS

We thank Markus Kesselring for his comments on our manuscript and for pointing out the connection between our lattice-surgery methods and the use of domain walls between the color code and surface code.

[1] T. J. Yoder and I. H. Kim, The surface code with a twist, Quantum **1**, 2 (2017).

[2] D. Litinski and F. v. Oppen, Lattice surgery with a twist: Simplifying Clifford gates of surface codes, Quantum **2**, 62 (2018).

[3] C. Chamberland, A. Kubica, T. J. Yoder, and G. Zhu, Triangular color codes on trivalent graphs with flag qubits, New J. Phys. **22**, 023019 (2020).

[4] P. Prabhu and B. W. Reichardt, in *16th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2021)*, edited by M.-H. Hsieh (Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021), Vol. 197, p. 5:1.

[5] A. M. Steane, Overhead and noise threshold of fault-tolerant quantum error correction, Phys. Rev. A **68**, 042322 (2003).

[6] E. Knill, Quantum computing with realistically noisy devices, Nature **434**, 39 (2005).

[7] P. Aliferis, D. Gottesman, and J. Preskill, Quantum accuracy threshold for concatenated distance-3 codes, Quant. Inf. Comput. **6**, 97 (2006).

[8] R. Chao and B. W. Reichardt, Quantum Error Correction with Only Two Extra Qubits, Phys. Rev. Lett. **121**, 050502 (2018).

[9] R. Chao and B. W. Reichardt, Fault-tolerant quantum computation with few qubits, npj Quantum Inf. **4**, 42 (2018).

[10] C. Chamberland and M. E. Beverland, Flag fault-tolerant error correction with arbitrary distance codes, Quantum **2**, 53 (2018).

[11] T. Tansuwannont, C. Chamberland, and D. Leung, Flag fault-tolerant error correction, measurement, and quantum computation for cyclic Calderbank-Shor-Steane codes, Phys. Rev. A **101**, 012342 (2020).

[12] B. W. Reichardt, Fault-tolerant quantum error correction for Steane's seven-qubit color code with few or no extra qubits, Quantum Sci. Technol. **6**, 015007 (2020).

[13] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, Topological and subsystem codes on low-degree graphs with flag qubits, Phys. Rev. X **10**, 011022 (2020).

[14] R. Chao and B. W. Reichardt, Flag Fault-Tolerant Error Correction for Any Stabilizer Code, PRX Quantum **1**, 010302 (2020).

[15] T. Tansuwannont and D. Leung, Fault-tolerant quantum error correction using error weight parities, Phys. Rev. A **104**, 042410 (2021).

[16] T. Tansuwannont and D. Leung, Achieving fault tolerance on capped color codes with few ancillas, arXiv e-prints, arXiv:2106.02649 (2021).

[17] C. Chamberland and A. W. Cross, Fault-tolerant magic state preparation with flag qubits, Quantum **3**, 143 (2019).

[18] C. Chamberland and K. Noh, Very low overhead fault-tolerant magic state preparation using redundant ancilla encoding and flag qubits, Npj Quantum Inf. **6**, 1 (2020).

[19] L. De Moura and N. Bjørner, in *Tools and Algorithms for the Construction and Analysis of Systems*, edited by C. R. Ramakrishnan and J. Rehof (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), p. 337.

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

[21] H. Bombín, Gauge color codes: Optimal transversal gates and gauge fixing in topological stabilizer codes, New J. Phys. **17**, 083002 (2015).

[22] A. Kubica and M. E. Beverland, Universal transversal gates with color codes: A simplified approach, Phys. Rev. A **91**, 032330 (2015).

[23] F. Thomsen, M. S. Kesselring, S. D. Bartlett, and B. J. Brown, Low-overhead quantum computing with the color code, arXiv:2201.07806 [quant-ph] (2022).

[24] A. Y. Kitaev, Fault-tolerant quantum computation by anyons, Ann. Phys. (N.Y.) **303**, 2 (2003).

[25] S. B. Bravyi and A. Y. Kitaev, Quantum codes on a lattice with boundary, arXiv e-prints arxiv:quant-ph/9811052 (1998).

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

[27] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, Phys. Rev. A **86**, 032324 (2012).

[28] J. P. B. Ataides, D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, The XZZX surface code, Nat. Commun. **12**, 1 (2021).

[29] C. Chamberland, K. Noh, P. Arrangoiz-Arriola, E. T. Campbell, C. T. Hann, J. Iverson, H. Putterman, T. C. Bohdanowicz, S. T. Flammia, A. Keller, G. Refael, J. Preskill, L. Jiang, A. H. Safavi-Naeini, O. Painter, and F. G. S. L. Brandão, Building a Fault-Tolerant Quantum Computer Using Concatenated Cat Codes, PRX Quantum **3**, 010329 (2022).

[30] For instance, in the bottom-up magic-state-preparation protocol of Refs. [17,18], a non-Clifford circuit involving controlled-Hadamard gates is viewed as a Clifford circuit suitably conjugated by T gates.

[31] C. Barrett and C. Tinelli, in *Handbook of Model Checking*, edited by E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem (Springer International Publishing, Cham, 2018), p. 305.

[32] J. Backes, S. Bayless, B. Cook, C. Dodge, A. Gacek, A. J. Hu, T. Kahsai, B. Kocik, E. Kotelnikov, J. Kukovec *et al.*, in *Computer Aided Verification*, edited by I. Dillig and S. Tasiran (Springer International Publishing, Cham, 2019), p. 231.

[33] L. De Moura and N. Bjørner, Satisfiability modulo theories: Introduction and applications, Commun. ACM **54**, 69 (2011).

[34] T. Weber, S. Conchon, D. Déharbe, M. Heizmann, A. Niemetz, and G. Reger, The SMT competition 2015-2018, J. Satisf. Boolean Model. Comput. **11**, 221 (2019).

[35] T. Balyo, M. Heule, and M. Jarvisalo, in *AAAI Conference on Artificial Intelligence* (2017), Vol. 31, https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14977/14018.

[36] B. Tan and J. Cong, in *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20 (Association for Computing Machinery, New York, 2020).

[37] J. Ding and S. Yamashita, Exact synthesis of nearest neighbor compliant quantum circuits in 2-D architecture and its application to large-scale circuits, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **39**, 1045 (2019).

[38] P. Murali, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, Formal constraint-based compilation for noisy

intermediate-scale quantum systems, Microprocess. Microsyst. **66**, 102 (2019).

[39] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Association for Computing Machinery, New York, NY, USA, 2019), p. 1015.

[40] We make this distinction since multiple faults can occure within the same stabilizer-measurement circuit.

[41] S. Bravyi and A. Kitaev, Universal quantum computation with ideal Clifford gates and noisy ancillas, Phys. Rev. A **71**, 022316 (2005).

[42] A. Paetznick and B. W. Reichardt, Universal Fault-Tolerant Quantum Computation with Only Transversal Gates and Error Correction, Phys. Rev. Lett. **111**, 090505 (2013).

[43] J. T. Anderson, G. Duclos-Cianci, and D. Poulin, Fault-Tolerant Conversion between the Steane and Reed-Muller Quantum Codes, Phys. Rev. Lett. **113**, 080501 (2014).

[44] H. Bombín, Dimensional jump in quantum error correction, New J. Phys. **18**, 043038 (2016).

[45] D. Litinski, Magic state distillation: Not as costly as you think, Quantum **3**, 205 (2019).

[46] M. E. Beverland, A. Kubica, and K. M. Svore, Cost of Universality: A Comparative Study of the Overhead of State Distillation and Code Switching with Color Codes, PRX Quantum **2**, 020341 (2021).

[47] D. Litinski, A game of surface codes: Large-scale quantum computing with lattice surgery, Quantum **3**, 128 (2019).

[48] The $H_m^{(d)}$ operation is only Clifford when the data qubits wires are conjugated by the T gates as shown in Ref. [18]. This means that $H_m^{(d)}$ is technically non-Clifford but the T-gate conjugation on the data qubits does not affect the fault-tolerance properties, so we are free to design the conjugated circuit using our techniques and then apply the T conjugation to the resulting circuit.

[49] A. Wiles, Modular elliptic curves and Fermat's last theorem, Ann. Math. **141**, 443 (1995).

[50] A. G. Fowler and C. Gidney, Low overhead quantum computation using lattice surgery, arXiv e-prints arXiv:1808.06709 [quant-ph] (2018).

[51] C. Chamberland and E. T. Campbell, Universal Quantum Computing with Twist-Free and Temporally Encoded Lattice Surgery, PRX Quantum **3**, 010331 (2022).

[52] C. Chamberland and E. T. Campbell, Circuit-level protocol and analysis for twist-based lattice surgery, Phys. Rev. Res. **4**, 023090 (2022).

[53] H. Bombin, C. Dawson, R. V. Mishmash, N. Nickerson, F. Pastawski, and S. Roberts, Logical blocks for fault-tolerant topological quantum computation, arXiv:2112.12160 [quant-ph] (2021).

[54] H. P. Nautrup, N. Friis, and H. J. Briegel, Fault-tolerant interface between quantum memories and quantum processors, Nat. Commun. **8**, 1 (2017).

[55] M. Vasmer and A. Kubica, Morphing quantum codes, arXiv preprint arXiv:2112.01446 (2021).

[56] A. Kubica and N. Delfosse, Efficient color code decoders in $d \geq 2$ dimensions from toric code decoders, arXiv e-prints arXiv:1905.07393 [quant-ph] (2019).

[57] J. Edmonds, Paths, trees, and flowers, Can. J. Math. **17**, 449 (1965).

[58] V. Strassen, Gaussian elimination is not optimal, Numerische Mathematik **13**, 354 (1969).

[59] S. Bravyi and D. Gosset, Improved Classical Simulation of Quantum Circuits Dominated by Clifford Gates, Phys. Rev. Lett. **116**, 250501 (2016).