

Universal Graph Representation of Stabilizer Codes

Andrey Boris Khesin^{*,†}, Jonathan Z. Lu[‡], and Peter W. Shor

Department of Mathematics, *Massachusetts Institute of Technology*, Cambridge, Massachusetts 02139, USA



(Received 14 May 2025; revised 11 August 2025; accepted 25 September 2025; published 5 November 2025)

While stabilizer tableaux have proven useful as a descriptive tool for additive quantum codes, they otherwise offer little guidance for concrete constructions or algorithm analysis. We introduce a representation of stabilizer codes as graphs with certain structures, and prove via the ZX calculus that this representation is related to stabilizer tableaux by an efficiently computable bijection. This gives a new universal recipe for code construction by way of finding graphs with nice properties. The graph representation gives insight into both code construction and algorithms. We construct as examples families of $[[n, \Theta(\frac{n}{\log n}), \Theta(\log n)]]$ and $[[n, \Omega(n^{4/5}), \Theta(n^{1/5})]]$ codes. We use graphs in a probabilistic analysis to extend the quantum Gilbert-Varshamov bound into a three-way distance-rate-weight trade-off. Moreover, code properties such as distance and encoding circuit depth are bounded by simple functions of the graph degree. We prove that key coding algorithms—distance approximation, minimum-weight generator selection, and decoding—are unified as instances of one optimization game on a graph. By studying this game, we construct an efficient greedy decoder and prove that it corrects all recoverable errors for all graphs with cycle lengths no shorter than 13 (reducible to 5 with mild extra constraints); these include the above two families. Our results suggest that graphs are generically useful for the study of stabilizer codes.

DOI: [10.1103/1gjs-2rhx](https://doi.org/10.1103/1gjs-2rhx)

I. INTRODUCTION

Most quantum algorithms which are believed to provide significant speedups to certain computational problems require both many qubits and error correction to succeed in practice [1–8]. As advancements in the experimental scaling of quantum computers have steadily marched forward, the problem of designing good quantum codes for the practical implementation of various quantum algorithms has in turn become of more pressing interest [9–13].

With a few exceptions, stabilizer codes have emerged as one of the most intensely studied families of codes due to their simple description [8, 14–19]. Roughly speaking, stabilizer codes are the most general quantum analog of linear codes for classical computing. More precisely, a $[[n, k]]$ stabilizer code encodes k logical qubits to n physical qubits by embedding 2^k logical degrees of freedom into a subspace

defined as the simultaneous $+1$ eigenspace of $n - k$ independent commuting n -qubit Pauli operators, known as the *stabilizers*. Stabilizers comprise a considerably large class of quantum codes, including all Calderbank-Shor-Steane (CSS) codes [20, 21].

Generally, a stabilizer code is specified by the *stabilizer tableau*, the $n - k$ Pauli operator strings which define the code space. This description is simple and elegant, and has led to remarkable results such as the Gottesman-Knill theorem [22], which efficiently simulates any Clifford operation classically, and the quantum Gilbert-Varshamov bound [8], which proves by a probabilistic technique that asymptotically good stabilizer codes exist. At the same time, the stabilizer tableau representation is prohibitive in many other ways. Without further structure, it is unclear as to how one might construct desirable codes by simply writing down a clever collection of $n - k$ Pauli strings, or how one might devise and analyze coding algorithms for a given tableau. For this reason, much of the constructive effort in quantum coding theory has relied on developing sophisticated CSS-type product operations that generate CSS codes from nice classical codes [23, 24]. Moreover, algorithms to, e.g., decode these constructions typically rely on reducing the problem to a classical code decoding problem, and then relying on classical decoders [21, 23, 25, 26].

In this paper, we argue that a different approach based on graphs to represent stabilizer codes may also prove use-

*Contact author: khesin@mit.edu

†Presently at Department of Computer Science, University of Oxford, Oxford OX1 3QG, United Kingdom.

‡Contact author: lujz@mit.edu

Published by the American Physical Society under the terms of the [Creative Commons Attribution 4.0 International](https://creativecommons.org/licenses/by/4.0/) license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

ful for the development and analysis of quantum codes. The employment of graphs in code construction has a rich history in coding theory. Classical codes are described by a $(n - k) \times n$ binary matrix H the kernel of which over $\mathbb{F}_2$ defines the code space. Just as it is not obvious which tables of Paulis constitute good stabilizer codes, so too is it *a priori* unclear which tables of bits correspond to classical codes. Consequently, graph representations such as Tanner graphs, which encode H into a bipartite graph, have been developed to give insight into classical code construction. Tanner graphs have built the bridge between expander graphs and classical codes and have significantly contributed to modern classical coding through the utilization of graph theory insights [27]. Even in the quantum realm, graph reasoning has already enabled breakthroughs in constructions of CSS codes. Most famously, the topological structure of Euclidean lattices with periodic boundary conditions has motivated the development of the toric and other surface codes [28,29]. The toric code has even been shown to be efficiently decoded by way of classical graph algorithms; in particular, Edmonds's minimum-weight perfect matching [30,31]. Outside of quantum coding, tensor networks have played a key role in quantum simulation and algorithm design.

The critical contributions of graphs in both classical and quantum coding in many specific constructive cases naturally lead to the question of whether one can use graph theory to study additive quantum codes in the most general sense. That is, can we devise graph techniques to reason about quantum stabilizer codes in an inherently general quantum manner, without having to restrict ourselves to CSS codes and thereby reduce the problem to classical codes? This paper takes a step toward answering this question by exploring (a) what the most general graph techniques on stabilizer codes are without relying on classical codes and CSS products and (b) how useful such a representation is for a variety of important coding tasks.

A. Contributions, scope, and conjectures

Our first contribution, in the direction of the first question, is the development of a simple graph representation of all stabilizer codes. The general structure of the graph is known as an *encoder-respecting form*, which has the structure of a semibipartite graph that maps k input-representing nodes to n output-representing nodes. By semibipartite, we mean that inputs may not be connected to each other while outputs generically may. Informally, this result may be stated as follows.

Theorem 1 (Informal). Every stabilizer code is essentially uniquely represented by an encoder-respecting graph satisfying some rules, and conversely every such graph essentially uniquely represents a stabilizer code. There is an efficient compilation algorithm that maps tableaus to graphs in $O(n^3)$ time and vice versa in $O(n^2)$ time.

Here, the term “essentially” masks a more exact statement in which we ignore by local equivalences that do not affect any code properties. The precise version of Theorem 1 is given in Secs. II and IV. This result establishes that our graph representation is sufficiently general to universally represent all stabilizer codes. We have devised the compilation algorithm from tableaus to graphs in part for the study of graph properties of well-known codes. As examples, we compile the famous five-qubit, seven-qubit, and nine-qubit quantum codes and show that each take simple geometric forms, respectively, of a cone, cube, and tree-star. Conversely, Theorem 1 enables us to essentially transform any semibipartite graph into a stabilizer code, in such a way that (a) any stabilizer code can be constructed this way and (b) the properties of the code are tied to the properties of the graph. As a consequence, we can leverage the tools of graph theory and graph algorithms to study stabilizer codes. We show that the key properties of a stabilizer code are controlled in a unified way, namely by the degree of the corresponding graph.

Theorem 2 (Informal). Let G be a graph and let $C(G)$ be the corresponding code. The following hold:

- (1) The distance of $C(G)$ is bounded above by roughly the minimum degree of G .
- (2) If G is bipartite, then $C(G)$ is a CSS code.
- (3) The stabilizer tableau produced by the algorithm in Theorem 1 has weights bounded by a quadratic function of the degree of certain nodes in G .
- (4) There exists an efficient algorithm which produces an encoding circuit for $C(G)$ the depth of which is bounded by (up to a small additive constant) twice the maximum degree of G .
- (5) Every constant-depth diagonal gate and the $\sqrt{X}$ can be implemented logically in $C(G)$ with a circuit of depth, respectively, about (up to an additive constant) twice the minimum degree and the maximum degree.

The formal versions of these statements are given in Sec. IV. We emphasize that our universal graph representation is fully distinct from Tanner graph representations, even in the CSS case when the graph is bipartite. For example, the Tanner graph of a code with a large distance but a low check weight, such as the toric code, would have a very low degree, while condition (1) above requires that a graph using our construction for the same code would have a high degree.

To further strengthen the connection between graphs and codes, we next define a simple, one-player optimization-type game on a graph which we call quantum lights out (QLO), in Sec. IV. We prove that QLO unifies many important tasks in quantum coding, which opens a path toward the study of coding algorithms via their correspondence to approximately optimal QLO strategies.

Theorem 3 (Informal). Many coding algorithms such as distance approximation, finding minimum-weight stabilizer generators, and decoding are all approximately optimal strategies in the QLO game.

The above results demonstrate that graphs in many senses unify various aspects of stabilizer coding in a simple manner. Overall, we give four pieces of evidence that graphs serve as a useful representation. In the first, which we have discussed above, we show that graphs enable efficient algorithms which produce encoding circuits and logical operators the depths and weights of which are bounded by the degree. Later on, when we study decoding, we are able to bound the depths directly by the distance of the code for graphs which satisfy a certain property.

In the second aspect, we explore the geometric and topological properties of graphs as tools for building codes. Noticing that simple geometric solids realize celebrated quantum codes, we construct codes based on other such solids, including platonic solids and topological transformations of them. For simpler constructions, we are able to calculate the distance by direct computation. As an example, we produce a non-CSS stabilizer code from a dodecahedron, which has parameters $[[16, 4, 3]]$. We also construct a $[[54, 6, 5]]$ code by lifting the icosahedron to a covering space. For more complicated constructions, we use more indirect analytical techniques based on decoding. The main advantage of graphs as a constructive tool is their flexibility: we show that graphs enable us to more easily produce codes that have a desired numerical distance, by finding graphs which have a similar degree. While the degree alone does not guarantee a distance comparable to the degree, adding additional structural constraints to the graph can achieve a distance comparable to the degree. Graphs also can be chosen to immediately satisfy desired experimental constraints based on locality, geometric structure, etc. Thus, the graph representation may prove helpful tools for code construction at constant-size scales fit for experimental use.

Our third avenue is decoding. In general, we believe that the graph representation serves as an excellent guide for the design of decoding algorithms. Historical evidence of minimum-weight perfect matching on the surface code suggests that graphs are indeed helpful. In this paper, we take a first step in that direction by designing a decoder that employs a simple greedy strategy on the QLO game. We give a general condition on graphs, which we call its *sensitivity*, that sufficiently characterizes the amenity of a graph to be decoded well greedily. For concreteness, we also design a simple family of $[[m2^m/(m+1), 2^m/(m+1)]]$ codes, which we call the hypercube codes. Their distance is provably at most m , but we show that the decoder can correct at least $\lfloor (m-3)/4 \rfloor$ errors on it, so that the distance is also at least $\lfloor (m-1)/2 \rfloor$. We conjecture, however, that a more careful analysis can prove that the distance is

actually m . A small case of the hypercube code family is $[[112, 16, 7]]$. Moreover, our results imply a lower bound on the distance in terms of the degree, which in turn lead to bounds on encoding circuit depth, logical-operator circuit depth, and stabilizer weight entirely in terms of distance.

Although the sensitivity is a natural condition for greedily decodable codes, it is a complicated definition that is difficult to build intuition for the purposes of code construction. To overcome this issue and build a stronger bridge between our work and the main branches of graph theory, we prove in Lemma 5 that any graph with girth (i.e., shortest cycle length) at least 13 is minimally sensitive and thus optimally decodable by the greedy decoder. In the same theorem, we prove parallel results for graphs with smaller girths (as small as 5) which satisfy some additional constraints, such as regularity and having certain nodes be sufficiently far from each other. Consequently, such graphs have a provable distance lower and upper bound. It is interesting that such a condition on a global property has no dependence at all on n , in contrast to classical Tanner graphs which require expansion, a property that explicitly depends on n . We are further able to show under certain conditions that the distance is exactly the minimum degree. The connection between girth and decodability is particularly interesting because it enables us to leverage results from extremal graph theory to construct codes. In particular, we use a well-known construction of girth-12 high-degree graphs [32] to build a family of $[[n, \Omega(n^{4/5}), \Theta(n^{1/5})]]$ codes in Theorem 22. These constructions in many senses maximally exploit the capabilities of the greedy QLO decoder.

Because QLO and the greedy decoder are fully general to all stabilizer codes, they allow us to reason about decoding in an inherently quantum manner, as opposed to reducing the decoding problem to classical decoding in the CSS prescription. As such, this approach opens the door to the study of coding algorithms for *non*-CSS codes, including codes which are not even locally equivalent to a CSS code. At the same time, because QLO is general, it is also helpful as a new way to study CSS decoding without finding the underlying classical codes and appealing to classical coding theory. Even with the toric code, which is CSS, decoding is often performed by minimum-weight perfect matching, an approach from graph theory, rather than studying the classical constituent codes. In our case, as an example, the $[[n, \Omega(n^{4/5}), \Theta(n^{1/5})]]$ code turns out to be CSS but is most easily decoded directly by the greedy QLO strategy rather than classical reductions.

Finally, we exploit the structure of graphs to give a more fine-grained argument about random codes. By randomly connecting edges of our graphs, we give in Theorem 25 a large class of codes for which almost all members satisfy a trade-off between rate, distance, and stabilizer weight. More precisely, for a chosen number of physical qubits n , rate $R = k/n$ (where k is the number of logical qubits), and

parameter d , most codes in the family have distance at least d . Furthermore, every code in the family has rate R and check weight at most $\sim R/(1-R)d \log n$, where check weight is the maximum weight of any stabilizer generator. This extends the result of the quantum Gilbert-Varshamov bound, which shows by a coarse-grained probabilistic stabilizer tableau analysis a similar result with a rate-distance trade-off but always near-maximal stabilizer weight. Such a result further exemplifies that graphs enable greater flexibility in code analysis.

Our paper argues at a very general scope, namely at the level of all stabilizer codes. The universality of our work therefore has limitations that do not exist for highly specific quantum low-density parity-check (qLDPC) constructions. In particular, even when optimized as much as possible with known techniques, code properties such as check weight, encoding circuit depth, and logical-operator circuit depth will scale with the degree and therefore the code parameters, though not necessarily by a great deal (e.g., logarithmic). Thus, our intention is not to use the graph representation as a means for continued study into asymptotically good qLDPC codes with very large constants in the manner of Refs. [33–36]. Rather, we consider our work to be complementary to qLDPC in the sense that we give general insight into stabilizer codes and new flexible ways to construct codes that may be tailored specifically for a given experimental implementation. As an example, a future quantum long-term memory device may seek a particular large numerical distance threshold but care little for comparably large check weights if the measurement error is sufficiently small. This philosophy underpins our explicit construction of codes which have desirable properties at small scales. We believe also that further applications of graph-theoretic techniques may further improve the techniques in this paper to build increasingly practical recipes for code constructions that flexibly adapt to experimental constraints.

More generally, we argue that graph theory provides a promising path forward *generally* across avenues of quantum coding theory. Recent breakthroughs in, e.g., qLDPC codes have relied on graphs in their construction to generalize the use of surface code lattices [29]. Perhaps further generalizations of these specific types of constructive techniques, as well as less universal but more fine-grained (i.e., finding certain graph constructions which map to codes in such a way that some properties scale favorably or not at all with parameters) refinements of the universal graph representation will lead to new constructions of and insights into quantum codes.

B. Related work

Graph forms have been exploited in many ways in the quantum computation literature, most generally via tensor network representations. One simple case of a graph

representation is a graph state. Hu and Khesin [37] have produced a generalized graph state representation of stabilizer states, i.e., a stabilizer code with 0 logical qubits. Our reduction from tableau to graph utilizes insights from Hu and Khesin [37], and in the special case of $n = k$ in our work, there is a transformation that reduces our tableau-to-graph map to their corresponding map. We remark that McElvanney and Backens [38], building upon Hu and Khesin [37], have derived a different unique canonical form for stabilizer states that are particularly convenient for measurement-based quantum computation. Independently, Yu *et al.* [39] have also utilized graph theory in quantum coding, albeit at a much more general level (including nonstabilizer codes) for constructive and classification purposes. Additionally, several other works have started exploring the connections between quantum codes and graphs, going in directions ranging from holography [40], code concatenation [41], and quadratic forms [42] to connections with group theory [43,44].

Parts of our work—particularly the reduction from tableau to graph—utilize rules and notation from the ZX calculus. Recent works have also explored the use of ZX calculus to diagrammatically express quantum codes. A recent work of Kissinger [45], developed independently of this work, also constructs diagrammatic forms for CSS codes. Their forms are substantially different from ours and serve a different purpose. Primarily, Kissinger establishes a correspondence between CSS codes and ZX diagrams, by nonuniquely mapping CSS codes to such diagrams and mapping a subset of the ZX diagrams (the phase-free ones) to CSS codes. The tableau-to-graph portion of our work extends the conceptual idea of diagrammatic forms inspired by ZX calculus to all stabilizer codes, but takes a significantly different approach and utilizes the result for constructive and algorithmic purposes.

More generally, ZX-calculus approaches to quantum error correction have been considered in various ways. Two relatively recent works which utilize the ZX calculus exemplify these different approaches. In the first way, Wu *et al.* [46] represent specifically graph codes with ZX-calculus diagrams. This approach, while not general to stabilizer or CSS codes, has shown promise in code analysis. The second, due to Chancellor *et al.* [47], uses the ZX calculus specifically to quantize classical codes in the spirit of CSS code construction. Both of these works give evidence that graphs enable new insights into quantum coding, both in their construction and analysis, and thus motivate our completely general theory of graph representations. Moreover, aside from our tableau-to-graph reduction algorithm which does use elements of the ZX calculus, our work builds everything essentially only from the standard definition graphs. We emphasize that the graphical diagrams of the ZX calculus require additional structure. Therefore, graph representations further simplify related previous works that also devise similar representations

based on the ZX calculus. The specific motivations, applications, and analysis in these papers are also distinct from those in this work.

Building upon a preliminary version of this work, Khesin and Li [48] have conducted further analysis of graph representations in the special case of CSS codes, wherein the graphs become bipartite. They have specifically analyzed different ways to further simplify the graph representation and the potential advantages in doing so.

II. FROM TABLEAUS TO GRAPHS

We begin by constructing the general graph form and compiling any stabilizer tableau into such a graph. More precisely, we will compile an equivalence class of encoding Clifford circuits corresponding to an equivalence class of stabilizer tableaus, and at the end of this section we discuss how to compile starting with a tableau [18]. As a starting point, we appeal to the ZX calculus, a graph language for linear maps that has become of great interest in quantum information research [45,49–53]. The ZX calculus produces visual diagrams that represent quantum states, circuits, and more. As with any formal logical system, the ZX calculus has a set of rules which may be iteratively applied to transform diagrams into equivalent diagrams. These rules are representations of identities in quantum circuits. The ZX calculus has become of more interest than ever in fault-tolerant quantum computation and quantum compiler theory because it can explicitly visualize quantum properties such as entanglement in an intuitive manner. It has recently been applied to a host of quantum computation problems, including lattice surgery [54] and quantum optimization [55]. Importantly, the ZX calculus is, for stabilizer tableaus, complete (equalities of tableaus can be derived from corresponding ZX diagrams), sound (vice versa), and universal (every quantum operation can be expressed in the ZX calculus) [49,56].

We are consequently motivated to leverage the ZX calculus as an intermediate step in the development of a graph presentation. Ultimately, our graphs will be simple graphs and not the more complicated and structured diagrams in the ZX calculus. The diagrams that we map our circuits into are a subset of ZX diagrams that are equivalent by a simple transformation to graphs. To more clearly elucidate the transformation from tableau to graph, we construct the map from a composition of simpler maps. In particular, we first convert the tableau into a circuit. Next, we show that up to an equivalence relation, we can efficiently biject the circuits into a class of ZX diagrams of a certain form that satisfies four rules. We thus denote these diagrams *ZX canonical forms* (ZXCFs). Lastly, we remove unnecessary structure from ZXCFs to obtain a graph representation of the code. The set of transformations that we can efficiently

perform is pictorially represented by the following maps:

$$\text{Tableau} \longleftrightarrow \text{Encoder} \longleftrightarrow \text{ZXCF} \xrightarrow{\text{LE}} \text{Graph}. \quad (1)$$

In the absence of the last step, everything is completely invertible and has an interpretation as a unique compiler. For the last step, the map between ZXCFs and graphs is invertible modulo *local equivalence* (LE); i.e., modulo local (one-qubit) Clifford operations on the outputs. An immediate consequence for this ZXCF compiler is a diagrammatic method of testing equality between stabilizer codes.

We begin the formalism by noting that properties of a code depend only on the code space, and thus there is a unitary degree of freedom on the logical space. In particular, encoding circuits for a given stabilizer tableau will produce the same stabilizer code if and only if they have the same image. We therefore make the following definition.

Definition 1. Two circuits are equivalent as *Clifford encoders* if they have the same image over all possible input states. Equivalently, $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent encoders if there exists a unitary U acting on the inputs of the circuit such that $\mathcal{C}_1 = \mathcal{C}_2 U$ or $\mathcal{C}_2 = \mathcal{C}_1 U$.

Any reference we make to encoding circuits will either implicitly or explicitly refer to them up to this equivalence, and indeed they all map to the same stabilizer code.

A. ZX canonical form construction

In the intermediate steps using ZX-calculus rules, we follow the standard notation of ZX-calculus graph diagrammatics, specified in Backens [56]; we refer the reader to that work for the basics of the ZX calculus and transformation rules.

A ZX diagram is a graph with additional structure. In particular, nodes may be either red or green, nodes may be decorated with local Clifford operators, and edges may be Hadamarded. Additionally, nodes may have a “free edge,” an edge connected only to one node and dangling on the other side. Green nodes are associated with Z operators, and red with X . Nodes may have phases, corresponding to an application of $R_Z(\theta) = e^{-i\frac{\theta}{2}Z}$ for green nodes and $R_X = e^{-i\frac{\theta}{2}X}$ for red nodes. We restrict phases to multiples of $\pi/2$ to keep our operations Clifford. We color an edge blue and make it dotted if it has a Hadamard gate on it [57] (some papers, e.g., Ref. [56], instead use a line with a box and we occasionally also use this convention when convenient). The circuit takes k input qubits to n output qubits, for $0 \leq k \leq n$.

The specific structure of our graphs will be those in the following form.

Definition 2. An *encoder-respecting form* $\mathcal{E}$ has only green (Z) nodes, and is structured as a *semibipartite graph*.

A semibipartite graph has a left and right cluster, such that left-cluster nodes may have edges only to right-cluster nodes, but right-cluster nodes have no such restrictions. In our case, the left and right clusters are denoted as the input and output clusters, respectively. The input cluster $\mathcal{I}$ has k nodes associated with the k input qubits of the corresponding encoder, and the output cluster $\tilde{\mathcal{O}}$ has n nodes, where $n \geq k$. Each input node has a free (not connected to any other nodes) edge—the input edge. Similarly, each output node has a free output edge. The output edges are numbered from 1 to n —in order from left to right on an incomplete stabilizer tableau or top to bottom on Clifford circuit output wires. For convenience, we refer to the node v connected to an output edge numbered i as the output node numbered i .

The design of the encoder-respecting form graphically illustrates how information propagates from input to output (which edges connect $\mathcal{I}$ to $\tilde{\mathcal{O}}$) as well as the entanglement structure (which edges connect $\tilde{\mathcal{O}}$ to $\tilde{\mathcal{O}}$) of the underlying encoder. We emphasize that this idea is only for intuition, as there exist equivalence transformation rules on ZX diagrams that appear to change the connectivity in nontrivial ways but nonetheless represent the same operator. Note that the structure of $\mathcal{E}$ gives rise to a natural binary “partial adjacency matrix” $M_{\mathcal{E}}$ of size $k \times n$, which describes the edges between $\mathcal{I}$ and $\tilde{\mathcal{O}}$ akin to the biadjacency matrix of bipartite graphs.

Although this structure appears intuitively appealing, it is insufficient because there exist several equivalence transformations in the ZX calculus that yield an equivalent (but not obviously so) diagram. We therefore constrain an encoder-respecting form into a ZXCF via four additional rules. For notational convenience, we will interchangeably refer to applying an operation on the free edge of a node as applying an operation on the node:

- (1) *Edge rule.* The ZXCF must have exactly one Z node per free edge and every internal edge must have a Hadamard on it.
- (2) *Hadamard rule.* No output node v may both have a Hadamard gate on its output edge and have edges connecting v to lower-numbered nodes or input nodes.
- (3) *RREF rule.* $M_{\mathcal{E}}$ must be in reduced row-echelon form (RREF).
- (4) *Clifford rule.* Let $\mathcal{P} \subseteq \tilde{\mathcal{O}}$ be the nodes associated with the pivot columns of the RREF matrix $M_{\mathcal{E}}$, so that $|\mathcal{P}| = k$. The only local Cliffords allowed on nodes are $\{I, S, Z, SZ, H, HZ\}$, but there can be no operations at all on input or pivot nodes. There can be no input-input edges or pivot-pivot edges.

We have elected for emphasis to include the Z -nodes-only constraint in the edge rule and the no input-input

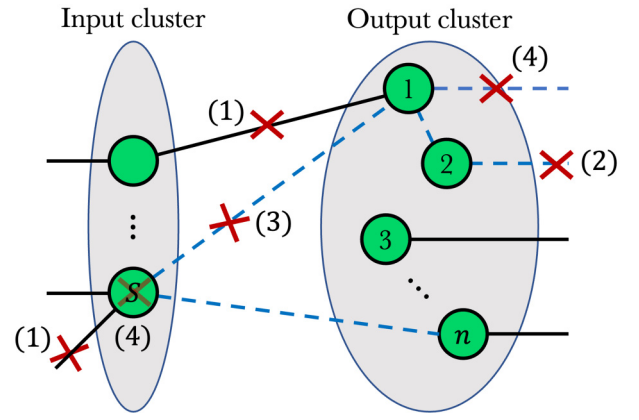


FIG. 1. An example of an encoder-respecting form and some ways in which it might violate the four rules. Recall that dotted blue edges indicate the presence of a Hadamard on that edge. Rule 1, the edge rule, is violated in two places: one of the internal edges does not contain a Hadamard gate and a Z does not have exactly one free edge. Rule 2, the Hadamard rule, is violated where the second output node has a Hadamard on its free edge, but since the node is connected to the lower-numbered first output node, this is not acceptable. Rule 3, the RREF rule, is violated where $M_{\mathcal{E}}$ is not in reduced row-echelon form, as we see that the first input is connected to the first output, meaning that no other inputs should be connected to that first output node. Rule 4, the Clifford rule, is violated twice: first in having a local Clifford operation, S , on an input node, and then where an H is placed on the first output node, which is a node associated with a pivot column of $M_{\mathcal{E}}$.

edges constraint in the Clifford rule even though these are redundant with that of the encoder-respecting form.

To provide some visual intuition on these four rules, in Fig. 1 we depict a generic example of some possible violations to the rules. We define the four rules such that the ZXCF gives a unique canonical form, is homogeneous in node type, and is minimal in terms of decorated operators and the number of vertices which do not correspond to a physical or logical qubit. It can be shown [56,58] that each of these rules corresponds to certain equivalence transformations. For example, row operations on $M_{\mathcal{D}}$ correspond to unitaries on the input. Rather than give all the rules, we will directly devise an encoder to ZXCF map and prove that such a map is efficient and bijective.

Theorem 4. Any Clifford encoder has a unique equivalent ZXCF satisfying the edge, Hadamard, RREF, and Clifford rules. There exists an algorithm, running in worst-case $O(n^3)$ time, that on an input of a Clifford encoder circuit outputs a corresponding ZXCF. This map is bijective up to encoder equivalence.

We briefly sketch the proof idea of Theorem 4 but defer the details to Appendix A. The proof involves a series of transformations via ZX equivalence rules. In the first step, we apply an isomorphism between circuits and states, and

proceed to apply a result of Hu and Khesin [37] to preliminarily represent the circuit as the ZX diagram of a corresponding state. We then transform the diagram into one representing a circuit, and carefully choose a series of equivalence transformations that iteratively force the diagram to satisfy a rule above without at the same time violating a different rule that has previously been satisfied. The resulting product is precisely a ZXCF. In fact, because the steps are equivalence transformations, distinct encoders map to distinct ZXCFs. The only remaining aspect of the proof of Theorem 4 is whether this one-to-one map is bijective up to encoder equivalence. Since equivalent encoders will have the same stabilizer tableau, we only have to demonstrate the bijection between ZXCFs and stabilizer tableaux. We prove this claim in Sec. II B.

1. Tableau to encoder

There are numerous algorithms for mapping a stabilizer tableau into a Clifford encoding circuit [8, 59], and it does not matter which algorithm one chooses to produce an encoding circuit. If we quotient out operations which leave the code space invariant, then any such map is also bijective. We describe one such algorithm for completeness. For a $(n - k) \times n$ tableau, we begin by drawing n output wires. At each step, we first simplify the tableau by applying a Clifford operation, and then we measure out one of the qubits to remove one row and column from the tableau. Repeating inductively yields a Clifford circuit that takes k qubits to n qubits.

The procedure at each step begins by finding a Clifford operation U that turns the first stabilizer into $Z_1 = Z \otimes I \otimes \cdots \otimes I$, in the manner dictated by the Gottesman-Knill theorem. U is prepended to the circuit under construction. We then multiply the remaining rows by the first until the entire first column of the tableau has only I , with the exception of the first row. This can always be done, since the n -qubit Pauli operators in each row of the matrix must commute. We then postselect on the $+1$ result of a computational basis measurement by applying $\langle 0|$ in the reverse direction. When reading the circuit in the forward direction, this is equivalent to initializing an ancillary qubit in the $|0\rangle$ state. The effect of U and the measurement is equivalent to applying $(I + P)/2$, where P is the first row of the stabilizer tableau. This is equivalent to postselecting on the $+1$ measurement result of that stabilizer. Having measured the qubit, we remove its corresponding row and column from the tableau, and repeat on a tableau of $n - 1$ qubits and $n - k - 1$ rows, until there are no rows left, at which point the remaining k wires that have not been measured out become the input qubits in the circuit. We conclude by drawing k input wires for those qubits.

B. Proof of canonicity

We next prove that our proposed ZXCF is indeed canonical. In other words, two equivalent stabilizer

tableaus—generators of the same subspace of the n -qubit Hilbert space—will map to the same ZXCF. We proceed by counting the number of stabilizer tableaux and ZXCFs. For notational convenience, let $\bar{k} := n - k$.

First, the number of stabilizer tableaux with $\bar{k}$ stabilizers on n qubits is

$$\frac{\prod_{i=1}^{\bar{k}} 2 \times 4^n / 2^{i-1} - 2 \times 2^{i-1}}{\prod_{i=1}^{\bar{k}} 2^{\bar{k}} - 2^{i-1}} = \prod_{i=1}^{\bar{k}} \frac{2^{2n-i+2} - 2^i}{2^{\bar{k}} - 2^{i-1}}. \quad (2)$$

For each row i , there are 2×4^n possible Pauli strings (including the sign), but the requirement that they commute with previous rows divides the count by 2^{i-1} . Of these $2 \times 4^n / 2^{i-1}$ valid strings, 2^{i-1} strings are linear combinations of previous strings, with a factor of 2 for strings that differ by a sign from previous strings.

We have over-counted, however, since there are many ways to find a set of stabilizer generators for a particular tableau. In particular, when choosing a set of generators for a Clifford encoder, we have $2^{\bar{k}} - 1$ choices for the first generator (subtracting the identity). There are then $2^{\bar{k}} - 2$ ways of choosing the next element, as we cannot pick anything in the span of the elements chosen so far. Repeating inductively gives the denominator above.

Next, the number of ZXCF diagrams for the same n and $\bar{k}$ can be expressed by the following fourfold recursive function f evaluated at $p = o = 0$:

$$f(n, \bar{k}, p, o) = \begin{cases} 1, & \text{if } n = \bar{k} = 0, \\ A_{n, \bar{k}, p, o} + B_{n, \bar{k}, p, o}, & \text{else,} \end{cases} \quad (3)$$

where

$$A_{n, \bar{k}, p, o} = 2^o f(n - 1, \bar{k}, p + 1, o) \quad (4)$$

if $n \neq \bar{k}$ and $A_{n, \bar{k}, p, o} = 0$ if $n = \bar{k}$, and where

$$B_{n, \bar{k}, p, o} = (2^{2p+o+2} + 2) f(n - 1, \bar{k} - 1, p, o + 1) \quad (5)$$

if $\bar{k} \neq 0$ and $B_{n, \bar{k}, p, o} = 0$ if $\bar{k} = 0$.

We may intuit the recursion as going through the n output vertices and assigning them to either be pivots or nonpivots. The case A represents the number of ways to assign the remainder of the nodes if we assign the current node to be a pivot. Similarly, the case B counts the cases for when the current node becomes a nonpivot output. This function is computed with a base case of an empty tableau when $n = k = 0$. A formal derivation of this function is left to Appendix B. We solve for an explicit form to obtain

$$f(n, \bar{k}, p, o) = 2^{o(n-\bar{k})} \prod_{i=1}^{\bar{k}} \frac{(2^{n+1} - 2^i)(2^{n-i+1+2p+o} + 1)}{2^{\bar{k}} - 2^{i-1}}. \quad (6)$$

One can verify by standard induction that $f(n, \bar{k}, 0, 0)$ gives the same expression as Eq. (2), proving that ZXCF is indeed canonical.

C. ZXCF to graph

In the final step, we reduce a ZXCF directly to a graph. We remark that any local Clifford operation must, by the Clifford rule, be placed only on nonpivot output nodes. However, local Cliffords do not change the weights of

the set of correctable errors. Thus, local Cliffords do not change the code distance or any other code parameters. If we also quotient out this equivalence, we may remove all local Cliffords entirely. Since all internal edges have Hadamards, we can remove those as well and simply remember that they exist for inversion. Similarly, we can remove all free edges. The remaining diagram is identically a graph, specified entirely by a collection of nodes and edges. As a consequence, we have completed the map sequence from Eq. (1), except for the graph to ZXCF map.

Theorem 5. For any stabilizer code, there exists at least one equivalent code that has a presentation as a semibipartite graph with no local Clifford operations.

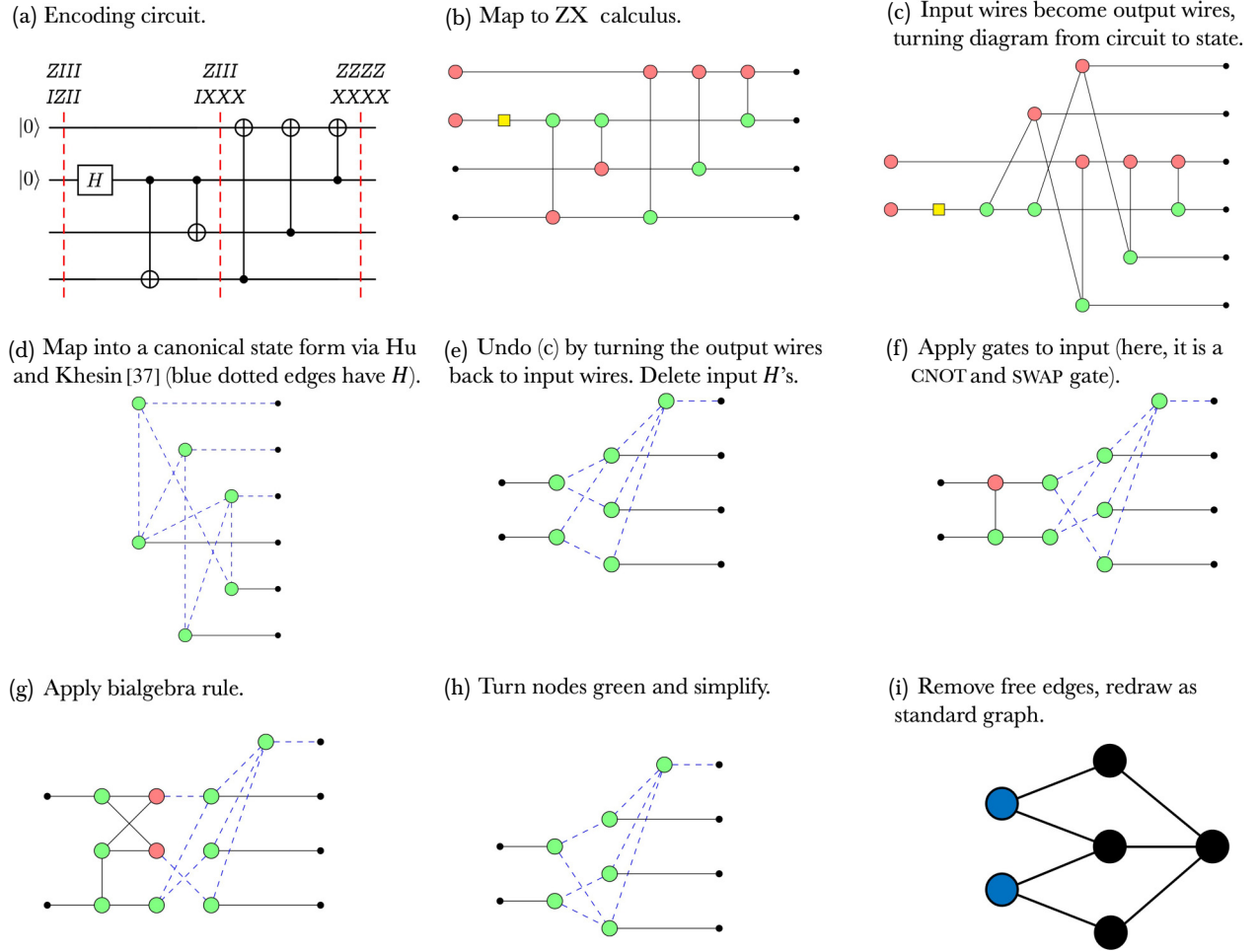


FIG. 2. An illustration of the compilation process of a $[[4, 2, 2]]$ code with stabilizers $XXXX$ and $ZZZZ$. In (a), we begin with the encoding circuit of the code. We then map it into the ZX calculus in (b), where yellow boxes (as well as dotted blue edges) represent Hadamards H , and the green and red nodes are the standard notation in the ZX calculus. In (c), we turn the circuit into a state by moving input wires to output wires. In (d), we implement the transformation from Hu and Khesin [37], turning it into a canonical form which satisfies the edge and Hadamard rules. We then return the state to a circuit in (e) by moving the output wires from (c) back to input wires. To satisfy the RREF rule, in (f) and (g) we apply gates on inputs and the bialgebra rule—an equivalence transformation in the ZX calculus. The details of these simplification steps are all discussed in Appendix A. Finally, in (h) and (i) we simplify and remove free edges to turn the diagram into a graph. In the final graph, the inputs are blue (leftmost column) and the outputs are black (middle and right columns).

For concreteness, Fig. 2 illustrates an example run of the compilation process on the $[[4, 2, 2]]$ code, which has stabilizers $XXXX$ and $ZZZZ$. The exact choice of equivalence transformations is dictated by Appendix A, but the key idea of Fig. 2 is that an encoding circuit turns into a ZX diagram, which is simplified by a series of equivalence transformations that ultimately yield a simple graph form. Here, we have colored the inputs blue and all other nodes black to emphasize that the final graph form can be studied independently of the ZX calculus.

III. APPLICATION TO QUANTUM CODES

It is illuminating to observe the graph representation of well-known quantum codes. Not only does it inform of geometric structure embedded in the code, but it also provides a guide as to how such codes may be generalized. We provide three examples for study. Consider first the nine-qubit code, due to Shor [17], which uses nine physical qubits to encode one logical qubit. The Shor code may be represented by the stabilizer tableau

$$\begin{bmatrix} Z & Z & I & I & I & I & I & I & I \\ Z & I & Z & I & I & I & I & I & I \\ I & I & I & Z & Z & I & I & I & I \\ I & I & I & Z & I & Z & I & I & I \\ I & I & I & I & I & I & Z & Z & I \\ I & I & I & I & I & I & Z & I & Z \\ X & X & X & X & X & X & I & I & I \\ X & X & X & I & I & I & X & X & X \end{bmatrix}. \quad (7)$$

We give the graph representation in Fig. 3(a) and the ZXCF in Fig. 3(b). We denote in the ZXCF the input node with **I**, and the free edges are dotted for emphasis. In the graph presentation, the input node is blue and the outputs are black. We first examine the ZXCF. There are three identical sectors of the outputs, two with Hadamarded outputs and one without. This resembles our expectations from examination of the unnormalized qubit representation of the Shor code, $(|000\rangle \pm |111\rangle)^{\otimes 3}$. The graph, on

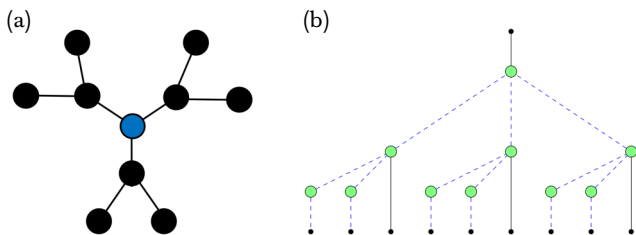


FIG. 3. The (a) graph presentation and (b) ZXCF of the nine-qubit code. The input in the graph (a), in the center, is shown in blue. In the ZXCF, the green nodes are those of standard ZX-calculus notation, and the dotted blue lines are Hadamarded edges. Note that the graph encoder uses a slightly different basis since its output edges do not contain Hadamard gates.

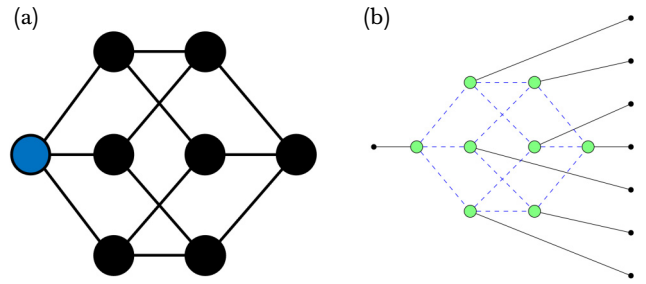


FIG. 4. (a) The (a) graph presentation and (b) ZXCF of the seven-qubit code. The input in the graph (a), given by the left-most node, is shown in blue. The ZXCF in (b) has standard ZX-calculus green nodes, and dotted blue lines are Hadamarded edges. Note that the graph encoder uses a slightly different basis as since its output edges do not contain Hadamard gates.

the other hand, is a star-shaped tree. It is easy to imagine how one might extend the Shor code into an infinite family of graphs, namely by recursively giving every outer (leaf) node two children.

Next, we compile Steane's seven-qubit code, a central construction in many fault-tolerant quantum computation schemes [20]. The code is represented by the following tableau:

$$\begin{bmatrix} I & I & I & X & X & X & X \\ I & X & X & I & I & X & X \\ X & I & X & I & X & I & X \\ I & I & I & Z & Z & Z & Z \\ I & Z & Z & I & I & Z & Z \\ Z & I & Z & I & Z & I & Z \end{bmatrix}. \quad (8)$$

In our ZXCF, this code takes the form given in Fig. 4. In particular, the nodes of the diagram are simply the corners of a cube. A similar picture has been given in a different context by Duncan and Lucas [60]. The generalization of the Steane code is also evident, via extensions into hypercubes of arbitrary dimension. We explore precisely such a coding scheme in Sec. VB.

We observe that the stabilizers of the seven-qubit code are positioned at exactly the 1-indices in the binary representation of a node label. We can see elegant symmetries in Fig. 4 with the positions of the nodes and their expressions in binary. The reason why the nodes adjacent to 0 are not 1, 2, and 4, (respectively 001, 010, and 100 in binary) is because the rules of the ZXCF require that the Hadamarded output edges not be connected to lower-numbered nodes.

As a final example we consider the five-qubit code, which is the smallest code one can achieve for correction of an arbitrary single-qubit error and has been studied experimentally [18,61]. Its tableau representation is given

by

$$\begin{bmatrix} X & Z & Z & X & I \\ I & X & Z & Z & X \\ X & I & X & Z & Z \\ Z & X & I & X & Z \end{bmatrix}. \quad (9)$$

Our procedure produces the ZXCF shown in Fig. 5.

The five-qubit code takes the shape of either a pentagon with an input in the middle or a cone, depending on how one sets up the graph. Consequently, a simple generalization is to use a n -gon, or equivalently add more nodes to the circle in the cone.

Such elegant and simple structure in the five-, seven-, and nine-qubit codes suggest that other graphs with similar elegant structure may yield interesting quantum codes, either other constant-size codes or larger codes in families in which these are merely small examples. We explore this idea in Sec. V A.

IV. THE INVERSE MAP: FROM GRAPHS TO STABILIZER CODES

The bijective relation between stabilizer codes and graphs—particularly those in the ZX canonical form—give justification that graphs are at least as powerful as stabilizer tableaus for the expression of codes. In this section, we lay the foundation for the use of graphs as *constructive tools* for code design. Such a mapping from graphs to stabilizer tableaus can be viewed as an “inverse map” to the algorithm given in Sec. II B, with some caveats relating to the fact that not all graphs can be equivalently represented by ZXCFs and therefore not all graphs can be mapped into codes.

Let G be a graph with $n + k$ nodes, separated into k input nodes $\mathcal{I}$, a choice of k pivot nodes $\mathcal{P}$, and $n - k$ nonpivot output nodes $\mathcal{O}$. In the notation of the encoder-respecting form from Definition 2, $\mathcal{P} \cup \mathcal{O} = \tilde{\mathcal{O}}$; we will henceforth directly write $\mathcal{P} \cup \mathcal{O}$ to refer to all output nodes. To ensure that our operations are well defined, there must exist equivalence operations that map G into a ZXCF.

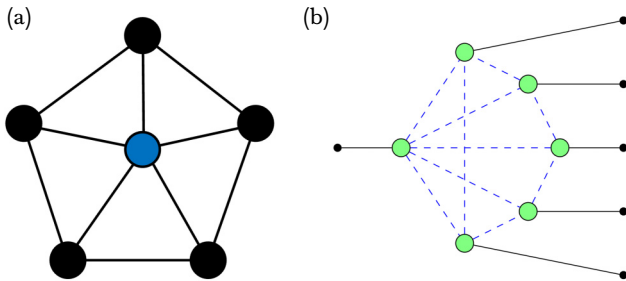


FIG. 5. The (a) graph presentation and (b) ZXCF of the five-qubit code. The input in the graph (a), in the center, is shown in blue. The ZXCF (b) has standard ZX-calculus green nodes and dotted blue lines are Hadamarded edges.

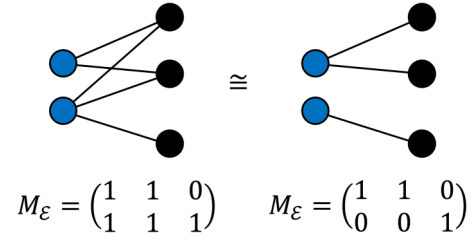


FIG. 6. A simple example for which the third output node must be chosen as a pivot. The inputs are on the left-hand side of each graph and colored blue and outputs on the right-hand side are black. The example is given on the left, and an equivalent (via row operations on the partial adjacency matrix between the left and right) graph is given on the right.

The only nontrivial restriction this requirement poses is on the choice of inputs and pivots. The RREF rule ensures that every distinct pivot is connected to exactly one distinct input; in other words, $\mathcal{P}$ and $\mathcal{I}$ are perfectly matched. Because row operations on the partial adjacency matrix are equivalences, it suffices to choose pivots such that there exists a row-reduced partial adjacency matrix $M_{\mathcal{E}}$ for which the chosen k pivot nodes correspond to the k pivot columns of $M_{\mathcal{E}}$. A simple example of this subtlety is shown in Fig. 6. Here, it is impossible to choose the first two outputs to be the pivots because the third output is forced to be a pivot when we row-reduce. Note, however, that this subtle restriction poses no issue *if every distinct pivot we choose is already connected to exactly one distinct input* (as in the case in Sec. V C) or if we can observe directly that the pivots we have chosen do correspond to the pivot columns after row reduction. It is also convenient to continue to enforce that inputs are not connected, so as to maintain encoder-respecting form. More generally, we say a choice of inputs and pivots is *valid* if no inputs are connected to each other and each pivot is connected to exactly one input. Henceforth, we will denote $G = (\mathcal{I} \sqcup \mathcal{P} \sqcup \mathcal{O}, E)$ (or equivalently, $G = (\mathcal{I} \cup \mathcal{P} \cup \mathcal{O}, E)$) as a graph for which a set of valid inputs and pivots $\mathcal{P}$ have been specified alongside a set of input nodes $\mathcal{I}$. The remaining nonpivot output nodes are $\mathcal{O}$ and the edge set is E . Usually, we will implicitly assume that inputs are not connected, and just say that a choice of pivots is valid.

Assuming that the choice of $\mathcal{P}$ is valid, G will be equivalent to a ZXCF by a series of equivalence transformations. First, we place a free edge on every node and place a Hadamard on all internal edges, satisfying the edge rule. The Hadamard rule is vacuously satisfied, as we will not place any Hadamard on output nodes. We can row-reduce the partial adjacency matrix to satisfy the RREF rule. Next, we strip input-input edges, which are unitaries on the inputs and thus represent equivalent encoders. Lastly, as discussed in Sec. II, there is an equivalence relation

that strips pivot-pivot edges. This procedure results in a ZXCF with no local Clifford on nodes at all. While there exist encoders that can only be expressed with local Clifford operations that this map cannot produce, the map can always produce a locally equivalent encoder, and is otherwise completely general.

In practice, we never transform our graphs into a ZXCF. In fact, the primary purpose of the ZXCF is to prove that a certain family of graphs is expressive enough to represent every possible stabilizer code. Now that we have established this fact, we can find a more convenient and explicit map from a graph directly to a stabilizer tableau. This map will work so long as every input is connected to exactly one pivot (and vice versa), but otherwise we do not need to strictly abide by the ZXCF rules. The primary way in which we will take advantage of this extra flexibility is to allow pivot-pivot edges. Although no ZXCF has a pivot-pivot edge, our map from graph to tableau remains well defined. Note that if we remove the pivot-pivot edges of a graph, we may in general change the corresponding code, but if we approach this from the direction of building graphs to make codes, it is worth noting that no additional codes will be expressible by adding pivot-pivot edges that could not have been produced without such edges. Aside from that, knowing that the graph is equivalent to a ZXCF, we can write down the stabilizer generators of its corresponding ZXCF directly. To do so, we first establish notation. Let $\Delta_{i=1}^n A_i = A_1 \Delta \dots \Delta A_n$ for sets $A_1, \dots, A_n$, where Δ is the symmetric difference. The symmetric difference is both commutative and associative, so this iterative procedure is well defined.

Definition 3 (Neighbor sets). Let $G = (V, E)$ be a graph with input set $\mathcal{I}$, pivot set $\mathcal{P}$, and nonpivot output set $\mathcal{O}$, so that $V = \mathcal{I} \sqcup \mathcal{O} \sqcup \mathcal{P}$. Let $v \in V$. We define $i(v) = \{w \in \mathcal{I} \mid w \sim v\}$ as the inputs connected to v and $p(v) = \{w \in \mathcal{P} \mid w \sim v\}$ as the pivots connected to v . Similarly, let $o(v) = \{w \in \mathcal{O} \mid w \sim v\}$ be the set of nonpivot outputs connected to v . Finally, let $N_o(v) = \{w \in \mathcal{O} \cup \mathcal{P} \mid w \sim v\} = o(v) \cup p(v)$ be all the output neighbors of v . More generally, we also define for a set $A \subseteq V$, $N_o(A) = \Delta_{v \in A} N_o(v)$ and likewise for $i(A), p(A), o(A)$.

If $v \in \mathcal{I}$, then $|p(v)| = 1$ since each pivot is connected to a unique input. If $v \in \mathcal{P}$, $|i(v)| = 1$ for the same reason. For this reason, we will frequently refer to the input connected to a pivot v as $i(v)$, even though technically the latter is a set containing one node. The same applies for the pivot $p(v)$ connected to input v . The symmetric difference is useful for correctly counting the parities of Pauli operations applied to certain nodes, i.e., if a Pauli is applied twice in the set, it is effectively not applied at all.

Every node of the graph represents a qubit—logical if the node is in $\mathcal{I}$ and physical otherwise. We can therefore speak of applying operations to the qubits by placing

operators to their corresponding nodes. Let $X(w)$ be a Pauli X on node w , and $X(A) = \bigotimes_{v \in A} X(v)$; Z operators are defined analogously. We may also write $X(A)$ and $Z(A)$ for placing a Pauli on every element of a subset $A \subseteq V$ of nodes. Note that we may only directly apply operators to output nodes because these are physical operators. With this notation, we define the following map $S : G \mapsto S$ from graph to stabilizer tableau, the output of which we call the *canonical stabilizers* of the code represented by G :

$$S := \{X(v)Z(N_o(v))X(p(i(v)))Z(N_o(p(i(v))))\}_{v \in \mathcal{O}}. \quad (10)$$

Denote each stabilizer S_v (or, occasionally when more convenient, $S(v)$) for $v \in \mathcal{O}$. These stabilizers are readily seen to be a valid generating set. There are exactly $n - k$ generators, and all are independent because only S_v has an X on v . Moreover, we claim that every pair of stabilizers S_v and S_w commutes. Partition the support of the Paulis in each stabilizer S_v into four groups: v , $N_o(v)$, $p(i(v))$, and $N_o(p(i(v)))$. S_v places X 's on the odd groups and Z 's on the even groups. We therefore need to check that whenever an odd and an even group intersect, the intersection size has even parity. We show that for two stabilizers S_v and S_w , in each of the four cases when an odd group intersects an even group, we can match each element of the intersection uniquely with a distinct element of the intersection. Therefore, the number of anticommuting overlaps between the stabilizers is even, so the stabilizers commute. We split this argument by the groups. Each case follows directly from the undirected structure of the edges of the graph. The fact that the graph is undirected implies that for $v_1, v_2 \in \mathcal{O} \cup \mathcal{P}$, $v_1 \in N_o(v_2)$ if and only if $v_2 \in N_o(v_1)$. For example then, if group 1 of v touches group 2 of w , then this means that $v \in N_o(w)$. But then $w \in N_o(v)$, so w is in group 2 of v and group 1 of w , giving the desired pairing. Analogous arguments prove the remaining three cases, which are depicted visually in Fig. 7. Moreover, we show in Fig. 8 the stabilizers of the graph of the $[[4, 2, 2]]$ code. Note that the ZXCF and graph formalism mod out by local (single-qubit) Cliffords at the end of the encoding process, and thus the stabilizers may differ from the typical presentation by a local Clifford. Here, the typical presentation of the $[[4, 2, 2]]$ code is given by stabilizers $XXXX$ and $ZZZZ$, while the canonical stabilizers are $XZZZ$ and $ZXXX$. They differ by a Hadamard on the first qubit.

We emphasize that Eq. (10) gives a recipe to construct codes that is essentially independent of ZXCFs (except that inputs and pivots must be matched), even though we will also show that the stabilizers given by Eq. (10) are actually the stabilizers for the ZXCF corresponding to the graph at hand. In principle, one could disregard the entirety of the previous section, *define* Eq. (10) as a map from graphs to stabilizers, and explore their properties. However, it is unclear *a priori* from this approach how expressive such

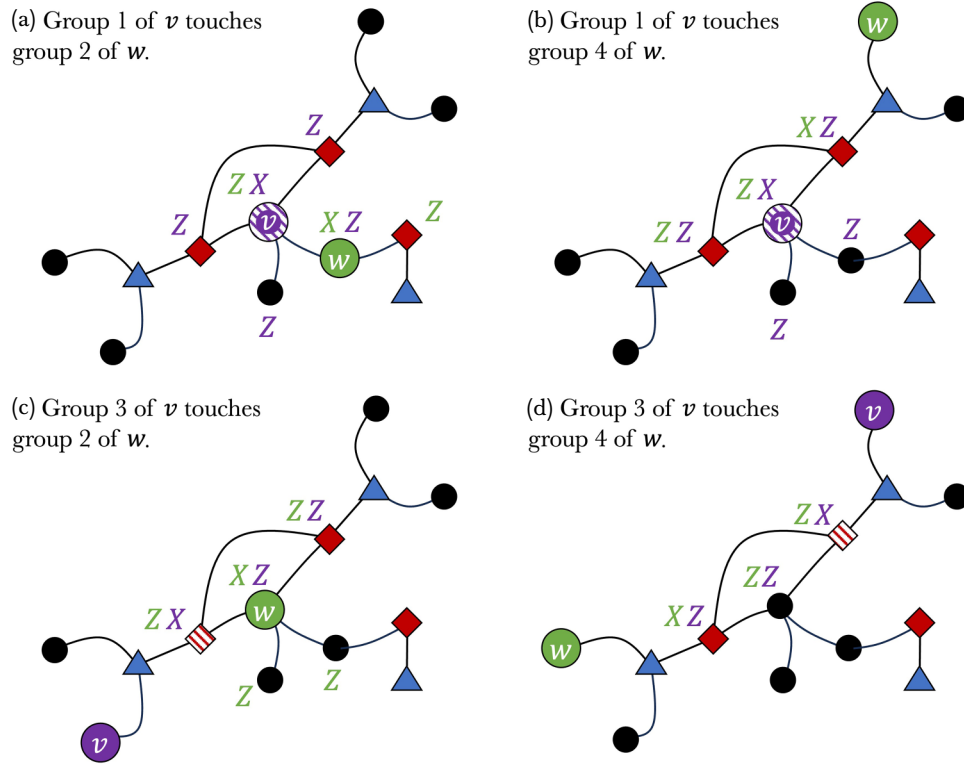


FIG. 7. A depiction of the four cases that the stabilizers given by Eq. (10) commute. Inputs are blue triangles, pivots are red diamonds, and nonpivot outputs are black circles. For each $v \in \mathcal{O}$, the support of the stabilizer S_v is partitioned into four groups: v , $N_o(v)$, $p(i(v))$, and $N_o(p(i(v)))$, using Definition 3. X 's are placed in the odd groups, and Z 's are placed on the even groups. Each case shows that when an odd group of a node $v \in \mathcal{O}$ (purple) and an even group of a node $w \in \mathcal{O}$ (green) have an intersection (striped node)—thus giving an anticommutation—there is always a corresponding intersection at another node which cancels out the anticommutation. Therefore, the stabilizers S_v and S_w commute.

a class of stabilizers would be and the motivation behind such an assertion would be obscure. The proof of canonicity with the ZXCF guarantees that this recipe is capable of generating all possible stabilizer codes. We next show that such a map generates precisely the stabilizer group of

the encoder that the corresponding ZXCF of G represents. Because graphs do not have local Clifford operations associated with them, we can treat $\mathcal{S}$ as the inverse map from graph to ZXCF, modulo local Cliffords, i.e., up to local equivalence. To do so, we give a fundamental equivalence rule for Paulis on graphs, which arise from equivalence rules in the ZX calculus.

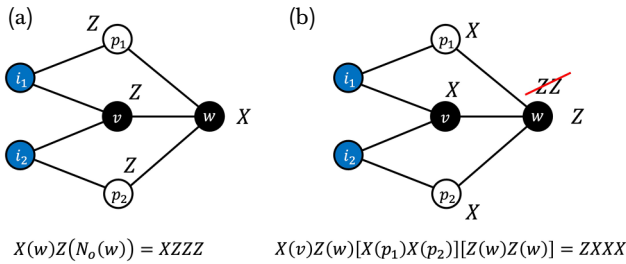


FIG. 8. The canonical stabilizers according to Eq. (10) of the $[[4, 2, 2]]$ code. Here, $i_1, i_2 \in \mathcal{I}$ are inputs, $p_1, p_2 \in \mathcal{P}$ are pivots, and $v, w \in \mathcal{O}$ are nonpivot outputs. The stabilizer in (a) corresponds to v and the stabilizer in (b) corresponds to w , with cancellations of two Z 's depicted. The stabilizers differ by a local Clifford, namely a Hadamard on w , from the standard presentation of the $[[4, 2, 2]]$ code with stabilizers $XXXX$ and $ZZZZ$.

Lemma 1 (ZX-graph rule). Let $G = (V, E)$ be a graph with $V = \mathcal{I} \sqcup \mathcal{O} \sqcup \mathcal{P}$. For any $v \in V$, a $X(v)$ is equivalent to $Z(N(v))$, where $N(v) = \{w \in V \mid (v, w) \in E\}$ is the neighbor set of v . Note that Paulis placed on input nodes $i \in \mathcal{I}$ act logically on the corresponding qubit.

Proof. If we endow every node in the graph with a free edge and recover a Hadamard on every internal edge as dictated by the ZXCF rules, then the rules of the ZX calculus show that an X placed on the free edge of a node equals a Z on all of its neighbors, proving the claim. ■

As a concrete example, we visualize the ZX-graph rule applied on a graph, which for concreteness is the graph of the $[[4, 2, 2]]$ code with stabilizers $XXXX$ and $ZZZZ$. This illustration is depicted in Fig. 9.

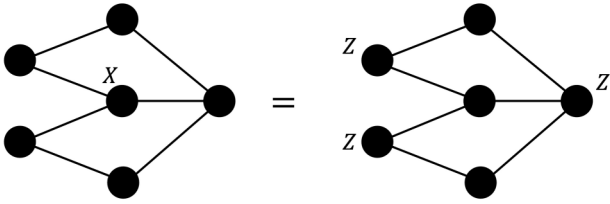


FIG. 9. A depiction of the ZX-graph rule on a graph of the $[[4, 2, 2]]$ code. An X placed on the central node of the graph is equivalent to a Z on all of its neighbors. The distinctions between inputs, pivots, and nonpivot outputs are intentionally not given, because this rule is agnostic to how we choose to classify the nodes.

The ZX-graph rule in Lemma 1 allows us to *derive* the stabilizers of a graph as given in Eq. (10). For each $v \in \mathcal{O}$, place $XX = I$ on v . Apply the ZX-graph rule on one X to obtain $Z(N(v))$, where $N(v)$ are the neighbors of v . However, some neighbors are inputs, and we can only act directly on physical qubits. We therefore must apply operations on other output nodes to cancel the Z 's on $i(v)$ that the input neighbors. Observe that for each $i \in i(v)$, placing $XX = I$ on the unique corresponding pivot p for i and applying the ZX-graph rule will cancel out the Z on i , but it will also create Z 's on $N(p)$. Since a pivot cannot be connected to more than one input, $N(p) = \{i\} \sqcup N_o(p)$, so the extra edges are in $N_o(p)$. Doing so on all pivots yields in total Z operators on $N_o(v)$ and $N_o(p(i(v)))$. Finally, we account for the X 's that we did not apply the ZX-graph rule to, giving $X(v)Z(N_o(v))X(p(i(v)))Z(N_o(p(i(v))))$,

which is precisely Eq. (10). Therefore, we have proven the following theorem.

Theorem 6 (Inversion). Given a graph G with a valid choice of inputs $\mathcal{I}$ and pivots $\mathcal{P}$, where a valid choice means that every pivot is connected to exactly one input, the stabilizer tableau associated with the ZXCF equivalent to G is given by Eq. (10).

We remark that if we choose a graph G with no inputs, then the stabilizer tableau of Eq. (10) is exactly that of a *graph state* of G , a stabilizer state where each of the n stabilizers are X on a node v and Z on its neighbors. An example derivation of the canonical stabilizers of the $[[4, 2, 2]]$ code is given in Fig. 10.

Application of the ZX-graph rule on *output* nodes yields the canonical stabilizers; we now show that application of the ZX-graph rule on *input* nodes yields a canonical set of logical Paulis. For each input node $i \in \mathcal{I}$, construct $\bar{X}_i$ by placing $X(i)$, which is by definition a logical operator on logical qubit i . To obtain the physical operator, we apply the ZX-graph rule to obtain Z 's on $N(i) = N_o(i)$. Hence, $\bar{X}_i = Z(N_o(i))$. Similarly, for $i \in \mathcal{I}$, $\bar{Z}_i = X(p(i))Z(N_o(p(i)))$, by placing a X on the corresponding pivot $p(i)$ and applying the ZX-graph rule. A visualization of the logical operators on the graph of the $[[4, 2, 2]]$ code is given in Fig. 11.

A useful consequence of the structure of the canonical stabilizers is that they reveal when a code is CSS. For our purposes, we say that a code is CSS if there exists a product of single-qubit Cliffords which when conjugated on the stabilizers makes every stabilizer either all X or all Z .

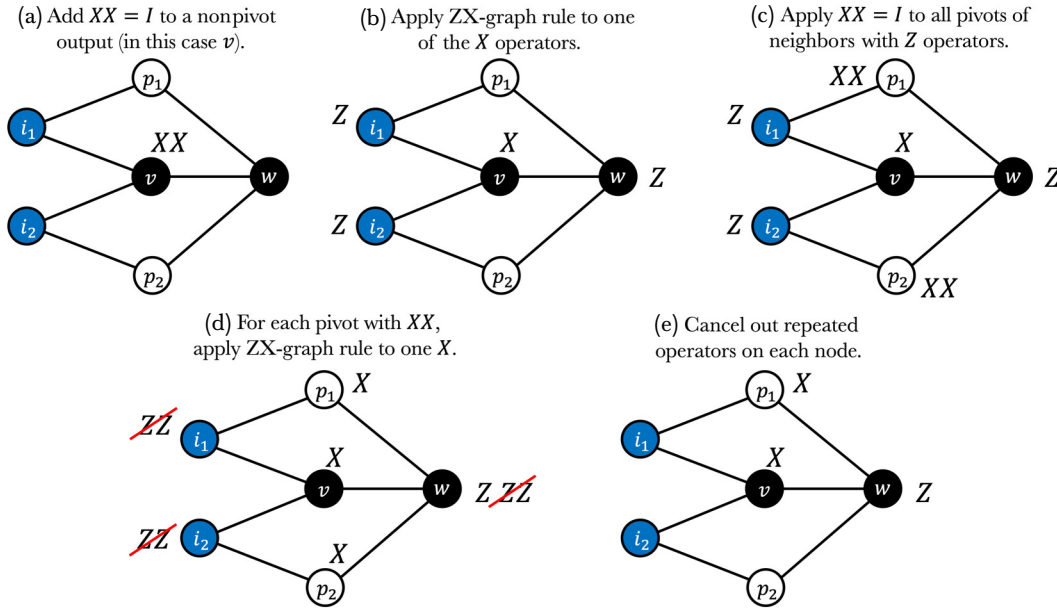


FIG. 10. An example derivation of a canonical stabilizer associated with node v in the $[[4, 2, 2]]$ code. Here, $i_1, i_2 \in \mathcal{I}$ are inputs, $p_1, p_2 \in \mathcal{P}$ are pivots, and the remaining nodes v, w are nonpivot outputs. We first apply $XX = I$ to v . Then, by applying the ZX-graph rule from Lemma 1 and applying $XX = I$ to pivots, we obtain the final stabilizer.

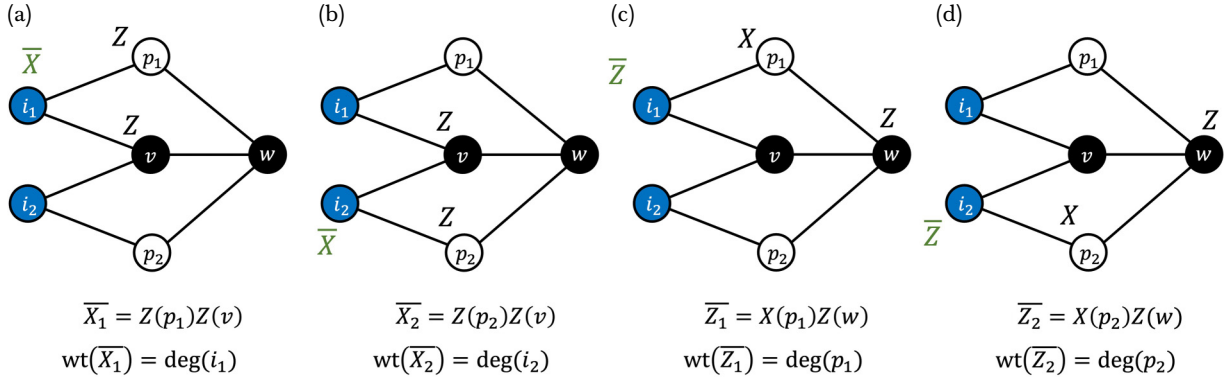


FIG. 11. The logical Paulis of the graph of the $[[4, 2, 2]]$ code. $i_1, i_2 \in \mathcal{I}$ are inputs, $p_1, p_2 \in \mathcal{P}$ are pivots, and $v, w \in \mathcal{O}$ are nonpivot outputs. The barred green Pauli is the logical Pauli, and the corresponding physical implementation of the Pauli is shown in unbarred black.

Theorem 7 (Bipartite graphs are CSS codes). Let $G = (V, E)$ be a graph where $V = \mathcal{I} \sqcup \mathcal{O} \sqcup \mathcal{P}$ and $\mathcal{I}$, $\mathcal{P}$, and $\mathcal{O}$ are, respectively, the input, pivot, and nonpivot output nodes. If G is bipartite, then the corresponding code given by the canonical stabilizers of Eq. (10) is a CSS code.

Proof. If G is bipartite, then we may define $\mathcal{L}$ and $\mathcal{R}$ such that $V = \mathcal{L} \sqcup \mathcal{R}$ and there are no edges between $\mathcal{L}$ and $\mathcal{R}$. Consider any nonpivot output node in the left $v \in \mathcal{L} \cap \mathcal{O}$. Then $N_o(v) \subseteq \mathcal{R}$, $p(i(v)) \subseteq \mathcal{L}$, and $N_o(p(i(v))) \subseteq \mathcal{R}$. Likewise, if $v \in \mathcal{R} \cap \mathcal{O}$, then $N_o(v) \subseteq \mathcal{L}$, $p(i(v)) \subseteq \mathcal{R}$, and $N_o(p(i(v))) \subseteq \mathcal{L}$. Therefore, applying a Hadamard H on all right output nodes $\mathcal{R} \cap (\mathcal{O} \cup \mathcal{P})$ turns all canonical stabilizers for $v \in \mathcal{L} \cap \mathcal{O}$ to have only X 's and all canonical stabilizers for $v \in \mathcal{R} \cap \mathcal{O}$ to have only Z 's. We give a visual proof of the theorem in Fig. 12. ■

We remark that while we will not need this stronger result in our work, a converse statement to Theorem 7 also holds true.

Fact 1 (Khesin and Li [48]). A code is CSS if and only if its corresponding ZXCF is bipartite.

As a consequence of our canonical logical-operator construction, we observe that the degree of G controls the upper bound of the distance as well as the stabilizer weights of $\mathcal{S}(G)$.

Proposition 1 (Distance upper bound). Let G be a graph with inputs $\mathcal{I}$, pivots $\mathcal{P}$, and nonpivot outputs $\mathcal{I}$. Let $\mathcal{S}(G)$ be the tableau of canonical stabilizers as given by Eq. (10). The distance of the code for which the stabilizer tableau is given by $\mathcal{S}(G)$ is at most the minimum degree of any vertex in $\mathcal{I} \cup N_o(\mathcal{I})$, where N_o is defined in Definition 3.

Proof. For each $v \in \mathcal{I}$, $X(v)$ is a logical operation which is equal to $Z(N_o(v))$. The weight of this operation

is equal to the degree of v . Similarly, for each $v \in N_o(\mathcal{I})$, the rules of the graph tell us that applying $X(v)$ as well as applying Z to all neighbors of v does not change the circuit effected by the graph. Since v neighbors at least one input node, this means that the logical operation formed by the Paulis applied to the inputs is equal to the Paulis applied to the outputs, including v . The size of this set is equal to $|N_o(v)| + 1$, which is at most the degree of v , since v neighbors an input. The construction of these logical operations proves the desired upper bound on the distance. This also shows that the statement of the corollary could be strengthened slightly by replacing the degree of input neighbors by the number of their output neighbors plus one. ■

A visualization of the proof of Proposition 1 is given by Fig. 13. The definition below is useful for proving structural properties of graphs as codes.

Definition 4 (t -spacing). Let $G = (V, E)$ be a graph with vertex set V and edge set E , and let $V_0 \subseteq V$ be a subset of nodes. We say that V_0 is t -spaced if the distance between every pair of nodes $v_1, v_2 \in V_0$ is at least t . By distance between nodes, we mean the minimum path length which begins at v_1 and ends at v_2 .

In the special case $t = 3$, the subset V_0 is known in graph theory as a 2-packing or a strongly independent set.

Proposition 2. Given a graph G , with inputs $\mathcal{I}$, pivots $\mathcal{P}$, and nonpivot outputs $\mathcal{O}$, define

$$\begin{aligned} \delta_{\mathcal{O}}^* &= \max_{v \in \mathcal{O}} \deg(v), & \delta_{\mathcal{OI}}^* &= \max_{v \in \mathcal{O}} |i(v)|, \\ \delta_{\mathcal{PO}}^* &= \max_{v \in \mathcal{P}} |N_o(v)|, \end{aligned} \quad (11)$$

where i and N_o are given in Definition 3. Let $\mathcal{S}(G)$ be the canonical stabilizers as in Eq. (10). Then $\max_{S \in \mathcal{S}(G)} |S| \leq$

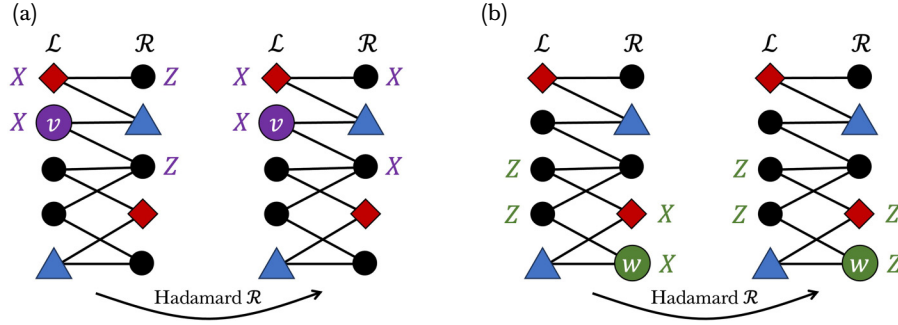


FIG. 12. A visualization that bipartite graphs are CSS. Inputs are blue triangles, pivots are red diamonds, and nonpivot outputs are circles. In (a), a left-node v is highlighted. Its stabilizer has only X 's in the left-hand side $\mathcal{L}$ and Z 's on the right-hand side $\mathcal{R}$. In (b), a right-node w is highlighted, and its stabilizer has only X 's in $\mathcal{R}$ and Z 's in $\mathcal{L}$. By Hadamarding all output nodes in $\mathcal{R}$, stabilizers associated with left nodes such as v become only X 's, and stabilizers associated with right nodes such as w become only Z 's.

$1 + \delta_{\mathcal{O}}^* + \delta_{\mathcal{O}\mathcal{I}}^* \delta_{\mathcal{P}\mathcal{O}}^*$, where $|\mathcal{S}|$ is the weight of a stabilizer $\mathcal{S}$. Moreover, if $\mathcal{I}$ is 3-spaced as in Definition 4 so that every node $u \in \mathcal{O}$ is connected to at most one node in $\mathcal{I}$, then $\max_{\mathcal{S} \in \mathcal{S}(G)} |\mathcal{S}| \leq \delta_{\mathcal{O}}^* + \delta_{\mathcal{P}\mathcal{O}}^*$.

Proof. This result is a direct consequence of Eq. (10). ■

In the next section, we will show that another important property, encoding depth, is also bounded above by the degree. We make two remarks regarding the utility of such degree bounds. First, for a general stabilizer code it is unclear as to how one may give a nontrivial upper bound on properties such as distance and circuit depth. The degree allows us to algorithmically do so by compiling a stabilizer tableau into a graph and then examining the degree. Second, it is possible, at least in certain cases, to upper bound the degree itself by some function of the code parameters. We show bounds of this type in Secs. VB and VC. When such a bound is possible, it provides nontrivial bounds on code properties in terms of functions of other parameters.

A. Encoding circuit with degree-bounded depth

A particularly appealing quality of the graph formalism is that it enables us to efficiently and canonically construct encoding circuits which are bounded by a small linear function of the *degree* of the graph. This construction can be taken advantage of in two ways. If we already have a stabilizer code and would like to construct a nice encoding circuit, we can compile the tableau of the code into a graph and then apply the procedure below. Note that Theorem 8 assumes a graph presentation, so if the ZXCF has local Cliffords, they are not included in the encoding circuit. They can be appended to the encoding circuit produced by the theorem, at the cost of increasing the depth by 1, since all local Cliffords are single-qubit operations. Alternatively, if we have a graph which by the previously discussed procedure represents a stabilizer code, we

can directly write down this encoding circuit and have a guarantee of its depth.

Theorem 8. Suppose that one may initialize qubits to the $|+\rangle$ state. Given a graph G with maximum degree $\mathcal{D}_{\max}$, there exists an efficient algorithm which outputs an encoding circuit $\mathcal{C}$ for the code with stabilizers $\mathcal{S}(G)$, with $\mathcal{S}(G)$ defined by Eq. (10), such that $\text{depth}(\mathcal{C}) \leq 2\mathcal{D}_{\max} + 3$. Here $\text{depth}(\mathcal{C})$ is the circuit depth of $\mathcal{C}$.

Proof. We construct explicitly the transformation from the graph to its encoding circuit [62]. An informal description is as follows. A graph with $|\mathcal{I}| = k$ and $|\mathcal{O} \cup \mathcal{P}| = n$ requires a circuit taking k input and $n - k$ ancillary qubits to n output qubits. We choose to send the k inputs to the k pivots on the output sides. Edges in G will generally become controlled- Z (CZ) gates in the encoding circuit, with the exception of edges from $\mathcal{I}$ to $\mathcal{P}$, which become a layer of Hadamards. More precisely, construct the circuit $\mathcal{C}$ as follows:

- (1) *Initialization.* Create n wires, one for each node in $\mathcal{O} \cup \mathcal{P}$. Label the output side of the wires by the node. For each wire associated with a pivot v , label the input side of the wire by the input corresponding to v . The remaining $n - k$ wires unlabeled on the input side are the ancillary qubits. Initialize these unlabeled wires to $|+\rangle$.
- (2) *Input-output edges.* For each input $u \in \mathcal{I}$ and $v \in \mathcal{O}$, if $v \in o(u)$ then add a $CZ_{u,v}$ to the circuit. The order does not matter since all CZ gates commute.
- (3) *Input-pivot edges.* For each input $u \in \mathcal{I}$ apply H_u . These will correspond to the edges between each input and its corresponding pivot.
- (4) *Output-output edges.* For each $u \in \mathcal{O} \cup \mathcal{P}$ and $v \in N_o(u)$, apply $CZ_{u,v}$.

For concreteness, we give an example of these four steps collectively constructing the encoding circuit for the

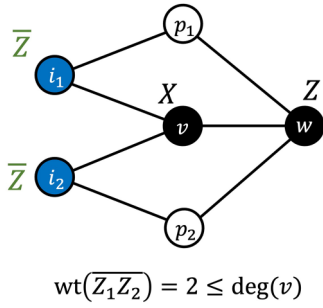


FIG. 13. The origin of the distance-degree bound in Proposition 1, visualized for the graph of the $[[4, 2, 2]]$ code. Here, $i_1, i_2 \in \mathcal{I}$ are inputs, $p_1, p_2 \in \mathcal{P}$ are pivots, and $v, w \in \mathcal{O}$ are non-pivot outputs. By placing an X on an input neighbor $v \in N_o(\mathcal{I})$ (where we recall that N_o is the set of neighbors in $\mathcal{O} \cup \mathcal{P}$) and applying the ZX-graph rule from Lemma 1, we transform it into a mix of logical and physical Paulis on its neighbors. We then remove all the extra physical Paulis on neighbors of v to cancel them out, leaving only the pure logical operator. The whole process requires at most $\deg(v)$ Paulis. This process works over all choices of placing X on $\mathcal{I}$ or $N_o(\mathcal{I})$ and applying the ZX-graph rule, so the degree bounds holds generally across nodes in $\mathcal{I} \cup N_o(\mathcal{I})$.

$[[4, 2, 2]]$ code in Fig. 14. There are two claims to show: (a) that this construction correctly yields an encoding circuit for the code described by $\mathcal{S}(G)$, and (b) that $\text{depth}(\mathcal{C}) \leq 2\mathcal{D}_{\max} + 3$. The first claim is essentially a direct consequence of the rules of the ZX calculus and the form of the ZXCF. Recall that G can be mapped to a ZXCF $\tilde{G}$ by Hadamarding all edges, endowing each node with a free edge. Initially, work qubits in ZX diagrams are given by Z nodes, which are the $|+\rangle$ states used in (1). As we turn the ZX diagram into a circuit, the edges which connect an input node to its corresponding node on the output side of the circuit pick up the gates between them. Since we have chosen the inputs to map to their corresponding pivot on the output side of the circuit, and the only gate between them is H , this accounts for (3) above. The remaining edges connect distinct qubits (rather than the same qubit from the input to output side), and such edges are precisely given by CZ gates. This map accounts for (2) and (4). Note that the gates are ordered in this particular way because of the causal structure of the ZX diagram. First, input edges must propagate information to the output edges, so the associated input wires must have their CZ gates to establish their interaction with the ancilla wires. Next, the circuit completes the interactions from input to output via the input-pivot Hadamards. Finally, the output edges interact, so the output-output CZ gates are placed on the ancillary wires. This completes the proof of the first claim, correctness.

To the second claim, we observe that since all CZ gates commute, the depth is given by the minimum number of groupings of CZ gates which act on disjoint sets of qubits.

Such a problem reduces to finding the optimal *edge coloring* of G , i.e., the minimum number of colors needed to color every edge, such that no two edges which share a node have the same color. Although finding the optimal edge coloring is known to be **NP**-complete, Vizing's edge coloring theorem guarantees that the minimal coloring is at most $\delta^* + 1$, and moreover there exists an efficient algorithm which finds such a coloring [63,64]. In sum, each of (2) and (4) has depth at most $\mathcal{D}_{\max} + 1$, and (3) has depth 1, so that the total depth is at most $2\mathcal{D}_{\max} + 3$ as claimed. ■

Although encoding in practice is often performed by a method to prepare the logical zero state, e.g., repeatedly measuring the stabilizers until they are all +1, Theorem 8 shows that for codes that have sufficiently small degree, it may be possible to have a much stronger guarantee, namely to efficiently encode any state directly via an encoding circuit. Section V A gives an example of such an encoding circuit for a code we have constructed, and discusses how the depth may actually be substantially smaller in practice by solving small cases of minimal edge coloring. Moreover, in Sec. V B, we will show that the degree, and thus the encoding circuit depth, can be bounded above by code parameters for certain families of codes. This result will thus directly upper bound the degree of the encoding circuit by code parameters.

B. Logical non-Pauli gates on graph codes

Earlier, we have shown that the logical Pauli operations can be extracted immediately from the graph representation. Specifically, $\bar{X}_v$ and $\bar{Z}_v$ for $v \in \mathcal{I}$ are implemented, respectively, by $Z(N_o(v))$ and $X(p(v))Z(N_o(p(v)))$. We have derived these logical Paulis by placing a Pauli on the free edge of a particular input node, and then applying ZX equivalence rules to push the Pauli through the node into the output.

For a completely general logical operation $\bar{U}$ and a code with graph G , the only algorithm to physically implement $\bar{U}$ is to unencode, apply U , and then reencode. Such a technique is typically undesirable, since the encoding circuit is necessarily high-depth to achieve a good distance, which implies that implementation of the logical operation strongly propagates errors and therefore resists fault tolerance. Nonetheless, if the encoding circuit has a depth which is sufficiently small, it is conceivable that in some cases this generic procedure may be of use. Moreover, we already have a generically optimized encoding circuit from Sec. IV A, which in turn implies the following general result.

Theorem 9 (Generic logical operation). Let G be a graph with maximum degree $\mathcal{D}_{\max}$, and let U be a unitary which can be implemented in depth ℓ_U . Then there is an

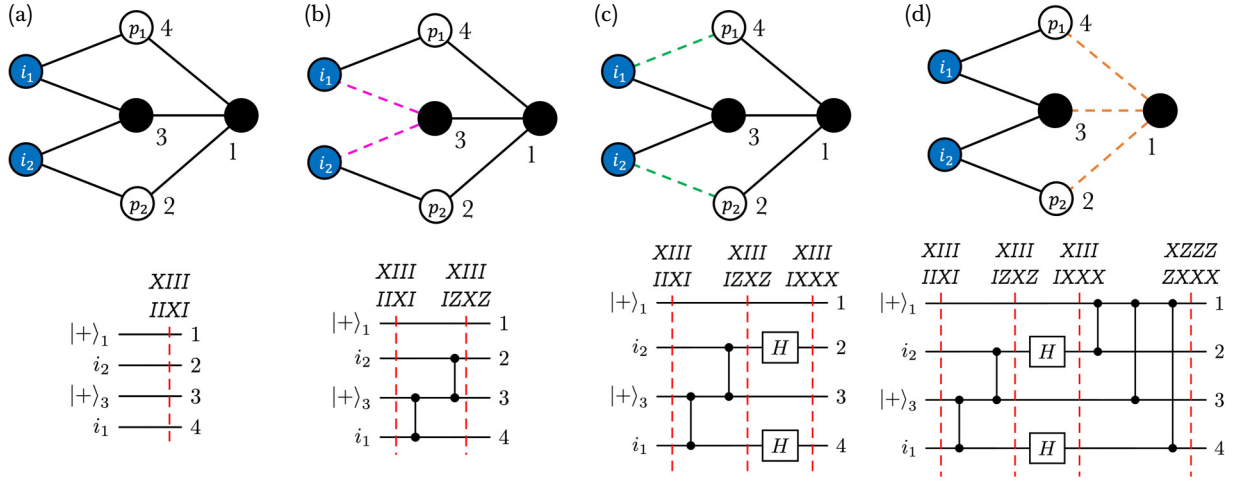


FIG. 14. A visualization of the encoding circuit construction from Theorem 8, depicted for the $[[4, 2, 2]]$ code with stabilizers $XXXX$ and $ZZZZ$. There are four steps in Theorem 8, which we illustrate here in (a)–(d). At each step, the graph of the code is shown, alongside the current status of the circuit construction. The current stabilizers are also shown at various steps in the circuit. (a) The inputs are labeled i_1 and i_2 and the outputs are labeled 1, 2, 3, and 4. The two pivots are labeled p_1 and p_2 . We draw four wires and label the right-hand side as the outputs. The input side has input wires for the $k = 2$ inputs i_1 and i_2 , and $|+\rangle$ states initialized on the remaining $n - k = 2$ nodes. (b) For every input-output edge (dotted, pink), we add a controlled-Z (CZ) gate between the output and the pivot corresponding to the input. (c) For every input (or equivalently, every input-pivot edge—here dotted, green), apply a Hadamard to the corresponding pivot. (d) For every output-output edge (dotted, orange), apply a CZ gate between the corresponding qubits. The final stabilizers are $XZZZ$ and $ZXXX$ which differ from $XXXX$ and $ZZZZ$ by a local Clifford, namely a Hadamard H_1 . The degree is 3, so the depth bound stated by Theorem 8 is 9 which the final circuit visibly satisfies.

efficient algorithm which constructs a circuit $\mathcal{C}_U$ that logically implements U on the code corresponding to G , such that $\text{depth}(\mathcal{C}_U) \leq 4\mathcal{D}_{\max} + 6 + \ell_U$.

Conceivably, if a depth of $\mathcal{D}_{\max}$ is not immediately prohibitive, an additional factor of 4 significantly reduces the likelihood of practical utility. However, we will show that many important gates, including diagonal Clifford, nondiagonal Clifford, and diagonal non-Clifford gates, can be implemented generically with a reduction on the constant factor. We remark that in the results below we do not claim that the discussed circuits are necessarily the optimal way to implement a particular logical gate. Rather, these results serve as demonstrations that graph techniques enable us to achieve nontrivial improvements in circuit depth relative to the trivial unencode-apply-encode strategy.

Theorem 10. For any graph G with maximum degree $\mathcal{D}_{\max}$, every diagonal gate U which can be implemented in depth ℓ_U can be implemented logically on the code corresponding to G with depth at most $2\mathcal{D}_{\max} + 4 + \ell_U$.

Proof. The circuit is still simply the unencode-apply-reencode operation, but with cancellations due to the diagonal structure. Let E be the encoding circuit from Theorem 8. The circuit is of depth at most $2\mathcal{D}_{\max} + 3$ because it is of the form $CZ(V)H(V)CZ(V)$, i.e., a layer of CZ's acting on all the nodes V in the graph, a layer of H , then another

layer of CZ's. These respectively contribute at most $\mathcal{D}_{\max} + 1$, 1, and $\mathcal{D}_{\max} + 1$ to the encoding circuit depth. The logical operation is given by $CZ(V)H(V)CZ(V) \cdot U \cdot CZ(V)H(V)CZ(V) = CZ(V)H(V) \cdot U \cdot H(V)CZ(V)$ since U commutes with all CZ operations. This circuit has a reduced depth of $(\mathcal{D}_{\max} + 1) + (1) + (\ell_U) + (1) + (\mathcal{D}_{\max} + 1) = 2\mathcal{D}_{\max} + 4 + \ell_U$. ■

As a corollary, the CZ, S , and T gates can all be generically applied with a reduction by a factor of 2 from Theorem 9.

For a certain nondiagonal Clifford gate, the $\sqrt{X}$ gate, we are able to even further reduce the constant to just 1. This improvement relies on a specific ZX equivalence relation involving a purely graph transformation known as a *local complementation about a vertex* (LCV). The LCV is defined for any graph and has been presented under various guises in the literature [37,56,58]. We have, in fact, already used the LCV in Appendix A to enforce the Clifford rule of the ZXCF. Below, we fix the vertex set V and consider different graphs which can be made with the nodes in V . Given a graph $G = (V, E)$, let $N(u) = \{v \in V : (u, v) \in E\}$ be the neighbors of $u \in V$ in G . We can assume throughout this paper that G does not permit self-edges; i.e., $\forall u \in V, (u, u) \notin E$.

Definition 5. Let V be fixed and $G = (V, E)$ be a graph. A *local complementation about a vertex* $v \in V$

is a graph transformation $\text{LCV} : \mathcal{G} \times V \rightarrow \mathcal{G}$ given by $(V, E) \mapsto (V, E \Delta K(N(v)))$, where $K(S)$ is the edge set of a complete graph formed by nodes in $S \subseteq V$.

Intuitively, a LC about a vertex v can be described as follows. Consider the edges of the complete graph built from $N(v)$, and then toggle all these edges in v . That is, if $e \in E \cap K(N(v))$, then remove e from E , and if $e \in K(N(v)) \setminus E$, add e to E .

The LCV builds a bridge between unitary operations on graph codes and inherent graph transformations. Suppose that we have a graph $G = (\mathcal{I} \cup \mathcal{P} \cup \mathcal{O}, E)$ and we wish to apply a logical unitary $\bar{U}$; i.e., $\bar{U}_v$ for $v \in \mathcal{I}$. Generically, if we attempt to use the ZX equivalence rules to push $\bar{U}_v$ to the output, the rules will add more local operators and change edges, resulting in $\bar{U}$ applied to the output $\mathcal{O}$ of a *different graph*, which corresponds to a different code. To make this distinction clearer, let $U_v(G)$ be the operator U applied to $v \in V$ on G . If $v \in \mathcal{I}$, then $U_v(G)$ is a logical operator; otherwise, $U_v(G)$ is a physical operator. For brevity, if $U = U^{(1)} \dots U^{(m)}$, we denote $U_v(G) = U^{(1)} \dots U^{(m)}_v(G)$. We also denote $U_{v_1}(G) \otimes \dots \otimes U_{v_m}(G) = U_{\{v_1, \dots, v_m\}}(G)$.

The specific connection between the LCV and the $\sqrt{X}$ gate is given by the following standard ZX equivalence relation, the proof of which can be found in Refs. [37,56,58].

Claim 1. Let $G = (V, E)$ be a graph. Then $\sqrt{X}_u(G) = HSH_u(G) = S_{N(u)}(\text{LCV}(G, u))$. Here, H is the Hadamard gate and $S = |0\rangle\langle 0| + i|1\rangle\langle 1|$ is the phase gate.

Therefore, in order to apply a logical $\sqrt{X}$ on some $u \in \mathcal{I}$, we apply the LCV about u , and then apply a S gate on all nodes in $N(u)$. Since $N(u) \subseteq \mathcal{O} \cup \mathcal{P}$, all the S gates are physical. Since edges are CZ gates according to the formalism of the ZX calculus, an equivalent statement of the $\sqrt{X}$ gate implementation on an input $u \in \mathcal{I}$ is to apply S gates on every neighbor of u , as well as CZ gates between every pair of neighbors of u . This implementation is depicted in Fig. 15.

Lemma 2. Given a graph G representing a code, the graph transformation $\text{LCV}(G, u)$ for $u \in \mathcal{I}$ can be implemented with a quantum circuit of depth at most $\deg(u)$. There is an efficient algorithm outputting this circuit.

Proof. $\text{LCV}(G, u)$ is implemented by CZ_{vw} for all $(v, w) \in K(N(u))$. Since $|N(u)| = \deg(u)$, each node in $K(N(u))$ has degree $\deg(u) - 1$. By Vizing's theorem, there is an efficient algorithm to arrange the CZ gates to have depth at most $(\deg(u) - 1) + 1 = \deg(u)$. ■

Theorem 11. Let G be a graph. There is an efficient algorithm which finds a circuit acting on the encoded

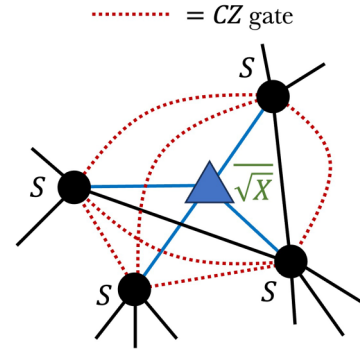


FIG. 15. Physical implementation of a logical $\sqrt{X}$ gate in the graph picture. The blue triangle is an input node (corresponding to a logical qubit) on which we wish to apply $\sqrt{X}$. This is accomplished by applying S gates on every neighbor, which are all outputs since input nodes are never connected. Furthermore, CZ gates are applied between every pair of neighbors, depicted as dotted red lines. The implementation of this gate does not depend on how the neighbors of the input are connected to each other. The transformation of edges on the graph is a local complementation about the input vertex.

space of the code given by G that implements the logical $\sqrt{X} = HSH$ gate on an input node u with depth at most $\deg(u) + 1$. Here, H is the Hadamard gate and $S = |0\rangle\langle 0| + i|1\rangle\langle 1|$ is the phase gate.

The proof is a direct consequence of Claim 1 and Lemma 2.

In general, we do not see a way to completely reduce the implementation depth of an arbitrary Clifford circuit because the Hadamard gate does not appear amenable to further optimization. To have a complete set of Clifford operations which can be implemented in depth strictly better than that of Theorem 9, one solution is to start with a self-dual CSS code, which admits a transversal Hadamard. In this case, the local complementation simplifications that push a H gate through to the output will cancel out in just the precise way as to produce H 's on all output nodes while preserving the graph structure. $\{H, \sqrt{X}, CZ\}$ is a generating set of the Clifford group, and thus the above results imply that any self-dual CSS code implements logical Cliffords at a depth that is reduced by at least a factor of 2 from the generic procedure of Theorem 9.

C. Game unification of stabilizer coding algorithms

Thus far, given a graph, we have described its corresponding stabilizers, given a set of logical operators, and provided a simple upper bound on the distance. To complete our fully graph formalism of stabilizer codes, we will show that important code algorithms—distance calculation or approximation, finding minimum-weight stabilizer

generators, and decoding—can all be expressed in a unified graph manner, namely by a single class of one-player games on the graph.

Definition 6 (Quantum lights out). An instance of a quantum lights out (QLO) game [65] on a graph G with inputs $\mathcal{I}$, pivots $\mathcal{P}$, and nonpivot outputs $\mathcal{O}$, is described as follows. Every node is endowed with both a light and a switch, both of which are binary (i.e., on or off), which are initially off. Flipping a switch at node v (a) permanently destroys the light (i.e., the light turns off forever and cannot be turned back on) at v if $v \notin \mathcal{I}$ and (b) toggles all intact (i.e., nondestroyed) lights on the neighbors of v . Let Alice be the player. The graph begins with some initial configuration of lights. The game consists of two rounds:

- (1) Depending on the instance, Alice may be allowed to flip switches in the input.
- (2) Alice chooses a set of n nodes $v_1, \dots, v_n \in \mathcal{P} \cup \mathcal{O}$, subject to some instance-dependent constraints. For each node v_i , in order, Alice chooses whether to toggle the switch at v_i or only to destroy the light on v_i . The game ends when the lights reach an instance-specific desired final configuration, usually with all noninput lights off or destroyed.

The specific instance of the QLO game specifies the initial light configuration, whether round 1 exists, the constraints in round 2, and the desired final configuration, e.g., all lights are turned off. Regardless of instance, Alice must make at least one move in round (2). We say that Alice wins in n moves if she makes n choices that lead to the desired final configuration of the game.

At its core, QLO is a simple way to reformulate the ZX-graph rule from Lemma 1, alongside some technical details. X operators are equivalent to Z on all neighbors, and in QLO we use a light to indicate the presence of a Z operator (so applying Z toggles a light, and the switch which toggles neighboring lights is simply a Pauli X operator with the ZX-graph rule. We use this observation repeatedly below.

By correctly choosing the instance of the game, we can represent many important stabilizer coding algorithms as strategies for QLO games. We begin by showing that finding the distance of a code is QLO. Note that the distance is invariant under unitaries on inputs, and hence every equivalent Clifford encoder has the same distance.

Theorem 12 (Distance is QLO). Given a graph G with inputs $\mathcal{I}$, pivots $\mathcal{P}$, and nonpivot outputs $\mathcal{O}$, denote the distance of the code represented by $d(G)$. $d(G)$ is calculated by the following QLO instance. Initially, all lights are off. In round 1, Alice may flip any number s of switches on input nodes $\mathcal{I}$. In round 2, Alice has no constraints and

may flip any noninput switch or destroy any noninput light. The final configuration consists of all noninput lights off or destroyed, and either at least one input light is on, or $s \geq 1$. Under this instance,

$$d(G) = \min_{\mathcal{A}} m_D(\mathcal{A}, G), \quad (12)$$

where $\mathcal{A}$ is Alice's strategy and $m_D(\mathcal{A}, G)$ is the number of moves she made in round 2. As a consequence, every distance approximation algorithm on G to multiplicative error ϵ is an ϵ -approximately optimal strategy in the distance QLO game instance. We illustrate distance calculation as a QLO game in Fig. 16.

Proof. As a consequence of the ZX-graph rules from Lemma 1, each of the switches flipped in the i th input corresponds to $\bar{X}_i$, i.e., logical X on logical qubit i . If the game allowed for flipping a single light at a node, this would correspond to applying a Z on the node. There is no point to applying a Z operator multiple times to the same output node, as they will cancel (and we can order them so that no phase occurs). Hence, instead we simplify by destroying the light altogether as a reminder that retoggling this light does not increase the weight of an operator. Next, toggling lights on all neighbors of a node v corresponds to applying $X(v)$. There is no additional Pauli weight if we apply both $X(v)$ and $Z(v)$, so for convenience we also destroy the light on v when we flip the switch. In sum, destroying lights and flipping switches correspond to physical Z and X operators. Lights turned on at inputs after all lights on outputs have been extinguished are logical Z operators. Consequently, the entire distance QLO game consists of applying some logical X operators and translating them into physical operators and logical Z operators. In other words, the physical operator represented by the n moves applies the logical product of the initial logical X 's and the final logical Z 's, which is a nontrivial logical operator if at least one of the two is not vacuous, i.e., if at least one input light is left on or $s \geq 1$. Since each move increases the weight of the physical operator by exactly 1, the minimum number of moves is precisely the distance. ■

Since finding the distance of a stabilizer code is **NP**-complete [66], we may immediately conclude that optimally playing QLO with the setup as above is also **NP**-complete.

Theorem 13 (Minimum-weight generator selection is QLO). Let $\{S_i\}$ be the canonical stabilizer generators of G as given by Eq. (10). Without loss of generality, fix S_1 the first stabilizer. Then the minimum-weight stabilizer S'_1 which can replace S_1 in the tableau, i.e., remains independent of $S_2, \dots, S_{n-k}$, is given by a QLO game instance. Initially, all lights are off. There is no round 1. In round 2, Alice may freely flip switches on any node except v_1 , the

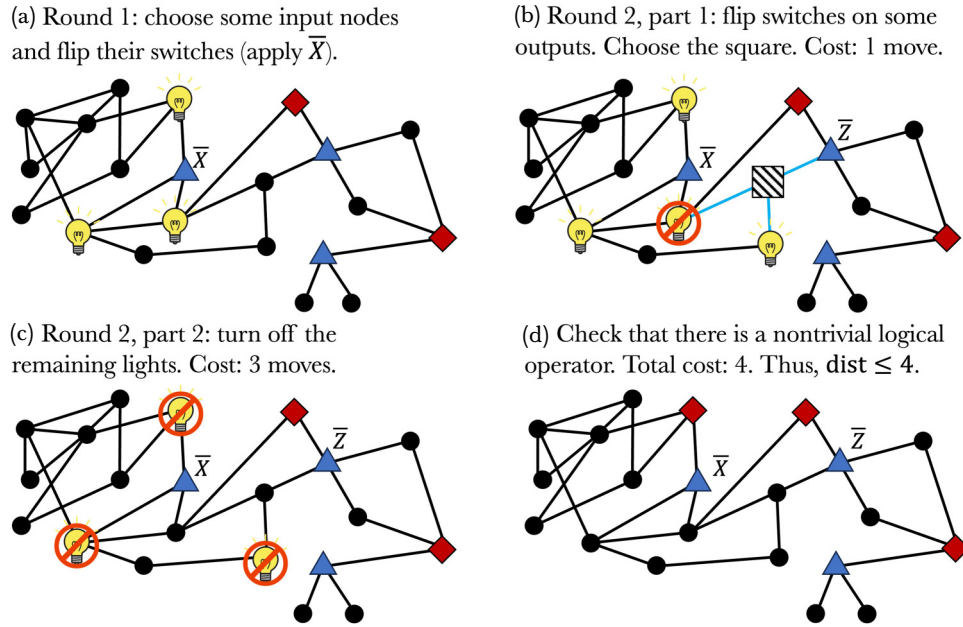


FIG. 16. A depiction of distance calculation as a quantum lights out (QLO) game. Here, inputs are blue triangles, pivots red diamonds, and nonpivot output nodes black circles. In (a)—round 1—Alice may choose to flip switches on inputs, which turn lights on their neighbors. Switch flipping corresponds to applying X operators by the ZX-graph rules of Lemma 1. In (b)—round 2—Alice flips output switches and then turns the remaining lights off directly. The total cost is the number of moves she makes in round 2, and the distance is at most the number of moves. In general, the distance is the minimum cost over all choices of moves, as shown in Theorem 12.

switch in $\mathcal{O}$ corresponding to S_1 . She must flip the switch at v_1 *exactly once*. She may also destroy any lights in $\mathcal{O} \cup \mathcal{P}$. The final configuration consists of all lights destroyed or off. Under this game instance,

$$|S'_1| = \min_{\mathcal{A}} m_{\text{WR}}(\mathcal{A}, G), \quad (13)$$

where $\mathcal{A}$ is Alice's strategy and $m_{\text{WR}}(\mathcal{A}, G)$ is the number of moves she made in round 2. Thus the minimum weight is equal to the minimum number of moves to end the game.

Proof. Initially, we apply $X(v_1)$. The canonical stabilizer represents the trivial strategy of flipping pivot switches to remove lights from inputs, and then destroying all remaining lights. However, any sequence of moves that turns all lights off, i.e., effectively applies the identity on the graph, is also a stabilizer. Because we restrict Alice from flipping the switch at v_1 , the resultant stabilizer is both nontrivial (i.e., not the identity) and independent from all other S_i , since S'_1 is still the only stabilizer at $X(v_1)$. Hence, the minimum number of moves, plus one for $X(v_1)$, represents the new weight as claimed. ■

By examining the proof, we also observe that the acts of writing down the canonical stabilizers and the canonical logical operators are also instances of substantially simpler QLO instances. In Fig. 17, we depict an example strategy to find a stabilizer generator on a node v for

a generic graph, which beats the strategy corresponding to the canonical stabilizer.

While the complexity of finding minimum-weight generators does not appear to be completely known, this problem is likely about as challenging as distance calculation or decoding since it is precisely the problem of finding short vectors in stabilizer subspaces.

Theorem 14 (Decoding is QLO). Suppose that after application of a noisy depolarizing (or in general, any mixture of Pauli errors) channel to the code word, we measure the canonical (or more generally, any known set of) stabilizer generators of a graph G with inputs $\mathcal{I}$, pivots $\mathcal{P}$, and nonpivot outputs $\mathcal{O}$. We thereby obtain syndrome bits $s_1, \dots, s_{n-k} \in \{\pm 1\}$. Then, the recovery map that should be applied is given by a QLO game instance. Initially, the light at each vertex $v_i \in \mathcal{O}$ corresponding to stabilizer S_i is turned on if $s_i = -1$. All remaining lights start off. In round 1, Alice may flip any number of switches in $\mathcal{I}$. In round 2, Alice has no constraints aside from only being able to act on nodes in $\mathcal{O} \cup \mathcal{P}$, as usual. The final configuration consists of all noninput lights off or destroyed. Under this instance,

$$r(\mathbf{s}, G) = \min_{\mathcal{A}} m_{\text{DC}}(\mathcal{A}, \mathbf{s}, G), \quad (14)$$

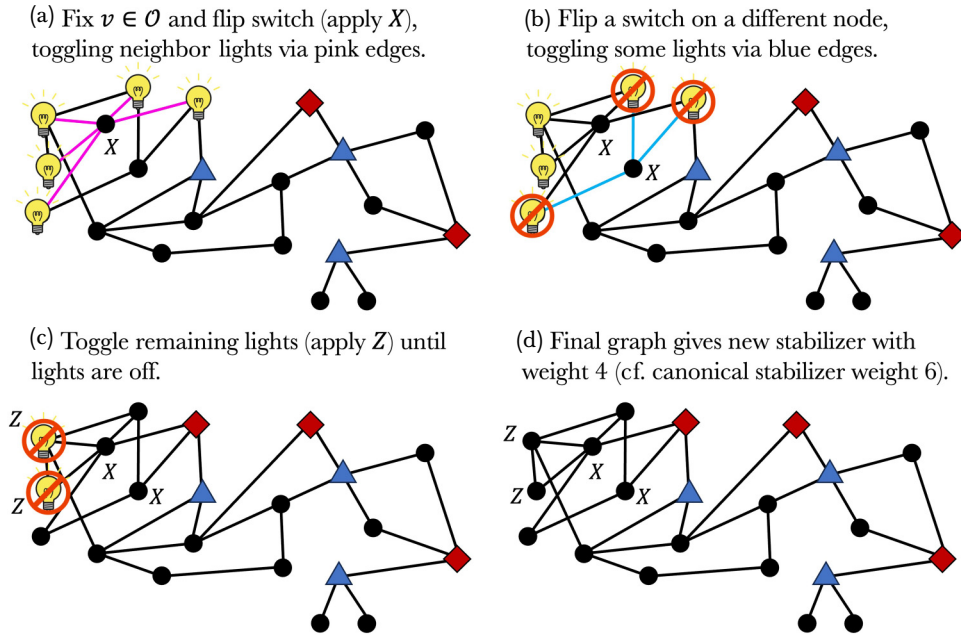


FIG. 17. A depiction of minimum-weight generator selection as a quantum lights out (QLO) game. Inputs are blue triangles, pivots red diamonds, and nonpivot outputs black circles. Flipping a switch corresponds to a Pauli X and toggling a light corresponds to a Pauli Z . In (a), we fix a nonpivot output node $v \in \mathcal{O}$, for the corresponding stabilizer of which we wish to find a minimum-weight version. We flip its switch to begin the game, and then in (b) we choose other switches to flip. We then toggle remaining lights in (c) so that all lights are off. The total cost is the number of moves made in the entire process, which in this case is 4. Note that the canonical stabilizers are given by one unique strategy—flip the switch on v then directly turn off all the lights one by one—and in this case the depicted strategy has lower weight than the canonical strategy.

where $\mathbf{s}$ is the vector of syndromes, r is the number of moves in the optimal recovery map, $\mathcal{A}$ is Alice's strategy, and $m_{\text{DC}}(\mathcal{A}, \mathbf{s}, G)$ is the number of moves she made in round 2. We note that input lights being on does not affect anything in round 2 or in the minimization condition. This means that, where convenient, input lights can be completely ignored.

Proof. Any decoding procedure can be split into steps, where each step involves applying a single-qubit Pauli to an output node. The final recovery map is then given by the product of all the single-qubit Paulis. We begin with a light on at v_i if $s_i = -1$ with no lights on at any pivot. At each step, we will apply a chosen Pauli on a chosen output node. Using casework, we analyze each possible choice of Pauli and node, and which stabilizers for which this operation will flip the syndrome:

- (1) $E = Z(v)$ for $v \in \mathcal{O}$. E anticommutes with stabilizers which contain $X(v)$. The unique canonical generator which contains $X(v)$ is $S(v)$. Therefore, the syndrome associated with v is flipped.
- (2) $E = Z(v)$ for $v \in \mathcal{P}$. E anticommutes with stabilizers which contain $X(v)$. These are the stabilizers associated with the *nonpivot output neighbors* of the *input* associated with v , i.e., $S(w)$ for all $w \in$

$o(i(v))$. Applying a Z to a node $v \in \mathcal{P}$ negates the syndromes of *all* stabilizers of the vertices $o(i(v))$.

- (3) $E = X(v)$ on any $v \in \mathcal{O} \cup \mathcal{P}$. E anticommutes with stabilizers which contain $Z(v)$. These are $S(w)$ for $w \in o(v)$ as well as $w \in o(i(p(v)))$ due to the presence of Z 's in $S(w)$ on $o(p(i(w)))$ (note that the order of i and p is backward when we consider stabilizers affected by a node). Applying an X to a node $v \in \mathcal{O} \cup \mathcal{P}$ therefore toggles all lights in $o(v) \Delta o(i(p(v)))$. The symmetric difference is present because syndromes corresponding to nodes in $o(v) \cap o(i(p(v)))$ are flipped twice and thus identically not flipped at all.

We wish to reframe the action of Paulis on the syndromes in such a way that we can identify all Paulis operators as a choice of either flipping a switch, which toggles lights on neighbors, or toggling a light. (For simplicity, we say toggling instead of destroying a light here, since destroying is merely a simplification so as to avoid double counting multiple Paulis applied on the same node.) Currently, Z on $v \in \mathcal{O}$ only affects the syndrome on v , so we may think of Z on v as toggling the light on v . If we apply Z on $v \in \mathcal{P}$, we toggle syndromes $oi(v)$. One way to achieve this by using switches and toggles is to flip the switch on $i(v) \in \mathcal{I}$ and then toggle v . By construction, flipping the

switch on $i(v)$ toggles lights in the neighbors $N(i(v))$, and $N(i(v)) = \{v\} \cup o(i(v))$ since $i(v)$ is only connected to a single pivot, which is by definition v . Hence, if we flip the switch on $i(v)$ and then toggle v , we equivalently toggle $o(i(v))$ as desired. But flipping switches on nodes in $\mathcal{I}$ does not cost moves and can be absorbed into the arbitrary allowed action in round 1. Hence, the application of Z on $v \in \mathcal{P}$ is equivalent to some costless moves in round 1 plus a single toggle of the light in v in round 2. Similarly, when applying X to $v \in \mathcal{O} \cup \mathcal{P}$, we flip the syndromes in $o(v) \Delta o(i(p(v)))$ and therefore want to toggle exactly those lights using switches and toggles. We may do so by flipping the switch at v as well as all switches in $i(p(v))$; this toggles $o(v) \Delta o(i(p(v)))$ as desired, and costs only a single move because the switches flipped in $i(p(v))$ are costless and absorbed into round (1). Conversely, any switch flip or light toggle can be associated with the construction of a recovery operator. Specifically, a switch flip on $\mathcal{O} \cup \mathcal{P}$ is associated with an X operator, and a light toggle on $\mathcal{O} \cup \mathcal{P}$ is associated with a Z operator. This association differs from the true action only by switch flips on input nodes, which are costless logical operations that do not affect the syndrome and can be absorbed into round 1. Finally, when all lights are off, by our construction, all the syndromes are zero, so the decoding process has finished. We note that destroying a light at a node after applying the switch at that node represents ambivalence between applying an X or a Y , but if Alice is interested in finding the proper recovery map instead of just the size of the map, she should leave all lights unbroken. Then, when she is done toggling switches, she can choose to turn off any lights at vertices the switches of which she toggled without having this count as a move in round 2. We conclude that decoding algorithms are QLO strategies. ■

We illustrate the origin of decoding as a QLO game in Fig. 18. Namely, errors are applied to output nodes, which affect certain stabilizers associated with nonpivot output nodes $v \in \mathcal{O}$. The affected stabilizers (those giving a violated syndrome) have their lights turned on according to a specific depicted pattern, and the sequence of moves to turn the lights off corresponds to the recovery operation. We note that construction of the recovery map is very closely related to the calculation of the distance. Indeed, finding the minimum-weight recovery operator is also **NP**-complete. There are, however, more general definitions of optimal recovery that one can consider, which factor in degeneracy classes. In such cases, decoding can become as hard as **#P**-complete [67].

It is illustrative to compare the decoding QLO game to a classical analog. For a classical code with a Tanner graph $G = (L, R, E)$, where L and R are, respectively, the left and right node sets and E is the edge set, we may associate every check node in R with a light and every data node in L with a switch, such that flipping the switch toggles

the neighboring lights. At the start of decoding, a light is on if the check corresponding to that node is violated. We then iteratively choose data bits to flip and update the syndromes. This is precisely equivalent to flipping switches on the data nodes, which toggle the check node lights. When all lights are off, the recovered data must be a code word. This classical version of lights out on Tanner graphs is thus **NP**-complete as well. At the same time, classical lights out can be solved efficiently if the graph is a sufficiently good expander, using a simple greedy algorithm [27]. We later show that if a graph of a stabilizer code satisfies a different but related structural property, then a greedy QLO strategy can also efficiently decode errors.

Had we been considering erasure channels instead of depolarizing channels, the QLO game would look very different. In the regimes where only a small number of errors has occurred, such as less than half the distance of a given code, then being able to pinpoint the locations of those errors leaves us able to follow a generic recovery algorithm that has no connection to QLO. For instance, we could replace each erased qubit with the maximally mixed state, measure all the stabilizers, and search for any recovery operator supported on the erased qubits, all in polynomial time. It is more interesting to study the erasure channel in the cases where the number of errors approaches the threshold of the code and not the distance of the code. We leave this question to future work.

The unified expression of all three algorithms, combined with the many complex techniques to perform optimizations on graphs that have been developed in past decades, suggests that the study of approximating optimal QLO strategies may be a promising pathway for devising coding algorithms.

V. APPLICATIONS OF THE GRAPH FORMALISM

We proceed to provide constructive evidence in three areas for the utility of a graph representation. First, in Sec. VA, we argue that for near-term experimental purposes, the graph formalism provides a simple and flexible prescription for the design of codes with a desired rate R (where $R = k/n$ and k is the number of logical qubits) and distance d . Intuitively, this advantage occurs because we can appeal to the geometrical and topological structure of graphs, as well as take inspiration from well-studied graphs, to get close to a distance d . We then describe improved results for random codes and construct an efficient graph decoder for a family of stabilizer codes constructed from graphs.

A. Flexible code design

Generally, stabilizer tableaus have proven more useful as a description rather than a constructive mechanism. Most well-known stabilizer codes constructed are actually CSS codes [26,28,29,36], and non-CSS stabilizer codes are

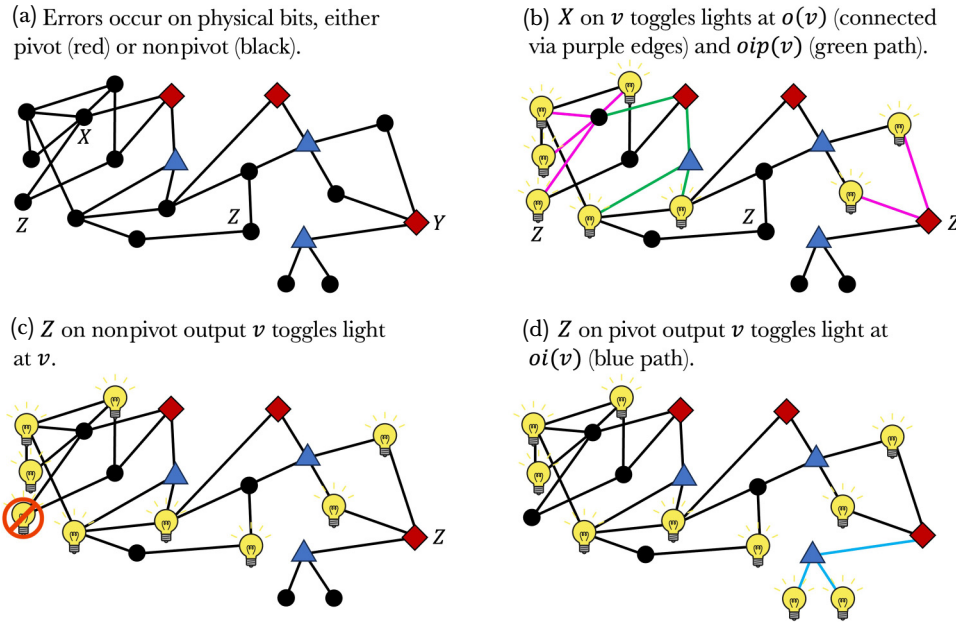


FIG. 18. An illustration of decoding a stabilizer code as a quantum lights out (QLO) game. Inputs are blue triangles, pivots are red diamonds, and nonpivot outputs are black circles. For shorthand, we let $oip(v) := o(i(p(v)))$ and $oi(v) := o(i(v))$, where $o(v)$, $i(v)$, and $p(v)$ are as in Definition 3. In (a), errors occur on physical bits, depicted by some Paulis on output nodes. Here, the lights are toggled not by the ZX-graph rule from Lemma 1, but by the nodes for which the corresponding stabilizer anticommutes with the error (i.e., stabilizers which measure to a violated syndrome). For X on an output node $v \in \mathcal{O} \cup \mathcal{P}$, as shown in (b), the affected stabilizers are those corresponding to $o(v)$ the nonpivot output neighbors and $oip(v)$ the nonpivot output neighbors of the inputs corresponding to the pivot neighbors of v ; their lights thus turn on. Generally, if a node is in $o(v) \cap oip(v)$, it will be toggled twice and not turn on; however, in the depicted graph $o(v)$ and $oip(v)$ are disjoint. A Z on a nonpivot output $v \in \mathcal{O}$ simply toggles the light on v as shown in (c), while a Z on a pivot $v \in \mathcal{P}$ toggles the light at $oi(v)$ —the nonpivot output neighbors of inputs neighboring v —as shown in (d). The QLO game then consists of finding the smallest number of moves to turn lights off (i.e., fix violated syndromes), and the sequence of moves corresponds to the recovery operation.

usually found by some alternative presentation. The lack of use of stabilizer tableaux is not unusual, as it is simply not obvious as to what collections of Pauli strings construct codes with high distance, easy decoding, gates, geometric properties, etc. The universality of the graph representation of stabilizer codes, combined with a rich history of graphs in error correction [27,29,45,68], suggests that a promising technique to construct certain codes lies in finding sufficiently “nice” graphs.

Because of its universal expressive power, the graph representation in this work has parameters that scale with distance. As we have shown previously, the distance is bounded above by degree parameters of the graph, and the canonical stabilizers have weights that also scale with the degree. Therefore, the graph representation is best suited for finding codes with a desired approximate numerical values of code parameters, as opposed to asymptotically good LDPC codes. In other words, graphs provide a flexible technique to generate potentially practical small-scale codes.

As an example of this flexibility, if one wanted to create a code with a particular distance, one could set the degree of the graph in accordance with Proposition 1. The degree

choice gives an upper bound on the distance, so having a sufficiently large degree is a necessary but not sufficient condition to achieve a code with the desired distance. To actually get a good distance, one can either check explicitly for small cases, or use graphs which have a provable distance lower bound. The codes in this section are sufficiently small that we can usually apply the former strategy. However, in a later section, when we investigate decoding algorithms, we will address the second strategy and provide explicit distance lower bounds based on the degree for graphs which satisfy certain structural properties. Achieving a certain rate translates to having a sufficiently high density of inputs, which in turn can influence the required degree of the input neighbors. Some experimental devices or algorithms may benefit from codes that have at most a particular number of physical qubits or encode a minimum number of logical qubits, both of which are easy to set in the graph formalism. If ensuring that the physical qubits behave similarly in a highly uniform manner, a graph could be chosen to be highly regular and symmetric, such as the five-, seven-, and nine-qubit codes from Sec. III. Further still, a graph can be chosen to be planar and local in two dimensions, or perhaps supporting

only a fixed number of long-range connections for ease of practical implementation. (To implement a code represented by a graph, we place physical qubits at every output node, so the qubits inherit the geometric properties of the graph. Moreover, the canonical stabilizers are local in the graph in the sense that every qubit supported in the stabilizer associated with $v \in \mathcal{O}$ is at most three edges away from v . We note, however, that there certainly exist codes which are, e.g., planar, but the graph of which in our representation is not planar.) On the other hand, a graph embedded into a higher-dimensional space, such as, e.g., a high-dimensional grid graph, could be easier to represent or simulate classically. We will give examples of codes with each of these properties below.

The distance calculations of codes in this section have been performed by direct computation. The code is freely available for use [69].

1. The dodecahedral code

We now showcase a few examples of novel small codes that we have constructed from the graph formalism. As a first example, we examine the seven-qubit Steane code in Fig. 4, which takes the shape of a cube. Such a geometric structure motivates the exploration of other highly regular geometric solids as graph codes, such as the platonic solids. While the graphs of a tetrahedron and octahedron perform rather poorly as codes due to their particular symmetries, the graph of a dodecahedron represents a relatively good code. In Fig. 19, we exhibit the dodecahedral code, which has parameters $[[16, 4, 3]]$. Since the degree of every node is 3, the dodecahedral code saturates its distance-degree bound. To maintain a distance equal to the degree, no two inputs may share a neighbor (as otherwise there exists a logical operator of lower weight), so the maximum number of inputs that can be selected is four. We remark that this graph is not a CSS code, being nonbipartite. Furthermore, having distance 3, the dodecahedral code does not improve upon Hamming-type quantum codes in parameters, e.g., the $[[15, 7, 3]]$ quantum Hamming code [70]. Nonetheless, it is interesting that simply inserting another platonic solid into the graph formalism has provided a new, nontrivial, and non-CSS code.

To complete our discussion of the dodecahedral code, we apply Theorem 8 to give the code a low-depth encoding circuit. We illustrate two representations of the encoding circuit in Fig. 20. On the left, we draw a ZX diagram which appears as an intermediate step in the algorithmic construction in Theorem 8. On the right, we translate the ZX diagram into standard quantum circuit notation. The translation is an immediate consequence of the ZX-calculus rules: a Z node with a single output is a $|+\rangle$ state and a Hadamarded edge between two Z nodes implements a CZ gate. Corresponding to the theorem, the CZ gates before

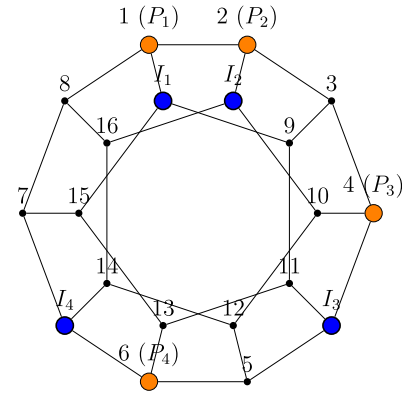


FIG. 19. The $[[16, 4, 3]]$ dodecahedral code. Blue nodes are inputs, black nodes are outputs, and orange nodes are pivots. This code is the optimal packing of input nodes into the dodecahedral graph so that no inputs have a path between them of length less than 3. As the dodecahedron has odd cycles, it is not a bipartite graph and therefore this code is a non-CSS stabilizer code. The labels indicate an ordering of the input and output nodes as well as the pivots (lowest-numbered nodes next to each input). There are no extra input-pivot edges but we allow pivot-pivot edges where this does not change our constructions. An ordering can be chosen on the nodes to avoid all pivot-pivot edges.

the layer of Hadamards form the input-output edges; the Hadamards effectively form input-pivot edges; and the CZ gates after the layer of Hadamards form the edges from noninputs to noninputs. We also observe that the application of the ZX-calculus simplification rules to the diagram in Fig. 20, such as merging Z nodes together, recovers the graph representation of the dodecahedral code itself. In this sense the encoding circuit appears almost immediately from the graph representation.

While our results about the worst-case encoder depth suggest that it may have a depth of up to 9, we have applied some analysis by inspection to further optimize the depth to 6. Further compilation of this circuit would allow us to reduce the depth further. This is because output-output edges commute with every gate in the circuit as they are not on the input-pivot wires containing the only nondiagonal gates. In particular, the five gates in the last block can be moved to the second, first, fourth, first, and first blocks, respectively, resulting in a circuit of depth 5. This is the minimum possible depth as that is the number of gates acting on some qubits, such as the first input.

2. Covering spaces and the 5-covered icosahedral code

We next consider a code based on the graph of an icosahedron, depicted in Figs. 21(a) and 21(b). Since the vertices of an icosahedron have degree 5, we can hope that a code based on this graph would have distance 5. Unfortunately, we find that the code under-performs the degree bound because a Y gate on two antipodal vertices is equivalent to applying a Z to each vertex. Hence, a

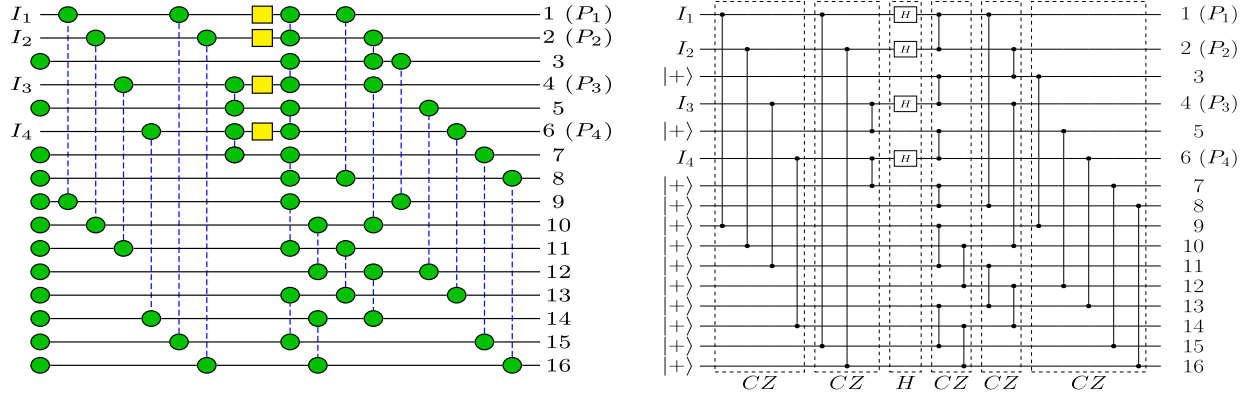


FIG. 20. Two presentations of the encoding circuit of the dodecahedral code, labeled as in Fig. 19. On the left, the presentation is of a ZX diagram. Yellow boxes indicate H gates and blue dashed edges are, as usual, Hadamarded. On the right, the presentation is of a standard quantum circuit. Each dashed block represents a set of transversal (depth-1) operations, so that the circuit depth is the number of blocks, in this case six. The presentation here is partially optimized, by inspection, to improve upon the upper bound given in Theorem 8. It is possible with further heuristics to produce a depth-5 (but more visually complicated) circuit, which is provably optimal.

logical Y on any input is a logical operation of weight 3, with one Y on the vertex opposite the input and two more on any two antipodal vertices. This observation also implies that if we wish to add more than one input to the icosahedron, our distance will be reduced further, as a logical operation would just consist of Y 's on two vertices opposite the inputs. To get around this issue, we lift the graph into a finite covering space. Specifically, we can imagine taking a double cover of the icosahedron by “spearing” the icosahedron through the centers of two opposite faces and attaching a winding number to each vertex, which is computed modulo 2. Such a construction yields a double-covered icosahedron. In Fig. 21, we illustrate the double-covered icosahedron in an intuitive manner, namely by gluing two icosahedrons together across the spear. We find that the double-covered graph with a single input has distance 5. In fact, we can improve the rate slightly by taking a t -cover of the icosahedron. Specifically, when $t = 5$ we can fit six input vertices, where the inputs are not vertices of the punctured faces, and each of which is wound 300° around from the previous input. In Fig. 21, if we puncture the two central faces, the inputs in the five copies of the icosahedron will be on copies of the six vertices along the perimeter, each rotated 300° from the last. This construction creates a $[[54, 6, 5]]$ code.

The ideas used to create the 5-covered icosahedron can be extended more generally to extend otherwise finite graphs. In fact, arbitrarily complicated covers can be proposed, leading all the way up to the universal cover of the graph. If regularity and a finite set of vertices are desired, some periodic boundary conditions can be established to keep the graph finite. In general, such covering space graphs may provide a promising path toward code design recipes in this framework. In the case of Cayley graphs, similar ideas have already been utilized

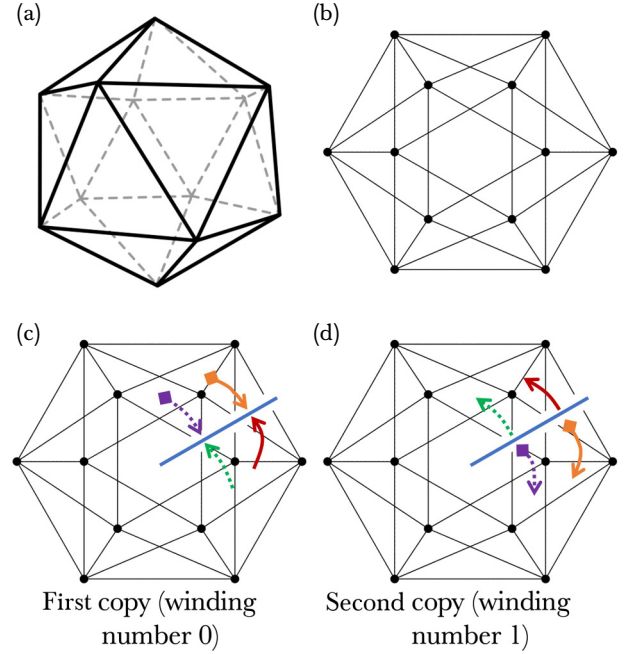


FIG. 21. Double covering of the icosahedron. We begin with an icosahedron, shown in (a). The graph version of this solid is given in (b). To double cover it, we make two copies of the icosahedron and “spear” it through the center, illustrated with a solid blue line. In the first copy, as we move (clockwise) counterclockwise and pass through the spear, we “teleport” to the second copy and continue as shown by the (solid orange arrows with the square tails) solid red arrows. Similarly, in the second copy, as we move (clockwise) counterclockwise and pass through the spear, we “teleport” to the first copy and continue as shown by the (dotted purple arrows with the square tails) dotted green arrows. This idea can be extended to a general t -covered icosahedron by associating each copy with a winding number. Going (clockwise) counterclockwise through the spear teleports to the copy with winding number one (lower) higher mod t .

to construct quantum LDPC codes from Cayley graphs of free groups with generators comprised of columns of a classical parity-check matrix [71].

3. Planar and higher-dimensional lattice-type codes

Even more generally, lattice-type codes enable the construction of families of codes with increasing degree (and thus, potentially, increasing distance) while also preserving some notions of regularity that are often desirable in code construction. If we wish to remain in two dimensions and also have a Euclidean lattice, we are restricted in the sense that the triangular lattice, which has degree 6, is the highest-degree such lattice. We numerically confirmed that such lattices do yield distance-6 codes. Note that while surface codes also have Euclidean planar connections, they can allow for much higher distances. This comes at the trade-off of having only a constant number of logical qubits. If we were to construct a large two-dimensional triangular lattice, we could achieve a different trade-off: capping our distance at 6, but being able to place numerous inputs in the lattice, achieving a constant rate.

Beyond degree 6, we can either raise the dimension or embed graphs in non-Euclidean spaces. For example, an embedding of triangular surfaces into hyperbolic spaces yields the Poincaré disk map. However, because the Poincaré disk packs infinitely many points into a finite space, we would have to construct finite approximations and then study the effect of artificially establishing a finite boundary. Analysis of such effects appears difficult and is beyond the scope of this work. Nonetheless, such classes of codes in non-Euclidean spaces may prove useful, and even connected to Pastawski-Yoshida-Harlow-Preskill (HaPPY) codes and other hyperbolic-type codes [72–74] which use graphs in hyperbolic space.

The other path—increasing the dimension—is more amenable to analysis. A simple, general recipe to construct a code from a lattice is to cut off the lattice in each dimension at a finite number of intervals, endow the finite version with periodic boundary conditions in each direction, and then embed some inputs and pivots within them such that the choices of pivots are valid and inputs are not connected.

One immediate advantage of this approach is a robustness to deformation that does not hold for CSS codes. In particular, we have shown in Theorem 7 that bipartite graphs correspond to CSS codes. For lattices with periodic boundary conditions, the bipartite condition is equivalent to the length of every single dimension of the torus being even. Such a requirement is very nonlocal and does not change the central architecture of the code. Even for CSS torii, the addition of one vertex far from any inputs such that it forms an odd-length cycle will make the code no longer CSS but will not meaningfully affect any of the parameters of the code.

When constructing families of codes in the graph formalism, it is important to consider the geometry of the graph as well as the placement of the inputs. To make use of the graph formalism for computing stabilizers and logical operations, the identification of pivot nodes is paramount. While the ZXCF does not allow pivot-pivot edges, these do not affect the expressions of the graph formalism. Thus, for easy pivot selection, it suffices for all of the inputs to be separated by paths of length at least 3 from each other in the graph. Although such a restriction is not necessary for code construction, it does allow for the input of any neighbor to be chosen as its pivot as that node would necessarily not border any other inputs by the distance condition. The down side to such graphs, however, is that their rate is bounded as $R \leq 1/q$, where q is the degree of the graph (assuming it is regular). In other words, they have a strong rate-distance trade-off. Nonetheless, such graphs may be interesting at small scales, as we have already seen.

In higher dimensions, perhaps the simplest lattice we can utilize to build a code is the Boolean hypercube in an arbitrary m dimensions. We represent the hypercube by labeling nodes as length- m bit strings and connecting two nodes if their Hamming distance is exactly 1, i.e., they differ by one bit flip. We observe immediately that the hypercube code is CSS for all m , since the hypercube is bipartite. To select inputs, we observe that if we wish to place inputs such that they are separated by paths of length at least 3, we can leverage classical coding theory to provide this guarantee automatically. Specifically, we consider the *classical* $[2^r - 1, 2^r - r - 1, 3]$ Hamming codes [75]. Letting $m = 2^r - 1$, we can equivalently express the parameters as $[m, m - \log(m + 1), 3]$, where the code is perfect when m is one less than a power of 2. Note that all logarithms in this paper are taken to be base 2. Since this code has $m - \log(m + 1)$ logical bits, there are a total of $2^m / (m + 1)$ code words represented as length- m bit strings. Because the distance of the Hamming code is 3, all such code words are guaranteed to be 3-spaced in the hypercube graph. Therefore, by choosing the input nodes in the m -dimensional hypercube to be those the bit string of which is a code word in the $[m, m - \log(m + 1), 3]$ Hamming code, we obtain a code with $2^m / (m + 1)$ logical qubits, $2^m - 2^m / (m + 1) = m 2^m / (m + 1)$ physical qubits, and a distance $d(m)$ bounded above by the degree m . That is, a $[[m 2^m / (m + 1), 2^m / (m + 1), d(m)]]$ family of codes. Due to the 3-spacing of inputs, any choice of pivots uniquely connected to a corresponding input will do; later, we will show that it is both possible and particularly convenient to choose the pivots such that they are also separated by paths of length at least 3. We will also later prove in Sec. VB that $d(m) \gtrsim m/2$, up to an additive error of 2, and give evidence that $d(m) = m$. For example, for $m = 7$, the code has parameters $[[112, 16, 7]]$. (This distance has been computed numerically; we later discuss this

in more detail alongside the lower bound.) Asymptotically, the code scales as $\llbracket n, \Theta(n/\log n), \Theta(\log n) \rrbracket$.

Definition 7. Let the family of *hypercube codes* be the set of $\llbracket m2^m/(m+1), 2^m/(m+1), D_m \rrbracket$ codes defined above, parametrized by $m \geq 3$.

To improve the rate further without the strong distance-rate trade-off barrier $R \leq 1/d$, we will need to use the most general form of graph codes, namely those with inputs separated by paths of length 2. We conclude this section with some remarks on how such codes may be constructed. In a grid lattice on a m -dimensional torus, we wish to pack inputs more closely while being mindful of the fact that our distance will be bounded above by the number of output nodes next to each output node. This would mean packing inputs with a graph distance of fewer than 3 between them. Such a packing will reduce the distance below the degree of the graph, but we can hope to try to not let the distance be reduced “too much” while (hopefully) considerably improving the packing density of inputs and thus the rate of the code.

One such construction is as follows. First, consider a foliation of the m -torus into sequentially numbered “layers” of $(m-1)$ -dimensional toric hypersurfaces that together stack into a m -dimensional torus. In each layer, we will consider sets of vertices grouped by their *index*, the sum of their integer coordinates taken mod 3. We can assume that the size of each torus dimension is a multiple of 3 to make sure these sets are well defined. Note that the index does not consider the m th coordinate, as that is determined by the number of the layer in which a node ends up:

- (1) In layer 1, we set all nodes of index 0 to be inputs, with pivots in layer 2.
- (2) In layer 2, we set all nodes of index 0 or 1 to be pivots.
- (3) In layer 3, we set all nodes of index 1 to be inputs, with pivots in layer 2.
- (4) We repeat this three-layer structure as many times as desired.

Such a pattern is essentially a three-layered, high-dimensional analog of a checkerboard pattern. Again, we see that our distance upper bound is d , as vertices such as those of index 1 in layer 1 have d inputs among their $2d$ neighbors. More sophisticated techniques are necessary, however, to lower bound the distance. However, the rate of this code is a constant $\frac{2}{7}$, which is achieved at any scale of this code, without the need for extreme numbers of qubits for the asymptotic behavior of the code parameters to be expressed. Additionally, we note that if an odd number of three-layer structures are used or if any of the dimensions of each layer is odd, the code graph will have a cycle of

odd length, meaning that the resulting code will not be a CSS code.

These examples showcase a wide variety of families and constructions, both small and large, that can be achieved with the graph formalism. While some are sufficiently small to admit brute-force computational distance calculations, others are more difficult to analyze and will require additional distance-bounding tools to understand. We take a first step in this direction in the next section.

B. Decoding algorithms and analysis

We continue our discussion by constructing a simple greedy decoding algorithm on graphs. We give a sufficient-condition characterization of graphs for which the greedy decoder successfully recovers errors up to half of the theoretical maximum. That is, graphs which satisfy this certain property have a provable distance *lower bound* based on its degree. Finally, we will show that the $\llbracket m2^m/(m+1), 2^m/(m+1) \rrbracket$ Boolean hypercube code, defined in Sec. V A, satisfies this characteristic and thus is efficiently decodable.

Denote for simplicity $oip(v) := o(i(p(v)))$ and $oi(v) = o(i(v))$, where $o(\cdot)$, $i(\cdot)$, and $p(\cdot)$ are as in Definition 3. In Theorem 14, we have shown that decoding amounts to strategies in QLO. There, we have shown that Pauli operations on nodes are equivalent to a combination of switches and toggles on outputs and switches on inputs, which could cleanly reframed to fit the round 1 and round 2 structure of the QLO game framework. Here, we will be more direct, and directly apply Paulis to toggle corresponding lights. Recall from the proof of Theorem 14 that Z on $v \in \mathcal{O}$ toggles the light on v , Z on $v \in \mathcal{P}$ toggles all lights on $oi(v)$, and X on $v \in \mathcal{O} \cup \mathcal{P}$ toggles all lights in $o(v) \Delta oip(v)$. (Y errors are treated as both an X and a Z error.) In this manner we may equivalently ignore the existence of lights on pivots entirely, as they are never toggled.

In analyzing strategies for decoding, we will find it helpful to introduce a metric, which we call the *illumination*. Informally, the *illumination* of a node v is a weighted count of how many lights are on, out of all those that are toggled by flipping v ’s switch. The set of lights that can be toggled by v are either in $o(v)$ or $oip(v)$ (we ignore the lights which are in both, as flipping v ’s switch toggles them twice, effectively doing nothing). We count each light in $o(v)$ as one light, but we count each light in $oip(v)$ only as a fraction of a light. This is because flipping a pivot switch $p \in \mathcal{P}$ can toggle many lights in $oip(v)$ at once. Therefore, we will see that the illumination quantifies how many moves we can save by flipping the switch at v .

Definition 8. Let $G = (V, E, \{o, i, p\})$ be a graph with input, output, and valid pivot nodes. For any node $v \in \mathcal{O} \cup \mathcal{P}$, let $L = o(v) \Delta oip(v)$. Then the *illumination* of v ,

denoted $\Lambda(v)$, is given by

$$\Lambda(v) = |\{\text{lights on in } o(v) \cap L\}| + \sum_{\substack{p \in p(v) \\ |oi(p) \cap L| > 0}} \frac{|\{\text{lights on in } oi(p) \cap L\}|}{|oi(p) \cap L|}. \quad (15)$$

Note that each pivot p around v has an associated set of nodes $oi(p)$ that can be toggled with one operation—applying a Z to p . Thus, the number of lights on is normalized to a weight of 1, as if all of the lights in $oi(p)$ were concentrated in a light at p . We also define the *maximum illumination* analogously, as

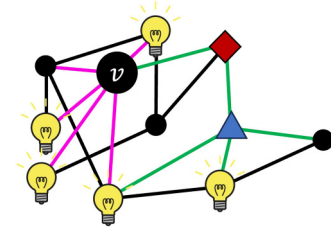
$$\Lambda_{\max}(v) := |o(v) \cap L| + |\{p \in p(v) \mid |oi(p) \cap L| > 0\}|, \quad (16)$$

which represents the maximum possible value of $\Lambda(v)$ across all light configurations.

For $v \in \mathcal{I}$, we define $\Lambda(v) = |\{w \in o(v) \mid \text{light at } w \text{ is on}\}|$ and $\Lambda_{\max}(v) = |o(v)|$. Note that both for $v \in \mathcal{O} \cup \mathcal{P}$ and for $v \in \mathcal{I}$, $\Lambda_{\max}(v) \leq \deg(v)$.

Applying X to a node $v \in \mathcal{O} \cup \mathcal{P}$ or Z to $v \in \mathcal{I}$ will decrease the illumination of v by $\Lambda(v) - (\Lambda_{\max}(v) - \Lambda(v)) = 2\Lambda(v) - \Lambda_{\max}(v)$ lights (if this value is negative, then the illumination has actually *increased*). Additionally, note that while the illumination Λ depends on a contextual “current” configuration of lights, the maximum illumination $\Lambda_{\max}$ depends only on the graph and its assignment of inputs and pivots. It is useful to compare the notion of illumination to a classical analog. Classically, when considering a Tanner graph of a code, we may label each data node by the number of check nodes it neighbors which are violated. Decoders such as the flip algorithm from Sipser and Spielman [27] use this metric to correctly deduce the error, by flipping data bits the node of which has more violated syndromes than not. The notion of illumination is quite similar in that it also captures violated syndromes around a node. In our graph representation, however, the presence of pivots complicates how we measure the amount of violation, giving rise to the more technical definition of illumination. A schematic explanation of illumination is given in Fig. 22.

The greedy decoder, given in Algorithm 1, is a strategy in the decoding instance of QLO. In particular, the operations permitted are Recover_X and Recover_Z , which apply the corresponding Paulis that effectively flip switches or destroy lights depending on the node v chosen. We have shown earlier in Theorem 14 how a QLO strategy maps to a physical recovery operation. The construction of the algorithm is based on the following intuition. If $v \in \mathcal{O}$ suffers an X error, the stabilizers at $o(v) \Delta oip(v)$ will measure -1 , so in the QLO game the lights in $o(v) \Delta oip(v)$ will turn on. Perhaps there will be some Z errors in $o(v) \subseteq$



$$\Lambda(v) = 3 + 1/2 = 3.5$$

$$\Lambda_{\max}(v) = 4 + 1 = 5$$

FIG. 22. A depiction of the illumination of a node v for a given configuration of lights. The set $L = o(v) \Delta oip(v)$ does not include the node directly below v , as it is connected by both a pink edge (denoting the paths from v to $o(v)$) and a green edge (denoting the paths from v to $oip(v)$). As there are three lights on in $o(v) \cap L$, this contributes 3 to the illumination of v . One of two lights is on in $oip(v) \cap L$, which contributes $\frac{1}{2}$ to the illumination. The maximum illumination is 5, which can be computed by turning on all the lights in the figure, or by adding $|o(v) \cap L| = 4$ and all pivot neighbors p of v such that $oi(p)$ has at least one light in L .

$\mathcal{O}$, each of which correspond to turning on a single light in $o(v)$. Therefore, assuming that it is difficult to turn on more than half of the lights in $o(v)$, a good guess is that if more than half of the lights are on in $o(v)$ then v has suffered an X error, at which point we can apply X on v to reverse the error, via Recover_X . However, applying X to v toggles both $o(v)$ and $oip(v)$. Toggling $oip(v)$ is the same as toggling $oi(p)$ for each pivot p in $p(v)$. Since this can be toggled by applying a single Z to p (effectively toggling “the light at p ,” or equivalently, $oi(p)$), we treat the status of the lights in $oi(p)$ as having the same collective importance as one light in $o(v)$, and we therefore normalize the contribution of $oi(p)$ to the gap by dividing the respective term by $|oi(p)|$ for each p . This is all summed up by the aggregate quantity $\Lambda(v)$, the illumination of v as defined in Definition 8. At each step, we apply the single move that will most decrease the illumination of any given node, starting with nodes in $\mathcal{O} \cup \mathcal{P}$ and continuing with nodes in $\mathcal{I}$. We only perform moves that decrease the illumination by at least 1, as that is how many moves it would take to decrease it by other means. We depict the action of Algorithm 1 in Fig. 23. We begin with the QLO decoding game from Fig. 18 built from the application of some Pauli error. When the algorithm terminates, we observe that the recovery operation is precisely the original Pauli error applied, indicating that the greedy algorithm decoded correctly in this example.

Applying Recover_X may itself turn on more lights in other places, so to avoid issues related to parallelism, we work in sequence and apply X recoveries one at a time, in order of what most decreases the illumination of the node, a process we will refer to as the first stage of the algorithm. A similar story holds for Z errors on $v \in \mathcal{P}$.

ALGORITHM 1. Greedy graph decoder.

```

Function GreedyDecoder( $G$ ):
  explored  $\leftarrow \emptyset$ ,  $(v_0, n_0) \leftarrow (\text{nil}, 0)$ 
  while True do // STAGE 1: X error recovery loop
    for  $v \in \mathcal{O} \cup \mathcal{P}$  do
      gap  $\leftarrow 2\Lambda(v) - \Lambda_{\max}(v)$ 
      // If the illumination of  $v$  would decrease by applying  $X(v)$ , 'gap' is positive
      if gap  $> n_0$  then
         $v_0 = v$ ,  $n_0 = \text{gap}$ 
      if  $v_0 \in \text{explored}$  or  $n_0 < 1$  then
        break
      Recover $_X(v_0)$  // toggle lights at  $o(v_0) \Delta oip(v_0)$ , destroy light at  $v_0$ 
      Append  $v_0$  to explored
  explored  $\leftarrow \emptyset$ ,  $(v_0, n_0) \leftarrow (\text{nil}, 0)$  // Reset tracking variables
  while True do // STAGE 2: Z error on pivots recovery loop
    for  $v \in \mathcal{P}$  do
      gap  $\leftarrow 2\Lambda(i(v)) - \Lambda_{\max}(i(v))$ 
      if gap  $> n_0$  then
         $v_0 = v$ ,  $n_0 = \text{gap}$ 
      if  $v_0 \in \text{explored}$  or  $n_0 < 1$  then
        break
      Recover $_Z(v_0)$  // toggle lights at  $oi(v_0)$ 
      Append  $v_0$  to explored
  for  $v \in \mathcal{O}$  do // STAGE 3: Z error on non-pivot outputs recovery loop
    if LightIsOn( $v$ ) then
      Recover $_Z(v)$  // Destroy light at  $v$ 
  return

```

They are detected by stabilizers in $oi(v)$, so we apply the analogous greedy recovery operation there in the second stage. Finally, for remaining lights on in $v \in \mathcal{O}$, we directly destroy the lights with Z operations in the third and final stage of Algorithm 1.

The key property within this intuition that seems to enable greedy-type decoding is the idea that it must be difficult to trick Algorithm 1 into believing that an X error on some v occurred by using a small amount of moves to increase the illumination of x . We now formalize this property to give a simple performance guarantee of the greedy decoder.

In this definition, we are specifically interested in how much an operation at node v can change the illumination of another node.

Definition 9. Let $G = (V, E, \{o, i, p\})$ be a graph with vertices $V = \mathcal{I} \sqcup \mathcal{P} \sqcup \mathcal{O}$, inputs $\mathcal{I}$, pivots $\mathcal{P}$, nonpivot outputs $\mathcal{O}$, edges E , and o , i , and p as in Definition 3. We say that G is B -sensitive or that the sensitivity of G is B , where $B \in \mathbb{R}$ is the maximum amount by which the illumination $\Lambda(v)$ of any node can change from the application of certain single-qubit Pauli operations, when starting with all lights off. Specifically, if $v \in \mathcal{O} \cup \mathcal{P}$, we allow any Pauli operation and if $v \in \mathcal{I}$, we only consider Pauli- Z operations. Quantitatively, denoting the change in the illumination of v due to applying the Pauli P on vertex w by

$\Delta_{P(w)}(\Lambda(v))$, we have

$$B := \max \left(\max_{\substack{v, w \in \mathcal{O} \cup \mathcal{P} \\ w \neq v \\ P \in \{X, Y, Z\}}} \Delta_{P(w)}(\Lambda(v)), \max_{\substack{v \in \mathcal{I} \\ w \in \mathcal{O} \cup \mathcal{P} \\ w \neq p(v)}} \Delta_{Z(w)}(\Lambda(v)) \right). \quad (17)$$

To give some intuition about what the notion of sensitivity represents, imagine a graph representing a social network in which nodes are people and edges connect people who are friends. If it is common for two vertices (people) to have many shared neighbors (mutual friends), then it will be possible to toggle many lights on around the first vertex by flipping the switch on the second. If mutual friends are scarce and two nodes never share more than one mutual friend, then it will be difficult to turn on many lights around a given vertex by flipping switches at other vertices. The precise description of illumination used in calculating sensitivity also includes terms such as $oip(v)$, which care about more distant vertices than merely the immediate neighbors, but counting mutual friends can serve as a good heuristic for the sensitivity of a graph before inputs and pivots have been assigned. Altogether, the sensitivity definition forms a sufficient condition to place a relatively tight guarantee on the distance of the code and the performance of the greedy decoder.

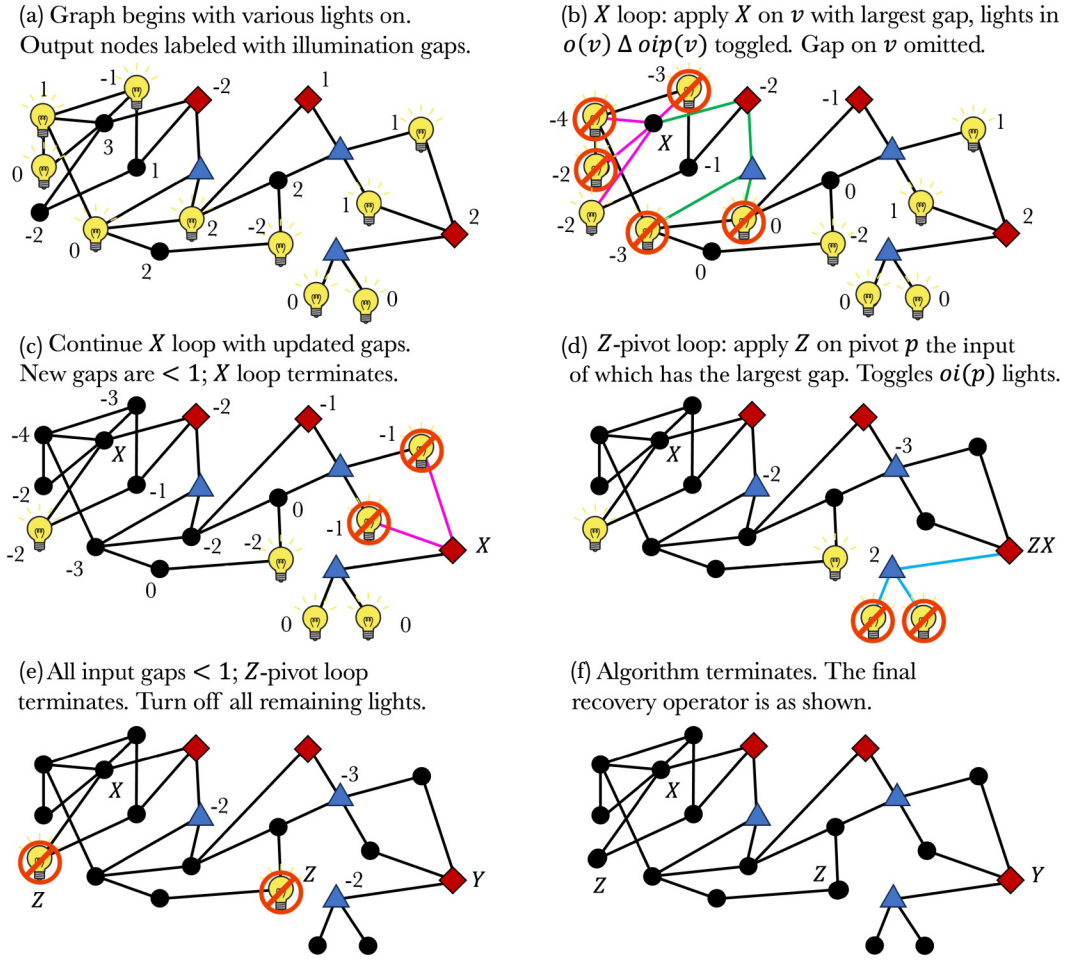


FIG. 23. A visualization of Algorithm 1 run on the final graph in Fig. 18. Inputs are blue triangles, pivots are red diamonds, and nonpivot outputs are black circles. In (a)–(c), nonpivot output nodes are labeled with their illumination gaps, given by $2\Lambda(v) - \Lambda_{\max}(v)$, where Λ and $\Lambda_{\max}$ are given in Definition 8. These are the nodes checked by the X loop. In (d) and (e), the illumination gaps of inputs are labeled; these are the nodes checked by the Z -pivot loop. The final recovery map shown is precisely the error we applied originally to the graph in Fig. 18.

Definition 10. Let $G = (V, E, \{o, i, p\})$ be a graph with vertices $V = \mathcal{I} \sqcup \mathcal{P} \sqcup \mathcal{O}$, inputs $\mathcal{I}$, pivots $\mathcal{P}$, nonpivot outputs $\mathcal{O}$, edges E , and o, i , and p as in Definition 3. We define the *brightness* $\ell \in \mathbb{N}$ of graph G as the minimum maximum illumination across all vertices in V , as defined in Definition 8. Specifically,

$$\ell := \min_{v \in V} \Lambda_{\max}(v), \quad (18)$$

where $\Lambda_{\max}$ is given in Definition 8. Note that by definition, $\ell \leq \min_{v \in V} \deg(v)$ since $\Lambda_{\max}(v) \leq \deg(v)$. By Proposition 1, $\ell \leq \text{dist}(C(G))$, where $C(G)$ is the code corresponding to G and dist is the distance.

Below, we will see that $\Lambda_{\max}(v)$ quantifies the difficulty of tricking Algorithm 1 at node v . The brightness ℓ uniformly lower bounds $\Lambda_{\max}(v)$ across all v , which will allow us to state guarantees and distance lower bounds in terms of ℓ .

Theorem 15 (Distance lower bound). Let G be a B -sensitive graph with brightness ℓ as defined in Definition 9 and Definition 10. Let $C(G)$ be the code that G represents. Then, $\text{dist}(C(G)) \geq \lceil \ell/B \rceil + 1$.

Proof. The intuition behind this proof is that a small sensitivity B makes it difficult to turn off the lights around a node in few moves, which forces the weight of any logical operator to be large. We continue to adopt the picture in which Z on $v \in \mathcal{O}$ toggles the light on v , Z on $v \in \mathcal{P}$ toggles lights in $oi(v)$, and X on $v \in \mathcal{O}$ toggles lights in $o(v) \Delta oip(v)$.

Every stabilizer contains an X on a node in $\mathcal{O}$, so any logical operator U must either include at least one X or have at least one Z on a pivot, where Y operations are treated as both X and Z . This is because any operator consisting of no X 's and no Z 's on pivots must contain only Z 's on vertices in $\mathcal{O}$, in which case it would either be the

identity operator or have an anticommuting stabilizer and thus not be a logical operator. We may split the operation of U into a sequence of single-qubit Paulis applied onto different qubits. Thus, without loss of generality, we may choose the first Pauli in this sequence to be either a Z operator on $v \in \mathcal{P}$ or an X operator on $v \in \mathcal{O} \cup \mathcal{P}$. This first operator hence increases the illumination of a vertex v from 0 to $\Lambda_{\max}(v) \geq \ell$.

At the same time, the final configuration has all lights off, and thus every node ends with illumination 0. Consequently, the remaining Paulis in the sequence must reduce the illumination at v from at least ℓ to 0. Suppose that the initial Pauli was X on $v \in \mathcal{O} \cup \mathcal{P}$. Then the illumination can only decrease by at most B with each move, as each move is equivalent to applying a Pauli recovery operation. Therefore, at least $\lceil \ell/B \rceil$ Paulis after the first are necessary to reduce the illumination at v to 0. Suppose instead that U has no X but has a Z on some pivot $p \in \mathcal{P}$. Then after applying $Z(p)$, the illumination $\Lambda(i(p))$ will be at least ℓ and will also require $\lceil \ell/B \rceil$ Pauli- Z operations to reduce to 0. Combined with the first move to increase the illumination, this results in a minimum of $\lceil \ell/B \rceil + 1$ moves, which by Theorem 12 is a lower bound for the distance. ■

Theorem 16 (Greedy guarantee). Let G be a B -sensitive graph with brightness ℓ as defined in Definition 9 and Definition 10. Let $C(G)$ be the code that G represents. Then the greedy graph decoder given by Algorithm 1 corrects at least $\lceil (\ell + 1)/2B \rceil - 1$ errors. In other words, the less sensitive a graph is, the more effective is the greedy decoder. The run-time of Algorithm 1 is $O(n\mathcal{D}_{\max}^2 + \min(n, \mathcal{D}_{\max}^4)\mathcal{W})$, where $\mathcal{D}_{\max}$ is the maximum degree of G and $\mathcal{W}$ is the weight of the error.

Proof. The decoder will fail if it ever corrects an error that did not occur, or if it ever fails to correct one that did. Suppose that the decoder has applied the correct recovery operators in both the X loop (stage 1) and the Z on pivots loop (stage 2). Then the recovery operation in stage 3 (Z on nonpivot outputs) must also be correct since at that point the only remaining possible Paulis that can be applied are Z 's on nonpivot outputs, each of which just turn the light of its own node off with no interaction with other lights. It suffices, therefore, to prove the correctness of stages 1 and 2. Additionally, if we focus on the first mistake the decoder makes, either applying an unnecessary recovery or moving on from a stage without recovering all errors. We can use the fact that at the time of the first error of the decoder, all preceding stages have been decoded correctly, and therefore we only have to be concerned about error types that we are meant to recover at the current or later stages. Specifically, when recovering Z errors on pivot nodes, we do not have to be concerned about remaining X operations to recover. This observation allows us to use our definition of

sensitivity from Definition 9, which we have carefully formulated to not consider how X operations may influence the illumination of input nodes.

Suppose that at some point the decoder applied a Z on a pivot or an X recovery operation on node w where none was needed. Then the gap variable on a corresponding node v must be at least 1, meaning that we have $1 \leq 2\Lambda(v) - \Lambda_{\max}(v) \leq 2\Lambda(v) - \ell$. Here, either $v = w$ if the error is X or $v = i(w)$ if $w \in \mathcal{P}$. This implies that $\Lambda(v) \geq (\ell + 1)/2$. But $\Lambda(v)$ can change by at most B per operation, and thus per error, so such a scenario can only occur if there are at least $\lceil (\ell + 1)/2B \rceil$ errors. Hence Algorithm 1 can still correct $\lceil (\ell + 1)/2B \rceil - 1$ errors.

Now suppose that at some point the decoder fails to recover a Z error on $w \in \mathcal{P}$ or an X error. Since everything was correctly recovered up to this point, the only way for the decoder to miss this recovery is if the error on w occurs alongside many other errors which collectively bring down the gap of some corresponding node v to < 1 . Again, $v = w$ if an X error occurred and otherwise $v = i(w)$. If so, then $1 > 2\Lambda(v) - \Lambda_{\max}(v) \geq \ell - 2(\Lambda_{\max}(v) - \Lambda(v))$, implying $\Lambda_{\max}(v) - \Lambda(v) > (\ell - 1)/2$. Thus, it would take more than $(\ell - 1)/2B$ or at least $\lfloor (\ell - 1)/2B \rfloor + 1$ additional errors to reduce the illumination enough for the decoder to make a mistake. Combined with the original error on w , this would require a total of $\lfloor (\ell - 1)/2B \rfloor + 2$ errors for a mistake, so Algorithm 1 can still correct $\lfloor (\ell - 1)/2B \rfloor + 1 \geq \lceil (\ell + 1)/2B \rceil - 1$ errors. This completes the proof of the correctness guarantee.

The run-time of the algorithm can be computed as follows. The very first act of the algorithm is to calculate gap on every node. This has the same asymptotic complexity as calculating $\Lambda(v)$, which takes $O(n\mathcal{D}_{\max}^2)$ time since $|o(v)| + |oip(v)| = O(\mathcal{D}_{\max}^2)$ for all v . Next, we iterate through the X loop. Whenever we make a recovery, we must update the gap. Any time we apply an X recovery operation, we affect $O(\mathcal{D}_{\max}^2)$ lights, each of which can be next to $O(\mathcal{D}_{\max})$ inputs, which in turn affect the illuminations of $O(\mathcal{D}_{\max})$ other nodes each. The total number of nodes for which the illumination is affected is thus $O(\mathcal{D}_{\max}^4)$, which we would need to update $\mathcal{W}$ times, where $\mathcal{W}$ is the weight of the error. However, this total number of affected nodes cannot exceed n , so we obtain that the total run-time is $O(n\mathcal{D}_{\max}^2 + \min(n, \mathcal{D}_{\max}^4)\mathcal{W})$. The X recovery loop subsumes the other two loops in complexity, so the asymptotics are the same when we account for the remaining loops. ■

The guarantees we have given are phrased in terms of brightness, assuming a sufficiently small sensitivity. Both of these definitions are technical, so it is useful to be able to lower bound these quantities generically in terms of simpler quantities, such as degree-type functions. We will first give a relatively general lower bound for the brightness, so that we can use the results of Theorem 15 and Theorem 16

to directly bound the distance and greedy guarantee from below by a simpler function of the graph. To do so, we will need to recall a standard definition from graph theory.

Definition 11 (Girth). Let G be a graph. We say that the *girth* of G is the length of the shortest cycle in G , where a cycle is a sequence of vertices that are adjacent in G such that last vertex is equal to the first. The length of the cycle does not double count the last vertex. We denote this $\text{girth}(G)$.

Recall from Definition 4 that a subset of nodes in a graph is t -spaced if every path between two nodes in the subset has length at least t .

Lemma 3 (Girth to brightness). Let $G = (\mathcal{I} \sqcup \mathcal{P} \sqcup \mathcal{O}, E)$ be a graph with inputs $\mathcal{I}$, pivots $\mathcal{P}$, outputs $\mathcal{O}$, and edge set E . Suppose that G satisfies at least one of the following conditions:

- (1) $\text{girth}(G) > 6$.
- (2) $\text{girth}(G) > 4$ and $\mathcal{P}$ is 3-spaced.
- (3) $\text{girth}(G) > 4$ and $\mathcal{I}$ is 3-spaced.

Then the brightness of ℓ of G , as given by Definition 10, satisfies

$$\begin{aligned} \ell &= \min \left(\min_{v \in \mathcal{O} \cup \mathcal{P}} |N_o(v)|, \min_{v \in \mathcal{I}} |o(v)| \right) \\ &= \min \left(\min_{v \in \mathcal{O} \cup \mathcal{P}} \deg(v) - |i(v)|, \min_{v \in \mathcal{I}} \deg(v) - 1 \right), \end{aligned} \quad (19)$$

where $N_o(v)$, $i(v)$, and $o(v)$ are as in Definition 3.

Proof. We note that the definition of brightness enforces that the quantity in the right-hand side of Eq. (19) is an upper bound for ℓ . This way, we just need to show that the brightness is no smaller than the quantity above. In the case of $v \in \mathcal{I}$, the maximum illumination gives us that $\ell \geq \min_{v \in \mathcal{I}} |o(v)|$, and since v neighbors 0 inputs and 1 pivot, $|o(v)| = \deg(v) - 1$. We now consider the case where $v \in \mathcal{O} \cup \mathcal{P}$.

One reason for why the maximum illumination $\Lambda_{\max}(v)$ could be less than $|N_o(v)|$ is if the some element $w \in o(v)$ was not in $o(v) \Delta oip(v)$, meaning that $w \in oip(v)$ as well. If $\text{girth} > 4$, this cannot occur, as the w would have to be both adjacent to v as well as connected to v with a sequence of three edges, passing through a pivot and an input, which would form a cycle of length 4.

In all cases of the claim we have $\text{girth}(G) > 4$, so by the above $o(v) \cap oip(v) = \emptyset$ and hence $o(v) \subseteq o(v) \Delta oip(v)$. The only other reason for why we might have $\Lambda_{\max}(v) < |N_o(v)|$ is if there are two pivots p_1, p_2 in $p(v)$ with nontrivial intersection: $oi(p_1) \cap oi(p_2) \neq \emptyset$. Then, since $oip(v) = \Delta_{p \in p(v)} oi(p)$, $oi(p_1) \cap oi(p_2)$ could be disjoint

from $oip(v)$. However, if the pivots are 3-spaced, $|p(v)| \leq 1$ so the above case cannot occur. Alternatively, if the inputs are 3-spaced, no node in $oi(p_1)$ can intersect with a node in $oi(p_2)$. Otherwise, this node in $\mathcal{O}$ would neighbor 2 inputs, giving a path of length 2 between inputs. Lastly, if no 3-spacing is guaranteed, then we must have a cycle of the form $v \rightarrow p_1 \rightarrow i_1 \rightarrow o \rightarrow i_2 \rightarrow p_2 \rightarrow v$, which is impossible if $\text{girth}(G) > 6$. These 3 sufficient conditions prove the lemma. ■

Corollary 1. If G is a q -regular graph such that $\text{girth}(G) > 4$ and the inputs of G are 3-spaced (as in Definition 4), then the brightness ℓ (as in Definition 10) of G satisfies $\ell = q - 1$.

Proof. By 3-spacing, $|i(v)| \leq 1 \forall v$, so the claim follows from condition (3) of Lemma 3. ■

Before we proceed to give a similarly general bound on sensitivity, we construct an explicit family of codes which has both low sensitivity and large brightness, so that the greedy decoder can correct errors almost up to the maximum weight allowed by the distance-degree upper bound of Proposition 1. Previously, we have defined in Sec. V A the hypercube code with parameters $\llbracket m2^m/(m+1), 2^m/(m+1) \rrbracket$. We label the nodes of the hypercube by their length- m binary string representation. Denote e_i the i th standard basis vector in $\mathbb{Z}_2^m$ where $1 \leq i \leq m$, i.e., $(e_i)_i = 1$ and $(e_i)_{j \neq i} = 0$. Moreover, we have shown by injection of the classical Hamming code that the input nodes of the hypercube code are 3-spaced. Before we calculate the sensitivity of the hypercube, we give a useful lemma.

Lemma 4 (Hypercube 3-spacing). It is possible to choose pivots on the hypercube code, as defined in Definition 7, such that the pivots of the code are 3-spaced, as defined in Definition 4.

Proof. The minimum path length between inputs is already 3. By homogeneity of the hypercube, it is enough to choose the pivots in a uniformly consistent way for every input. One example which works is to set $p(v) = \{v + e_1\}$ for all $v \in \mathcal{I}$. This assignment will never choose $p(v) \in \mathcal{I}$ since the inputs are separated by paths of length at least 3. Now suppose that there exists a path between two pivots p_1 and p_2 with length less than 3. In other words, with at most two bit flips, we can go from p_1 to p_2 , so $p_2 = p_1 + e_a + e_b$ for some $a, b \in [m]$. Since $p_1 = i_1 + e_1$ and $p_2 = i_2 + e_1$, where i_j is the input corresponding to p_j , $i_2 = i_1 + e_a + e_b$, which contradicts the fact that inputs are separated by paths of length at least 3. ■

We note that there are several other transformations that can be applied to the hypercube to choose a set of

pivots. More generally, to assign a set of pivots on the hypercube, it suffices to choose any map from the set of vertices of the hypercube to itself, such that the graph distance between two nodes is preserved and every vertex is mapped to an adjacent one. In the proof of Lemma 4, we have used a reflection about the first axis by adding e_1 , although we could have just as easily chosen any e_i . Another acceptable map would have been to choose any two axes, and rotate the hypercube 90° (or 270°) in the plane formed by those axes. This would leave most of the components of each vector intact, and modify the two bits corresponding to the chosen axes forward (or backward) in a cycle $00 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 00$. This would ensure that every vertex moves to a neighboring spot, and as rotations are isometries, the distance condition would also be satisfied.

Theorem 17 (Hypercube sensitivity). The m -dimensional hypercube code, as defined in Definition 7, has a sensitivity of $B \leq 2 + 2/(m - 2)$.

Proof. The nodes of the hypercube are labeled by their length- m bit string. The sensitivity is a maximum over a certain quantity over all nodes. By symmetry of the hypercube, the sensitivity is exactly this certain quantity for any fixed node, and without loss we choose the node to be $v = 0^m$. The general idea is that when we take neighbor-type operations $p(\cdot), o(\cdot), i(\cdot)$ on a bit string with weight M , the resultant weight of the bit string can only change by 1 due to the connectivity of the hypercube.

We need to show that no operation allowed by Definition 9 on a vertex w can change the illumination of v by more than $2 + 2/(m - 2)$. We also recall Lemma 4, so our inputs and pivots will both be 3-spaced. First note that if $v \in \mathcal{I}$, no Z operation can toggle more than one light in $o(v)$ since the inputs are 3-spaced.

Suppose that we apply X on some $w \in \mathcal{O} \cup \mathcal{P}$, which toggles lights in $o(w)$ and $oip(w)$. In order for this operation to affect the illumination of v , $o(w) \Delta oip(w)$ must intersect $o(v)$ or $\bigcup_{p \in p(v)} oi(p)$. These are all an odd distance from v ; likewise $o(w)$ and $oip(w)$ are an odd distance from w . But the hypercube is bipartite, so if the X on w is to affect the illumination of v , the bit strings corresponding to v and w must have the same parity. Thus, we only need to consider w for which the bit string has even weight. We have that $v = 0^m$, so if the bit string of w has too large a weight, X on w cannot affect $o(v)$ or $oip(v)$ at all. If w has weight greater than 6, it cannot affect the illumination of v , as $oip(v)$ and $oip(w)$ cannot intersect. If w has weight exactly 6, then X on w affecting the illumination of v would require a distance of 2 between a weight 2 input $ip(v)$ and a weight 4 input $ip(w)$, which is impossible since the inputs are 3-spaced.

Suppose that w has weight 2. Without loss of generality, the vertex w has bit string $110 \dots 0$ and $o(w)$ intersects

$o(v)$ at $10 \dots 0$ and $010 \dots 0$. However, for an operation at w to affect v any further requires either $oip(w)$ intersecting $o(v)$ or $oip(v)$ intersecting $o(w)$. In either case, we need to draw a chain of four steps from v to w , which cannot go through $10 \dots 0$ and $010 \dots 0$. Without the trailing zeros, this chain will look like $000 \rightarrow 001 \rightarrow (101 \text{ or } 011) \rightarrow 111 \rightarrow 110$ without loss of generality. However, the central node in the chain would then have to be the input $ip(w)$ or $ip(v)$, in which case either 100 or 010 would be a neighbor of this input. Let us call this weight-1 node u . We note that u is a neighbor of all three of v, w , and the input above, which is either $ip(w)$ or $ip(v)$. The node u would therefore *not* be in at least one of the symmetric differences $o(v) \Delta oip(v)$ or $o(w) \Delta oip(w)$, meaning that either the light at u was not turned off by w or it was not one of the lights illuminating v to begin with.

Meanwhile, if w has weight 4, then we only need to consider the interactions of $o(v)$ and $oip(w)$ as well as $o(w)$ and $oip(v)$. This case is illustrated in Fig. 24. Any two nodes in the hypercube share at most two neighbors. Applying this fact to v and $ip(w)$ as well as w and $ip(v)$ shows that $oip(w)$ decreases the illumination of v by at most 2, while $o(w)$ turns off at most 2 lights in $oip(v)$. Since $ip(v)$ and v share the neighbor $p(v)$, they will also share one other weight-1 neighbor, which will therefore not be in the set $o(v) \Delta oip(v)$. Hence $|oip(v) \cap (o(v) \Delta oip(v))| = m - 2$, by subtracting out from the m neighbors the above node as well as one additional node which is a pivot. Thus, any operation on w toggles at most two lights in $o(v)$ and at most two lights out of $m - 2$ relevant ones in $oip(v)$, changing the illumination of v by up to $2 + 2/(m - 2)$. ■

The greedy guarantee of Theorem 16 states that a graph with low sensitivity is greedy-correctable up to about ℓ errors, where ℓ is the brightness from Definition 10. To complete our arguments of the hypercube, we show that its brightness is close to the degree. Since degree upper bounds the distance by Proposition 1 and the brightness approximately lower bounds the distance (up to the sensitivity factor), this will imply that the hypercube distance is at least a constant factor away from its upper bound, and that the greedy decoder succeeds up to this factor. Since the hypercube has cycles of length 4, Lemma 3 does not apply. Instead, we prove the brightness lower bound directly by studying the structure of the hypercube.

Proposition 3 (Hypercube brightness). Let ℓ be the brightness from Definition 10. Then the m -dimensional hypercube code from Definition 7 has $\ell = m - 2$.

Proof. Every input is adjacent to m vertices of which one is a pivot and none are inputs. Thus, the maximum illumination of any input vertex is $m - 1$. As the pivots of

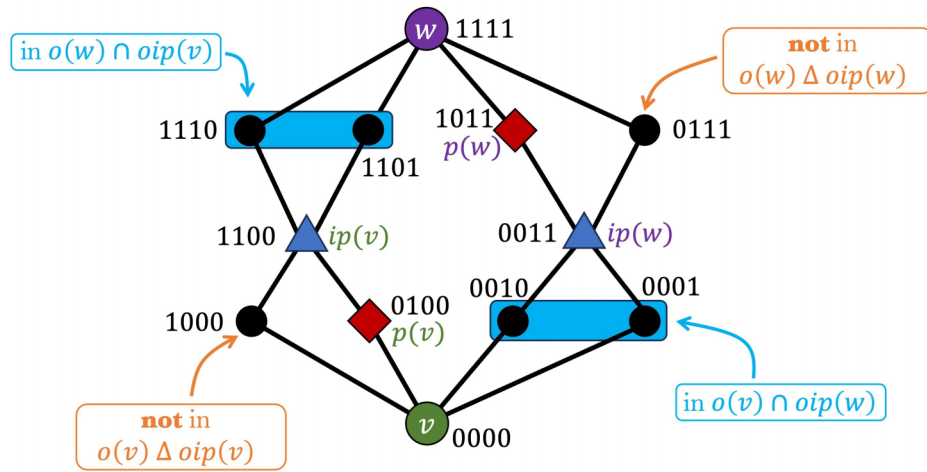


FIG. 24. A depiction of how a switch flipped at w can affect the illumination of v for the m -dimensional hypercube code. Nodes are labeled with the first four bits of a m -bit bit string identifying their vertex in the hypercube graph. Pivots are shown as red diamonds and inputs as blue triangles. Flipping the switch at w toggles the lights in $o(w) \Delta oip(w)$. This does not include the top right vertex, 0111, due to the symmetric difference. This can toggle lights in $o(v)$ in the bottom right, affecting the illumination of v by 2, as well as in $oip(v)$ (top left), affecting the illumination of v by up to $\frac{2}{m-2}$, since the input node shown at 1100 has m neighbors, of which one is a pivot and another, shown as 1000 in the bottom left, might not be in $o(v) \Delta oip(v)$.

the hypercube code are 3-spaced by Lemma 4, we know that for any $v \in \mathcal{O} \cup \mathcal{P}$, $p(v)$ contains at most one vertex. Furthermore, $ip(v)$ will share exactly two neighbors with v , one of which is $p(v)$ and the other one of which is in $o(v)$. Thus, v neighbors m vertices of which one might be an input and one might be removed by the symmetric difference of $o(v)$ and $oip(v)$, so $\Lambda_{\max}(v) \geq m - 2$. Since ℓ is the smallest maximum-illumination, this proves the claim. ■

Corollary 2. Let $d(m)$ be the distance of the m -dimensional hypercube code where $m \geq 3$. Then $\left\lfloor \frac{m-2}{2+\frac{m-2}{2}} \right\rfloor + 1 = \left\lfloor \frac{m-1}{2} \right\rfloor \leq d(m) \leq m$. Notably, $d(m) \in \Theta(m)$. Moreover, Algorithm 1 can correct all Pauli errors of weight $\left\lfloor \frac{m-1}{4+\frac{m-2}{2}} \right\rfloor - 1 = \left\lfloor \frac{m-1}{4} \right\rfloor - 1$ or less.

Proof. This is a direct consequence of Proposition 3, Theorem 15, and Theorem 16. ■

This example shows that we can use the graph formalism to construct graphs that admit good algorithms for decoding.

Conjecture 1. For the m -dimensional hypercube code, as defined in Definition 7, $d(m) = m$ (where $d(m)$ is the distance of the m -dimensional hypercube code) when $m \geq 7$ and a greedy decoding algorithm can decode up to $\lfloor (m-1)/2 \rfloor$ errors.

In the case of the $[[112, 16]]$ ($m = 7$) code as well as the $[[240, 16, 8]]$ ($m = 8$) code, we have verified numerically

that the distance portion of our conjecture holds [69]. For $m < 7$, we have verified that the conjecture does not hold and we believe this to be due to low-dimensional effects.

We next proceed to upper bound the sensitivity. Recall that the girth of a graph G is the length of the shortest cycle in G . Moreover, recall that a subset of nodes is 3-spaced if every path between these nodes has length at least 3 (see Definition 4). Previously, we have shown that girth and 3-spacing are sufficient conditions to lower bound the brightness. As we now show, the same properties are sufficient to upper bound the sensitivity. Specifically, a larger girth implies that the graph has a lower sensitivity, with there being options to relax the girth requirement at the cost of introducing requirements on the spacing of inputs or pivots. In conjunction with the previous results on brightness upper bounds, we obtain a tight guarantee for efficient greedy decodability up to nearly the information-theoretic limit (given by the distance) for all graphs of sufficient girth and spacing.

Lemma 5 (Large girth implies sensitivity). Let G be a graph with inputs $\mathcal{I}$, pivots $\mathcal{P}$, and sensitivity B . The following all hold:

- (1) If $\text{girth}(G) > 12$, $B = 1$.
- (2) If $\text{girth}(G) > 10$ and G is q -regular, $B \leq 1 + 1/(q-1)$.
- (3) If $\text{girth}(G) > 8$ and $\mathcal{I}$ is 3-spaced, $B = 1$.
- (4) If $\text{girth}(G) > 8$ and $\mathcal{P}$ is 3-spaced, $B = 1$.
- (5) If $\text{girth}(G) > 8$ and G is q -regular, $B \leq 2$.
- (6) If $\text{girth}(G) > 6$ and $\mathcal{P}$ is 3-spaced, $B \leq 2$.

- (7) If $\text{girth}(G) > 6$, G is q -regular, and $\mathcal{P}$ is 3-spaced, $B \leq 1 + 1/(q-1)$.
- (8) If $\text{girth}(G) > 4$ and $\mathcal{P}$ is 3-spaced, $B \leq 3$.
- (9) If $\text{girth}(G) > 4$, G is q -regular, and $\mathcal{P}$ is 3-spaced, $B \leq 2 + 2/(q-1)$.

Here, girth is as in Definition 11, sensitivity as in Definition 9, and 3-spacing as in Definition 4.

Proof. In all cases, the girth of the graph is greater than 4. Therefore if $v \in \mathcal{I}$ is an input, then for any pivot $p \in \mathcal{P}$, $oi(p)$ can intersect with $o(v)$ on at most 1 vertex, as otherwise there would be a cycle of length 4. Consequently for all such cases, the illumination of any *input* is not changed by more than 1. It is enough therefore to consider the case of $v \in \mathcal{O} \cup \mathcal{P}$. We know that for w to affect the illumination of v by more than 1, the operation on w must affect one or both of $o(v)$ and $oip(v)$, and this operation can only do so while toggling the lights in $o(w)$ and $oip(w)$, which gives us four total cases. These cases are illustrated [1]–[4] in Fig. 25. We refer to each of these cases as *types* of connections. Each type of connection independently contributes to the sensitivity. Note that if a Y operation is used instead of an X operation, this also toggles the light at w . For the purposes of girth, it is always stronger for w to connect to the light in $o(v)$ or $oip(v)$ that it wants to toggle rather than being this light itself, as that is what lengthens the relevant cycles the most. Doing both (being a light in one of these sets and being connected to a light in one of these sets) is never an option as that would imply the existence of a cycle of length 3, which is impossible if the girth of G exceeds 4.

The four cases described above are illustrated in Fig. 25, which we will refer to repeatedly in this proof, particularly the connections numbered [1]–[4] between v and w , as well as edges [5] and [6]. This figure illustrates all the possible families of connections that are relevant to the conditions in the theorem, as follows:

- (1) When $\text{girth}(G) > 12$, at most one connection from v to w is possible. If there were two such connections of type [3], we would obtain a cycle of length 12 since each connection has length 6. Thus $B = 1$.
- (2) If $\text{girth}(G) > 10$, then we might have many length-6 connections, such as copies of type [3]. These connections come from the intersection of $oip(v)$ and $oip(w)$ which means that paths start with v to a pivot and end with a pivot to w . If G is q -regular, each neighbor set $oi(p)$ for some pivot neighbor $p \in \mathcal{P}$ has size $q-1$, and at most q of these length-6 connections will be made, so $B \leq q/(q-1) = 1 + 1/q - 1$.
- (3) If $\text{girth}(G) > 8$, then we might have a length-4 connection such as [2], a $oip(v)$ and $o(w)$ intersection, or [4], a $o(v)$ and $oip(w)$ intersection. There could

also be some length-6 connections as in [3] from the previous case. If $\mathcal{I}$ is 3-spaced, however, [3] cannot exist, because [3] requires two inputs to have a distance of length 2. Moreover, if there were more than one connection of type either [2] or [4], there would be a cycle of length 8 (see cycle made of going through [2] and back through [4]). Hence, only one connection of type [2] or [4] is possible, so $B = 1$.

- (4) As in case (3), the only possible connections are of type [2], [4], or [3]. Since the girth is > 8 , again only one connection of type either [2] or [4] is possible. If there were more than one connection of type [3], there would exist a pair of pivots with distance 2 from each other. Thus there is at most one connection of type [3] as well, and $B = 1$.
- (5) As in cases (3) and (4), the only connections allowed are a single one of type [2] or [4], and then potentially many connections of type [3]. If G is q -regular, then as in case (2), each connection of type [3] can contribute at most $1/(q-1)$ to the illumination of v . Since connections of type [2] and [4] can modify the illumination by 1, we have in total $B \leq 1 + (q-1)1/(q-1) = 2$.
- (6) If $\text{girth}(G) > 6$, then we might have either connection [1] and copies of [3] (possibly with shared or overlapping central vertices between the copies), or we might have copies of [2], [3], and [4]. If $\mathcal{P}$ is 3-spaced, however, we are restricted to a single connection each of type [2], [3], and [4]. Moreover, a connection to v of type either [2] or [4] cannot coexist with a connection to v of type [3] by pivot 3-spacing, and likewise with w instead of v . Thus, we have at most either a single connection each of types [2] and [4], or a single connection each of types [1] and [3]. In either case, these two connections guarantee $B \leq 2$.
- (7) This case is identical to case (6), with the addition of q -regularity as a condition. q -regularity implies that the connections that begin with a pivot, namely [3] and [4], can affect the illumination of v by at most $1/(q-1)$, so $B \leq 1 + 1/(q-1)$.
- (8) If $\text{girth}(G) > 4$, we may have at most one connection of type [1], as two would give a cycle of length 4. The other connections *a priori* may be plentiful. However, since $\mathcal{P}$ is 3-spaced, there is at most one connection to v each of type [2], [3], and [4]. Further, a connection to v of type [3] cannot coexist with either a connection to v of type [2] or [4]. The same holds for v replaced with w . By enumerating the possible cases, we find that the structure that maximizes connectivity arises from the existence of connections [1], [2] on the v side only, [3], [4] on the w side only, and the further addition of “staggered edges” given by edges [5] and [6]. This

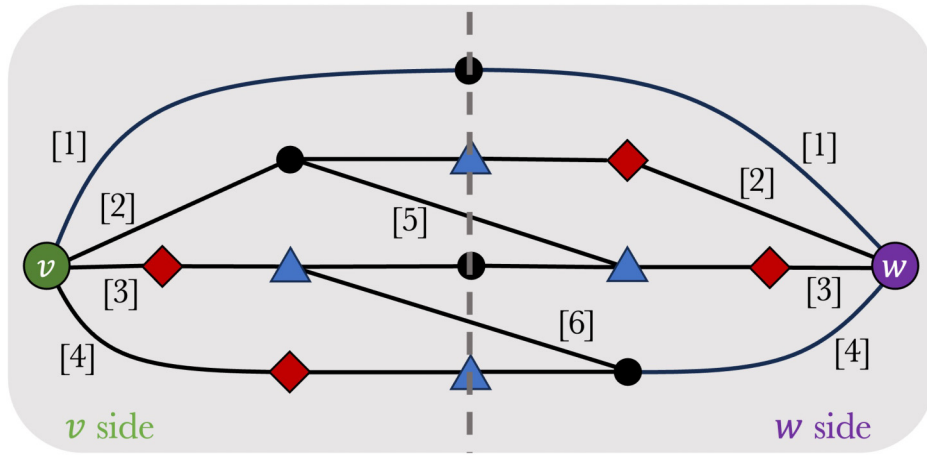


FIG. 25. A visualization of the various paths by which flipping a switch at w can affect the illumination of v . Blue triangles are inputs and red diamonds are pivots. The connecting arcs [1] to [4] correspond to intersections of the pairs $o(v)$ and $o(w)$, $o(v)$ and $oip(w)$, $oip(v)$ and $oip(w)$, and $oip(v)$ and $o(w)$, respectively. These are paths through which w can affect the illumination of v and bounds on the girth and spacing of inputs and pivots can affect which and how many of these connections can be present between v and w . The edges labeled [5] and [6] are used only when necessary to highlight particular possible constructions.

connectivity gives two lights in $o(v)$ being toggled and two lights in $oip(v)$ when we operate on w . As the latter two contribute to the illumination as part of the same pivot node, we have $B \leq 2 + 1 = 3$.

- (9) This is identical to case (8), but now adding the condition that G is q -regular means that the two lights toggled in $oip(v)$ contribute only $2/(q-1)$ toward the illumination since they go through the same pivot. Thus, $B \leq 2 + 2/(q-1)$.

■

While it may at first appear surprising that the entire notion of sensitivity can be simply bounded by the girth and spacing constraints which are both small constants, we note for intuition that the formulation of sensitivity from Definition 9 is highly local. That is, when an error affects a certain qubit, the stabilizers which can detect that error are no more than a *small, constant* number of edges away from the affected qubit. Thus, if two nodes are separated by a sufficiently large constant minimum path length, then errors on one have no effect on the other. This condition of separation by a long path length is closely related to the girth, since if a node v is in a short cycle, then the path from v to the furthest-away node w contained in the cycle is about half the length of the cycle.

Theorem 18 (Master theorem of graph girth). Let G be a graph with inputs $\mathcal{I}$, pivots $\mathcal{P}$, and nonpivot outputs $\mathcal{O}$. Let d be the distance of the code corresponding to G and let T be the number of errors correctable by the decoder in given in Algorithm 1. If one of the following conditions

holds, then the brightness of G is

$$\ell^* = \min \left(\min_{v \in \mathcal{O} \cup \mathcal{P}} \deg(v) - |i(v)|, \min_{v \in \mathcal{I}} \deg(v) - 1 \right) \quad (20)$$

and the bounds d and T are as stated. Note that if G is q -regular, $\ell \leq q - 1$:

- (1) If $\text{girth}(G) > 12$, $d \geq \ell^* + 1$ and $T \geq \lceil (\ell^* - 1)/2 \rceil$.
- (2) If $\text{girth}(G) > 10$ and G is q -regular, $d \geq \ell^* + 1$ and $T \geq \lceil ((\ell^* + 1)(q - 1))/2q \rceil - 1 \geq \lfloor (\ell^* - 1)/2 \rfloor$.
- (3) If $\text{girth}(G) > 8$ and $\mathcal{I}$ is 3-spaced, $d \geq \ell^* + 1$ and $T \geq \lceil (\ell^* - 1)/2 \rceil$.
- (4) If $\text{girth}(G) > 8$ and $\mathcal{P}$ is 3-spaced, $d \geq \ell^* + 1$ and $T \geq \lceil (\ell^* - 1)/2 \rceil$.
- (5) If $\text{girth}(G) > 8$ and G is q -regular, $d \geq \lceil \ell^*/2 \rceil + 1$ and $T \geq \lceil (\ell^* + 1)/4 \rceil - 1$.
- (6) If $\text{girth}(G) > 8$, G is q -regular, and $\mathcal{I}$ is 3-spaced, $d = q$ and $T = \lfloor (q - 1)/2 \rfloor$.
- (7) If $\text{girth}(G) > 6$ and $\mathcal{P}$ is 3-spaced, $d \geq \lceil \ell^*/2 \rceil + 1$ and $T \geq \lceil (\ell^* + 1)/4 \rceil - 1$.
- (8) If $\text{girth}(G) > 6$, G is q -regular, and $\mathcal{P}$ is 3-spaced, $d \geq \ell^* + 1$ and $T \geq \lceil ((\ell^* + 1)(q - 1))/2q \rceil - 1 \geq \lfloor (\ell^* - 1)/2 \rfloor$.
- (9) If $\text{girth}(G) > 6$, G is q -regular, $\mathcal{P}$ is 3-spaced, and $\mathcal{I}$ is 3-spaced, $d = q$ and $T \geq \lfloor q/2 \rfloor - 1$.
- (10) If $\text{girth}(G) > 4$ and $\mathcal{P}$ is 3-spaced, $d \geq \lceil \ell^*/3 \rceil + 1$ and $T \geq \lceil (\ell^* + 1)/6 \rceil - 1$.
- (11) If $\text{girth}(G) > 4$, G is q -regular, and $\mathcal{P}$ is 3-spaced, $d \geq \lceil \ell^*/2 \rceil + 1$ and $T \geq \lceil ((\ell^* + 1)(q - 1))/4q \rceil - 1 \geq \lfloor (\ell^* - 1)/4 \rfloor$.
- (12) If $\text{girth}(G) > 4$, G is q -regular, $\mathcal{P}$ is 3-spaced, and $\mathcal{I}$ is 3-spaced, $d \geq \lfloor q/2 \rfloor + 1$ and $T \geq \lceil (q - 1)/4 \rceil - 1$.

Proof. The claim is a direct consequence of Lemma 5, Lemma 3, Corollary 1, Theorem 15, and Theorem 16. ■

As long as any of the various conditions in Theorem 18 are satisfied, we obtain a code that has distance $\Omega(\ell^*)$, which is a simple function of the degree as we had originally desired. As long as no node has more than a constant fraction of neighbors as inputs, $\ell^* = \Omega(\mathcal{D}_{\min})$ (where $\mathcal{D}_{\min} = \min_{v \in V} \deg(v)$) and so we can correct $\Omega(\mathcal{D}_{\min})$ errors, matching the upper bound from Proposition 1 up to constants.

We remark that while Theorem 18 provides several sufficient conditions for which the distance has a lower bound near the upper bound and the greedy decoder decodes a number of errors also near the upper bound, it is not necessary. If one wishes to construct graphs with girth ≤ 4 , the corresponding code is not necessarily immediately low distance or not greedily decodable. Rather, the graph must instead be carefully designed to still have high brightness and low sensitivity. The most direct example of this phenomenon is the hypercube code. The hypercube is full of length 4 cycles, e.g., $00 \dots \rightarrow 01 \dots \rightarrow 11 \dots \rightarrow 10 \dots \rightarrow 00 \dots$, and yet by carefully studying its symmetries, we were able to show in Theorem 3 and Theorem 17 that the family of hypercube codes indeed has near-maximal distance and greedy decodability.

For the remainder of this work, however, we will use girth to our advantage. Theorem 18 suggests a completely graph-theoretic recipe for finding good codes that are optimally decodable by the greedy strategy. Specifically, we seek families of graphs which satisfy the following:

- (1) Any of the conditions in Theorem 18 hold for G ; e.g., having $\text{girth}(G) > 10$ and being m -regular.
- (2) The minimum degree of G is as large as possible, ideally at least $\omega(\log n)$ to beat the distance of the hypercube code, where ω is little-omega notation. That is, a function $f(n)$ is $\omega(g(n))$ if $\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$.

These two simple conditions invite the application of extremal graph theory for code construction. To that end, we consider the following seminal result of extremal graph theory.

Theorem 19 (Benson [32]). There exists an explicit family of bipartite graphs G_q , where q is any power of an odd prime, which satisfies

- (a) G_q is $(q+1)$ -regular,
- (b) G_q has $2 \sum_{k=0}^5 q^k$ nodes, and
- (c) G_q has girth at least 12.

We refer to such graphs as Benson graphs. The only remaining obstacle between a Benson graph and a greedy-optimal graph code is the choice of input and pivots. Although the maximum packing of inputs is generically hard, we show that it is still possible to do so up to a good approximation ratio. We give two techniques for such an input packing, both of which are completely generic and have no dependence on the Benson graph. They thus apply equally well toward transforming any other graph into a code. The first approach is very simple but only gives a guarantee $k = \Omega(n/\mathcal{D}_{\max}^2)$ in the worst case, where $\mathcal{D}_{\max} = \max_{v \in V} \deg(v)$. In Theorem 19, $q = \mathcal{D}_{\max}$. The second approach is more sophisticated and achieves the guarantee $k = \Omega(n/\mathcal{D}_{\max})$. Recall that for a node $v \in V$, where V is the vertex set of a graph G , $N(v) = \{w \in V : (v, w) \in E\}$ for E the edge set of G .

For concreteness, we depict an example run of Algorithm 2 in Fig. 26.

Claim 2. Let G be a graph with m nodes. Algorithm 2 gives a valid assignment of k inputs, k pivots, and $n - k$ nonpivot outputs such that $n + k = m$ and $k = \Omega\left(\frac{n}{\mathcal{D}_{\max}^2}\right)$. By valid, we mean that every pivot is connected to exactly one input. Moreover, the set of inputs $\mathcal{I}$ and the set of pivots $\mathcal{P}$ are both 3-spaced, as in Definition 4.

Proof. At each step of the loop, all nodes within a distance ≤ 2 of the newly assigned input and pivot are marked, and therefore cannot be later made inputs or pivots. Hence, all inputs and pivots are separated by paths of length at least 3. In addition, each input has a uniquely connected pivot, giving a valid assignment. Moreover, at each step at most $2\mathcal{D}_{\max}^2$ nodes are marked, so the loop runs $\Omega(m/\mathcal{D}_{\max}^2)$ times. ■

Informally, Algorithm 3 takes advantage of the observation that Algorithm 2 is maximally conservative at packing inputs. It is not strictly necessary for all inputs and pivots to be separated by distance at least 3, so long as we are careful. Thus, Algorithm 3 tries to pack input nodes as closely as possible without packing them “too closely.” Specifically, if we desire that the brightness of the graph remains within a constant fraction of the minimum degree of the graph, then we can have at most a constant fraction of the nodes around v be inputs or pivots. More generally, we define a parameter $\gamma > 1$ which quantifies how closely we are willing to pack our inputs. To be more precise, we construct our algorithm so that no output node can have more than $\mathcal{D}_{\min}/\gamma$ neighboring inputs or pivots. A convenient technique to enforce this constraint is to give each node an *availability*, initialized to $\mathcal{D}_{\min}/\gamma$, which indicates how many more inputs or pivots can be placed around that node. When the availability of a node v drops to zero, all nodes in $N(v)$ become ineligible to be chosen as the next input or pivot. Algorithm 2 is the special case of this new approach

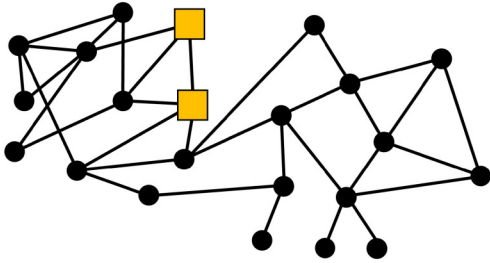
ALGORITHM 2. Simple input-pivot assigner.

```

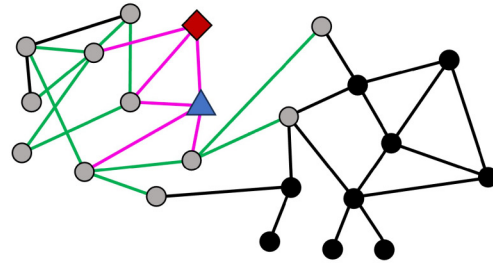
Function SimpleInputPivot( $G$ ):
  for  $v \in V$  do
     $\text{marked}_v \leftarrow \text{False}$ 
  while  $\exists v, w \in V : (v, w) \in E$  and  $\text{marked}_v = \text{marked}_w = \text{False}$  do
     $\text{marked}_v \leftarrow \text{True}$ 
     $\text{marked}_w \leftarrow \text{True}$ 
    set  $v$  as input
    set  $w$  as pivot( $v$ )
    for  $u \in N(N(v) \cup N(w))$  do
       $\text{marked}_u \leftarrow \text{True}$ 
  return

```

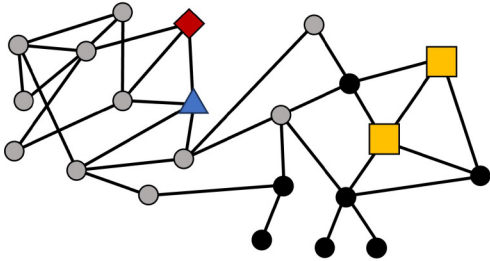
(a) Pick a connected free (black) pair (highlighted as orange squares).



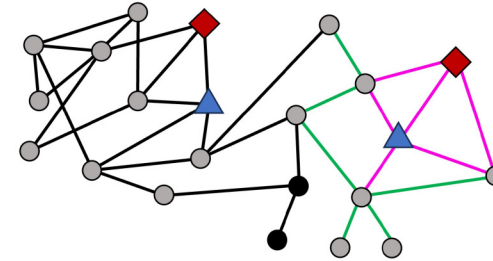
(b) Assign one as input (triangle) and other as pivot (diamond). “Gray out” neighbours (connected via pink) and next-neighbours (green).



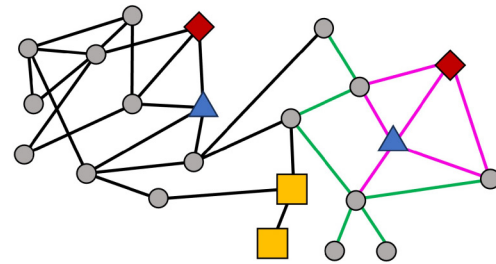
(c) Repeat step (a).



(d) Repeat step (b), and so on until there are no more connected free pairs.



(e) Pick a connected free pair.



(f) No free pairs remaining. Algorithm ends.

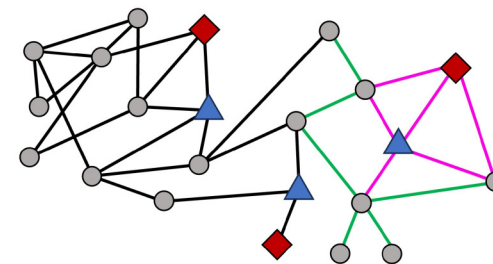


FIG. 26. An example run of the simple input-pivot assignment process given by Algorithm 2. Initially, all nodes are free, depicted by black circles. We begin in (a) by picking a connected free pair (orange squares). One is assigned as input (blue triangle) and the other as pivot (red diamond). Both the neighbors of the pair and the neighbors of those neighbors are grayed out in (b), making them no longer free. We repeat this process in (c)–(e) until there are no free pairs. The algorithm then terminates with the final assignment as in (f).

when $\gamma = \mathcal{D}_{\min}$. As γ decreases the rate increases while the distance decreases, but not necessarily at the same rate. We can therefore choose γ carefully to improve the rate significantly without degrading the distance by too much. In fact, we show that, up to constant factors, the following algorithm improves strictly upon Algorithm 2 in terms of distance and rate. The cost, however, is that the dense packing of inputs causes our bound on the check weight to go up from a linear to a quadratic bound, as discussed in Proposition 2.

We depict also an example run of Algorithm 3 in Fig. 27.

Claim 3. Let G be a graph with m nodes and minimum (maximum) degree $\mathcal{D}_{\min}$ ($\mathcal{D}_{\max}$). Let $\gamma > 2$ be a constant and suppose $\mathcal{D}_{\min} \geq \gamma$. Then Algorithm 3 gives a valid assignment of k inputs, k pivots, and $n - k$ nonpivot outputs such that $n + k = m$ and $k = \Omega\left(\frac{n\mathcal{D}_{\min}^2}{(\gamma+1)\mathcal{D}_{\max}^3}\right)$. By valid, we mean that every pivot is connected to exactly one input.

Proof. First, by construction each input has a unique pivot neighbor. Moreover, since all neighbors of the newly assigned input-pivot pair are marked at each step, no two inputs or pivots can be neighbors. The assignment is thus valid.

Next, at each step, v_1, v_2 , and $N(v_1) \cup N(v_2) \setminus \{v_1, v_2\}$ are marked explicitly (ignoring for the moment the markings due to the change of availability). Thus, each step marks $\leq 2(\mathcal{D}_{\max} - 1) + 2 = 2\mathcal{D}_{\max}$ nodes explicitly. We analyze the effect of the markings due to a change in availability by appealing to the pigeonhole principle. When the availability of some node drops to zero, $\leq \mathcal{D}_{\max}$ nodes are marked. However, in each step, the total availability summed across all nodes drops by $\leq 2(\mathcal{D}_{\max} - 1)$. Therefore, after s steps, the total availability across all nodes has changed by $\leq 2s\mathcal{D}_{\max}$, so at most $2s\mathcal{D}_{\max}/\mathcal{D}_{\min}/\gamma$ nodes have reached availability 0. Hence, at most $2s\mathcal{D}_{\max}/\mathcal{D}_{\min}/\gamma \cdot \mathcal{D}_{\max}$ nodes are marked at after s steps due to a change in availability. The total number of marked nodes after s steps is therefore at most $2s\mathcal{D}_{\max} + 2\gamma s\mathcal{D}_{\max}^2/\mathcal{D}_{\min} \leq 2(\gamma + 1)s\mathcal{D}_{\max}^2/\mathcal{D}_{\min}$. Consequently, it takes at least $m\mathcal{D}_{\min}/2(\gamma + 1)\mathcal{D}_{\max}^2$ steps before all nodes are marked and therefore no more input-pivot pairs can be added.

To complete the proof, we show that for $\Omega\left(\frac{m\mathcal{D}_{\min}^2}{(\gamma+1)\mathcal{D}_{\max}^3}\right)$ steps, there exists a node which is unmarked and has an unmarked neighbor, so that the loop runs for $\Omega\left(\frac{m\mathcal{D}_{\min}^2}{(\gamma+1)\mathcal{D}_{\max}^3}\right)$ steps and thus assigns $k = \Omega\left(\frac{m\mathcal{D}_{\min}^2}{(\gamma+1)\mathcal{D}_{\max}^3}\right)$ inputs. The condition of the existence of an unmarked node with an unmarked neighbor is equivalent to the existence of an edge in the subgraph

induced by the unmarked nodes. Each time a node is marked, it removes at most $\mathcal{D}_{\max}$ edges from the unmarked subgraph. After s steps, at most $2(\gamma + 1)s\mathcal{D}_{\max}^2/\mathcal{D}_{\min}$ nodes are marked, which in turn remove at most $2(\gamma + 1)s\mathcal{D}_{\max}^3/\mathcal{D}_{\min}$ edges. There are, to begin with, at least $m\mathcal{D}_{\min}/2$ edges. Therefore, for $s \leq m\mathcal{D}_{\min}^2/4(\gamma + 1)\mathcal{D}_{\max}^3$ steps, the unmarked subgraph has at least one edge. ■

Although we have, for the sake of generality, included γ in the asymptotic scaling, we will typically choose γ to be a small constant greater than 1, such as 2. This choice is sufficient for all constructive purposes which might involve Algorithm 3. Combining Claim 2, Claim 3, Theorem 15, Theorem 16, Lemma 3, and Corollary 1, we obtain the following general results.

Theorem 20 (Sensitive girth-5 graphs as codes). If G is a B -sensitive graph with $m = n + k$ nodes, $\text{girth}(G) > 4$, maximum degree $\mathcal{D}_{\max}$, and minimum degree $\mathcal{D}_{\min}$, Algorithm 2 describes an input packing which yields a $[[n, k, d]]$ code, where

$$n = \Theta(m), \quad k = \Omega\left(\frac{m}{\mathcal{D}_{\max}^2}\right), \quad d = \Omega\left(\frac{\mathcal{D}_{\min}}{B}\right). \quad (21)$$

The greedy decoder can (efficiently) correct $\lceil \mathcal{D}_{\min}/2B \rceil - 1$ errors.

Proof. Since each node has at most one input and one pivot around it and since G has $\text{girth}(G) > 4$, this guarantees that $\ell = \mathcal{D}_{\min} - 1$ by Lemma 3, where ℓ is the brightness of G . Since the graph is B -sensitive, the distance of the code will be at least $\lceil \ell/B \rceil + 1 = \lceil (\mathcal{D}_{\min} - 1)/B \rceil + 1$ by Theorem 15. The efficient decoding of the code is guaranteed by Theorem 16. The quantities n and k are produced by Claim 2. ■

Theorem 21 (Sensitive girth-7 graphs as codes). If G is a B -sensitive graph with $m = n + k$ nodes, $\text{girth}(G) > 6$, maximum degree $\mathcal{D}_{\max}$, and minimum degree $\mathcal{D}_{\min}$, a choice of γ for Algorithm 3 describes an input packing which yields a $[[n, k, d]]$ code, where

$$n = \Theta(m), \quad k = \Omega\left(\frac{m\mathcal{D}_{\min}^2}{(\gamma+1)\mathcal{D}_{\max}^3}\right),$$

$$d = \Omega\left(\frac{\mathcal{D}_{\min}(1 - 1/\gamma)}{B}\right). \quad (22)$$

The greedy decoder can (efficiently) correct $\lceil (\mathcal{D}_{\min}(1 - 1/\gamma) + 1)/2B \rceil - 1$ errors.

Proof. Since each node has at most $\mathcal{D}_{\min}/\gamma$ inputs or pivots around it and since G has $\text{girth}(G) > 6$, this guarantees that $\ell \geq \mathcal{D}_{\min}(1 - 1/\gamma)$ by Lemma 3, where

ALGORITHM 3. Clever input-pivot assigner.

Function `CleverInputPivot`(G, γ):

```

 $\mathcal{D}_{\min} \leftarrow \min_{v \in V} \deg(v)$ 
for  $v \in V$  do
   $\text{availability}_v \leftarrow \lfloor \mathcal{D}_{\min} / \gamma \rfloor$ 
   $\text{marked}_v \leftarrow \text{False}$ 
while  $\exists v_1, v_2 \in V : (v_1, v_2) \in E$  and  $\text{marked}_{v_1} = \text{marked}_{v_2} = \text{False}$  do
  set  $v_1$  as input
  set  $v_2$  as pivot( $v_1$ )
   $\text{marked}_{v_1} \leftarrow \text{True}, \text{marked}_{v_2} \leftarrow \text{True}$ 
  for  $w \in (N(v_1) \cup N(v_2)) \setminus \{v_1, v_2\}$  do
     $\text{marked}_w \leftarrow \text{True}$ 
     $\text{availability}_w \leftarrow \text{availability}_w - 1$ 
    if  $w \in N(v_1) \cap N(v_2)$  and  $\text{availability}_w > 0$  then
       $\text{availability}_w \leftarrow \text{availability}_w - 1$ 
      // Availability decrements once for each time it appears in a neighbouring set,
      // unless already 0
    if  $\text{availability}_w = 0$  then
      for  $u \in N(w)$  do
         $\text{marked}_u \leftarrow \text{True}$ 
return

```

ℓ is the brightness of G . Since the graph is B -sensitive, the distance of the code will be at least $\lceil \ell/B \rceil + 1 \geq \lceil \mathcal{D}_{\min}/B(1 - 1/\gamma) \rceil + 1$ by Theorem 15. The efficient decoding of the code is guaranteed by Theorem 16. The quantities n and k are produced by Claim 3. ■

With the input packing considered, we return to Theorem 19 and its construction of a $(q + 1)$ -regular graph with girth at least 12, where q is any power of an odd prime. We may apply Algorithm 3, choosing $\gamma = \Theta(1)$, e.g., $\gamma = 2$, to obtain an explicit construction of a code by packing input-pivot pairs into Benson graphs.

Theorem 22 (A greedy-optimal code with high distance). There exists an explicit family of $[[n, \Omega(n^{4/5}), \Theta(n^{1/5})]]$ CSS codes, based on Benson graphs, which are efficiently decodable by the greedy algorithm. The corresponding stabilizers in $\mathcal{S}(G)$, by Proposition 2, have weight bounded above by $O(n^{2/5})$.

By using Algorithm 2 instead of Algorithm 3, we may trade off the check weight bound and rate to construct a $[[n, \Omega(n^{3/5}), \Theta(n^{1/5})]]$ with check weight $O(n^{1/5})$. These trade-offs are, however, generic in that they do not depend on the structure of Benson graphs at all. It is possible that a more careful analysis into the geometry of the Benson graph (beyond its girth and degree properties) will reveal a more clever assignment of input-pivot pairs which ensures that inputs and pivots are separated by distance at least 3, giving a code with rate $\Omega(n^{4/5})$ and check weight bound $O(n^{1/5})$.

Our primary purpose for constructing codes from Benson graphs in Theorem 22 is to demonstrate concretely that the technique of using girth can readily give greedily

decodable code constructions the distance of which scales relatively well with n , i.e., as a power law n^τ for constant τ rather than something constant or logarithmic. The Benson graph code, however, is not immediately practical in the sense of fault tolerance because the check weight will scale as around the distance. Nonetheless, we believe that further investigation into graph constructions may reveal improved results, including larger distance codes as well as ways to find better choices of stabilizer generators by devising QLO strategies of minimum-weight generator selection as in Theorem 13.

As a final example to show the flexibility of Theorem 18, we construct a code with a girth smaller than 12. Specifically, a construction due to Benson [32] and independently Singleton [76] gives a $(q + 1)$ -regular graph (for q a prime power) with girth 8 and $n = \Theta(q^3)$ vertices. We refer to this family as the *Benson-Singleton* graphs. Theorem 18 requires for girth 8 that the pivots be 3-spaced (i.e., all pivot nodes have distance at least 3 from each other on the graph). Therefore, we will assign inputs and pivots according to Algorithm 2. This ensures that inputs and pivots are both 3-spaced, with a logical dimension guarantee $k = \Omega(n/(q + 1)^2) = \Omega(n^{1/3})$. Along with the regularity of the Benson-Singleton graph, allows us to use case (9) of Theorem 18, so that the distance of the code is $q + 1$ and the greedy decoder can correct at least $\lfloor (q + 1)/2 \rfloor - 1$ errors. We have thereby proven the following result.

Theorem 23 (Benson-Singleton graph codes). There is an explicit family of $[[n, \Omega(n^{1/3}), d = \Theta(n^{1/3})]]$ codes, based on Benson-Singleton graphs, which are efficiently decodable up to $\lfloor d/2 \rfloor - 1$ errors via Algorithm 1. The check weight of the code is at most $O(n^{1/3})$, by Proposition 2.

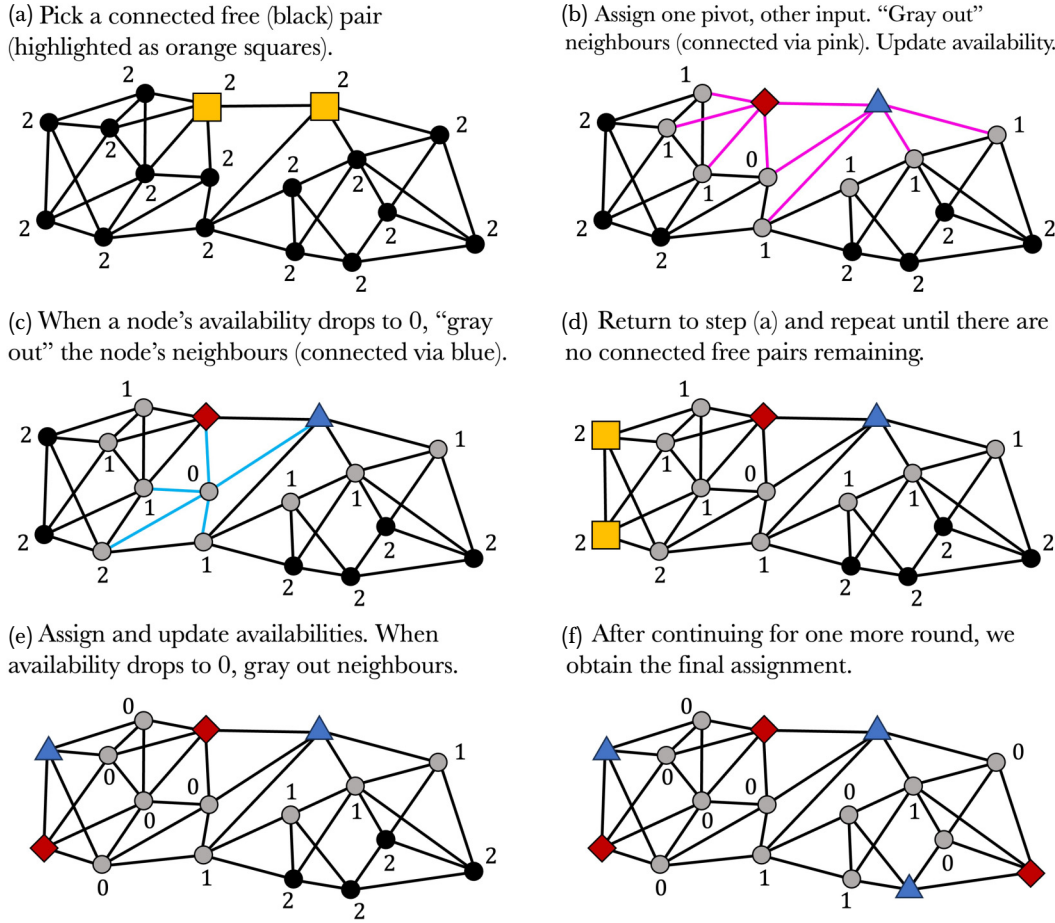


FIG. 27. An example run of the clever input-pivot assignment process given by Algorithm 3. The graph shown has minimum degree $D_{\min} = 4$ and we choose the parameter $\gamma = 2$, so that the availability is $D_{\min}/\gamma = 4/2 = 2$. The graph begins in (a) with all nodes free (shown as black circles) with availability 2 on every node. We choose some connected free pair (orange squares) and assign one as input (blue triangle) and the other as pivot (red diamond). The neighbors are “grayed out,” i.e., marked as not free, and their availability is decreased in (b). The availability of each neighbor drops by 1 if it neighbors only the pivot or only the input, or 2 if it neighbors both the input and the pivot. In (c), all nodes the availabilities of which drop to 0 have their own neighbors grayed out. This process is repeated in (d) and (e), and so on, until the final assignment is reached in (f).

As with the Benson graph codes, this check weight is too large to be practical for fault-tolerance purposes, but further improvements on techniques to reduce check weight it is plausible that constructive techniques of this form may prove useful for practical purposes.

C. Random codes with reduced stabilizer weights

One direct technique for which a graph formalism enables more controlled analysis is random codes. Under the stabilizer tableau representation, the quantum Gilbert-Varshamov bound gives an asymptotic (large n) relation between code parameters n , k , and d that, if satisfied, guarantees the existence of a $[[n, k, d]]$ code.

Theorem 24 (Quantum Gilbert-Varshamov). For parameters $d \leq n/2$, k which depend implicitly on n , as $n \rightarrow \infty$ there exists a $[[n, k, d]]$ code if $nH(d/n) + d \log 3 < n -$

k , where $H(p) := -p \log p - (1 - p) \log(1 - p)$ is the binary entropy function of $p \in [0, 1]$.

We give a self-contained proof of Theorem 24 in Appendix C. The proof proceeds probabilistically, namely by generating a uniformly random stabilizer tableau and arguing that such a tableau has a nonzero probability of having distance d if the bound is satisfied. However, because the expected weight (i.e., number of nonidentity elements) of a random Pauli string $P_1 \otimes \cdots \otimes P_n$, where $P_i \in \{I, X, Y, Z\}$, is $O(n)$, the random codes generated in the Gilbert-Varshamov fashion will have linear stabilizer weight. A natural question is whether the average weight can be decreased by a stochastic algorithm that is more fine grained than uniform randomness. Such a task has proven difficult in the stabilizer tableau representation due to the challenge of generating Paulis that both commute and are

not uniformly distributed. On the other hand, we can easily generate random graphs of a certain structure by defining first the possible edges that may be included, and then randomly choosing edges to include in the graph. This insight enables us to extend the result of the quantum Gilbert-Varshamov bound by way of the following theorem. Recall that a function $f(n)$ is $\omega(g(n))$ if $\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$.

Theorem 25. Let n be the number of physical qubits and define $R(n) < 1$ and $d(n) \in [\omega(1), n/\log n]$. There exists an efficiently samplable family of distributions $\mathcal{S}_n$, parameterized by n , over stabilizer codes that as $n \rightarrow \infty$ satisfy three properties:

- (a) The stabilizer weights are

$$|S_i| \leq \min \left(n, \left(4 + 2 \frac{R}{1-R} \right) d \log n \right), \quad (23)$$

where S_i is a stabilizer from a tableau sampled from $\mathcal{S}_n$.

- (b) The rate ratio of logical to physical qubits is $R(n)$.
 (c) With probability at least $1 - n^{-d+o(d)}$, a random code from $\mathcal{S}_n$ will have distance $d(n)$.

Theorem 25 is positive in that it provides a nontrivial bound on the stabilizer weight of random codes for many choices of $R(n)$ and $d(n)$, and it gives a strong probabilistic concentration of $1 - n^{-d+o(d)}$. For any $d \in [\omega(1), O(n/\log n)]$, this concentration implies that almost all codes in $\mathcal{S}_n$ have the desired properties. On the other hand, Theorem 25 complements—as opposed to subsuming—the Gilbert-Varshamov bound because it provides a nontrivial weight reduction at substantially sublinear distances. It also forces the stabilizer weight—at least, for the generating set we construct—to be at least as large as the distance, which is an artifact of our proof technique. However, the result nonetheless remains interesting because of the guarantees it is able to provide and the flexibility that it offers. Indeed, practically useful codes are seldom asymptotically good. Since the distance and the rate are both determining factors for the stabilizer weight, we obtain a three-way distance-rate-weight trade-off of random codes that can be leveraged to study sublinear-distance codes in which we care much less about one of the three properties than the others. As with the other claims in this paper, this result is thus complementary to LDPC constructions that are asymptotically good but the utility of which lies primarily in complexity theory rather than practical usage, due to, e.g., the smallest- n instance of such quantum LDPC families being far larger than a practical block length. Instead, we interpret this result as evidence that for moderately large codes—e.g., $n \sim 50$ at a scale where we expect little- o asymptotics to take effect—there is a great deal of

flexibility in the graph construction to build codes of certain ranges of R and d which also may have a relatively small stabilizer weight.

We proceed to the proof of Theorem 25.

Proof. We construct elements of $\mathcal{S}_n$ graphically as shown in Fig. 28; denote this graph G . In particular, we begin with an encoder-respecting form, separating the inputs $\mathcal{I}$, pivots $\mathcal{P}$, and (nonpivot) outputs $\mathcal{O}$. We choose a spacing Σ so that every Σ output nodes, we place an input-pivot pair. The rate satisfies $R = 1/(\Sigma + 1)$, since for every $\Sigma + 1$ outputs (including pivots) there is one input. Next, we assign each set of nodes a *locality diameter*, Δ_I , Δ_P , and $\Delta_O + 1$, respectively (the $+1$ since the output node is included in its diameter but does not have a self-edge). The boundary conditions are periodic. Each node may only connect to output nodes within their locality diameter. If two nodes may be connected, we say one is *reachable* from the other. We sample from $\mathcal{S}_n$ by iterating through each connectable edge and randomly including or excluding it from the graph with probability $1/2$; the resulting stabilizer tableau is the one defined by the collection $S := \{S_v := X(v)Z(N_o(v))X(p(i(v)))Z(N_o(p(i(v))))\}_{v \in \mathcal{O}}$ as described in Sec. IV. For notation, we say that S_v contains a Pauli X_i (Z_i) if the i th element of S_v is X_i or Y_i (Z_i or Y_i). Our analysis of this stochastic algorithm adapts the techniques in the analysis of Theorem 24 to the graph picture. We fix a nonidentity Pauli P and consider four cases:

- (1) $P = Z^{\otimes T}$ for $T \subseteq [n]$ and T does not contain any pivots. Pick $i \in T$ and consider $v_i \in G$. Then S_{v_i} must anticommute with P because S_{v_i} contains $X(v_i)X(p(i(v_i)))$ which only overlaps P in one position. Hence, P does not commute with S .
- (2) $P = Z^{\otimes T}$ for $T \subseteq [n]$ and T contains at least one pivot $p \in \mathcal{P}$. Let $i \in \mathcal{I}$ be the unique input connected to p . Each of the Δ_I output nodes v within the diameter Δ_I of i have probability $1/2$ of connecting to i . S_v contains $X(p)$ if and only if i and v are connected. Since the (anti)commutativity of P and that of S_v are determined only by the parity of the number of qubits on which X 's in S_v overlap with Z 's in P , the probability that S_v and P commute is $1/2$ regardless of the other Paulis in S_v and P . Furthermore, consider any other S_w where w is also in the diameter of i . Because w has an edge connecting to i distinct from that of v , the probability of S_w and P commuting is independently $1/2$. This argument is general to all output nodes in the diameter of i , and remaining stabilizer act on disjoint subspaces. Hence the probability that P commutes with S is $2^{-\Delta_I}$.
- (3) P contains at least one $X(p)$ for $p \in \mathcal{P}$. Analogously, $\Pr[P \text{ commutes with } S] = 2^{-\Delta_P}$ by considering the

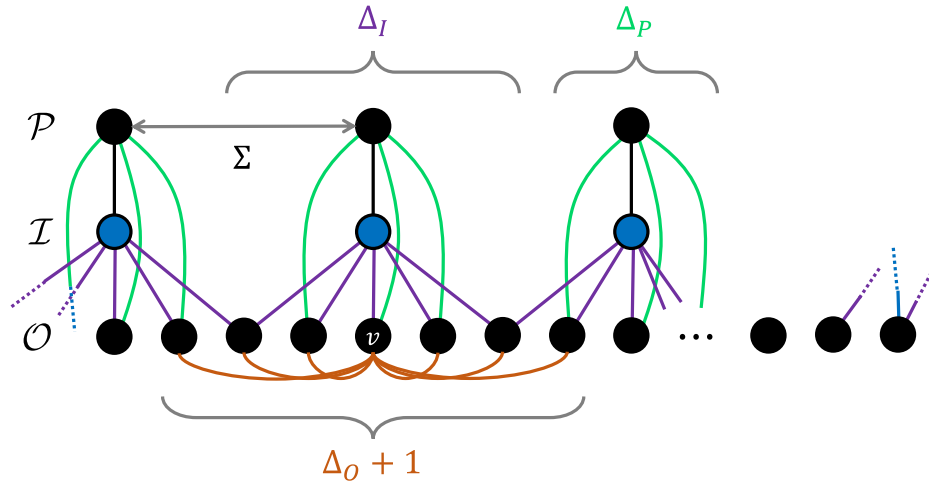


FIG. 28. General localized random graph construction. Nodes are separated into k input nodes $i \in \mathcal{I}$ (which are spaced apart by Σ , depicted as 4 in the figure), k pivot nodes $p \in \mathcal{P}$, and $n - k$ nonpivot output nodes $o \in \mathcal{O}$. Each pivot p (input i) can connect to output nodes within a diameter Δ_P of p (Δ_I of i); here, $\Delta_P = 3$ ($\Delta_I = 5$) and the possible lines are green (purple). We emphasize that the edges drawn represent edges which appear with nonzero probability in the random graph construction, but the actual randomly drawn graph will not include all drawn edges. Similarly, output nodes o may connect to outputs in a diameter Δ_O of o . For clarity, only the possible connections for a single output v is shown, and here $\Delta_O = 6$.

Δ_P nodes in $\mathcal{O}$ each of which independently have probability $1/2$ of having a $Z(p)$.

- (4) P contains at least one $X(v)$ for $v \in \mathcal{O}$. $\Pr[P \text{ commutes with } S] \leq 2^{-\Delta_O}$ by an analogous argument. We neglect some stabilizers which could anticommute with P by acting through neighboring pivots, i.e., the $Z(N_o(p(i(v))))$ component, since that can only decrease the probability and complicates the expression.

Set $\Delta_O = \Delta_P = \Delta_I := \Delta$. We proceed to bound the probability of the event $A(S)$ that there exists nonidentity P with weight $\leq d$ which commutes with S . As shown in Appendix C, there are $2^{nH(d/n) + d \log 3 + O(\log n)}$ such Paulis, where $H(p) = -p \log p - (1-p) \log(1-p)$ is the binary entropy. By the union bound,

$$\Pr[A(S)] \leq 2^{nH(d/n) + d \log 3 + O(\log n) - \Delta} \leq \epsilon, \quad (24)$$

where ϵ bounds the fraction of codes which do not have distance at least d . We therefore solve the equation

$$\Delta \geq \log \frac{1}{\epsilon} + nH\left(\frac{d}{n}\right) + d \log 3 + O(\log n). \quad (25)$$

Since $d/n \leq 1/\log n$, we can asymptotically Taylor expand the binary entropy as $H(d/n) \rightarrow (1/\ln 2 - \log(d/n))d/n$, giving

$$\Delta \geq \log \frac{1}{\epsilon} + d \log n + o(d \log n). \quad (26)$$

Here, we have used our assumption that $d = \omega(1)$, so that $O(\log n) = o(d \log n)$. Let $\epsilon = n^{-d+o(d)}$ (where the $o(d)$

absorbs the $o(d \log n)$ term), so that if $\Delta = 2d \log n$ at least $1 - n^{-d+o(d)}$ of codes asymptotically have distance d . Lastly, we calculate the stabilizer weights.

For $v \in \mathcal{O}$, we consider the weight of $S_v = Z(N_o(v))X(p(i(v)))Z(N_o(p(i(v))))$ which has four distinct contributions. The first term has weight 1. The second and fourth terms are all in $\mathcal{O}$, and collectively may only have weight at most $2\Delta - 1$. This is because the number of nodes in $\mathcal{O}$ that can be affected by this is at most $2\Delta - 1$, as the locality diameter dictates that the last term can reach Δ different nodes from a given input on v 's left-hand side, including v . An additional Δ nodes can be reached on the right-hand side, again including v . But we have double counted v , so we have in total $2\Delta - 1$ distinct nodes that the fourth term can reach. The contribution of the fourth term subsumes that of the first and second. Hence, the total contribution of the first, second, and fourth terms is thus $2\Delta - 1$. As for the third term, we count the number of reachable input nodes, which is $\leq \Delta/\Sigma + 1$. This gives us a total stabilizer weight of at most $(2 + 1/\Sigma)\Delta$. Since $\Sigma = 1/R - 1$, we have

$$|S_v| \leq \left(2 + \frac{1}{\Sigma}\right) \Delta = \left(2 + \frac{1}{1/R - 1}\right) \Delta \quad (27)$$

$$= \left(2 + \frac{R}{1 - R}\right) \Delta = \left(2 + \frac{R}{1 - R}\right) 2d \log n \quad (28)$$

$$= \left(4 + 2\frac{R}{1 - R}\right) d \log n. \quad (29)$$

■

Obviously, $|S_v| \leq n$, which gives the claimed result.

VI. CONCLUSION AND OUTLOOK

We have introduced a graph structure which universally represents all stabilizer codes. Stabilizer tableaux can be efficiently compiled into such graphs, and vice versa. Our primary motivation for such a representation has been to gain access to natural graph properties and notions—degree, geometry, connectivity—and then leverage them to improve code construction and analysis. As first steps in this direction, we have chosen several geometric shapes discretized into graphs and analyzed them as code representations. In doing so, we have found a number of constant-size codes with desirable rates and reasonably large numerical distances, as well as a family of hypercube codes that have near-linear rate, logarithmic distance, and good encoding/decoding properties. In similar spirit, we have also constructed a class of codes for which, given a desired rate R and distance d , a random code drawn from the class has high probability of having rate R , distance d , and, so long as R and d were not both too large, nontrivially bounded stabilizer weights. This analysis extends the result of the quantum Gilbert-Varshamov bound, which has a distance-rate trade-off but no bound on stabilizer weight due to its coarse-grained analysis.

More generally, we have shown that graph properties are intimately tied to code properties. Key code properties (weight, distance, encoding, and logical-operator circuit depths) are all bounded by small linear functions of graph degrees, and key coding algorithms (distance approximation, minimum-weight generator selection, and decoding) are all strategies on various instances of quantum lights out (QLO) games. The former observation has allowed us to leverage graph algorithms to efficiently construct encoding circuits with controlled depth upper bounds as well as to build logical operation circuits the depths of which, in the case of diagonal or certain Clifford gates, are conceptually distinct and quantitatively better than simply unencoding, applying, and reencoding. The latter result has led us to design an efficient greedy QLO strategy for decoding, and prove that it succeeds up to a small constant factor on at least a certain class of graphs which include the hypercube code. We believe that further study in graph-based decoding algorithms will reveal efficient decoders for many other families of graphs. In sum, the universal graph representation enables insights and improvements that are general across all or large classes of stabilizer codes, by providing avenues of construction and analysis that are substantially less obvious via conventional representations of additive quantum codes.

There are several directions in which to explore from this point. The first is general. Classically, graph representations have shown a strong correspondence between graph properties and code properties beyond simple objects such as the degree. Notably, spectral expander graphs have led to good classical codes. A characterization effort

connecting graph properties such as spectrum, expansion, cycle sizes, density, geometry, and topology to distance, decoding, stabilizer weight, etc. could significantly extend the insight of this work into stabilizer codes.

A second pressing direction involves more deeply studying the quantum lights out game, as insights into its strategies have equivalent interpretations on coding algorithm performance. This paper has studied the simplest QLO strategy and derived performance guarantees for graphs of sufficient girth. Such a connection is reminiscent of belief-propagation insights from classical coding theory, which have led to significant progress for classical error correction. By devising more clever QLO strategies, could we weaken the constraints on graphs decoded by the strategy, and therefore construct efficiently QLO-decodable codes with distance large enough to be practical?

A third direction is directly constructive. Given particular architectures or algorithms of experimental interest, is it possible to systematically design graphs the codes of which are particularly suitable for such devices or algorithms in terms of locality of connectivity, geometry, rate, and distance? As a starting point, one can conduct a more thorough investigation into extremal graph theory to obtain the best explicit constructions of graphs with a particular girth (as dictated by the master theorem of girth, Theorem 18). Such graphs come with guaranteed efficient decodability and distance lower bounds nearly matching the upper bound. Even further, one can devise nontrivial strategies for selecting a better choice of stabilizer generators under certain structural conditions, in hopes of chaining these ideas together in order to create high-distance, low-check-weight, and efficiently decodable stabilizer codes. It is plausible that such approaches involve modifying the representation itself. Representations that are universal are doomed in certain aspects of coding because their generality implies that properties scale with distance. If there exists a collection of graphs that, in exchange for representing a restricted subset of stabilizer codes, satisfies low-weight stabilizers, it may open new paths forward for practical code constructions and corresponding algorithms that are fundamentally not based on topological or classical (i.e., CSS-type product) reasoning. Such a representation would essentially lie in an optimal middle ground between the approach of surface-type code constructions and the universal graph representation.

ACKNOWLEDGMENTS

We are grateful to Zhiyang He (Sunny) for fruitful discussions and insightful observations. We also thank Adam Hesterberg for suggesting a closer investigation into the girths of graphs. A.B.K. was supported by the National Science Foundation (NSF) under Grant No. CCF-1729369. J.Z.L. was supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship. P.W.S. was

supported by the NSF under Grant No. CCF-1729369, by the NSF Science and Technology Center for Science of Information under Grant No. CCF-0939370, by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Co-design Center for Quantum Advantage (C2QA) under Contract No. DE-SC0012704, and by NTT Research Grant No. AGMT DTD 9.24.20.

DATA AVAILABILITY

The data that support the findings of this paper are openly available [69].

APPENDIX A: TABLEAU TO GRAPH PROOFS

In the main text, we have sketched briefly the algorithm that maps encoders to ZXCFs. Here, we give a detailed explication. The first part of our algorithm maps the encoder into *some* ZX diagram; we can then transform it via a sequence of equivalence rules into a ZXCF. To accomplish a transformation into ZX diagrams, we first observe prior work by Hu and Khesin [37] for stabilizer states.

Theorem A1 (Hu and Khesin [37]). There exists a ZXCF for stabilizer quantum states, which we call the HK form.

Although Hu and Khesin did not use ZX calculus in their work, their construction uses graph states decorated with single-qubit operators, which can be mapped directly to ZX calculus by a manner we will briefly describe. A HK diagram is constructed by starting with a graph state—a state corresponding to an undirected graph wherein the nodes become $|+\rangle$ qubits and the edges become CZ gates. Each node is then endowed with a single *free edge*, which does not connect to any other node. Local Clifford operations in $\langle H, S, Z \rangle$ are then applied to the qubits. Such a construction has the capacity to express any stabilizer state. A direct mapping enables the presentation of a HK diagram into the ZX calculus:

- (1) HK vertices become Z nodes, as a vertex in a graph state begins in the state $|+\rangle$, which is a Z node with a single output.
- (2) HK (nonfree) edges become (nonfree) edges with Hadamard gates, as both of these correspond to CZ gates.
- (3) HK local operations of S , Z , or SZ become phases on the corresponding node of $\pi/2$, π , and $3\pi/2$, respectively. The local operation H becomes a Hadamard gate on the free edge of the node.

Thus, we will without loss consider HK forms to be in the ZX-calculus representation and denote them *ZX-HK forms*.

We remark that transformations of similar decorated graph state families into ZX-calculus presentations have been considered in other contexts, such as in Backens [56].

The work of Hu and Khesin [37] is limited to stabilizer states. However, by vector space duality one can equivalently transform operators into states and vice versa. (The formal statement of this duality in quantum information theory is given by the Choi-Jamiołkowski isomorphism.) The composition of such a transformation with the result above gives the following lemma.

Lemma A1. There exists an efficient transformation of a ZX encoder diagram with k input edges and n output edges into a corresponding ZX presentation of a stabilizer state in ZX-HK form, with $n + k$ free edges.

Proof. We will turn the encoder diagram into a Clifford circuit, and then map the output of that circuit (with input the all- $|+\rangle$ state) to HK form. Any X (red) nodes in the ZX encoder diagram can be transformed into Z (green) nodes with the same phase surrounded by Hadamard gates. Now, we can interpret the ZX encoder diagram as a state by treating the input edges as output edges (the ZX version of the Choi-Jamiołkowski isomorphism).

Our ZX diagram now computes a state, so we can express it entirely in terms of the following elementary operations and their circuit analogs. First, we could have a Z node with only one output or only one input. In a circuit, this is a $|+\rangle$ qubit or a postselected $\langle +|$ measurement, respectively. We can also have a Hadamarded edge or a node with $\pi/2$ phase, which are represented as H or S gates in circuits, respectively. Lastly, we can have a Z node of degree 3, with either two inputs and one output or one input and two outputs that acts as a merge or a split. This is equivalent to applying a CX operation between two qubits where the second qubit is either initialized to $|0\rangle$ before the CX gate or postselected by the measurement $\langle 0|$ after the CX gate, respectively.

Any ZX diagram can be expressed in terms of these operations [56]. Furthermore, as we apply each of these operations we can keep track of the current HK form for our state. (The Hu and Khesin [37] method shows us how to keep a diagram in HK form as each operation is applied.) All of these steps can be done efficiently [37], so this gives us an efficient procedure for turning a ZX diagram into a corresponding stabilizer state in HK form. ■

Application of Lemma A1 results in a ZX-HK form. That is, there is one Z node per vertex of the graph in HK form, with any internal edge having a Hadamard gate on it. Free edges can only have Hadamard gates on them if their phase is a multiple of π , corresponding to the local Clifford gates H and HZ in HK form, and if the associated node is not connected to any lower-numbered nodes. Nodes for which the free edges have no Hadamard gate are

free to have any multiple of $\pi/2$ as a phase, corresponding to local Clifford operations I , S , Z , or SZ , respectively.

We are now equipped with a ZX-HK diagram that represents a state. This diagram has only output edges. To return it into an encoder diagram, we partition the $n + k$ free edges into k input edges and n output edges. In the circuit representation, this is equivalent to turning bras into kets and vice versa to map between an operator and a state. For example, $|00\rangle\langle 1| - |11\rangle\langle 0| \leftrightarrow |001\rangle - |110\rangle$. Now, if there are any edges between input nodes (those in $\mathcal{I}$), we can simply remove them. They correspond to CZ operations, and we can take off any unitary operation on the input by the encoder definition. The same goes for local Cliffords on $\mathcal{I}$.

At this stage, the obtained ZX-HK form is in encoder-respecting form. It also satisfies the edge and Hadamard rules—there are no input edges in ZX-HK form so the analogous rule is enforced only on lower-numbered nodes, but if we number the nodes from the beginning such that the input nodes are lower-numbered than all output nodes, then the transformed ZX-HK form will satisfy our ZXCF Hadamard rule. So, all that remains is to simplify the diagram to obey the RREF and Clifford rules.

Lemma A2. An encoder diagram in ZX-HK form can be efficiently transformed to satisfy the RREF rule, while continuing to satisfy the edge and Hadamard rules.

Proof. This proof relies on two basic properties of ZX calculus. The first is that a CX gate can be expressed on a pair of qubits by applying a pair of spiders connected by an edge, with a Z spider on the control and an X spider on the target. The second fact is a rewrite rule called the *bialgebra rule* [56], which allows us to rewrite a green node connected to a red node by the following sequence of steps. First, remove the two spiders and the edge between them. Next, add a red node on each edge previously leading into the green node; do the same for the red node by adding green nodes. Lastly, connect each red node to each green node.

Applying a CX gate to a pair of inputs is a unitary operation and does not change our code. The control spider of the CX gate fuses with the first input. The target spider is used to perform the bialgebra rule with the second input. This results in two green nodes where the CX target used to be. The first is merged with the first input while the second takes the place of the recently destroyed second input. Meanwhile, the red nodes on each input-output edge previously connected to the second input now find themselves connected to both the first and second input, and turn green after commuting through the Hadamard gate on the input-output edge, merging with the output on the other side. Any doubled edges are removed by the Hopf rule [56]. The result is that we have added the row of M_D corresponding to the first input to the row of the second. This allows us to

do arbitrary row operations since we are working in a field of characteristic 2. This concludes the proof of the fact that M_D can be turned into RREF without affecting the edge and Hadamard rules. ■

The only remaining task is to enforce the Clifford rule. For any pivot node $p \in \mathcal{P}$ with a nonzero phase, we denote by v_{in} its associated input node. We apply a local complementation about v_{in} , which notably does not change the entries of M_D . However, this operation also increases the phase of each of the neighbors of v_{in} by $\pi/2$ (due to multiplication by S) so we repeat this process until the phase of that pivot vanishes. The local complementation transformation is a ZX equivalence identity, and therefore is a valid operation on our diagram [56].

Lastly, if there are any edges between pivots p_1 and p_2 , we can remove them by applying a different transformation ϕ to their pair of associated input nodes v_1 and v_2 . Let N_1 be the set of the neighbors of v_1 as well as v_1 itself and let N_2 be defined respectively. The effect of ϕ is to toggle all edges in the set $N_1 \times N_2$, including multiplicity. This means that edges that get toggled twice are not affected. Any self-edge of the form (v, v) is treated as a Z operation on vertex v . In addition, ϕ also applies an additional local operation of HZ to each of v_1 and v_2 , but this can be removed as v_1 and v_2 are input vertices. As a result of this, we have swapped the neighbors of v_1 and v_2 as well as toggled the one pivot-pivot edge that we wish to flip, (p_1, p_2) . Note that ϕ does not violate the RREF rule because once this process is complete we can simply swap the neighbors back (without any toggling) via a row operation. This also does not add any local operations to the nodes p_1 and p_2 , as each is only in one of N_1 and N_2 , and thus does not receive a Z operation. We repeat until no pivot-pivot edges remain. ϕ can be shown to be a valid ZX equivalence identity from van de Wetering [58, Eq. (101)].

With that step, the transformation from Clifford encoder to ZXCF is complete. Although this process may seem lengthy, all of these steps can be done systematically in an efficient manner, without having to go back to fix earlier rules. Specifically, this algorithm takes $O(n^3)$ time. Creating the HK diagram for the encoder requires us to apply $O(n)$ Pauli projections, each of which can be applied in $O(n^2)$ time. After the HK form is created, operations such as row-reducing a matrix also take $O(n^3)$ time. We suspect that the time complexity cannot be improved in the worst case, but that there is a lot of room for heuristic run-time improvements and optimizations.

APPENDIX B: DERIVATION OF THE ZXCF COUNTING RECURSION

In this appendix, we derive the recursive counting formula of ZXCF diagrams, given in Eq. (3). To derive f , imagine that, starting with two empty bins, we must assign

the $\bar{k} := n - k$ output nodes to be pivots (case *A*) or non-pivots (case *B*), where pivots need to be matched with input nodes. The current number of pivots is tracked by p and the number of nonpivot outputs is tracked by o .

Suppose that we want the next output node to be a pivot (in $\mathcal{P}$). Since there is a one-to-one correspondence between pivots and inputs, we can add pivots only if $n > k$, and thus $\bar{k} > 0$. The matching between pivots and input nodes is fully constrained by the RREF rule, which sorts the inputs and pivots together. The Clifford rule says that no pivot nodes may have local Clifford operations, so we just need to choose the edges connecting them to nodes we have already assigned. Since there are no pivot-pivot edges and the pivot connects to only one input, we have exactly 2^o possibilities. Having made an assignment, p increases by 1, and n decreases by 1 (both input and output are decremented since the pivot matches with an input).

Suppose instead that we want the next output node to be a nonpivot (in $\mathcal{O}$). Then we have to choose the local Clifford operation as well as its edges to previously assigned vertices. If we choose to connect the node to none of the p assigned inputs, p assigned pivots, or o assigned vertices, then we are allowed to place any of the six local Cliffords on the node. If any of those edges are present, however, we cannot apply a Hadamard to the output edge due to the Hadamard rule. Hence, four choices remain for the local Clifford operation. This works out to a total of $4(2^{2p+o} - 1) + 6 = 2^{2p+o+2} + 2$ possibilities. To finish, we decrement the number of output qubits without changing the number of inputs.

APPENDIX C: QUANTUM GILBERT-VARSHAMOV

The quantum Gilbert-Varshamov bound (Theorem 24) is a somewhat folklore result but is discussed in, e.g., Nielsen and Chuang [8]. Here, we provide a self-contained proof.

Proof. We proceed via a probabilistic method. That is, we compute the probability that a random code has parameters $[[n, k, d]]$ with n large. To construct a random stabilizer code, we begin with $Z_1, \dots, Z_{n-k}$ as stabilizers, and then choose a random Clifford operator U on n qubits. The random stabilizer code is given by $S_1 := UZ_1U^\dagger, \dots, S_{n-l} := UZ_{n-k}U^\dagger$. Such a code is a uniformly random stabilizer code because (a) all generators commute and (b) a random Clifford takes distinct nonidentity Paulis to uniformly random nonidentity Paulis.

Fix a n -qubit nonidentity Pauli P and note that $P' := UPU^\dagger$ is a fixed uniformly random nonidentity Pauli. Let $p = \Pr[[P, S_1] = 0, \dots, [P, S_{n-k}] = 0] = \Pr[[P', Z_1] = 0, \dots, [P', Z_{n-k}] = 0]$ be the probability that P commutes with the stabilizer tableau. If P' commutes with Z_i , then the Pauli on the i th qubit must be either I or

Z . There are $2^{n-k}4^k - 1$ nonidentity Paulis which have I or Z on the first $n - k$ qubits, and $4^n - 1$ total nonidentity Paulis. For large n then,

$$p = \frac{2^{n-k}4^k - 1}{4^n - 1} \approx \frac{2^{n+k}}{4^n} = \frac{1}{2^{n-k}}. \quad (C1)$$

On the other hand, the total number of nonidentity Paulis with weight at most d is

$$N = \sum_{j=1}^d 3^j \binom{n}{j} \leq d 3^d \binom{n}{d}, \quad (C2)$$

using the assumption that $d \leq n/2$ so that the last term is the greatest. By Stirling's approximation,

$$N \longrightarrow 2^{nH(d/n) + d \log 3 + O(\log n)}. \quad (C3)$$

To complete the proof, we apply the union bound. Let $A(S, P)$ be the event that P commutes with $S_1, \dots, S_{n-k}$, and let $A(S) = \bigcup_{P \neq I} A(S, P)$. Then

$$\begin{aligned} \Pr[A(S)] &\leq \sum_{P \neq I} \Pr[A(S, P)] \\ &\leq 2^{nH(d/n) + d \log 3 + O(\log n)} 2^{k-n}. \end{aligned} \quad (C4)$$

A $[[n, k, d]]$ code exists if and only if $\Pr[A(S)] < 1$. By setting the right-hand side of Eq. (C4) to < 1 and solving, we obtain the quantum Gilbert-Varshamov bound. ■

-
- [1] L. K. Grover, in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (ACM, Philadelphia, PA, 1996), p. 212.
 - [2] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Rev.* **41**, 303 (1999).
 - [3] Z. Brakerski, P. Christiano, U. Mahadev, U. Vazirani, and T. Vidick, A cryptographic test of quantumness and certifiable randomness from a single quantum device, *J. ACM (JACM)* **68**, 1 (2021).
 - [4] E. Farhi, D. Gosset, A. Hassidim, A. Lutemirski, and P. Shor, in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (ACM, Cambridge, MA, 2012), p. 276.
 - [5] I. M. Georgescu, S. Ashhab, and F. Nori, Quantum simulation, *Rev. Mod. Phys.* **86**, 153 (2014).
 - [6] H.-Y. Huang, R. Kueng, and J. Preskill, Predicting many properties of a quantum system from very few measurements, *Nat. Phys.* **16**, 1050 (2020).
 - [7] A. Harrow, P. Hayden, and D. Leung, Superdense coding of quantum states, *Phys. Rev. Lett.* **92**, 187901 (2004).
 - [8] M. A. Nielsen, and I. Chuang, *Quantum computation and quantum information* (2002).

- [9] A. Kandala, K. Temme, A. D. Córcoles, A. Mezzacapo, J. M. Chow, and J. M. Gambetta, Error mitigation extends the computational reach of a noisy quantum processor, *Nature* **567**, 491 (2019).
- [10] M. V. Larsen, X. Guo, C. R. Breum, J. S. Neergaard-Nielsen, and U. L. Andersen, Deterministic generation of a two-dimensional cluster state, *Science* **366**, 369 (2019).
- [11] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, Quantum supremacy using a programmable superconducting processor, *Nature* **574**, 505 (2019).
- [12] H. Wang, J. Qin, X. Ding, M.-C. Chen, S. Chen, X. You, Y.-M. He, X. Jiang, L. You, Z. Wang *et al.*, Boson sampling with 20 input photons and a 60-mode interferometer in a 10^{14} -dimensional Hilbert space, *Phys. Rev. Lett.* **123**, 250503 (2019).
- [13] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, N. Lacroix, G. J. Norris, M. Gabureac, C. Eichler, and A. Wallraff, Repeated quantum error detection in a surface code, *Nat. Phys.* **16**, 875 (2020).
- [14] A. Y. Kitaev, in *Quantum Communication, Computing, and Measurement* (Springer, Shizuoka, Japan, 1997), p. 181.
- [15] S. B. Bravyi, and A. Y. Kitaev, Quantum codes on a lattice with boundary, [arXiv:quant-ph/9811052](https://arxiv.org/abs/quant-ph/9811052).
- [16] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, *Phys. Rev. A* **86**, 032324 (2012).
- [17] P. W. Shor, Scheme for reducing decoherence in quantum computer memory, *Phys. Rev. A* **52**, R2493 (1995).
- [18] D. Gottesman, An introduction to quantum error correction and fault-tolerant quantum computation, [ArXiv:0904.2557](https://arxiv.org/abs/0904.2557).
- [19] D. Gottesman, Stabilizer Codes and Quantum Error Correction. Thesis, California Institute of Technology, Pasadena, California, 1997. [ArXiv:quant-ph/9705052](https://arxiv.org/abs/quant-ph/9705052)
- [20] A. Steane, Multiple-particle interference and quantum error correction, *Proc. R. Soc. London. Series A: Math., Phys. Eng. Sci.* **452**, 2551 (1996).
- [21] A. R. Calderbank, and P. W. Shor, Good quantum error-correcting codes exist, *Phys. Rev. A* **54**, 1098 (1996).
- [22] S. Aaronson, and D. Gottesman, Improved simulation of stabilizer circuits, *Phys. Rev. A* **70**, 052328 (2004).
- [23] J.-P. Tillich, and G. Zémor, Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength, *IEEE Trans. Inf. Theory* **60**, 1193 (2013).
- [24] P. Panteleev, and G. Kalachev, Quantum LDPC codes with almost linear minimum distance, *IEEE Trans. Inf. Theory* **68**, 213 (2021).
- [25] N. P. Breuckmann, and J. N. Eberhardt, Balanced product quantum codes, *IEEE Trans. Inf. Theory* **67**, 6653 (2021).
- [26] P. Panteleev, and G. Kalachev, Degenerate quantum LDPC codes with good finite length performance, *Quantum* **5**, 585 (2021).
- [27] M. Sipser, and D. A. Spielman, Expander codes, *IEEE Trans. Inf. Theory* **42**, 1710 (1996).
- [28] A. Y. Kitaev, Fault-tolerant quantum computation by anyons, *Ann. Phys.* **303**, 2 (2003).
- [29] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, High-threshold and low-overhead fault-tolerant quantum memory, *Nature* **627**, 778 (2024).
- [30] J. Edmonds, Maximum matching and a polyhedron with 0,1-vertices, *J. Res. Natl. Bur. Stand. B* **69**, 55 (1965).
- [31] J. Edmonds, Paths, trees, and flowers, *Can. J. Math.* **17**, 449 (1965).
- [32] C. T. Benson, Minimal regular graphs of girths eight and twelve, *Can. J. Math.* **18**, 1091 (1966).
- [33] T. Bergamaschi, L. Golowich, and S. Gunn, in *Proceedings of the 56th Annual ACM Symposium on Theory of Computing* (2024), p. 1507.
- [34] T. Rakovszky, and V. Khemani, The physics of (good) LDPC codes II. Product constructions, [ArXiv:2402.16831](https://arxiv.org/abs/2402.16831).
- [35] A. Ashikhmin, S. Litsyn, and M. A. Tsfasman, Asymptotically good quantum codes, *Phys. Rev. A* **63**, 032311 (2001).
- [36] P. Panteleev, and G. Kalachev, in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing* (ACM, Rome, Italy, 2022), p. 375.
- [37] A. T. Hu, and A. B. Khesin, Improved graph formalism for quantum circuit simulation, *Phys. Rev. A* **105**, 022432 (2022).
- [38] T. McElvanney, and M. Backens, Complete flow-preserving rewrite rules for MBQC patterns with Pauli measurements, *Electron. Proc. Theor. Comput. Sci.* **394**, 66 (2023).
- [39] S. Yu, Q. Chen, and C. H. Oh, Graphical quantum error-correcting codes, [arXiv:0709.1780](https://arxiv.org/abs/0709.1780).
- [40] G. A. Munné, V. Kasper, and F. Huber, Engineering holography with stabilizer graph codes, *npj Quantum Inf.* **10**, 48 (2024).
- [41] S. Beigi, M. Grassl, P. Shor, and B. Zeng, Graph concatenation for quantum codes, *J. Math. Phys.* **52**, 022201 (2009).
- [42] M. Grassl, A. Klappenecker, and M. Rotteler, in *Proceedings, IEEE International Symposium on Information Theory* (IEEE, Lausanne, Switzerland, 2002), p. 45.
- [43] D. Schlingemann, Stabilizer codes can be realized as graph codes, *Quantum Inf. Comput.* **2**, 307 (2002).
- [44] D. Schlingemann, and R. F. Werner, Quantum error-correcting codes associated with graphs, *Phys. Rev. A* **65**, 012308 (2001).
- [45] A. Kissinger, Phase-free ZX diagrams are CSS codes ($\dots$ or how to graphically grok the surface code), [ArXiv:2204.14038](https://arxiv.org/abs/2204.14038).
- [46] Z. Wu, S. Cheng, and B. Zeng, A ZX-calculus approach to concatenated graph codes, [ArXiv:2304.08363](https://arxiv.org/abs/2304.08363).
- [47] N. Chancellor, A. Kissinger, S. Zohren, J. Roffe, and D. Horsman, Graphical structures for design and verification of quantum error correction, *Quantum Sci. Technol.* **8**, 045028 (2016).
- [48] A. B. Khesin, and A. Li, Equivalence classes of quantum error-correcting codes, [ArXiv:2406.12083](https://arxiv.org/abs/2406.12083).
- [49] B. Coecke, and R. Duncan, in *International Colloquium on Automata, Languages, and Programming* (Springer, Reykjavik, Iceland, 2008), p. 298.
- [50] T. Peham, L. Burgholzer, and R. Wille, Equivalence checking of quantum circuits with the ZX-calculus, *IEEE J. Emerg. Sel. Top. Circuits Syst.* **12**, 662 (2022).
- [51] A. Cowtan, and S. Majid, Quantum double aspects of surface code models, *J. Math. Phys.* **63**, 042202 (2022).

- [52] J. van de Wetering, Constructing quantum circuits with global gates, *New J. Phys.* **23**, 043015 (2021).
- [53] R. D. P. East, J. van de Wetering, N. Chancellor, and A. G. Grushin, AKLT-states as ZX-diagrams: Diagrammatic reasoning for quantum states, *PRX Quantum* **3**, 010302 (2022).
- [54] N. de Beaudrap, and D. Horsman, The ZX calculus is a language for surface code lattice surgery, *Quantum* **4**, 218 (2020).
- [55] A. Kissinger, and J. van de Wetering, Reducing the number of non-Clifford gates in quantum circuits, *Phys. Rev. A* **102**, 022406 (2020).
- [56] M. Backens, The ZX-calculus is complete for stabilizer quantum mechanics, *New J. Phys.* **16**, 093021 (2014).
- [57] R. Duncan, A. Kissinger, S. Perdrix, and J. Van De Wetering, Graph-theoretic simplification of quantum circuits with the ZX-calculus, *Quantum* **4**, 279 (2020).
- [58] J. van de Wetering, ZX-calculus for the working quantum computer scientist, *ArXiv:2012.13966*.
- [59] D. Gottesman, Surviving as a quantum computer in a classical world, Textbook manuscript preprint (2016).
- [60] R. Duncan, and M. Lucas, *Proceedings of Quantum Physics and Logic (QPL)* (Barcelona, Spain, 2013).
- [61] E. Knill, R. Laflamme, R. Martinez, and C. Negrevergne, Benchmarking quantum computers: The five-qubit error correcting code, *Phys. Rev. Lett.* **86**, 5811 (2001).
- [62] A related discussion of such a transformation is given in a simultaneous work by one of the authors in Khesin and Li [48].
- [63] V. G. Vizing, On an estimate of the chromatic class of a p-graph, *Diskret. Analiz.* **3**, 25 (1964).
- [64] J. Misra, and D. Gries, A constructive proof of Vizing's theorem, *Inf. Process. Lett.* **41**, 131 (1992).
- [65] We adopt the name quantum lights out because the game appears to be a generalization of a classical game known as lights out, in which a player aims to turn all lights off on a grid with some lights initially on, using moves such that flipping a switch at a given cell toggles all the lights of the adjacent cell but not their own. QLO generalizes this notion onto a general graph, and adds more flexibility on how lights may be toggled.
- [66] U. Kapshikar, and S. Kundu, On the hardness of the minimum distance problem of quantum codes, *IEEE Trans. Inf. Theory* **69**, 6293 (2023).
- [67] P. Iyer, and D. Poulin, Hardness of decoding quantum stabilizer codes, *IEEE Trans. Inf. Theory* **61**, 5209 (2015).
- [68] R. Tanner, A recursive approach to low complexity codes, *IEEE Trans. Inf. Theory* **27**, 533 (1981).
- [69] J. Z. Lu, and A. B. Khesin, graphcodes (GitHub Repository), 2024, <https://github.com/jz-lu/graphcodes>
- [70] R. Chao, and B. W. Reichardt, Fault-tolerant quantum computation with few qubits, *npj Quantum Inf.* **4**, 42 (2018).
- [71] A. Couvreur, N. Delfosse, and G. Zémor, A construction of quantum LDPC codes from Cayley graphs, *IEEE Trans. Inf. Theory* **59**, 6087 (2013).
- [72] F. Pastawski, B. Yoshida, D. Harlow, and J. Preskill, Holographic quantum error-correcting codes: Toy models for the bulk/boundary correspondence, *J. High Energy Phys.* **2015**, 1 (2015).
- [73] O. Higgott, and N. P. Breuckmann, Constructions and performance of hyperbolic and semi-hyperbolic Floquet codes, *PRX Quantum* **5**, 040327 (2024).
- [74] N. P. Breuckmann, and J. N. Eberhardt, Quantum low-density parity-check codes, *PRX Quantum* **2**, 040101 (2021).
- [75] R. W. Hamming, Error detecting and error correcting codes, *Bell Syst. Tech. J.* **29**, 147 (1950).
- [76] R. Singleton, On minimal graphs of maximum even girth, *J. Comb. Theory* **1**, 306 (1966).