Surface Code Compilation via Edge-Disjoint Paths

Michael Beverland,[1] Vadym Kliuchnikov,[1] and Eddie Schoute[2,3,4,*]

[1] *Microsoft Quantum*
[2] *Joint Center for Quantum Information and Computer Science, University of Maryland*
[3] *Institute for Advanced Computer Studies, University of Maryland*
[4] *Department of Computer Science, University of Maryland*

(Received 31 December 2021; accepted 16 February 2022; published 25 May 2022)

We provide an efficient algorithm to compile quantum circuits for fault-tolerant execution. We target surface codes, which form a two-dimensional grid of logical qubits with nearest-neighbor logical operations. Embedding an input circuit's qubits in surface codes can result in long-range two-qubit operations across the grid. We show how to prepare many long-range Bell pairs on qubits connected by edge-disjoint paths of ancillae in constant depth that can be used to perform these long-range operations. This forms one core part of our *edge-disjoint path compilation* (EDPC) algorithm, by easily performing many parallel long-range Clifford operations in constant depth. It also allows us to establish a connection between surface code compilation and several well-studied edge-disjoint path problems. Similar techniques allow us to perform non-Clifford single-qubit rotations far from magic state distillation factories. In this case, we can easily find the maximum set of paths by a max-flow reduction, which forms the other major part of EDPC. EDPC has the best asymptotic worst-case performance guarantees on the circuit depth for compiling parallel operations when compared to related compilation methods based on SWAP gates and network coding. EDPC also shows a quadratic depth improvement over sequential Pauli-based compilation for parallel rotations requiring magic resources. We implement EDPC and find significantly improved performance for circuits built from parallel controlled-NOT (CNOT) gates, and for circuits that implement the multicontrolled X gate C^{k}NOT.

DOI: 10.1103/PRXQuantum.3.020342

I. INTRODUCTION

Quantum hardware will always be somewhat faulty and subject to decoherence, due to inevitable fabrication imperfections and the impossibility of completely isolating physical systems. For large computations, it becomes a certainty that faults will occur among the many qubits and operations involved. *Fault-tolerant quantum computation* (FTQC) can be implemented despite this by encoding the information in a quantum error correcting code and applying logical operations that are carefully designed to process the encoded information with an acceptably low effective error rate.

The surface code [1,2] provides a promising approach to implement FTQC. Firstly, it can be implemented using geometrically local operations on a patch of qubits in a two-dimensional (2D) grid, which is the natural setting for many hardware platforms, including superconducting [3,4] and Majorana [5] qubits. Secondly, the logical qubits it encodes remain protected even for relatively high noise rates, with a threshold of around 1% [6]. Thirdly, a sufficiently general set of elementary logical operations can be performed fault tolerantly on qubits encoded in the surface code using *lattice surgery* [7]. By tiling the plane with surface code patches, a 2D grid of logical qubits is formed, where the elementary operations are geometrically local; see Fig. 1. When combined with magic state distillation [8], these operations become universal for quantum computing. Indeed, this approach, which we refer to as the *surface code architecture*, is seen as among the most promising by many research groups and companies working in quantum computing [3,9–11].

In this work, we seek to minimize the resources required to fault tolerantly implement a quantum algorithm using the surface code architecture, which we refer to as the *surface code compilation problem*. For concreteness, we assume that the input quantum algorithm is expressed as a quantum circuit composed of preparations and destructive measurements of individual qubits in the Z or X

[*] eschoute@lanl.gov

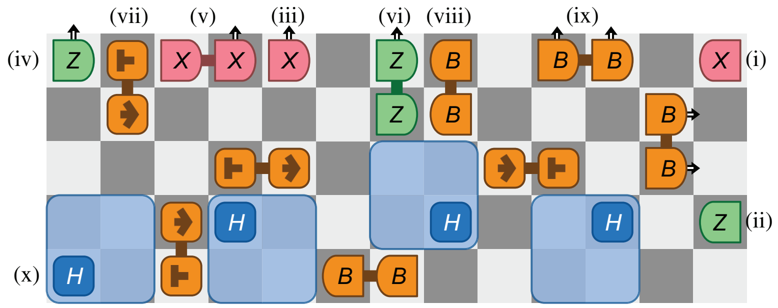


FIG. 1. Logical qubits (light and dark gray patches) encoded in the surface code form a 2D grid. The elementary operations can be applied on any lattice translations of those shown. Their times in units of surface code logical time steps are as follows. *0 logical time steps:* single-qubit preparation in the X basis (i) and the Z basis (ii). Destructive single-qubit measurement, which moves the patch outside of the code space, in the X basis (iii) and the Z basis (iv) take 0 steps. *1 logical time step:* two-qubit measurement of XX (v) and ZZ (vi). A move of a logical qubit from one patch to an unused patch (vii). Two-qubit preparation (viii) and destructive measurement (ix) in the Bell basis. *3 logical time steps:* a Hadamard gate, which uses three ancilla patches (x). See Appendix A for further details.

basis, controlled-NOT (CNOT), Pauli-X, Pauli-Y, Pauli-Z, Hadamard (H), phase (S), and T gates. Our results can be easily generalized to broader classes of input quantum circuits. The output is the quantum algorithm executed using the elementary logical surface code operations shown in Fig. 1. Ultimately, we would like to minimize the *physical space-time cost*, which is the product of the number of physical qubits and the time required to run an algorithm. To avoid implementation details, we instead minimize the more abstract *logical space-time cost*, which is the number of logical qubits (the circuit width) multiplied by the number of logical time steps (the circuit depth) of the algorithm expressed in elementary surface code operations. The logical and physical space-time costs are expected to be 1-to-1 and monotonically related (see Appendix B), such that minimizing the former should minimize the latter.

A well-established approach to implement surface code compilation is known as sequential Pauli-based computation [13], where non-Clifford operations are implemented by injection using Pauli measurements, and Clifford operations are conjugated through the circuit until the end. The circuit that is run in this approach then consists of a sequence of high-weight Pauli measurements that can have overlapping support, leading them to be measured one after the other. For large input circuits, this can be problematic because highly parallel input circuits can become serialized with prohibitive runtimes.

A major challenge to solve the surface code compilation problem is that quantum algorithms typically involve operations between logical qubits that are far apart when laid out in a 2D grid. One approach to deal with a long-range gate is to swap logical qubits around until the pair of interacting qubits are next to one another [14]. However, this can result in a deep circuit; see Fig. 2(a). A more efficient approach is to create long-range entanglement by producing Bell pairs, which, for example, can be used to implement a long-range CNOT gate with a constant-depth circuit [12,15,16] [see Fig. 2(b)]. Both of these approaches

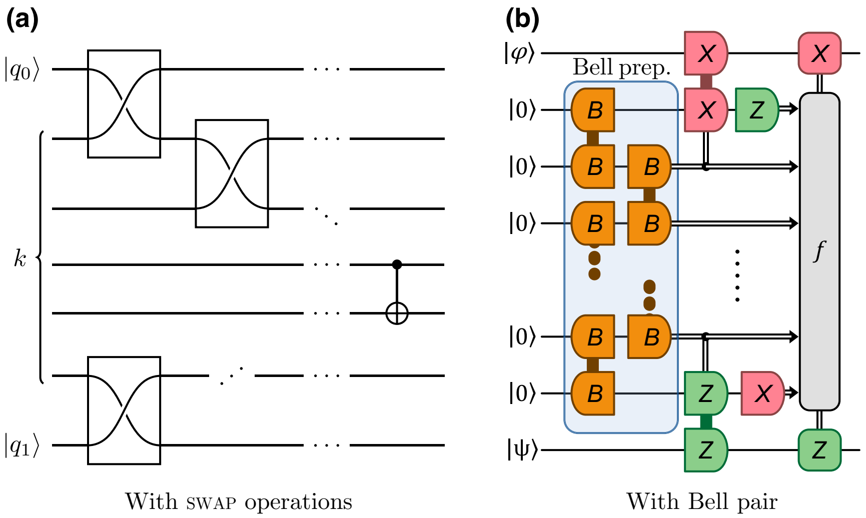


FIG. 2. Application of a CNOT(q_0, q_1) gate on qubits at distance $k+1$ using surface code operations in two ways. (a) Using a SWAP-based approach requires $\Omega(n)$ depth using operations from Fig. 1, while (b) generating and consuming a Bell pair [12] can be implemented in constant depth. The classical function f computes Pauli corrections on the output qubits.

can be implemented with the elementary operations of the surface code.

Moreover, algorithms typically consist of many long-range operations that can ideally be performed in parallel. For swap-based approaches, this can be done by considering a permutation of the logical qubits that is implemented by a sequence of SWAP gates [17–19]. Finding these SWAP circuits reduces to a routing problem on graphs [14,20]. There are efficient algorithms that solve this problem for certain families of graphs [14,21], but finding a minimal depth solution is NP-hard in general [22]. Alternatively, *linear network coding* can be used to prepare many long-range pairs in constant depth [23–28], and then these Bell pairs can be used to implement operations on pairs of distant qubits. But a major barrier for using linear network coding is the lack of known efficient algorithms to find linear network codes.

In this paper, we provide a solution to the surface code compilation problem that generalizes the use of entanglement for long-range CNOT gates discussed above to the implementation of many long-range operations in parallel. In particular, we propose the edge-disjoint path compilation (EDPC) algorithm, which is a computationally efficient classical algorithm tailored to the elementary operations of the surface code architecture. We find evidence that our EDPC algorithm significantly outperforms other approaches by performing a detailed cost analysis for the execution of a set of quantum circuit benchmarks.

EDPC reduces the problem of executing quantum circuits to problems in graph theory. Logical qubits correspond to graph vertices, and there is an edge between qubits if elementary surface code operations can be applied between them. We show how to perform multiple long-range CNOT gates in constant depth along a set of edge-disjoint paths in the graph. In other words, long-range CNOT gates can be performed simultaneously, in one round, if their controls and targets are connected by edge-disjoint paths. This leads to the well-studied problem of finding maximum edge-disjoint path sets [29]. The ability to perform long-range CNOT gates along with the elementary operations allows compilation of Clifford operations. We also give a construction for edge-disjoint path sets that are asymptotically optimal in the depth of worst-case sets of independent CNOT gates.

The final operations that complete our gate set for universal quantum computation with the surface code are T gates. The T gates are not natural operations on the surface code, but can be implemented fault tolerantly by consuming specialized resource states, called *magic states*. Magic states can be produced using a highly optimized process called magic state distillation, which we assume occurs independently of the computation on our code. We assume that logical magic states are available in a specified region of the grid. EDPC reduces magic state delivery to simple MAX FLOW instances that have known efficient algorithm [30]. We compare the depth of input circuits compiled using surface code compilation algorithms in the literature and EDPC in Table I.

TABLE I. A comparison in the depth of surface code compilation algorithms [that use $\Theta(n)$ space] for various input circuits of width n. We compare the worst-case performance for a single long-range CNOT gate, for CNOT circuits with $n/2$ parallel CNOT gates, and for k rotations with $k \in \mathbb{N}$ that need to be performed at the boundary.

	Input circuit (compiled depth)		
Algorithm	One CNOT gate	$n/2$ parallel CNOT gates	k parallel rotations
Sequential Pauli	0	0	$\Theta(k)$
SWAP	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$
Network coding	$\Theta(1)$	$\Omega(\sqrt{n})$	$\Omega(\sqrt{k})$
EDPC	$\mathbf{\Theta(1)}$	$\mathbf{\Theta(\sqrt{n})}$	$\mathbf{\Theta(\sqrt{k})}$

The outline of the paper is as follows. In Sec. II, we construct key higher-level components from the basic surface code operations in Fig. 1, including simple long-range operations. These long-range operations allow us to perform many parallel CNOT operations given vertex-disjoint and edge-disjoint paths that connect the data qubits in Sec. III. Because of its importance to the algorithms, there we also compare the state-of-the-art graph algorithms for finding vertex-disjoint or edge-disjoint sets of paths and analyze their relation to our algorithms. We complete our gate set by giving an algorithm for efficient remote rotations using magic states at the boundary in Sec. IV. Putting parallel long-range CNOT and remote rotations together, we construct our circuit compilation algorithm, EDPC, in Sec. V. Finally, we compare the performance of EDPC to prior surface code compilation work in Sec. VI, note its connections to network coding, and give numerical results comparing the space-time performance with a SWAP-based compilation algorithm.

II. KEY CIRCUIT COMPONENTS FROM SURFACE CODE OPERATIONS

Recall that our goal in this work is to develop an efficient classical compilation algorithm that reexpresses a quantum algorithm into one that uses the elementary operations of the surface code with a low logical space-time cost. In Appendix A we give an overview of the surface code and justify the resource costs of the elementary operations shown in Fig. 1. The initial quantum algorithm is assumed to be expressed as a circuit diagram involving preparations and measurements of individual qubits in the computational basis, controlled-not (CNOT), Pauli-X, Pauli-Y, Pauli-Z, Hadamard (H), phase (S), and T gates. In this section we build and calculate the cost of some key circuit components from the elementary surface code operations in Fig. 1. The contents of this section are

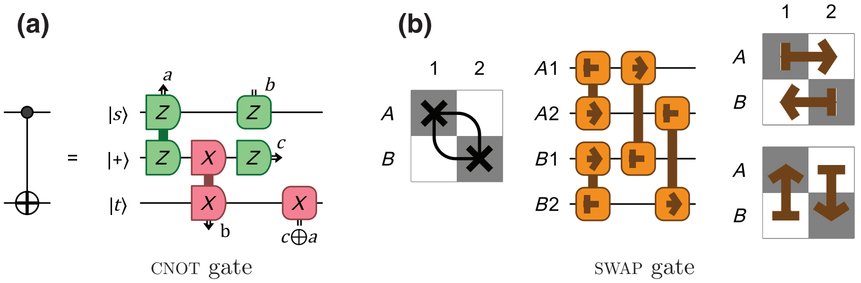


FIG. 3. A CNOT gate can be implemented in depth 2 using ZZ and XX joint measurements with a $|+\rangle$ ancilla state, followed by classically controlled Pauli corrections. The SWAP gate can be implemented using four move operations and two ancillae in depth 2.

reproductions or straightforward extensions of previously known circuits.

A. Single-qubit operations

Some of the operations of the input circuit can be implemented directly with elementary surface code operations, namely the preparation and measurement of individual qubits in the measurement basis, and the Hadamard gate (provided three neighboring ancillary patches are available as ancillae; see Fig. 1). Pauli operations do not need to be implemented at all since they can be commuted through Clifford gates and arbitrary Pauli gates [31] and can therefore be tracked classically and merged with the final measurements. For this reason, while we occasionally explicitly provide the Pauli corrections where instructive, we often show equivalence of two circuits only up to Pauli corrections. The remaining single-qubit operations in the input circuit, namely the S and T gates, can be implemented using magic states and are addressed in Sec. IV.

B. Local CNOT and SWAP gates

An important circuit component is the CNOT gate, which can be implemented as shown in Fig. 3(a) [32]. The qubits involved in this example are stored in adjacent patches, i.e., it is local. Another useful operation is a SWAP of a pair of qubits stored in nearby patches. The surface code's move operation shown in Fig. 1 gives a straightforward way to implement this, as shown in Fig. 3(b). With these implementations, the CNOT gate requires one ancilla patch, while a SWAP gate requires two. Both are depth 2.

C. Long-range CNOT gates using SWAP gates

Typical input circuits for surface code compilation will involve CNOT operations on pairs of qubits that are far apart after layout. A very intuitive approach to apply a long-range CNOT(q_1, q_2) gate is shown in Fig. 4. This involves making use of SWAP gates to first move the qubits q_1 and q_2 so that they are near one another, and then use the local CNOT gate in Fig. 3(a). Let the path $P = v_1 v_2 \cdots v_k$ for $k \in \mathbb{N}$, where $v_1 = q_1$ and $v_k = q_2$. As each SWAP gate has depth 2, we get a circuit of depth $2\lceil (k-1)/2 \rceil$ since we can perform SWAP gates on either end simultaneously. Afterwards, the two qubits are adjacent and we simply perform a CNOT gate in depth 2.

A lower bound on the depth it takes to perform a long-range CNOT gate using SWAP gates is proportional to the length of the shortest q_1-q_2 path. To move a qubit k patches using a SWAP gate takes depth exactly $2k$. Therefore, to move the control and target to the middle of the shortest path connecting them, it must take time proportional to at least half the length of the path.

D. Long-range CNOT gate using a Bell pair

A circuit component that we make extensive use of in this paper is the long-range CNOT using a Bell pair [12]. This allows us to apply CNOT gates in depth 2 between any pair of qubits (provided there is a path of ancilla qubits that connects them).

To understand the construction, we first show in Fig. 5(a) how to prepare a longer-range Bell pair from two Bell pairs. By iterating this construction one can form a circuit to prepare a long-range Bell pair at the ends of any path of adjacent ancilla patches in depth 2. Next, we show

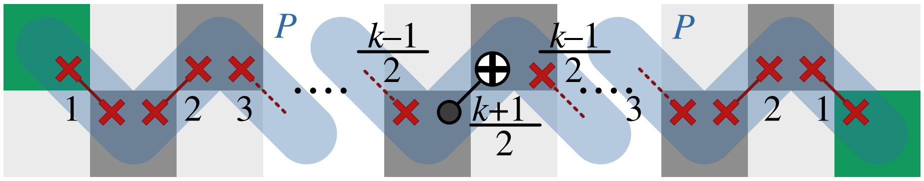
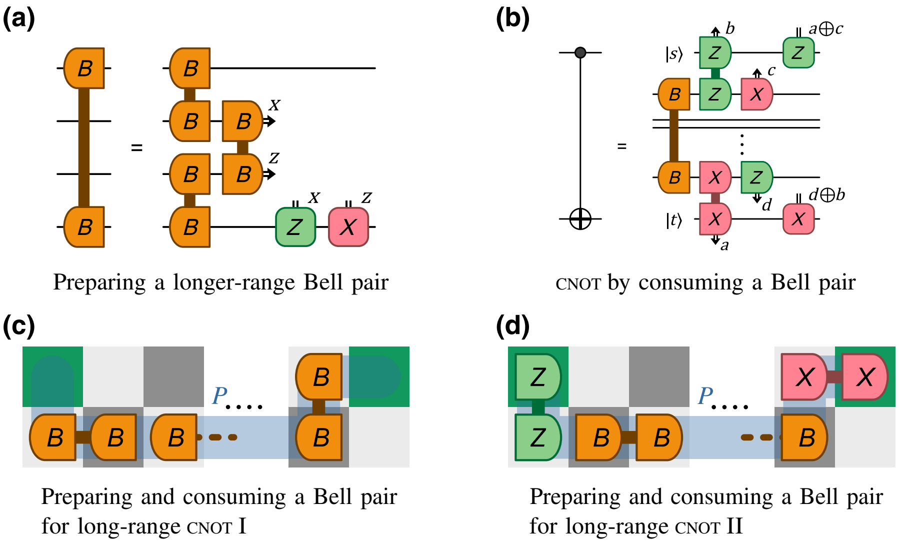


FIG. 4. A nonlocal CNOT can be implemented using SWAP gates that take depth $2\lceil (k-1)/2 \rceil$ using a zigzag of ancilla patches along the path P of length k. The figure shows the case when k is odd and the depth of the SWAP circuit is $2(k-1)$. The patches on the path can store other logical information, which will simply be moved during the SWAP gates. The patches adjacent to the path are ancillae that are used to implement the SWAP gates.



Preparing a longer-range Bell pair

CNOT by consuming a Bell pair

Preparing and consuming a Bell pair for long-range CNOT I

Preparing and consuming a Bell pair for long-range CNOT II

FIG. 5. A long-range CNOT can be implemented in depth 2 by first preparing a Bell pair. (a) Joining Bell pairs with Bell measurements. This can be iterated to form a long-range Bell pair along any path of ancillae in depth 2. (b) A Bell pair can be used to apply a CNOT. (c),(d) The first and second steps of a depth-2 circuit that implements a CNOT between a pair of patches at the end of a path of ancilla patches by preparing and consuming a Bell pair.

in Fig. 5(b) how to implement a CNOT operation between qubits stored in patches neighboring a pair of patches storing a Bell pair. Putting these together, using a path of ancilla patches between a pair of qubits, a long-range CNOT gate can be implemented in depth 2 in a two-step circuit shown in Figs. 5(c) and 5(d), respectively. This approach can be used to implement the CNOT in a depth-2 circuit using any path from the control to the target qubit that starts with a vertical edge and ends with a horizontal edge. There is also flexibility in the precise arrangement of the Bell pairs and Bell measurements along the path using the circuits in Appendix C.

Note that here we have focused on implementing a long-range CNOT gate by constructing and consuming a Bell pair. However, a similar strategy (of first preparing a long-range Bell pair in the patches at the ends of a path of ancillae) can be used to implement other long-range operations, such as teleportation.

III. PARALLEL LONG-RANGE CNOT GATES USING BELL PAIRS

Here, we generalize the use of Bell pairs from the setting of compiling an individual nonlocal CNOT gate into surface code operations to the setting in which a set of parallel nonlocal CNOT gates are compiled. In Fig. 1 and the circuit components in Sec. II, ancilla qubits are used to perform some operations on data qubits. To consider the compilation on large sets of qubits, we must specify the location of data and ancilla qubits: here we assume a 1 data to 3 ancilla qubit ratio, as illustrated in Fig. 6.

In Sec. III A we discuss some relevant background on sets of vertex-disjoint paths (VDPs) and sets of edge-disjoint paths (EDPs) in graphs. Then in Sec. III B we define the *VDP subroutine* and the *EDP subroutine* that apply parallel CNOT gates at the ends of a particular type of VDP or EDP set. In Sec. III C, we show how to use the EDP subroutine to compile more general CNOT circuits and prove bounds on the performance of this approach.

A. Vertex-disjoint paths and edge-disjoint paths

In Sec. II D we saw that a long-range CNOT gate could be implemented with the use of a Bell pair produced with a path of ancilla qubits connecting the control and target of the CNOT gate. A barrier to implement multiple CNOT gates simultaneously can arise when an ancilla resides in the paths associated with multiple different CNOT gates. This motivates us to review some relevant theoretical background concerning sets of paths on graphs.

Given a graph G, a set of paths $\mathcal{P}$ is said to be a *VDP* set if no pair of paths in $\mathcal{P}$ share a vertex, and an *EDP* set if no pair of paths in $\mathcal{P}$ share an edge. Note that a set of vertex-disjoint paths is also edge disjoint. Further consider a set of *terminal pairs* $\mathcal{T} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ for terminals $s_i, t_i \in V(G)$, the vertices of G, and $i \in [k]$. We then say that a set of paths $\mathcal{P}$ is a *VDP set for* $\mathcal{T}$ (respectively an *EDP set for* $\mathcal{T}$) if $\mathcal{P}$ is a VDP set (respectively an EDP set), and each path in $\mathcal{P}$ connects a distinct pair in $\mathcal{T}$. These path sets do not necessarily connect all pairs in $\mathcal{T}$. In what follows, we pay special attention to the square grid graph [see Fig. 7(a)]. The grid graph is relevant for qubits in the surface code as shown in Fig. 1, where the vertices correspond to code patches and edges connect vertices associated with adjacent patches [33].

The problems of finding a maximum (cardinality) VDP set for $\mathcal{T}$ or a maximum EDP set for $\mathcal{T}$ have been well studied and there are known efficient algorithms capable of finding approximate solutions to each. Unfortunately, on grids it is particularly hard to approximate the maximum VDP set. In particular, for $N := |V(G)|$, there exist terminal sets for which no efficient algorithm can find an approximate solution to within a $2^{O(\log^{1-\epsilon} N)}$ factor of the maximum set size for any $\epsilon > 0$, unless

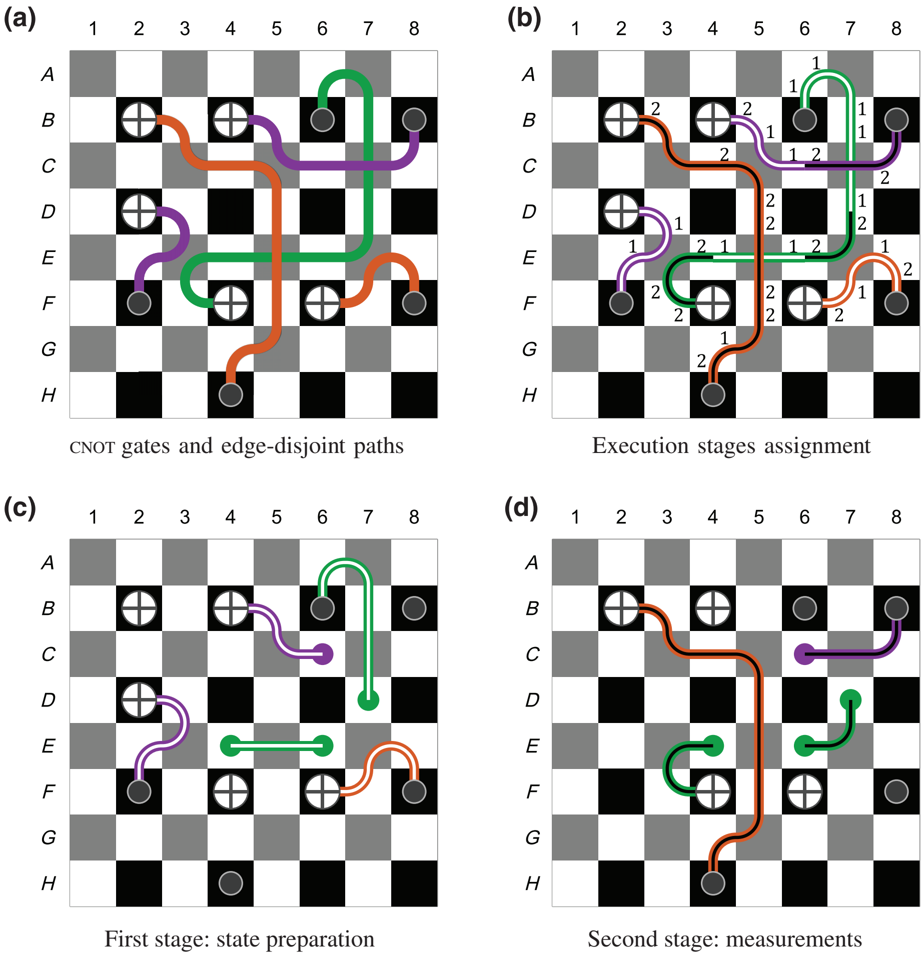

(a)
1 2 3 4 5 6 7 8
A B C D E F G H
CNOT gates and edge-disjoint paths
(b)
Execution stages assignment
(c)
First stage: state preparation
(d)
Second stage: measurements

FIG. 6. The EDP subroutine implements a set of parallel CNOT gates connected by an operator EDP set. We assume a qubit ratio of 1 to 3 of data (black) to ancilla (gray and white). (a) The input to the EDP subroutine is a set of CNOTs and an associated EDP set. (b) We fragment the EDP set into two VDP sets consisting of segments of the original paths, and implement the compiled circuit over two depth-2 stages, one for each of these sets. (c) During the first stage we prepare a Bell pair between the ends of the segments in the first VDP set. (d) During the second stage we perform joint Bell measurements between the ends of segments in the second VDP set, producing long-range Bell pairs on ancillae adjacent to the control and target of each CNOT. Then, long-range CNOTs can easily be applied by using the long-range Bell pairs (Sec. II D). See Figs. 8 and 9 for further details of the long-range operations used here.

$\mathrm{NP} \subseteq \mathrm{RTIME}(N^{\mathrm{poly}\log N})$ [34]. However, efficient algorithms are available if one is willing to accept a looser approximation to the optimal solution. For example, a simple greedy algorithm is an $O(\sqrt{N})$-approximation algorithm for finding the maximum VDP set [35,36], i.e., it produces a VDP set to within an $O(\sqrt{N})$ multiplicative factor of the optimal solution for any graph, not just the grid. For grids, the best efficient algorithm that is known is an $\tilde{O}(N^{1/4})$-approximation algorithm [37], where $\tilde{O}(\cdot)$ hides logarithmic factors of $O(\cdot)$.

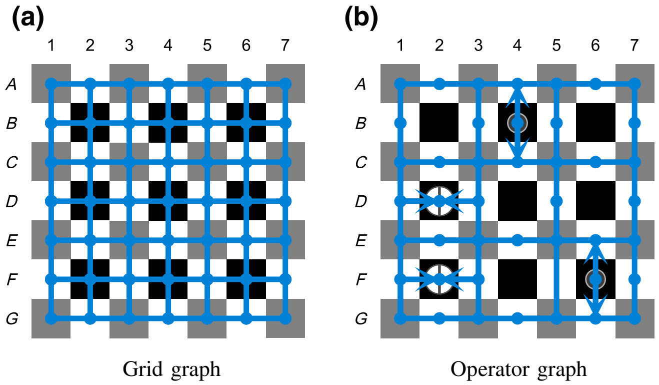


FIG. 7. The graphs used in this paper. (a) The grid graph where each surface code patch corresponds to a vertex and is connected to its neighbors. (b) The operator graph for a set of terminal pairs $\mathcal{T}$ that correspond with a parallel CNOT circuit. EDP sets for $\mathcal{T}$ on this graph are also operator EDP sets.

The situation is better for approximation algorithms of the maximum EDP set: there is a $\Theta(\sqrt{N})$-approximation algorithm [38] for any graph, and on grids Aumann and Rabani [39] showed an $O(\log N)$-approximation algorithm that was later improved to an $O(1)$-approximation algorithm [29,40]. In practice, these algorithms can be technical to implement and can have large constant prefactors in their solutions that can be prohibitive for the instance sizes that we consider. A simple greedy algorithm forms an $O(\sqrt{N})$-approximation algorithm [35] for finding a maximum EDP set on the two-dimensional grid and does not suffer from the constant prefactors of the asymptotically superior alternatives. The dominant runtime complexity of this greedy algorithm is mainly in finding shortest paths for each terminal pair, giving an $O(|\mathcal{T}|N\log N)$ runtime upper bound by Dijkstra's algorithm [41].

It is informative to consider the comparative size of the maximum EDP and VDP sets for the same terminal set $\mathcal{T}$. Since any VDP set is also an EDP set, the size of the maximum VDP set for $\mathcal{T}$ cannot be larger than the maximum EDP set for $\mathcal{T}$. Moreover, one can construct some cases of $\mathcal{T}$ on the grid [29] in which the maximum EDP set is a factor $\sqrt{N}$ larger than the maximum VDP set [29]. For example, consider the set of terminal pairs $\mathcal{T} = \{[(i,1),(L,i)] \mid i \in [L]\}$ of an $L \times L$ grid graph, where vertex (i,j) denotes the vertex in row i and column j. All terminals can be connected by edge-disjoint paths but the maximum VDP set is of size one.

In Sec. III B, we show that both VDP and EDP sets for $\mathcal{T}$ can be used to form constant-depth compilation subroutines for disjoint CNOT circuits. Ultimately, as will become clear in Sec. III B, each path in the EDP or VDP sets for $\mathcal{T}$ allows us to implement one more CNOT gate in parallel by a compilation subroutine. In this work, we focus on EDPs rather than VDPs for two main reasons. Firstly, as mentioned above, better approximation algorithms exist for finding maximum EDP sets than for finding maximum VDP sets on the grid. Although, in practice, we make use of the greedy $O(\sqrt{N})$-approximation algorithm for finding maximum EDP sets in this work. Secondly, as was also mentioned above, the maximum EDP set is at least as large as the maximum VDP set.

An important open problem that could ultimately influence the performance of the surface code compilation algorithm we present in this work is whether an alternative approximation algorithm for finding maximum EDP sets can be used that performs better in practical instances.

B. Long-range CNOT subroutines using VDPs and EDPs

Here we present one of our main technical contributions, namely a description of how to implement a set of long-range CNOT gates at the end of VDP and EDP sets using surface code operations. This is central to our overall surface code compilation algorithm presented in Sec. V.

Consider the $L \times L$ square grid graph G [see Fig. 7(a)], which consists of vertices $V(G) = [L] \times [L]$ for $[L] := \{1, \dots, L\}$ and undirected edges

$$E(G) = \{[(i,j),(i,j+1)] \mid i \in [L], j \in [L-1]\} \\ \cup \{[(i,j),(i+1,j)] \mid i \in [L-1], j \in [L]\}. \quad (1)$$

Here, vertices correspond to qubits stored in surface code patches, and edges connect qubits on adjacent patches (see Fig. 1). We color the vertices of G with three colors: black, gray, and white (see Fig. 6). All vertices with both even row and even column indexes are colored black and correspond to data qubits (where data qubits correspond to qubits in the input circuit). The vertices (corresponding to ancilla qubits) with both odd row and odd column indexes are colored white, and all remaining vertices are colored gray. This gives us a 1 : 3 data qubit to ancilla qubit ratio. We set n to equal the number of black vertices, i.e., the number of data qubits.

Because of the designation of some vertices as data qubits and others as ancilla vertices in our layout, and due to the asymmetry of two-qubit operations along horizontal and vertical edges in Fig. 1, we add some restrictions to the paths we consider. We define an *operator path* to be a path $P = v_1 v_2 \cdots v_k$ for $k \in \mathbb{N}$ such that v_1 and v_k correspond to data qubits and its *interior* $v_2 \cdots v_{k-1}$ are all ancilla qubits. Moreover, v_1 to v_2 must be a vertical edge, and v_{k-1} to v_k must be a horizontal edge. Then an *operator VDP (respectively EDP) set* is a set of vertex-disjoint (respectively edge-disjoint) operator paths. In addition, we require that the ends of the paths in the operator EDP set do not overlap. With the coloring assignments of the grid graph G, it is easy to see that the first and last vertexes of an operator path are colored black. In what follows, we show how we can implement CNOT gates between the data qubits at the ends of the paths in an operator VDP (EDP) set in constant depth.

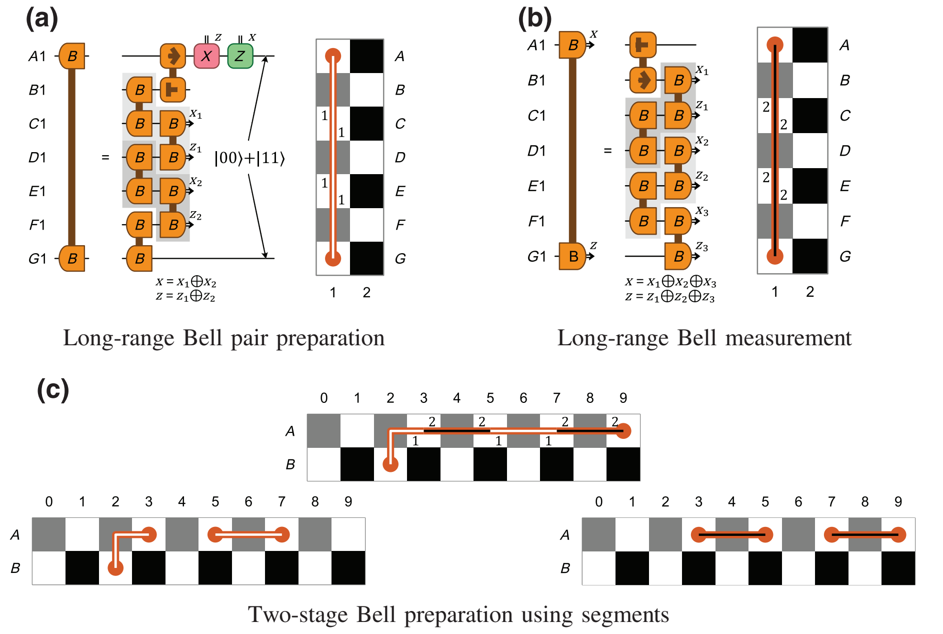
First consider an operator VDP set $\mathcal{P}$. It is straightforward to see that we can simultaneously apply long-range CNOT gates along each $P \in \mathcal{P}$ as in Fig. 5 in depth 2. We call this the *VDP subroutine*.

Now consider an operator EDP set $\mathcal{P}$. An EDP set can have intersecting paths, and the ancilla qubits at intersections appear in multiple paths, preventing us from simultaneously producing Bell pairs at their ends. We circumvent this by producing Bell pairs across a path in two stages by splitting the path into segments; see Fig. 8. We show that $\mathcal{P}$ can be *fragmented* into two VDP sets $\mathcal{P}_1$ and $\mathcal{P}_2$ that, together, form $\mathcal{P}$. More precisely, each path $P \in \mathcal{P}$ can be built by composing paths contained in $\mathcal{P}_1$ and $\mathcal{P}_2$ such that each path in either $\mathcal{P}_1$ or $\mathcal{P}_2$ appears in precisely one path in $\mathcal{P}$. We say that the paths in $\mathcal{P}_1$ and $\mathcal{P}_2$ are *segments* of paths in $\mathcal{P}$. This forms the basis of the *EDP subroutine*, which is presented in Algorithm 1 and illustrated with an example in Fig. 6.

We show the following lemma, which restricts the adjacency of *crossing* vertices. As will become clear later, the adjacent crossing vertices impose systems of constraints on fragmenting $\mathcal{P}$, and their restricted adjacency of any operator EDP set ensures that a fragmentation into two VDP sets always exists.

Lemma 1: *Given an operator EDP set $\mathcal{P}$, a crossing vertex is a vertex contained in more than one path in $\mathcal{P}$. Let the set of crossing vertices be V_c. Then the induced subgraph $G[V_c]$ contains only three kinds of connected components:*

1. *isolated vertices;*
2. *a horizontal path, where each vertex (i,j) in the connected component can only be adjacent to $(i-1,j)$ and $(i+1,j)$;*



Long-range Bell pair preparation

Long-range Bell measurement

Two-stage Bell preparation using segments

FIG. 8. (a) For segments marked in white, we use long-range Bell pair preparation in depth 2. (b) For segments marked in black, we then use long-range Bell pair measurement in depth 2. (c) The Bell measurements in stage 2 stitch together the Bell pairs made in phase 1, resulting in a Bell pair in the qubits at the ends of the full path.

3. *a vertical path, where each vertex (i,j) in the connected component can only be adjacent to $(i,j-1)$ and $(i,j+1)$.*

Proof. We consider all possible colors of a vertex (i,j) in a connected component of $G[V_c]$. Black vertices cannot be crossing vertices by the definition of an operator EDP set so cannot be contained in V_c. It is then easy to see that white vertices in V_c satisfy the lemma.

Therefore, the only relevant case is when (i,j) is a gray vertex. The vertices $(i+1,j)$ and $(i,j+1)$ are white and $(i+1,j+1)$ is black. We show that these white vertices cannot both be crossing vertices. Suppose that they are; then both edges between the white vertices and the black vertex, $[(i+1,j),(i+1,j+1)]$ and $[(i,j+1),(i+1,j+1)]$, are in $\mathcal{P}$. This is a contradiction with the fact that the interior of operator EDPs cannot contain a black vertex so it must be at the end of two paths, but an operator EDP set cannot contain two paths ending at the same vertex. By the same argument applied to the other white neighbors of (i,j) we see that only $(i-1,j)$ and $(i+1,j)$ or $(i,j-1)$ and $(i,j+1)$ can both be crossing vertices, and the claim follows. ■

We now prove that $\mathcal{P}$ can be fragmented.

Theorem 2: *We can fragment an operator EDP set $\mathcal{P}$ to produce vertex-disjoint sets of segments $\mathcal{P}_1$ and $\mathcal{P}_2$. If $\mathcal{P}$ is vertex disjoint then $\mathcal{P}_1 = \mathcal{P}$ and $\mathcal{P}_2 = \emptyset$.*

Proof. We assign edges for inclusion in segments in $\mathcal{P}_1$ or $\mathcal{P}_2$ by an edge labeling $l(e): E(G) \to \{1,2\}$. Given a labeling of all edges e in the paths of $\mathcal{P}$, we can assign edges $l(e) = b$ to segments in $\mathcal{P}_b$. Therefore, given a labeling of

```
Input : An operator EDP set $\mathcal{P}$
$\mathcal{P}_1, \mathcal{P}_2 \leftarrow$ fragment $\mathcal{P}$ in two VDP sets of segments    // Theorem 2
for segment $P \in \mathcal{P}_1$ :
    if $P$ connects two data qubits then
        execute long-range CNOT gate along $P$
    else
        execute phase 1 operation along $P$ [Fig. 8(a), or 9(b), or 9(c)]
for segment $P \in \mathcal{P}_2$ :
    if $P$ connects two data qubits then
        execute long-range CNOT gate along $P$
    else
        execute phase 2 operation along $P$ [Fig. 8(b), or 9(d), or 9(e)]
```

Algorithm 1. *EDP subroutine*: to apply CNOTs to the data qubits at the endpoints of a set of edge-disjoint paths $\mathcal{P}$, where the interior of each path is supported on ancilla qubits. The depth is at most 4.

all edges in paths in $\mathcal{P}$, it is easy to construct $\mathcal{P}_1$ and $\mathcal{P}_2$. We now label all edges in the paths in $\mathcal{P}$ and prove that their labeling guarantees the vertex-disjointness property of $\mathcal{P}_1$ and $\mathcal{P}_2$.

We constrain the labeling around every crossing vertex v so that the VDP property is satisfied. Clearly, v is contained in the interior of exactly two paths, P_1 and P_2. Let v be contained in edges e_1 and e_1' of P_1, and edges e_2 and e_2' of P_2; then we impose the constraints

$$l(e_1) = l(e_1'), \tag{2}$$

$$l(e_2) = l(e_2'), \tag{3}$$

$$l(e_1) \neq l(e_2), \tag{4}$$

guaranteeing the vertex disjointness of segments at v since a segment of P_1 must span both e_1 and e_1', and a segment of P_2 must span both e_2 and e_2' with a different label.

We show there always exists a feasible solution given these constraints. If we consider the graph $G[V_c]$ induced by crossing vertices V_c then we see that every connected component in $G[V_c]$ gives a system of constraints. The adjacency of $G[V_c]$, by Lemma 1, is such that each system has one degree of freedom, which we decide arbitrarily.

Finally, for every vertex-disjoint path $P \in \mathcal{P}$, assign $l(e) = 1$ to all edges e in P. All remaining edges can be labeled arbitrarily. ■

The depth of a CNOT circuit produced by the EDP subroutine for an operator EDP set $\mathcal{P}$ is at most 4. If $\mathcal{P}$ happens to be vertex disjoint then the depth is 2 since all paths are assigned to phase 1 by Theorem 2.

C. Compiling parallel CNOT circuits with the EDP subroutine

In this section we consider how to compile input parallel CNOT circuits using the EDP subroutine. We define the terminal pairs $\mathcal{T} \subseteq V(G) \times V(G)$ to be the pairs of control and target qubits for each CNOT gate in the parallel CNOT circuit. To use the EDP subroutine, we need to find operator EDP sets $\mathcal{P}_1, \dots, \mathcal{P}_k$ that connect all terminal pairs in $\mathcal{T}$. We refer to any such set $\{\mathcal{P}_1, \dots, \mathcal{P}_k\}$ as a *$\mathcal{T}$-operator set*. The depth of the compiled implementation is minimized when the size k of the $\mathcal{T}$-operator set is minimized.

There are reasons to believe that the compilation strategy for parallel CNOT circuits formed by finding a minimal $\mathcal{T}$-operator set and applying the EDP subroutine should produce low-depth output circuits. For sparse input circuits, i.e., those with a small number of CNOTs, one can expect a small $\mathcal{T}$-operator set to exist, giving a low-depth output. On the other hand, we now prove that there are dense CNOT circuits for which the EDP subroutine with a minimal size $\mathcal{T}$-operator set produces a compiled circuit with optimal depth (up to a constant multiplicative factor).

Theorem 3: *Let a parallel input CNOT circuit with corresponding terminal pairs $\mathcal{T}$ be given, and let the n qubits of the input circuit be embedded in a grid among $3n$ ancilla qubits according to the layout in Fig. 6. For simplicity, we assume that n is both even and the square of an integer. We can find a $\mathcal{T}$-operator set of size at most $2\sqrt{n} - 1$ in polynomial time.*

Proof. For each CNOT gate, we construct an operator path and argue that all such paths can be grouped into $O(\sqrt{n})$ disjoint EDP sets. For simplicity, in the following, we specify paths by a sequence of key vertices, with each consecutive pair of key vertices connected by the shortest path (which is a horizontal or a vertical line).

We now construct an operator path for each CNOT operation, where the associated control vertex is $v = (v_x, v_y) \in V(G)$ and the target vertex is $u = (u_x, u_y) \in V(G)$. We can always form an operator path to connect u and v given by the following sequence of five key vertices: v, $(v_x, v_y - 1)$, $(u_x - 1, v_y - 1)$, $(u_x - 1, u_y)$, u. This path consists of one vertical end segment, one horizontal interior segment, one vertical interior segment, and finally a horizontal end segment.

Having assigned a path to each CNOT gate, we now show that any of these operator paths can share an edge with at most $2(\sqrt{n} - 1)$ of the other paths. Since the operator paths have distinct endpoints, two different paths cannot share an edge on either of their end segments $v, (v_x, v_y - 1)$ and on $(u_x - 1, u_y), u$. Therefore, pairs of these operator paths can only share an edge on their interior segments. The horizontal interior segment of the operator path from v to u can share an edge with at most $\sqrt{n} - 1$ other paths. To see this, consider an operator path from $v' = (v_x', v_y') \in V(G)$ to $u' = (u_x', u_y') \in V(G)$ that shares at least one horizontal edge with the operator path from v to u. Explicitly, this means that the segment $(v_x, v_y - 1)$, $(u_x - 1, v_y - 1)$ shares an edge with the segment $(v_x', v_y' - 1)$, $(u_x' - 1, v_y' - 1)$, which implies that $v_y = v_y'$. Since the terminals are unique, there can only be $\sqrt{n} - 1$ other CNOTs with the control sharing the v_y coordinate. An analogous argument applies for vertical segments, such that the operator path from u to v can share an edge with at most $2(\sqrt{n} - 1)$ other operator paths.

Let us construct a graph H where each vertex represents an operator path as constructed above. We connect two vertices in H if the associated paths share an edge. Every vertex in H has degree at most $2(\sqrt{n} - 1)$; therefore, H is $(2\sqrt{n} - 1)$ colorable using the (polynomial time) greedy coloring algorithm. We construct a $\mathcal{T}$-operator set of size $2\sqrt{n} - 1$ by grouping the paths associated with each color in a set of edge-disjoint paths. ■

We now show a general lower bound on compiling parallel CNOT circuits to the surface code architecture. Our strategy will be to consider a parallel CNOT circuit with control data qubits in an area with small boundary that generates an amount of entanglement across the boundary proportional to the area for a given initial state. However, each elementary surface code operation is local such that only those operations acting at the boundary can increase the entanglement across it. The depth of any implementation of the CNOT circuit is then lower bounded by the entanglement that it generates over the boundary size [42,43].

Theorem 4: *Consider a surface code architecture of n data qubits embedded in a grid where all ancilla qubits are in the $|0\rangle$ state. For any positive integer $k \leq n/2$, there exists a parallel* CNOT *circuit of k* CNOT *gates with associated terminal pairs $\mathcal{T}$ that needs depth $\Omega(\sqrt{k})$ to be implemented on the surface code architecture.*

Proof. Consider a CNOT circuit with terminal pairs $\mathcal{T}$ with control qubits on data vertices in a square region, V_L, and target qubits on vertices outside V_L. We initialize the $2k$ data qubits associated with $\mathcal{T}$ to a product state $|+\rangle^k\,|0\rangle^k$, with $|+\rangle$ on control qubits and $|0\rangle$ on target qubits (the remaining data qubits are initialized in an arbitrary product state and ignored). After applying the CNOT circuit, we obtain k Bell pairs. Therefore, the (von Neumann) entropy of the reduced state of the data qubits in V_L has increased from 0 to k.

Consider a circuit $\mathcal{C}$ of depth d that implements the parallel CNOT circuit. Any elementary operation of the surface code acting only within V_L or within $\bar{V}_L := V(G) \setminus V_L$ or classical communication (together, LOCC) cannot increase the entropy of the state on V_L. Moreover, as we show below, each elementary operation that acts both on V_L and on $\bar{V}_L$ can increase the entropy by at most a constant 4. We can therefore upper bound the increase in entropy due to $\mathcal{C}$ by $4d$ times the number of vertices adjacent to V_L, which is proportional to $\sqrt{k}$. To attain the k increase in entropy, we therefore need $d = \Omega(\sqrt{k})$.

We now bound the increase in entropy of any elementary operations acting on V_L and $\bar{V}_L$ to at most 4. All such elementary operations are built from a single XX or a ZZ measurement and single qubit operations (Appendix A), which cannot increase the entropy. It is possible to implement XX and ZZ measurements acting on V_L and $\bar{V}_L$ using two CNOT gates and operations acting only within V_L or within $\bar{V}_L$. The increase in entropy in V_L by a CNOT operation is bounded by 2 [44, Lemma 1]. Therefore, XX measurements, ZZ measurements, and indeed any elementary operation of the surface code can increase the entropy by at most 4. ■

In practice, it can be difficult to find minimal-size $\mathcal{T}$-operator sets. However, when the minimal size $\mathcal{T}$-operator set is k, in the following theorem we show that a $\mathcal{T}$-operator set $\{\mathcal{P}_1, \ldots, \mathcal{P}_l\}$ with size at most $l = O(k \log|\mathcal{T}|)$ can be found by a greedy algorithm that iteratively finds the maximum operator EDP set for remaining terminals in $\mathcal{T}$.

Theorem 5: *On the grid of n vertices, the greedy algorithm for finding $\mathcal{T}$-operator sets repeats the following two steps, for $i = 1, \ldots, |\mathcal{T}|$, until there are no more terminal pairs to connect:*

1. *find a maximum operator EDP set $\mathcal{P}_i$ for $\mathcal{T}$,*
2. *remove all terminal pairs in $\mathcal{P}_i$ from $\mathcal{T}$.*

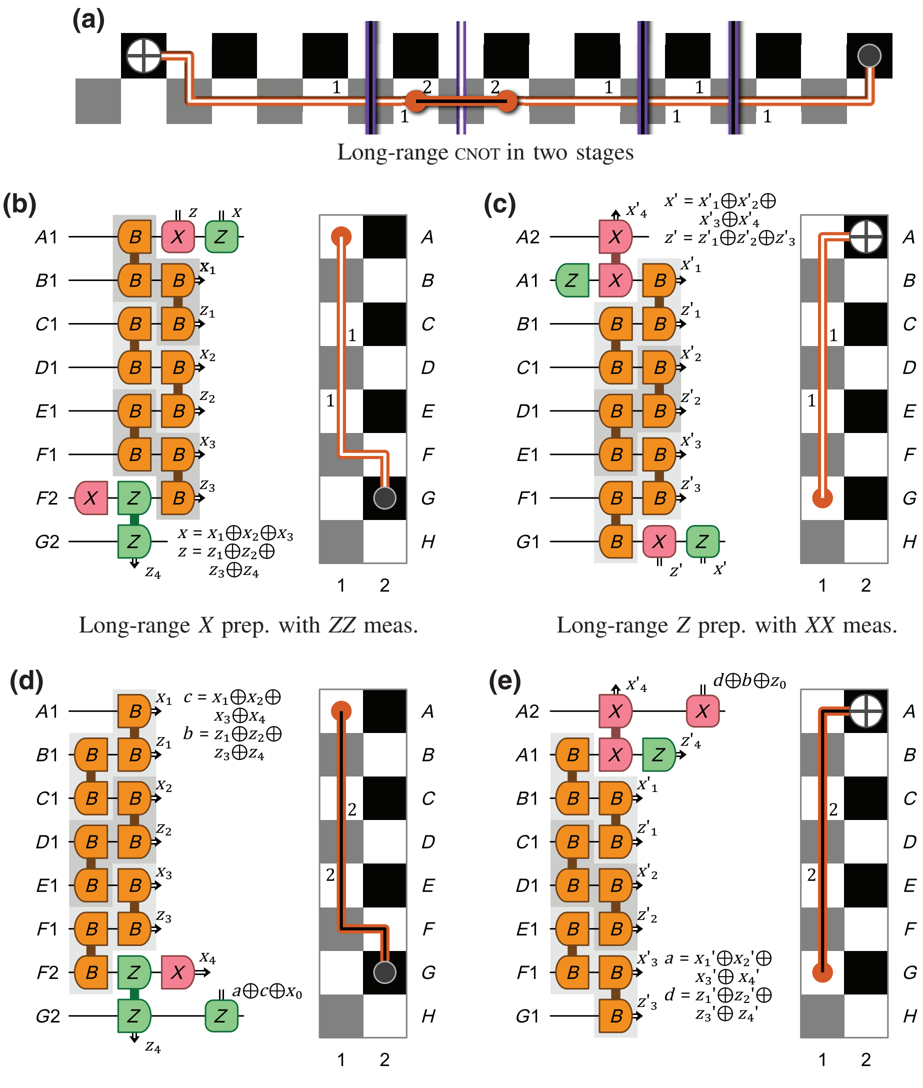
The set $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ is a $\mathcal{T}$-operator set and is an $O(\log|\mathcal{T}|)$-approximation algorithm for finding minimum-size $\mathcal{T}$-operator sets.

Proof. We base our proof on Ref. [39]. Assume that the minimum-size $\mathcal{T}$-operator set is $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_K\}$ for some size K. Then there is an operator EDP set $\mathcal{Q}_i$, for $i \in [K]$, such that $|\mathcal{Q}_i| \geq |\mathcal{T}|/K$. Therefore, the number of unconnected terminal pairs is reduced by at least a factor $(1 - 1/K)$ each iteration and it will require at most $O(K \log|\mathcal{T}|)$ iterations to connect all terminal pairs [45]. ■

To make use of Theorem 5, we would ideally like to have an algorithm to find maximum operator EDP sets on the grid; however, the efficient algorithms we discussed in Sec. III A fall short of this in two ways. Firstly, they find EDP sets rather than operator EDP sets, and secondly they provide approximate maximum sets rather than maximum sets. Fortunately, we find an equivalence between operator EDP sets on the grid and EDP sets on a graph that we call *the $\mathcal{T}$-operator graph* [see Fig. 7(b)]. The $\mathcal{T}$-operator graph is a copy of the grid graph but with all vertices corresponding to control qubits in $\mathcal{T}$ only having vertical outgoing edges, and with all vertices corresponding to target qubits in $\mathcal{T}$ only having horizontal incoming edges, and all remaining vertices corresponding to data qubits are removed. An EDP set for terminal pairs $\mathcal{T}$ on the $\mathcal{T}$-operator graph is an operator EDP set on the grid. It is easy to see that a maximum operator EDP set for $\mathcal{T}$ on the grid is equivalent to a maximum EDP set for $\mathcal{T}$ on the $\mathcal{T}$-operator graph. Using an approximation algorithm for finding the maximum operator EDP set also still gives approximation guarantees for minimizing the $\mathcal{T}$-operator set, as shown in the following corollary.

Corollary 6: *The greedy algorithm for finding minimum $\mathcal{T}$-operator sets, but with a κ-approximation algorithm for finding maximum operator EDP sets, gives an $O(\kappa \log|\mathcal{T}|)$-approximation algorithm for finding minimum $\mathcal{T}$-operator sets.*

Proof. We modify the proof of Theorem 5 such that every iteration we connect a $(1 - \kappa/K)$ fraction of unconnected terminal pairs using the κ-approximation algorithm


(a)
1
2
2
1
1
1
1
1
(b)
A1
B1
C1
D1
E1
F1
F2
G2
B
X
Z
x_1
z_1
x_2
z_2
x_3
z_3
z_4
$x = x_1 \oplus x_2 \oplus x_3$
$z = z_1 \oplus z_2 \oplus z_3 \oplus z_4$
A B C D E F G H
1 2
(c)
A2
A1
B1
C1
D1
E1
F1
G1
x'_4
$x' = x'_1 \oplus x'_2 \oplus x'_3 \oplus x'_4$
$z' = z'_1 \oplus z'_2 \oplus z'_3$
x'_1
z'_1
x'_2
z'_2
x'_3
z'_3
z'
x'
(d)
$c = x_1 \oplus x_2 \oplus x_3 \oplus x_4$
$b = z_1 \oplus z_2 \oplus z_3 \oplus z_4$
x_4
$a \oplus c \oplus x_0$
(e)
$d \oplus b \oplus z_0$
z'_4
$a = x_1' \oplus x_2' \oplus x_3' \oplus x_4'$
$d = z_1' \oplus z_2' \oplus z_3' \oplus z_4'$

Long-range CNOT in two stages

Long-range X prep. with ZZ meas.

Long-range Z prep. with XX meas.

Long-range teleport with ZZ meas.

Long-range teleport with XX meas.

FIG. 9. Detailed implementation of the steps in Fig. 6. For each segment that is scheduled in phase 1, we use (b) and (c); and for each subpath that is scheduled in phase 2, we use (d) and (e). In (d) variables x_0 and z_0 are equal to the total parity of all long-range Bell measurements applied during stage 1 on the CNOT path. Each of these operations takes depth 2. Panels (d) and (e) share variables a and b.

for finding maximum operator EDP sets. Therefore we obtain a $O(\kappa \log|\mathcal{T}|)$-approximation algorithm for finding minimum $\mathcal{T}$-operator sets. ■

The equivalence between operator EDP sets on the grid and EDP sets on the $\mathcal{T}$-operator graph motivates us to seek an efficient algorithm to find approximate maximum EDP sets on the $\mathcal{T}$-operator graph as a key part of our EDPC algorithm. The algorithms we discussed in Sec. III A come close to doing this, but some of them are intended for finding approximate maximum EDP sets on the grid rather than on the $\mathcal{T}$-operator graph and even if they are adapted, the guarantees of the size of the approximate minimum EDP sets they produce may not apply in the case of the $\mathcal{T}$-operator graph. The algorithms described in Refs. [39,40] for finding approximate maximum EDP sets on the grid do not directly apply to the operator graph. While it seems straightforward to adapt the $O(\log n)$-approximation algorithm [39], the algorithms in Refs. [39,40] are complex to implement and have large constant-factor overheads, which can make them impractical on small instance sizes.

In EDPC, we instead combine the theoretical worst-case bounds of Theorem 3 with the pragmatic performance of a greedy approach, which does not have a large constant overhead, in Algorithm 2. By Theorem 4 this gives us asymptotically tight performance in the worst case. The runtime of this algorithm is dominated by $O(|\mathcal{T}|)$ iterations of approximately maximizing the operator EDP set in time $O(|\mathcal{T}|n \log n)$. We leave it as an open question to find better approximation algorithms for finding maximum operator EDP sets that give improved performance outside the worst case and that may also improve the runtime since less iterations over $\mathcal{T}$ are required.

IV. REMOTE ROTATIONS WITH MAGIC STATES

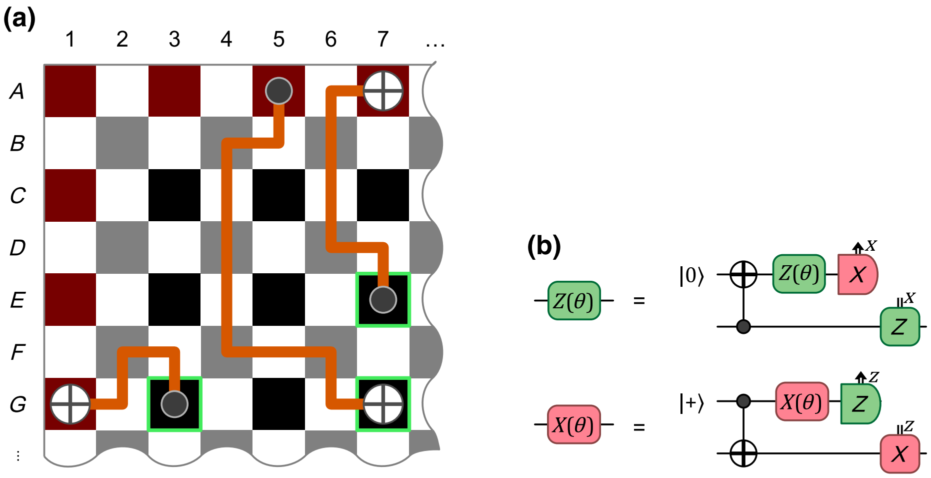
Thus far, we have discussed the surface code compilation of all the input circuit operations listed in Sec. I

```
Input   : $\mathcal{T}$ terminal pairs
$\mathcal{Q}_1 \leftarrow$ the $\mathcal{T}$-operator set given by Theorem 3 for $\mathcal{T}$
$\mathcal{Q}_2 \leftarrow \emptyset$
while $\mathcal{T} \neq \emptyset$ :        // Greedy approx. minimum-size $\mathcal{T}$-operator set
    $\mathcal{P} \leftarrow$ approximately maximize operator EDP set using greedy EDP
      algorithm [34] on operator graph
    remove connected terminal pairs in $\mathcal{P}$ from $\mathcal{T}$
    $\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{\mathcal{P}\}$
return minimum-size set between $\mathcal{Q}_1$ and $Q_2$
```

Algorithm 2. *Bounded $\mathcal{T}$-operator set algorithm*: an approximation algorithm for minimizing the $\mathcal{T}$-operator set size that combines the theoretical guarantees from Theorem 3 with pragmatic performance using the greedy algorithm of Theorem 5.

except for the single-qubit rotation gates $S = Z(\pi/4)$ and $T = Z(\pi/8)$. In this section we design a subroutine for the compilation of parallel rotation circuits. The S and T gates can be implemented by using specially prepared *magic states* $|S\rangle$ and $|T\rangle$, respectively. Magic states can be prepared using a highly optimized process known as *magic state distillation* [46], which distills many faulty magic states that are easy to prepare into fewer robust states. Still, producing both $|S\rangle$ and $|T\rangle$ involves considerable overhead. The $|S\rangle$ state is used to apply the S gate in a "catalytic" fashion, whereby the state $|S\rangle$ is returned afterwards. On the other hand, the state $|T\rangle$ is consumed to apply the T gate. The reason for this distinction is rooted in the fact that the S gate is Clifford but the T gate is non-Clifford.

In this work, we do not address the mechanism by which magic states are produced, but instead assume that these states are provided at specific locations where they can be used to implement gates. More specifically, we assume that rotation gates S and T [and also Clifford variations of these such as $X(\pi/8) = T_x$ and $X(\pi/4) = S_x$] can be applied as a resource on specific ancilla qubits $B \subseteq V(G)$ at the boundary of a large array of logical qubits [Fig. 10(a)]. This will allow sufficient space outside the boundary where highly optimized magic state distillation and synthesis circuits can be implemented. Because a large number of magic states are used in the computation, we consider having magic state distillation adjacent to and concurrent with computation we are concerned with in this paper to be a reasonable allocation of resources.



Long-range CNOT operations for diagonal gates   Remote $Z(\theta)$ and $X(\theta)$

FIG. 10. We assume the capability of performing S and T gates at the boundary qubits (red) where it is easy for us to supply the requisite S and T magic states. We can then execute S or T gates in the Z or X basis for our circuit by using long-range CNOT gates and the circuits in (b). For example, to execute S or T on qubits $G3$, $E7$, and HTH on $G7$, we apply long-range CNOT gates between pairs $(G3, G1)$, $(E7, A7)$, $(A5, G7)$, and then execute S or T on $G1$, $A7$, HTH on $A5$. We can continue applying other Clifford gates to qubits $G3$, $E7$, and $G7$ right after performing the long-range CNOT gate, without waiting for the Z correction, since we can propagate the correction through Clifford operations.

We need a technique to apply *remote rotations* to data qubits that can be far from the boundary, making use of the rotations that can be performed at the boundary. We make use of the property that any Z rotation (including T or S) has the same action when applied to either qubit in the state $\alpha\,|00\rangle + \beta\,|11\rangle$. In particular, these two qubits need not be close to one another, so we can apply Z rotations *remotely*. A similar notion holds for X rotations (including $T_x = HTH$ or $S_x = HSH$) and $\alpha\,|{++}\rangle + \beta\,|{--}\rangle$. Given a qubit q that needs to perform a Z rotation requiring a magic state, we apply a *remote Z rotation* [Fig. 10(b)]: by performing a long-range $\textsc{cnot}(q, q')$ to a boundary ancilla $q' \in B$ prepared in $|0\rangle$. Therefore we can apply the Z rotation remotely and use an X measurement on q' to collapse the state back to one logical qubit. Similarly, for a qubit q that needs to perform an X rotation requiring a magic state, we apply a long-range $\textsc{cnot}(q', q)$ to an ancilla q' prepared in $|+\rangle$ on the boundary, giving

$$\textsc{cnot}\,|+\rangle\,(\alpha\,|+\rangle + \beta\,|-\rangle) = \alpha\,|{++}\rangle + \beta\,|{--}\rangle\,. \tag{5}$$

Therefore we apply the X rotation remotely and collapse the state back by a single-qubit Z measurement of q'.

The task of compiling a parallel rotation circuit therefore reduces to applying a set of CNOT gates from the boundary to the sites of the rotation gates. This can be achieved by finding an appropriate EDP set and running the EDP subroutine of Algorithm 1. Compared to the task of finding an EDP set for parallel CNOT gates of Sec. III, there is one simplifying condition here: any boundary qubit can be used for each CNOT gate when applying remote rotations. As we explain below, we can find the maximum EDP set for the compilation of remote rotations by solving the following (unit) MAX FLOW problem [47].

Definition 7 (MAX FLOW)**:** Given a directed graph G and source and sink vertices $s, t \in V$, we wish to find a flow for all edges of $G, f(e)\colon E(G) \to \mathbb{R}$, that is skew symmetric, $f[(u,v)] = -f[(v,u)]$, and, for $v \in V(G) \setminus \{s,t\}$, must respect the constraints

$$f(e) \leq 1, \tag{6}$$

$$\text{and} \quad \sum_{u:(v,u)\in E(G)} f[(v,u)] = 0, \tag{7}$$

such that the outgoing source flow $|f| := \sum_{u:(s,u)\in E(G)} f[(s,u)]$ is maximized.

To understand why this yields a maximum EDP, we first point out that a solution for which f has binary values provides an EDP set by building paths from those edges e for which $f(e) = 1$. Moreover, this EDP set must be maximum, because a larger EDP set would imply a larger flow than f, which is the maximum flow by definition. Indeed, the Ford-Fulkerson algorithm [30] solves MAX FLOW in runtime bounded by $O(|E(G)||f|)$ and finds flow values $f(e) \in \{0,1\}$ on all $e \in E(G)$ because of the unit capacity constraints, $f(e) \leq 1$. Therefore, f corresponds to a maximum EDP set [47, Sec. 7.6].

The *remote rotation subroutine* (Algorithm 3) executes a set of parallel single-qubit rotations. Each iteration can be performed in depth 4 using the EDP subroutine. On the surface code architecture, we can give strong guarantees on the number of iterations required to execute a set of parallel rotations by the MAX FLOW to min-cut equivalence.

Theorem 8: *The remote rotation subroutine executes all rotations in $\mathcal{G}_m$ in depth $O(\sqrt{|\mathcal{G}_m|})$.*

Proof. The function `max_rotations` $(\mathcal{G}_m)$ that is a part of the remote rotation subroutine finds a maximum flow connecting the data qubits performing rotations to the boundary where every additional unit of flow is one

```
Input : Connectivity graph $G$ with vertices corresponding to boundary
        qubits $B \subseteq V(G)$ and a set of parallel rotations $\mathcal{G}_m$
function max_rotations($\mathcal{G}_m$):
    $W \leftarrow$ vertices associated with qubits in $\mathcal{G}_m$
    create virtual vertices $s$ and $t$
    $G' = (V(G) \cup \{s,t\}, E(G) \cup \{(s,s') \mid s' \in W\} \cup \{(t',t) \mid t' \in B\})$
    $f \leftarrow$ solve MAX FLOW on $G'$ using the Ford-Fulkerson algorithm
    $\mathcal{P} \leftarrow$ construct edge-disjoint set of $s$–$t$ paths from $f$
    return $\mathcal{P}$ with $s$ and $t$ removed from each $P \in \mathcal{P}$
while $\mathcal{G}_m$ is not empty :
    $\mathcal{P} \leftarrow$ max_rotations($\mathcal{G}_m$)
    execute remote rotations at boundary with EDP subroutine given $\mathcal{P}$
    remove executed rotations from $\mathcal{G}_m$
```

Algorithm 3. *Remote rotation subroutine*: executes parallel single-qubit rotations that require magic states at the boundary by a MAX FLOW reduction. Using the EDP subroutine (Algorithm 1), we can perform remote rotations (Fig. 10) on each set of qubits connected to the boundary by $\mathcal{P}$ in depth 4.

more rotation executed. This maximum flow is equal to the minimum edge cut separating the data qubits from the boundary [30]. The boundary of a rectangle containing $|\mathcal{G}_m|$ vertices on the grid is of size $\Omega(\sqrt{|\mathcal{G}_m|})$, giving a minimum cut size of $\Omega(\sqrt{|\mathcal{G}_m|})$. Thus, at most $O(\sqrt{|\mathcal{G}_m|})$ iterations of the while loop in the remote rotation subroutine are necessary to implement all remote rotations, as claimed. ■

We bound the runtime of the remote rotation subroutine by $O(n^2\sqrt{|\mathcal{G}_m|})$ as follows. At most $O(\sqrt{|\mathcal{G}_m|})$ iterations of the while loop are necessary (see the proof of Theorem 8). Each iteration, the call to `max_rotations` $(\mathcal{G}_m)$ is dominated by solving a MAX FLOW instance using the Ford-Fulkerson algorithm [30], which has a runtime bounded by $O(n^2)$.

One could consider a number of generalizations and variations of this compilation subroutine for parallel rotation circuits. For instance, when the number of rotation gates is small, it may be useful to find VDP sets rather than EDP sets so that the VDP subroutine rather than the EDP subroutine can be applied. There is a different reduction to MAX FLOW in this case that can be obtained by replacing each vertex with two vertices, one with an incoming edge and one with an outgoing edge, connected by a directed edge with capacity 1. This guarantees that only one flow can pass through every vertex.

Although we do not consider other single-qubit rotations in our input circuit for compilation, it is worth noting that any single-qubit rotation gate $Z(\theta)$ can be approximately synthesized to arbitrary precision [48] using $|S\rangle$ and $|T\rangle$ states along with the surface code operations shown in Fig. 1. The approach used to apply S and T gates shown in Fig. 10(a) can also be used to apply any rotation $Z(\theta)$ within the grid of surface codes by synthesizing the rotation at the boundary. However, if one considers more general rotations in the input circuit, the time needed for synthesis at the boundary will need to be accounted for and accommodated by other aspects of the overall surface code compilation algorithm. Another extension that can be considered is if multiqubit diagonal gates are allowed in the input circuit. We show how X and Z rotations generalize to multiqubit diagonal gates in Appendix D, although we do not use this in our surface code compilation algorithm.

V. EDPC SURFACE CODE COMPILATION ALGORITHM

In this section we construct the EDPC algorithm for compiling universal input circuits into surface code operations by combining the subroutines in Algorithms 1 and 3 for compiling long-range CNOT gates and Z/X rotations, respectively. First we provide a more formal definition of surface code compilation.

Definition 9 (Surface code compilation): Consider an input quantum circuit of operations $\mathcal{C} = g_1 g_2 \cdots g_\ell$, which is a list of length ℓ of operations g_i for $i \in [\ell]$, consisting of state preparation in the X or Z basis; the single-qubit operators X, Y, Z, H, S, T, $S_x = HSH$, and $T_x = HTH$; CNOT operations; and X, Z measurements. Then a surface code compilation produces an equivalent output circuit $\mathcal{O}$ in terms of surface code operations (Fig. 1) on a grid of surface codes with S, T, S_x, and T_x rotations applied only at the grid's boundary.

The surface code compilation algorithm EDPC (Algorithm 4) combines the bounded $\mathcal{T}$-operator set algorithm for parallel CNOT gates with the remote rotation subroutine. Note that the input circuit is considered to be a sequence of operations rather than a series of time steps that specify the operations in each time step, such that l is the number of operations of the input circuit, not the depth.

We bound the classical runtime of EDPC given an input circuit with depth D acting on n qubits. It is useful to note that each of the D layers of the input circuit can be decomposed into a set of parallel rotations followed by a set of parallel CNOT gates, each acting on at most n qubits. Recall that the remote rotation subroutine has a runtime bounded by $O(n^{2.5})$, whereas compiling a set of parallel CNOT gates has a runtime of at most $O(n^3 \log n)$. Thus, EDPC has a runtime bounded by $O(Dn^3 \log n)$.

Circuits compiled by EDPC can be bounded in depth as listed in Table I. Our claim for a single CNOT gate is

```
Input : Circuit $\mathcal{C}$ with Pauli gates commuted to the end and merged with
        measurement
while available operations in $\mathcal{C}$ :
    execute all available state preparation, measurement, and Hadamard gates
    run remote rotation subroutine on available rotations
    $\mathcal{T} \leftarrow$ terminal pairs associated with available CNOT gates
    $\mathcal{Q} \leftarrow$ run bounded $\mathcal{T}$-operator set Algorithm 2 on $\mathcal{T}$
    for $\mathcal{P} \in \mathcal{Q}$ :
        run EDP subroutine (Algorithm 1) on $\mathcal{P}$
```

Algorithm 4. *EDPC*: a surface code compilation algorithm for any circuit $\mathcal{C} = g_1 \cdots g_\ell$. An operation g_i is *available* if it has not been executed and all operations g_j with overlapping support, for $j < i$, are executed.

trivial. Theorems 3 and 4 show that parallel CNOT circuits are compiled to a depth of $\Theta(\sqrt{n})$, and Theorem 8 shows that k parallel rotations are compiled to a depth of $O(\sqrt{k})$. It is then easy to see that a circuit of depth D compiles to a circuit of depth at most $O(D\sqrt{n})$. If we assume that a remote rotation must be performed for each rotation requiring magic states at the boundary (in particular, it requires a long-range CNOT gate as in EDPC), then Theorem 4 shows an $\Omega(\sqrt{k})$ lower bound on the depth to apply k CNOT operations with the boundary.

There are various modifications of EDPC that are worth considering. Firstly, the bounded $\mathcal{T}$-operator set algorithm (Algorithm 2) can be improved by better algorithms for finding maximum operator EDP sets. Secondly, the requirement to execute all available gates before moving on to the next set could be relaxed. This could increase the number of long-range gates that are performed in parallel, but would require careful scheduling with Hadamard gate execution, which may block some paths. Lastly, EDPC leans heavily on finding operator EDPs and the EDP subroutine, but a similar surface code compilation algorithm could be constructed from operator VDPs and the VDP subroutine instead. We believe that larger maximum EDP sets allows EDPC to apply more gates simultaneously (see Sec. III A), and more so if algorithms for approximation maximum operator EDP sets can be adopted from EDP approximation algorithms [39,40]. Both of these features can give asymptotic improvements at only a 2 times depth increase over the VDP subroutine. However, it is not difficult to construct instances where a VDP-based approach would give a lower depth, motivating a more nuanced trade-off between our EDP-based approach and a VDP-based approach.

VI. COMPARISON OF EDPC WITH EXISTING APPROACHES

In this section, we compare EDPC with other approaches in the literature. We first mention some of the features and shortcomings of the well-established approach of Pauli-based computation Sec. VI A. Then we address a more recently proposed compilation approach based on network coding in Sec. VI B. In Sec. VI C we specify a SWAP-based compilation algorithm [14] and use this as a benchmark for numerical studies of the performance of an implementation of EDPC in Sec. VI D.

A. Surface code compilation by Pauli-based computation

One well-established surface code compilation approach is known as *Pauli-based computation*, which is described in Ref. [13]. For an algorithm expressed in terms of Clifford and T gates, Pauli-based computation first involves reexpressing the algorithm as a sequence of joint multiqubit Pauli measurements along with additional ancilla qubits prepared in T states. This reexpressed circuit has no Clifford operations, and the circuit depth can be straightforwardly deduced from the input circuit since each T gate results in two [49] joint Pauli measurements [50]. This reexpression of the circuit essentially comes from first replacing each T gate by a small gate teleportation circuit consisting of an ancilla in a T state and a two-qubit joint Pauli measurement, and then commuting all Clifford operations to the end of the circuit. The main advantage of the Pauli-based computation approach is that all Clifford gates are removed from the input circuit, resulting in no cost for CNOT circuits in Table I.

That said, this approach has a major drawback. When a Clifford circuit is commuted through a two-qubit joint Pauli measurement, it is transformed into Pauli measurements that can have support on all logical qubits. Therefore, the resulting circuit may contain measurements with large overlapping support that need to be performed sequentially (even when the T gates in the input circuit are acting on disjoint qubits during the same time step). The sequential nature of the joint measurements causes a fixed rate of T-state consumption that does not grow with the number of logical qubits and results in a $\Theta(k)$ depth for k parallel rotations, as listed in Table I. The depth for parallel rotations is significantly higher than EDPC and could lead to a larger space-time cost for circuits with many T gates per time step.

A modified version of this Pauli-based computation compilation algorithm can be used to implement more T gates in parallel [13, Sec. 5.1]. However, as highlighted in Sec. V.A of Ref. [49], this results in a significant increase of total logical space-time cost when compared to the standard Pauli-based computation compilation algorithm, even when disregarding the increased T-factory costs that would be needed to achieve a higher T-state production rate.

In contrast with Pauli-based computation, one of our goals when designing the EDPC algorithm was to maintain the parallelism present in the input circuit, such that input circuits with higher numbers of T gates per time step are compiled to circuits with a higher T-state consumption rate.

B. Surface code compilation by network coding

Another approach to surface code compilation, based on the field known as *linear network coding* [51], can be built from the framework put forward in Ref. [28]. Similar to our EDPC algorithm, the essential idea in this compilation scheme is to generate sets of Bell pairs in order to implement operations acting on pairs of distant qubits.

In the abstract setting of network coding [52], one is given a directed graph G_{NC} and a set of terminal pairs $\mathcal{T} = \{(s_1, t_1), \dots, (s_k, t_k)\}$ for source terminals $s_i \in V(G_{\mathrm{NC}})$ and target terminals $t_i \in V(G_{\mathrm{NC}})$ for $i \in [k]$. Messages are passed through edges according to a linear rule. Namely, the value of the message associated with an edge is given as a specific linear combination of the values of those edges that are directed at the edge's head. One can consider the task of "designing a linear network code" by specifying the linear function at each edge in the graph such that, when any messages are input via the source vertices $s_1, \ldots, s_k$, then those same messages are copied over to the corresponding output via the target vertices $t_1, \ldots, t_k$.

A number of works have considered how linear network coding theory can be applied to the quantum setting [23–27]. In Ref. [28] a construction for a constant-depth circuit is given to generate Bell pairs across the terminal pairs $\mathcal{T}$ on a set of ancilla qubits corresponding to the vertices of G_{NC} with CNOT gates allowed on the edges of G_{NC}. This is similar to, but not precisely the same scenario as, what we consider for surface code compilation in this paper since the basic operations are CNOT gates rather than the elementary operations of the surface code, and since only ancilla qubits are considered without any data qubits. However, it should be quite straightforward to modify the approach in Ref. [28] to form a surface code compilation algorithm. For example, one could use a layout similar to that which we use for EDPC in Fig. 6, with G_{NC} corresponding to a connected subset of ancilla qubits among a set of data qubits. The Bell pairs produced by the linear network coding approach could then be used to compile long-range operations between data qubits.

In such a network-coding-based compilation algorithm, the task of compiling an input circuit into surface code operations would largely rely on subroutines for (1) identifying $\mathcal{T}$ to implement the circuit's long-range gates, and (2) designing a linear network code for $\mathcal{T}$. A major barrier to forming a usable compilation algorithm with linear network coding is that we are unaware of the existence of any efficient algorithm to design linear network codes, or even to identify if a given terminal pair set admits any linear network code. Even if such a linear network code can be found efficiently, there exist sets $\mathcal{T}$ for which network coding cannot provide a depth advantage over EDPC.

```
Input : A circuit $\mathcal{C}$ with all Pauli gates commuted to the end and merged
        with measurement
function cost(mapping $\pi$, vertices $v_1, v_2$):
    return depth and edge attaining
      $\min_{e \in \mathcal{M}}$ depth(route($\pi + \{v_1 \mapsto e_1, v_2 \mapsto e_2\}$))
while available gates in $\mathcal{C}$ :
    $\mathcal{G} \leftarrow$ available gates in $\mathcal{C}$
    execute all Hadamard gates and measurements in $\mathcal{G}$
    $G \leftarrow$ surface code grid graph
    $\pi \leftarrow$ empty mapping of $V(G) \to V(G)$
    // Start modification for operations requiring magic
       states
    Set $B \subseteq V(G)$ as the set of boundary vertices
    $\mathcal{G}_m \leftarrow \{g \in \mathcal{G} \mid g \text{ is } S, T, S_x, T_x\}$
    while $\mathcal{G}_m \neq \emptyset$ and $B \neq \emptyset$ :
        $g \leftarrow$ pop random gate from $\mathcal{G}_m$
        $\pi \leftarrow \pi + \{v \mapsto u\}$, for closest $u \in B$ to $v$
        remove $u$ from $B$ and $G$
    // End modification
    $\mathcal{M} \leftarrow$ maximum matching of $G$
    $\mathcal{G}_c = \{g \in \mathcal{G} \mid g \text{ is CNOT}\}$
    while $G_c \neq \emptyset$ and $\mathcal{M} \neq \emptyset$ :
        $g^*, e^* \leftarrow \max_{g \in \mathcal{G}_c}$ cost($\pi + \{v_1 \mapsto e_1, v_2 \mapsto e_2\}$) for $v_1, v_2$
          current location of $g$
        $\pi \leftarrow \pi + \{v_1 \mapsto e_1, v_2 \mapsto e_2\}$, for $v_1, v_2$ current location of $g^*$
        remove $g^*$ from $\mathcal{G}_c$
        remove $e^*$ from $\mathcal{M}$
    execute the SWAP gates found by route($\pi$)
    execute gates on qubits mapped by $\pi$ since they are now local
```

Algorithm 5. SWAP *compilation*: we construct an algorithm based on the *greedy depth* mapper algorithm from Ref. [14]. Let us implicitly define route(π), for mapping π, which finds a SWAP circuit for implementing partial permutations [14]. We can compute the required partial permutation from the current mapping of qubits, and the given future mapping π.

Any surface code compilation algorithm of CNOT circuits with k parallel CNOT gates, including EDPC and algorithms using network coding, is lower bounded in the worst case by Theorem 4 to a depth of $\Omega(\sqrt{k})$. This bound is loose when k is superconstant and sublinear in n since EDPC has a trivial upper bound of $O(k)$ and a bound of $O(\sqrt{n})$ by Theorem 3 on the compiled circuit depth. Therefore, it remains an open question whether network coding can give an advantage for the compiled circuit depth for such k.

C. Surface code compilation by SWAP gates

Here we specify a SWAP-based compilation algorithm, stated in Algorithm 5, which we use to benchmark our EDPC against in Sec. VI D. We assume the 1-to-1 ancilla-to-data qubit ratio as illustrated in Fig. 11. This is more qubit efficient than the 3-to-1 ratio we use for EDPC, and it allows the SWAP gadget in Fig. 3(b) to be implemented between diagonally neighboring data qubits.

The first step of the SWAP-based compilation algorithm is to assign each of the input circuit's qubits to a data qubit in the layout. Then, the gates in the input circuit are collected together into sets of disjoint gates. Before each set of gates, a permutation built from SWAP gates is applied, which repositions the qubits so that the gates in the set can be applied locally. We assume that the available local operations are the same as for our EDPC algorithm. In particular, we assume that the rotation gates (S, T, S_x, and T_x) can only be implemented at the boundary and that other single-qubit operations are performed as described in Sec. II A. One exception is that we make the simplifying assumption that the Hadamard gate can be performed without the need of three ancilla patches to simplify our analysis—this assumption could lead to an underestimate of the resources required for this SWAP-based compilation algorithm.

There are two main components of our SWAP-based algorithm that remain to be specified: how the permutations are implemented, and how we choose to separate the input circuit into a sequence of sets of disjoint gates. To permute the positions of data qubits, sequences of SWAP operations are used. Any permutation of the n vertices in a square grid can be achieved in at most $3\sqrt{n}$ rounds of nearest-neighbor swap gates [20]. To do this involves three stages, with the first and third stages each involving rounds of SWAP-gates within rows only, and the second stage involving rounds of SWAP-gates within columns only. A round of SWAP gates within either rows only or within columns only is implemented with surface code operations as shown in Fig. 11. This immediately shows that this approach is asymptotically tight for parallel circuits because the depth of a SWAP-based approach is lower bounded by the $\sqrt{n}$ diameter of the architecture grid for one long-range CNOT or rotation gate from the center of the grid. Therefore, a parallel input circuit is compiled by the SWAP-based algorithm to an output circuit with depth $\Theta(\sqrt{n})$, including all the examples in Table I.

There is considerable freedom in how to collect together gates from the input circuit into sets of disjoint gates. In our implementation in Algorithm 5, we use the *greedy depth* mapper algorithm from Ref. [14], with a small modification to ensure that the S and T gates are performed at the boundary. This algorithm also incorporates some further optimizations as described in Ref. [14], including a partial mapping of qubits to locations, leaving the remaining qubits to go anywhere in an attempt to minimize the SWAP circuit depth.

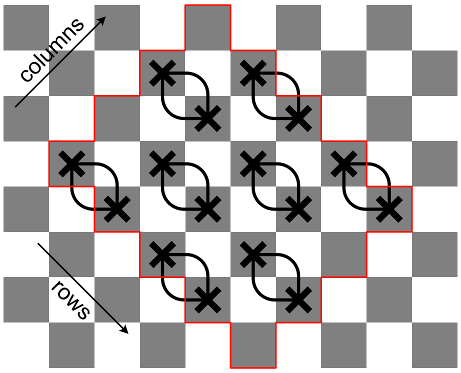


FIG. 11. On a rotated $L_1 \times L_2$ grid (here, 4×5), we can implement an odd-even pattern of swap gates on data qubits (gray) using ancillae (white). Row-wise and columnwise SWAP gates used in SWAP routing on a grid [21] can be modified as shown above so that the ancillae used for SWAP gates do not overlap. Therefore, any arbitrary permutation on a rotated grid can be implemented in space-time $4(L_1 + 1) + 2(L_2 + 1)$.

D. Numerical results

Here we numerically compare the performance of EDPC with the SWAP-based compilation algorithm (Algorithm 5) when applied to a number of different input circuits. Note that our implementation of the EDPC compilation algorithm here differs slightly from that given in Algorithm 4, by greedily executing CNOT gates earlier where possible. See Appendix E for details of the implementation.

Our first input circuit example consists of random parallel CNOT circuits of different gate densities. The density n_{CNOT} of a circuit is how many of the data qubits are involved in a CNOT gate in any such set. Therefore, $n_{\text{CNOT}} = 0.1n$ means that 10% of all qubits (n) are performing a CNOT gate in each set. For each data point, we sample ten random circuits and plot the mean space-time cost in Fig. 12 with the standard error of the mean in the shaded region. The runtime of the SWAP protocol is bounded by

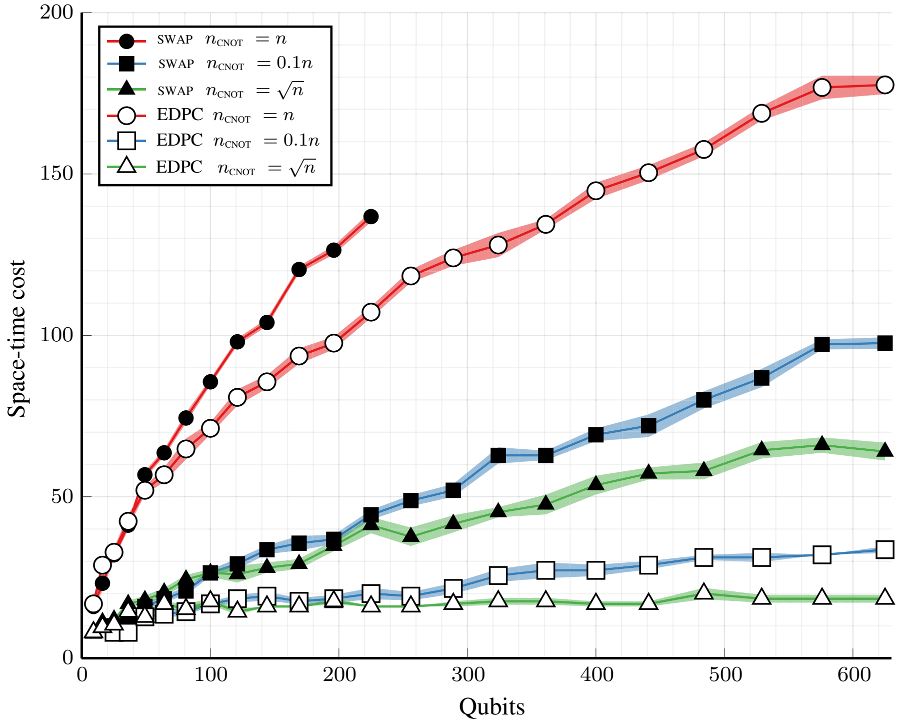


FIG. 12. Space-time cost of a randomly sampled set of disjoint CNOT gates with standard error of the mean (shaded region) compiled to the surface code using EDPC and SWAP compilation. We generate ten random circuits for each number of qubits (n) consisting of a set of disjoint CNOT gates of varying density; the number of randomly selected qubits involved in a CNOT gate is given by n_{CNOT}. At all densities we see improved performance and scaling using EDPC.

2 days, which is insufficient for larger instances of these random circuits at high densities.

We also consider a more structured input circuit, namely implementing half of a multicontrolled-X gate, C^{k}NOT. We consider decompositions of a C^{k}NOT gate for k integer powers of 2, but only compile the first half of the circuit, given in Fig. 13(a). A T-efficient implementation of the C^{k}NOT gate uses measurement and feedback for uncomputation [53], which are not captured in our model (see Sec. VII). We plot the space-time cost of compiling the half C^{k}NOT gate in Fig. 13(b). We see that the dependence on the number of qubits k is worse for SWAP-based compilation, and results in a larger space-time cost starting at 64 qubits. Unfortunately, the SWAP-based compilation is quite slow: we ran the algorithm for at most 3 days and 9 hour at each data point and were only able to obtain results up to 128 qubits. However, the data we were able to obtain indicates a crossover for compiling a half C^{k}NOT gate. The SWAP-based compilation has better space-time performance for small instances, while EDPC has a better space-time performance for compiling large C^{k}NOT gate.

VII. CONCLUSION

In this paper, we have introduced the EDPC algorithm for the compilation of input quantum circuits into operations that can be implemented fault tolerantly with the surface code. The heart of this algorithm lies in the EDP

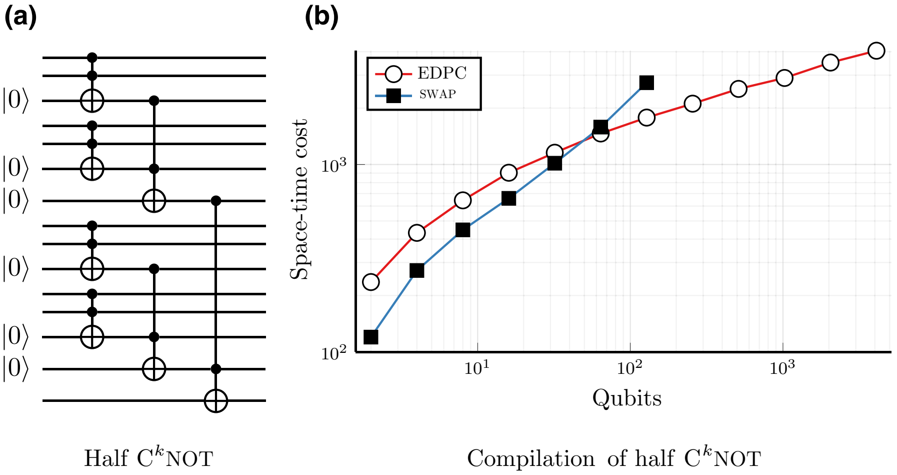


FIG. 13. We compare the space-time cost of compiling a T-gate optimized circuit decomposition for a half C^{k}NOT gate to the surface code using EDPC and SWAP compilation. We see in the log-log plot (b) that dependence of the space-time cost on n gives a higher scaling dependence in the case of SWAP compilation than EDPC. This results a lower space-time cost for EDPC starting from 64 qubits.

subroutine, which can implement both sets of parallel long-range CNOT gates and sets of parallel rotations in constant depth using existing efficient graph algorithms to find sets of edge-disjoint paths. EDPC has advantages over other compilation approaches, including Pauli-based computation, network-coding-based compilation, and SWAP-based compilation. We numerically find that EDPC significantly outperforms SWAP-based circuit compilation in the space-time cost of random CNOT circuits for a broad range of instances, and for larger C^{k}NOT gates. However, many details of EDPC can be improved, as it is only a first step towards using long-range operations for surface code compilation.

EDPC requires sets of constrained edge-disjoint paths, which we call operator paths and run almost entirely along ancilla qubits. Better algorithms for finding maximum sets of edge-disjoint operator paths could improve EDPC. It seems likely that an $O(\log n)$-approximation algorithm for finding maximum EDP sets on grids [39] can be modified to give an algorithm for finding maximum sets of edge-disjoint operator paths on grids. A polylogarithmic approximation algorithm for this task would imply an approximation algorithm for minimizing the depth, up to a polylogarithmic factor, of compiling parallel CNOT gates using the EDP subroutine. In practice, it is, however, also important to find approximation algorithms with reasonable constant prefactors.

The runtime complexity of EDPC for an input circuit of depth D acting on n qubits is $O(Dn^3 \log n)$. This is significantly faster than the SWAP-based compilation in Sec. VI C, which was found to be $O(Dn^5)$ in Ref. [14]. We found that our implementation of the SWAP-based compilation implementation runtime is much slower than that of EDPC on small instances, and found that the SWAP-based algorithm had impractically long runtimes when applied to circuits beyond a few hundred qubits, the regime of large-scale applications of quantum algorithms [54,55]. Potential ways to further improve EDPC's runtime include using a dynamical decremental all-pair shortest path algorithm in the greedy approximation of the maximum EDP set, or by finding faster and better approximation algorithms for finding the maximum set of edge-disjoint operator paths.

Any diagonal gates in the Z (or X) basis can be performed remotely on the boundary, including CCZ gates [56] (see Appendix D). Therefore, our results on applying $Z(\theta)$ rotations can be extended to diagonal gates, which will benefit circuit depth.

Even with the capability to perform long-range operations, it may still be helpful to localize the quantum information on some part of the architecture such as by permuting the data qubits. In particular, the size of the EDP set is bounded above by the minimum edge cut separating the terminals. Therefore, it may be beneficial to first redistribute quantum information where it is needed to ensure that large EDP solutions exist. It is straightforward to construct a long-range move of a data qubit to an ancilla in depth 2 from a long-range CNOT gate, by performing the CNOT gate targeting a $|0\rangle$ ancilla state and measuring the source in the X basis up to Pauli corrections. It is also straightforward to adapt the EDP subroutine to perform sets of these long-range moves along operator paths, now ending at the ancilla, in depth 4. The depth to permute only a few qubits a long distance can be improved significantly by this technique. For example, a SWAP of the two corners of an $L \times L$ grid architecture takes $O(1)$ depth using long-range move operations, as opposed to $\Omega(L)$ depth using conventional SWAP gates. It remains an open question how to trade off permuting data qubits (using SWAP gates or long-range moves) and directly using long-range CNOT gates.

We have assumed that classical feedback is not present in the input circuit for clarity of presentation. EDPC can readily be extended to the setting of classical feedback in the input circuit to form a "just-in-time" surface code compilation algorithm. To do so, a larger computation would be broken up into a sequence of circuit executions without classical feedback, where prior measurement results specify the next circuit to compile and execute.

ACKNOWLEDGMENTS

We would like to thank Alexander Vaschillo and Dmitry Vasilevsky for helpful discussion, and Thomas Haner for coming up with a simple correctness proof for delayed remote execution gadgets and many helpful discussions. A slightly modified version of this proof is presented in Appendix D.

E.S. acknowledges support from a Microsoft internship, an IBM PhD Fellowship, and the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Quantum Testbed Pathfinder program (Award No. DE-SC0019040).

APPENDIX A: SURFACE CODE ARCHITECTURE

Here we review some basic details of the surface code, focusing on the elementary logical operations shown in Fig. 1. This is intended as a high-level overview to provide some intuition of how the logical operations in Fig. 1 arise and what their resource costs are. For more thorough reviews of surface codes, see Refs. [3,57,58].

To implement the surface code, we assume that *physical qubits* are laid out on the vertices of a 2D grid, with nearest-neighbor interactions allowed. For concreteness, we describe here an implementation of lattice surgery with the rotated surface code with half-moon boundary [59], although our EDPC algorithm can use other implementations. A single surface code patch encodes a single *logical qubit* in $2d^2 - 1$ physical qubits, where the odd parameter d is known as the *code distance* that corresponds to the level of noise protection; see Fig. 14(a). For clarity,

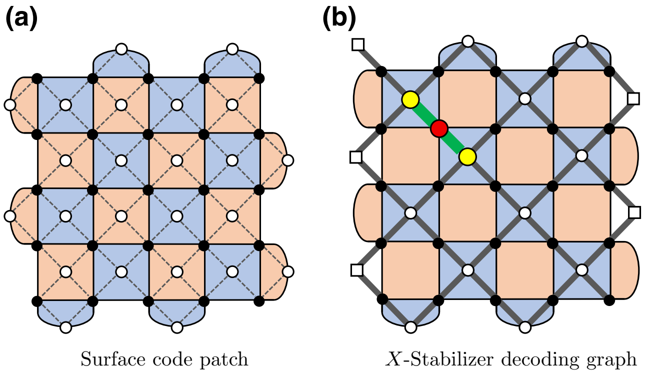


FIG. 14. (a) A $d = 5$ surface code patch implemented in a grid of data physical qubits (black disks), and ancilla physical qubits (white disks). Error correction is implemented with single-qubit operations and CNOT gates between pairs of qubits connected by a dashed edge. The Z- and X-type stabilizers are associated with alternating red and blue faces. (b) A decoding graph that is defined by associating an edge with each qubit and a vertex for each stabilizer. If stabilizers are measured perfectly, Z errors on data qubits (marked in red) can be corrected by finding a minimum weight matching (green edges) of vertices associated with unsatisfied X stabilizers (yellow disks).

within this section of the appendix we refer to physical qubits and logical qubits explicitly; however, in other sections we often drop the word "logical" when referring to logical qubits for brevity.

We designate every odd physical qubit as a *data physical qubit* in the patch, and every even physical qubit as an *ancilla physical qubit* to facilitate a *stabilizer* measurement; see Fig. 14(a). The code space of a surface code consists of those states of the data physical qubits that are simultaneous $+1$ eigenstates of the set of stabilizer generators. The stabilizer generators can be associated with faces and are either $X \otimes X \otimes X \otimes X$ or $Z \otimes Z \otimes Z \otimes Z$ operators for the bulk (interior) of the code or $X \otimes X$ or $Z \otimes Z$ operators on the boundary. We can see that the logical Z operator, Z_L, defined as any path of single-qubit Z operators on physical qubits connecting the rough boundaries, commutes with all stabilizers. Similarly, the logical X operator, X_L, is a path of X operators connecting the smooth boundaries.

For quantum error correction, it is necessary to repeatedly measure stabilizer generators. Stabilizer generators can be measured by running small circuits consisting of the preparation of the ancilla physical qubit, CNOT gates between the ancilla physical qubit, and the data physical qubits, followed by measurement of the ancilla physical qubit. Error correction can be performed by associating qubits with edges and stabilizer generators with vertices of a so-called decoding graph; see Fig. 14(b). A classical algorithm known as a decoder is used to infer a set of edges (specifying the support of the X or Z correction) given a subset of vertices (corresponding to unsatisfied Z or X stabilizers, that is, stabilizer generators with measurement outcome -1). Figure 14(b) shows an example of this in the setting of perfect stabilizer measurements, although this can be generalized to handle faulty measurements by repeating measurements.

Logical operations can be implemented fault tolerantly on logical qubits encoded in surface codes. For example, a destructive logical X measurement of a patch is implemented by measuring all data qubits in the X basis, and then using a decoder to process the physical outcomes and reliably identify the logical measurement outcome. Another important logical operation is the nondestructive measurement of a logical joint Pauli operator using an approach known as lattice surgery [7], as shown in Fig. 15(a). To simplify lattice surgery by lining up the boundary stabilizers of neighboring patches, we consider a tiling of the plane using two versions of distance d surface code patches as shown in Fig. 15(b) that forms a grid of logical qubits. Logical $Z_L \otimes Z_L$ can be measured between vertical neighbor patches while $X_L \otimes X_L$ can be measured between horizontal neighbor patches.

The allowed fault-tolerant logical operations that we assume throughout the paper and the resources they require are listed in Fig. 1. These are largely based on the rules specified in Ref. [13]. Here we justify the resource requirements for the logical operations in Fig. 1 not

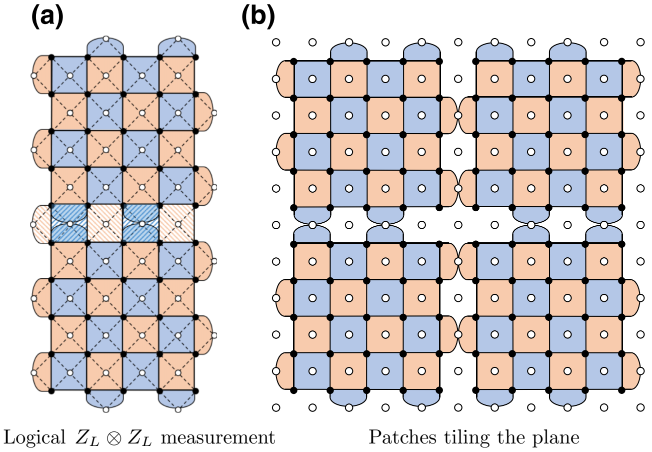


FIG. 15. (a) A logical $Z_L \otimes Z_L$ measurement is performed by lattice surgery in the following steps. (i) Stop measuring the weight-two stabilizers along the horizontal boundary between the patches. (ii) Reliably measure the bulk faces for a single vertically extended patch. Note that $Z_L \otimes Z_L$ can be inferred from the product of the outcomes of the newly measured red faces. This temporarily merges the patches to form a single extended surface code patch. (iii) Reliably measure once more the weight-two faces along the horizontal boundary between the patches. This separates the pair of patches. (b) Two types of patches tile the plane, with $Z_L \otimes Z_L$ measurements possible between vertically neighboring patches, and $X_L \otimes X_L$ measurements possible between horizontally neighboring patches.

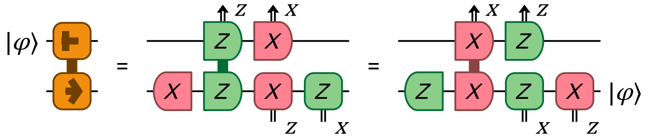


FIG. 16. The move operation can be implemented in depth 1 by local and neighboring Pauli measurements. A horizontal move operation can be implemented by preparing a single-qubit patch in |0⟩, applying joint XX measurement, and then measuring the original patch in the Z basis (up to Pauli corrections). The vertical move operation follows from applying a Hadamard gate to the source qubit $|\phi\rangle$ and a Hadamard gate on the output. Simplifying the circuit gives the right-hand side in the figure, with a ZZ measurement that is available vertically.

covered in Ref. [13] on a distance-d surface code. For space analysis, we work in units of full surface code patches such that if any qubits from a patch are needed to implement an operation, the full patch is counted. We show how to implement the operations in terms of more elementary Pauli measurements. The move operation can be implemented in depth 1 with the target qubit as ancilla, as shown in Fig. 16. The Hadamard gate can be implemented in depth 3 with three ancillae patched along with the move operation, as shown in Fig. 17. Finally, Bell measurement and preparation can be implemented in depth 1, as shown in Fig. 18.

It is worth mentioning that there is considerable freedom in the detailed choice and implementation of the surface code that could have an impact on the space-time cost of logical operations, both at the physical level but also in some cases at the logical level. For example the Hadamard gate could be performed using just one logical ancilla patch if each patch is padded with extra qubits. We do not explore these alternatives here, but note that our EDPC algorithm can still be applied if these alternatives are used.

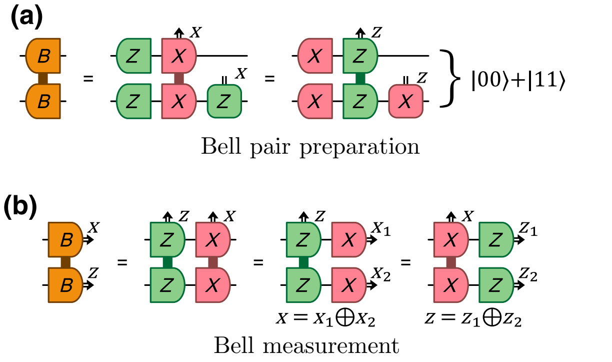


FIG. 18. We can implement Bell preparation and measurement in terms of single- and two-qubit Pauli measurements in depth 1 as given in Fig. 18 [13]. (a) A Bell pair can be prepared from a (horizontal) joint XX measurement of |00⟩ or a (vertical) joint ZZ measurement of |++⟩, up to Pauli corrections. (b) A destructive Bell measurement can be implemented by a joint XX measurement followed by individual Z basis measurements, or by a joint ZZ measurement followed by individual X basis measurements.

APPENDIX B: LOGICAL SPACE-TIME COST AS A PROXY FOR THE PHYSICAL SPACE-TIME COST

Here we provide a justification for our use of the logical space-time cost as a proxy for the physical space-time cost. As we have seen in Fig. 1 and Appendix A, logical operations implemented with the surface code require physical time that scales as d and physical space that scales as d^2. For a logical circuit written in terms of a total of A_{logical} elementary logical operations implemented using surface codes of distance d, the physical space-time cost A_{physical} is

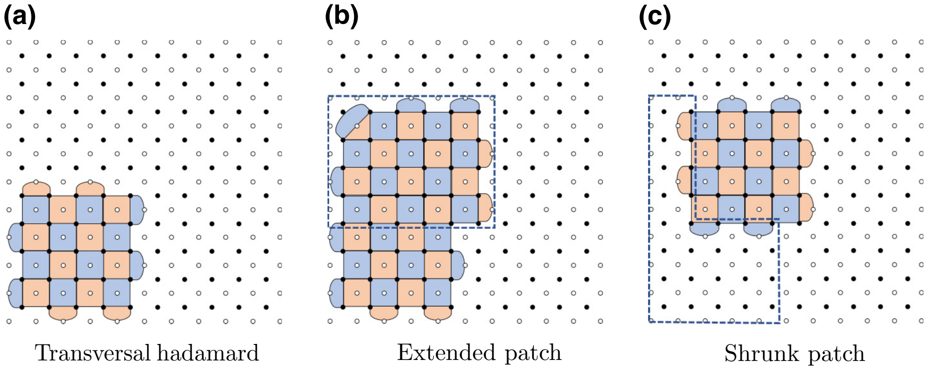
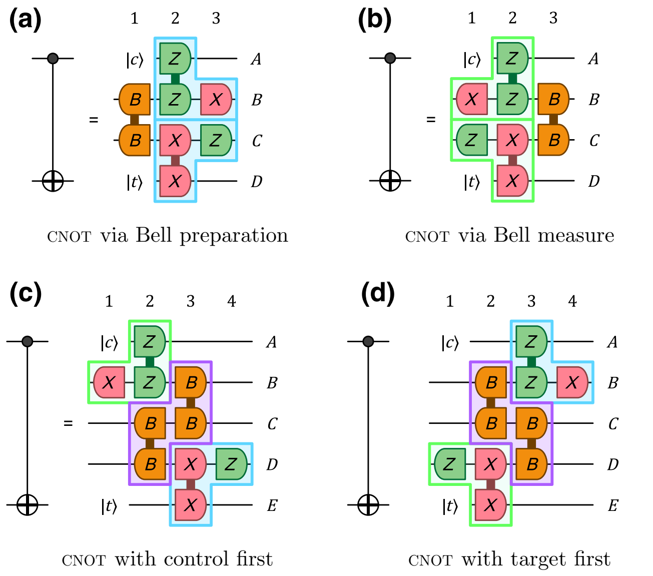


FIG. 17. Implementation of a Hadamard operation in depth 3 with three ancilla patches. (a) A transverse Hadamard operation is applied in depth 0 to each physical data qubit, which switches the arrangement of X and Z stabilizer generators compared to the standard configuration. (b) The patch is extended in depth 1 so that a segment of the standard boundary type is introduced on the right. (c) The patch is shrunk into a standard surface code patch of the form of the top-left corner of the region [see Fig. 15(b)] in depth 1, but with its location shifted by a (code distance) d-independent amount. This allows us to shift the patch into the top-left corner in 0 depth (not shown). Then we move the logical qubit to the bottom-left corner in depth 1.



CNOT via Bell preparation

CNOT via Bell measure

CNOT with control first

CNOT with target first

FIG. 19. Various implementations of a CNOT gate with intermediate ancilla qubits and Bell operations. In particular, we are able to apply the control and the target either before (green) or after (teal) Bell preparation and measurement steps, while keeping the depth at 2.

approximately

$$A_{\text{physical}} \sim A_{\text{logical}} d^3. \tag{B1}$$

The probability of any of these elementary operations resulting in a logical failure scales as $p_{\text{fail}} \sim (p/p^*)^{d/2}$, where the fixed system parameters are the physical error rate p and the fault-tolerant threshold for the surface code p^*. Moreover, we assume that $p_{\text{fail}} \sim 1/A_{\text{logical}}$ to ensure that the logical circuit is reliable with as small a code distance as possible. This suggests that the code distance behaves as

$$d \sim \frac{2 \log A_{\text{logical}}}{\log p^* - \log p}. \tag{B2}$$

Therefore we see that the physical and logical space-time costs are monotonically related, i.e.,

$$A_{\text{physical}} \sim A_{\text{logical}} (\log A_{\text{logical}})^3. \tag{B3}$$

APPENDIX C: CNOT VIA BELL OPERATIONS

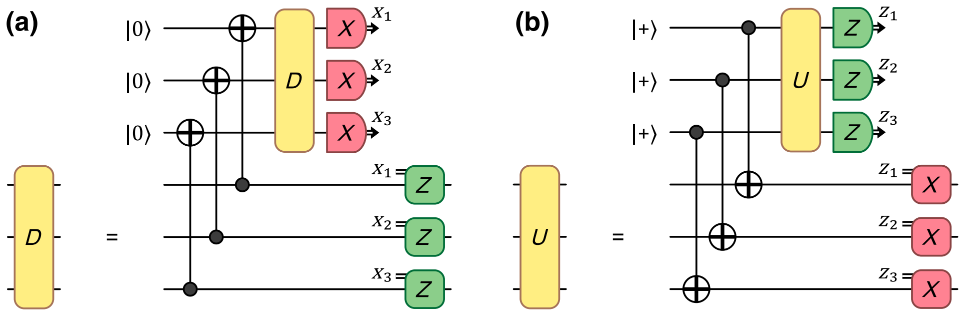
We list more variations of the standard CNOT gate [Fig. 3(a)] that use intermediate Bell preparation and measurements on ancillae in Fig. 19. By choosing the right subcircuit, we see that the long-range operations in Fig. 9 implement a CNOT gate.

APPENDIX D: REMOTE EXECUTION OF DIAGONAL GATES

A gate D diagonal on k-source qubits in the computational basis can be executed on k ancillae by first entangling these ancilla qubits using CNOT gates. We call this *remote* execution. Let the computational basis be $|\ell\rangle$ for $\ell \in [2^k]$; then $D\,|\ell\rangle = \exp(i\phi_\ell)\,|\ell\rangle$. We saw one use for remote gates in applying rotations at the boundary requiring magic states (Sec. IV).

We execute D remotely as follows (see Fig. 20). First, we initialize the ancillae in the state $|0\rangle^{\otimes k}$. Let the source qubits be in some pure state $\sum_\ell \alpha_\ell\,|\ell\rangle$ for $\alpha_\ell \in \mathbb{C}$. Then we apply k transversal CNOT gates controlled on source qubits so that the overall state becomes $\sum_\ell \alpha_\ell\,|\ell\rangle \otimes |\ell\rangle$. We now apply D to the ancillae instead:

$$(\mathbb{1} \otimes D) \sum_\ell \alpha_\ell\,|\ell\rangle \otimes |\ell\rangle = \sum_\ell \alpha_\ell \exp(i\phi_\ell)\,|\ell\rangle \otimes |\ell\rangle\,. \tag{D1}$$



Diagonal gate in computational basis

Diagonal gate in Hadamard basis

FIG. 20. (a) Any k-qubit gate diagonal in the computational basis can be remotely executed on k dedicated ancillae by first using CNOT gates. We use this technique to apply remote $Z(\theta)$ rotations [Fig. 10(b)] with magic states at the boundary. (b) Similarly, gates diagonal in the Hadamard basis also have a remote implementation. Since the Pauli corrections can be commuted through Clifford circuits, Clifford circuits can be executed immediately after executing the CNOT operations with no need to wait on the remote operations.

```
Input : Circuit $\mathcal{C}$ with Pauli gates commuted to the end and merged with
        measurement
1  while available operations in $\mathcal{C}$ :
2      execute all available state preparation, measurement, and Hadamard gates
3      $\mathcal{G}_m \leftarrow \{$available operations $g \in \mathcal{C} \mid g$ is $S, T, S_x$, or $T_x\}$
4      $\mathcal{G}_c \leftarrow \{$available operations $g \in \mathcal{C} \mid g$ is CNOT$\}$
5      while $\mathcal{G}_m \cup \mathcal{G}_c \neq \emptyset$ :
6          $G \leftarrow$ surface code grid graph
7          $\mathcal{P}_m \leftarrow$ max_rotations($\mathcal{G}_m$)          // see Algorithm 3
8          remove edges in each $P \in \mathcal{P}_m$ from $G$
9          $\mathcal{P}_c \leftarrow$ approximate max operator EDP set on $G$ with $\mathcal{G}_c$
10         execute concurrent remote rotations along $\mathcal{P}_m$ and long-range
             CNOT gates along $\mathcal{P}_c$ using EDP subroutine
11         remove executed rotations from $\mathcal{G}_m$ and CNOTs from $\mathcal{G}_c$
```

Algorithm 6. *EDPC implementation*: the EDPC algorithm (Algorithm 4) differs from our implementation in that it greedily tries to execute CNOT gates earlier.

We now disentangle the ancillae by measuring them in the X basis. Let the measurement give outcomes $\mathbf{x} \in \{0,1\}^k$; then the state on the source qubits is mapped to

$$\sum_{\ell} \alpha_\ell \exp(i\phi_\ell)(-1)^{(\mathbf{x},\ell)} |\ell\rangle , \tag{D2}$$

where $(\mathbf{x},\ell)$ is the inner product modulo 2 between $\mathbf{x}$ and the binary representation of ℓ. Applying a Z correction to each qubit $j \in [k]$ controlled on measurement result $\mathbf{x}_j$ maps the state to $\sum_\ell \alpha_\ell \exp(i\phi_\ell)\,|\ell\rangle$, as required.

This technique can be extended to any unitary operator U since it can be unitarily diagonalized as $U = VDV^\dagger$ by the spectral theorem, for V unitary and D diagonal operators. A particularly simple case are unitary operators that are diagonal in the Hadamard basis, where $V = H^{\otimes k}$. We write $U = H^{\otimes k}DH^{\otimes k}$ on the source qubits and apply remote execution of D using our techniques above. We then simplify the circuit to obtain Fig. 20(b).

APPENDIX E: EDPC IMPLEMENTATION

Here we provide Algorithm 6, which specifies the implementation of EDPC used for our numerical results presented in Sec. VI D, here called EDPCI for clarity. EDPCI differs slightly from EDPC (Sec. V) and we highlight the differences. Up until line 7 of Algorithm 6, EDPCI is the same as EDPC. Then, EDPCI greedily attempts to execute long-range CNOT gates earlier than would occur in EDPC. In particular, EDPC only executes CNOT gates after all available rotations have been executed, whereas EDPCI finds a set $\mathcal{P}_c$ on line 9 such that $\mathcal{P}_c \cup \mathcal{P}_m$ forms an EDP set. Now EDPCI concurrently executes long-range CNOT gates using any edges left over from remote rotations. Moreover, we note that EDPC uses the bounded $\mathcal{T}$-operator set algorithm (Algorithm 2) to execute a parallel CNOT circuit, which additionally finds a bounded $\mathcal{T}$-operator set $\mathcal{Q}_1$ on line 1, whereas EDPCI only finds $\mathcal{Q}_2$ from the bounded $\mathcal{T}$-operator set algorithm if given a parallel CNOT circuit. As a consequence of this difference, while a parallel input CNOT circuit is guaranteed to compile to an output circuit whose depths is upper bounded by $O(\sqrt{n})$, EDPCI does not have this guarantee.

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

[2] S. B. Bravyi and A. Yu. Kitaev, Quantum codes on a lattice with boundary, Nov. 20, 1998. ArXiv:quant-ph/9811052v1. Pre-published.

[3] Austin. G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland, Surface codes: towards practical large-scale quantum computation, Phys. Rev. A **86**, 032324 (2012).

[4] Christopher Chamberland, Guanyu Zhu, Theodore J. Yoder, Jared B. Hertzberg, and Andrew W. Cross, Topological and Subsystem Codes on Low-Degree Graphs with Flag Qubits, Phys. Rev. X **10**, 011022 (2020).

[5] Torsten Karzig, C. Knapp, R. M. Lutchyn, P. Bonderson, M. B. Hastings, C. Nayak, J. Alicea, K. Flensberg, S. Plugge, Y. Oreg, and C. M. Marcus, Scalable designs for quasiparticle-poisoning-protected topological quantum computation with Majorana zero modes, Phys. Rev. B **95**, 235305 (2017).

[6] David S. Wang, Austin G. Fowler, and Lloyd C. L. Hollenberg, Surface code quantum computing with error rates over 1%, Phys. Rev. A **83**, 020302(R) (2011).

[7] Clare Horsman, Austin G. Fowler, Simon Devitt, and Rodney Van Meter, Surface code quantum computing by lattice surgery, New J. Phys. **14**, 123011 (2012).

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

[9] Rui Chao, Michael E. Beverland, Nicolas Delfosse, and Jeongwan Haah, Optimization of the surface code design for Majorana-based qubits, Quantum **4**, 352 (2020).

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

[11] Amir Fruchtman and Iris Choi, Technical roadmap for fault-tolerant quantum computing. Research rep. University of Oxford, Sept. 2016, https://nqit.ox.ac.uk/content/technical-roadmap-faulttolerant-quantum-computing.html (visited on 09/15/2021).

[12] Daniel Litinski and Felix von Oppen, Braiding by Majorana tracking and long-range CNOT gates with color codes, Phys. Rev. B **96**, 205413 (2017).

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

[14] Andrew M. Childs, Eddie Schoute, and Cem M. Unsal, in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, Ed. by Wim van Dam and Laura Mancinska. Vol. 135. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, p. 3:1, ISBN: 978-3-95977-112-2.

[15] H.-J. Briegel, W. Dür, J. I. Cirac, and P. Zoller, Quantum Repeaters: The Role of Imperfect Local Operations in Quantum Communication, Phys. Rev. Lett. **81**, 5932 (1998).

[16] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R. Brown, Margaret Martonosi, and Frederic T. Chong, in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, Oct. 2017.

[17] L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, and C. G. Almudever, Mapping of lattice surgery-based quantum circuits on surface code architectures, Quantum Sci. Technol. **4**, 015005 (2018).

[18] Prakash Murali, Jonathan M. Baker, Ali Javadi Abhari, Frederic T. Chong, and Margaret Martonosi, in *ASPLOS '19: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence RI, United States, Apr. 13–17, 2019). Ed. by Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck. New York NY, United States: The Association for Computing Machinery, Jan. 30, 2019, pp. 1015–1029, p. 1015, ISBN: 978-1-4503-6240-5.

[19] Alwin Zulehner and Robert Wille, Compiling SU(4) quantum circuits to IBM QX architectures (ACM Press, New York, NY, USA), p. 185, ISBN: 978-1-4503-6007-4.

[20] Damian S. Steiger, Thomas Häner, and Matthias Troyer, Advantages of a modular high-level quantum programming framework, Microprocess. Microsyst. **66**, 81 (2019).

[21] Noga Alon, F. R. K. Chung, and R. L. Graham, Routing permutations on graphs via matchings, SIAM J. Discrete Math. **7**, 513 (1994).

[22] Indranil Banerjee and Dana Richards, in *Fundamentals of Computation Theory*. FCT: International Symposium on Fundamentals of Computation Theory (Bordeaux, France, Sept. 11–13, 2017). Ed. by Ralf Klasing and Marc Zeitoun. Lecture Notes in Computer Science 10472. Berlin, Germany: Springer, 2017, p. 69.

[23] Debbie Leung, Jonathan Oppenheim, and Andreas Winter, Quantum network communication—the butterfly and beyond, IEEE Trans. Inf. Theory **56**, 3478 (2010).

[24] Hirotada Kobayashi, François Le Gall, Harumichi Nishimura, and Martin Rötteler, in *Automata, Languages and Programming* (Springer Berlin Heidelberg, 2009), p. 622.

[25] Hirotada Kobayashi, François Le Gall, Harumichi Nishimura, and Martin Rötteler, in *2011 IEEE International Symposium on Information Theory Proceedings* (IEEE, July 2011).

[26] Takahiko Satoh, François Le Gall, and Hiroshi Imai, Quantum network coding for quantum repeaters, Phys. Rev. A **86**, 032331 (2012).

[27] Frederik Hahn, A. Pappa, and Jens Eisert, Quantum network routing and local complementation, npj Quantum Inf. **5**, 1 (2019).

[28] Niel de Beaudrap and Steven Herbert, Quantum linear network coding for entanglement distribution in restricted architectures, Quantum **4**, 356 (2020).

[29] Jon M. Kleinberg, *Approximation algorithms for disjoint paths problems*. PhD thesis, MIT EECS, May 1996, http://hdl.handle.net/1721.1/11013.

[30] L. R. Ford and D. R. Fulkerson, Maximal flow through a network, Can. J. Math. **8**, 399 (1956).

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

[32] Oded Zilberberg, Bernd Braunecker, and Daniel Loss, Controlled-NOT gate for multiparticle qubits and topological quantum computation based on parity measurements, Phys. Rev. A **77**, 012327 (2008).

[33] Later we consider a modification of the square grid graph because our algorithms require some further restrictions on the paths, for example preventing them from passing through those vertices associated with data qubits. It is unclear if all of the results in this section also apply for these modified graphs.

[34] Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat, in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing - STOC 2018* (ACM Press, 2018).

[35] Stavros G. Kolliopoulos and Clifford Stein, Approximating disjoint-path problems using packing integer programs, Math. Program. **99**, 63 (2004).

[36] Jon Kleinberg and Éva Tardos, *In Algorithm Design* (Boston, Pearson/Addison-Wesley, 2006), 1st ed., Chap. 11, ISBN: 0321295358.

[37] Julia Chuzhoy and David H. K. Kim, in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, Ed. by Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim. Vol. 40. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, p. 187. ISBN: 978-3-939897-89-7.

[38] Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd, An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow, Theory Comput. **2**, 137 (2006).

[39] Yonatan Aumann and Yuval Rabani, in *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, 3600 University City Science Center, Philadelphia, PA, United States: Society for Industrial and Applied Mathematics, 1995, p. 567. ISBN: 0898713498.

[40] J. Kleinberg and E. Tardos, in *Proceedings of IEEE 36th Annual Foundations of Computer Science* (IEEE Comput. Soc. Press, 1995).

[41] It may be possible to improve the runtime by using a decremental dynamic all-pair shortest path algorithm; it may be quicker to maintain a data structure for all shortest paths that can quickly be updated when edges are removed.

[42] Nicolas Delfosse, Michael E. Beverland, and Maxime A. Tremblay, Bounds on stabilizer measurement circuits and obstructions to local implementations of quantum LDPC codes (2021), ArXiv:2109.14599.

[43] Aniruddha Bapat, Andrew M. Childs, Alexey V. Gorshkov, and Eddie Schoute, Advantages and limitations of quantum routing. In preparation.

[44] C. H. Bennett, A. W. Harrow, D. W. Leung, and J. A. Smolin, On the capacities of bipartite Hamiltonians and unitary gates, IEEE Trans. Inf. Theory **49**, 1895 (2003).

[45] David S. Johnson, Approximation algorithms for combinatorial problems, J. Comput. Syst. Sci. **9**, 256 (1974).

[46] E. Knill, Fault-tolerant postselected quantum computation: schemes, Feb. 2004. ArXiv:quant-ph/0402171.

[47] Jon Kleinberg and Éva Tardos, in *Algorithm Design* (Boston, Pearson/Addison-Wesley, 2006), 1st ed., Chap. 7, ISBN: 0321295358.

[48] Neil J. Ross and Peter Selinger, Optimal ancilla-free Clifford+T approximation of z-rotations, Quantum Inf. Comput. **16**, 901 (2016).

[49] Christopher Chamberland and Earl T. Campbell, Universal quantum computing with twist-free and temporally encoded lattice surgery, (2021) ArXiv:2109.02746.

[50] In the scheme presented in Ref. [13] only one joint Pauli measurement is needed per T gate, but additional features are required of the surface code such as twist defects that were avoided in Ref. [49], and which we have avoided in this paper.

[51] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung, Network information flow, IEEE Trans. Inf. Theory **46**, 1204 (2000).

[52] April Rasala Lehman and Eric Lehman, in *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2004, p. 142. ISBN: 089871558X.

[53] Cody Jones, Low-overhead constructions for the fault-tolerant Toffoli gate, Phys. Rev. A **87**, 022328 (2013).

[54] Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer, Elucidating reaction mechanisms on quantum computers, Proc. Nat. Acad. Sci. **114**, 7555 (2017).

[55] Craig Gidney and Martin Ekerå, How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits, Quantum **5**, 433 (2021).

[56] Craig Gidney and Austin G. Fowler, Flexible layout of surface code computations using AutoCCZ states, May 22, 2019. ArXiv:1905.08916.

[57] H. Bombin, in *Quantum Error Correction*, edited by D. A. Lidar and T. A. Brun. Cambridge: Cambridge University Press, Nov. 1, 2013. ISBN: 9780521897877. ArXiv:1311.0277.

[58] Benjamin J. Brown, Katharina Laubscher, Markus S. Kesselring, and James R. Wootton, Poking Holes and Cutting Corners to Achieve Clifford Gates With the Surface Code, Phys. Rev. X **7**, 021029 (2017).

[59] Andrew J. Landahl and Ciaran Ryan-Anderson, Quantum computing by color-code lattice surgery (2014), July 18, 2014. ArXiv:1407.5103.