S.T. Yau High School Science Award

Research Report

The Team
Name of team member: Alexander Li
School: C. Leon King High School
City, Country: Tampa, FL, USA

Name of supervising teacher: Andrey Boris Khesin
Job Title: Graduate student
School/Institution: MIT
City, Country: Cambridge, MA, USA

Title of Research Report
Canonical Forms and Equivalence Classes of Quantum Clifford Codes in ZX Calculus

Date
August 19, 2023

Canonical Forms and Equivalence Classes of Quantum Clifford Codes in ZX Calculus

Alexander M. Li[1]

[1] *C. Leon King High School*

(Dated: August 19, 2023)

Theoretical work on quantum computation has enabled the creation of quantum error-correcting codes that can provide quantum computers with the ability to combat the inherent noise present in quantum mechanical systems. These error-correcting codes can be presented using ZX calculus, a graphical approach used to display quantum systems and processes. We aim to derive a canonical form for quantum Clifford codes by using the formalism of ZX calculus. First, we derive canonical forms for selected surface and Toric codes using manual manipulation of ZX diagrams and specialized diagrammatic software called Quantomatic. For these codes, the permutation of outputs nodes will be considered to give different codes. Next, we relax the restraints on output nodes so that permuting the output nodes results in an equivalent code, and we analyze and tabulate the equivalence classes of Clifford encoders by writing code in Java and outputting parameters, such as equivalence class sizes and the presence (or lack) of bipartite forms. This work builds on previous works in converting stabilizer tableaus into a form in ZX calculus that is intuitive and explainable, and this paper provides critical work in deriving improved canonical forms for Clifford codes.

Keywords: canonical form, Clifford codes, error-correcting codes, normal form, quantum compilers, quantum computing, quantum mechanical system, surface code, Toric code, ZX calculus

CONTENTS

I. INTRODUCTION

The work done in the past half century on quantum computing have brought large-scale quantum computers closer to reality. Today, quantum computers employing a low number (up to a few hundred) of qubits, in the form of photons and nuclear spins [1], have been created that are mainly used for experiments [2, 3]. These quantum computers differ from classical computers and classical supercomputers by employing the use of qubits rather than bits. The properties of quantum mechanics inherent in qubits, including superposition and entanglement, allow quantum computers to simulate quantum systems, which can make certain calculations much more efficient than classical computers.

However, as with classical information processing systems, quantum information processing systems also face noise that can disrupt information transmission between a sender and receiver. One of the principle challenges in quantum computing is to account for this noise, due to the fragility of quantum bits. To this end, quantum error-correcting codes are used to transmit quantum information successfully in the presence of noise [4]. While classical computers can copy bits, quantum mechanics does not allow the cloning of unknown qubits, and the measurement of a qubit eliminates the information available in the qubit. As such, the construction of suitable quantum error-correcting codes presents new challenges when compared to the construction of classical error-correcting codes. For decades, a quantum error-correcting code against general errors seemed impossible, until the Shor code [5] was first published in 1995 and Steane code [6] in 1996. Other examples of quantum error-correcting codes are the five qubit code [7] and the Toric code [8].

A number of approaches have been created to represent the components of quantum error-correcting codes. The stabilizer formalism is a method that expresses quantum error-correcting codes in terms of stabilizers, operators that, when acted on a stabilizer state, preserve the state [9]. This approach borrows ideas from group theory to represent the whole class of stabilizers with a finite number of generators. To make the idea of quantum error-correcting codes visual, recent advances have made progress on the topic of representing stabilizer states as graph states [10, 11].

Following the work on graph states, work has been done on representing Clifford codes using ZX calculus, a diagrammatic approach developed in these papers [12–14]. The properties of ZX calculus that allow it to replace the stabilizer tableau formalism (a tabulated form of the generators of the stabilizers) are its universality (it can express every quantum operation), soundness (tableaus can derive equiva-

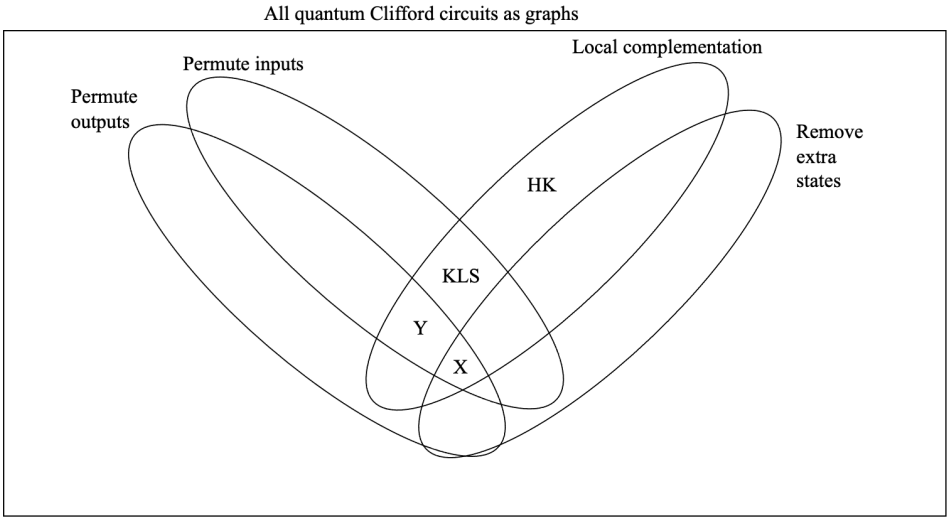


FIG. 1: Shown is a Venn diagram classifying quantum Clifford circuits as graphs, with the universal set of graphs surrounding the diagram. Khesin-Lu-Shor (KLS) forms are determined based on equivalence through permuting inputs and local complementation. The Hu-Khesin (HK) form is in the category of only local complementation because it does not have inputs to permute.

lence of ZX calculus diagrams), and completeness (ZX calculus diagrams can derive equivalence of tableaus) [12, 14]. This graphical language has had various applications in quantum information [15–18] and quantum computation problems [19, 20].

Expressing quantum Clifford circuits as graphs and finding canonical forms for equivalent graphs has had recent advances in the past few years. The Hu-Khesin (HK) form from [11] provides a canonical form for quantum Clifford states. In the context of quantum encoders, this is equivalent to having no inputs and only outputs. Then, the Khesin-Lu-Shor (KLS) form from [21] built on the HK form, providing a canonical form for Clifford encoders. The KLS paper gives a detailed process of transforming stabilizer tableaus into ZX calculus, then performing operations that preserve equivalence to find a canonical form. The first part (section III) of the present work expands on the work from KLS and focuses more topologically on the general shape the vertices and edges that the ZX diagram forms, and we specifically analyze selected surface and Toric codes to see how we can simplify the diagram into an intuitive canonical form. The second part similarly focuses on the general properties of the ZX graph, taking into account the bipartite portion of the graph between the input and output nodes while also allowing the permutation of output nodes to stay in the same equivalence class.

See Figure 1 for a summary of the work done in quantum Clifford encoders' graphical representations. The set of all quantum Clifford circuits as graphs is split into sectors depending on whether we consider the operation as giving equivalent Clifford codes. For example, the sector labeled KLS is positioned at the intersection of "permute inputs" and "local complementation." Therefore, the KLS canonical form relied on the encoders staying in the same equivalence class upon these two operations while the encoders changed equivalence classes when the output nodes are permuted or extra states (nodes unconnected to input nodes) are removed from the output nodes. The second part of this paper considers the sector labeled Y, as output nodes will be allowed to permute.

In this paper, Section II contains key definitions and background on ZX calculus and Clifford encoders. Section III contains our work on Calderbank-Shor-Steane (CSS) codes, specifically surface and Toric codes, in making an intuitive canonical form for select encoders. This builds on recent work from Kissinger [22, 23] that introduced the normal form of of CSS codes, which are "explainable" in that it is efficient to determine the stabilizers from the ZX normal form. For an $(n-2k)$-to-n encoder with k X-checks and k Z-checks, the resulting normal form will consist of $(n-2k)+n+k = 2n-k$ nodes, which are the input nodes, output nodes, and internal nodes representing the X-checks (for the ZX normal form) or the Z-checks (for the XZ normal form). Section III focuses on presenting a canonical form of the codes that eliminates these internal nodes while keeping the diagram for the encoder as intuitive and elegant as possible.

Section IV provides another definition of equivalence, permitting outputs to be permuted as a valid operation among equivalent graphs. The reason this definition of equivalence is also considered is that changing the order of the outputs does not change the "amount" of entanglement that the encoder puts the input qubits through. Section V expands on this definition by omitting parts of the set of quantum encoders that will not be necessary for the remainder of the paper. Since the complexity and overall structure of an encoder does not change with the addition of local quantum gates, for example, section IV explains how we omit them. Sections VI and VII provide the work we have done towards identifying equivalence classes and finding canonical forms. Section VI explains a method of representing encoders as integers so as to sort them into equivalence classes using a Java program. Then, it goes on to revealing more information about these equivalence classes, including sizes and the presence or lack of bipartite forms. Section VII expands on the equivalence classes containing bipartite forms.

II. BACKGROUND

In this section, we define key terms and background on error-correcting codes and the ZX calculus.

First, we define the following matrices.

Definition A. The *Pauli matrices* are

$$I \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$Y \equiv \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The Pauli matrices numerically represent quantum gates that can act on qubits and alter their state. Then, the *Pauli operators on n qubits* are n-fold tensor products of Pauli matrices, multiplied to a factor of the form i^k where $k \in \{0, 1, 2, 3\}$ and $i = \sqrt{-1}$.

Pauli operators are able to act on states in multi-qubit systems. For example, in a three qubit system, the tensor product $Z \otimes Z \otimes I$ will make Z act on the first qubit, Z act on the second qubit, and I act on the third qubit. This can be denoted as $Z_1 Z_2$, with the subscripts showing which qubit the operators are acting on. We can similarly denote other tensor products of operators.

With the operators from Definition A, we can construct quantum error-correcting codes that correct arbitrary errors for multiple qubits.

Other quantum gates that are useful are the Hadamard, CNOT, phase, and $\pi/8$ gates:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{\pi i/4} \end{pmatrix}.$$

These operations have the property of universality, that is, their ability to approximate any operator to arbitrary accuracy [4].

Clifford codes can be represented as a group of generators of stabilizers that, when multiplied together, form the whole class of stabilizers. These stabilizers are Pauli operators on n qubits and listing out the generators determines the whole encoder. In this paper, the encoders take $n - k$ inputs, which are logical qubits, and give n outputs, which are physical qubits. The generators are chosen to be independent, so each degree of freedom going from inputs to outputs requires an additional generator for the stabilizers, implying there are k generators for this $(n - k)$-to-n code.

ZX calculus makes the representation of Clifford codes graphical, and we use the conventions as described in [14, 21]. For an $(n - k)$-to-n encoder, the $n - k$ input nodes are connected with edges to the n output nodes, and the output nodes share edges amongst each other. Any connections among input nodes can be removed since these connections are just controlled-Z operations, and unitary operations on the inputs do not change the stabilizers.

Furthermore, each input node is connected to an input edge while each output node is connected to an output edge. Local operations on any input edges can be removed since these are also unitary operations. Any wires that contain exactly one Hadamard gate are denoted by a light blue edge while those that contain no gates are denoted by a black edge. An example of a Clifford code expressed in ZX calculus is shown in Figure 2.

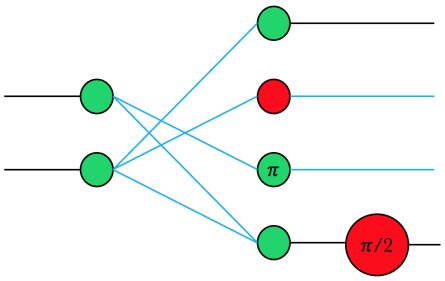


FIG. 2: Example of an encoder. The incoming edges from the left side are input edges and the outgoing edges on the right side are output edges. Note the local operations applied on the output qubits.

The canonical forms described in section III are based on the KLS canonical form [21], which consists of four rules that can be efficiently checked on a given ZX graph. From a given ZX graph, it is efficient to transform to the KLS canonical form using a series of operations.

One of the operations commonly used to transform equivalent ZX diagrams between each other is defined below.

Definition B. Consider a simple graph $G = (V, E)$, where V is the set of vertices and E is the set of (un-directed) edges between vertices. Take a vertex $v \in V$. The neighborhood of v, $N(v)$, is the set of all vertices in V adjacent to v, not including v. By performing a *local complementation* about vertex v, all edges connecting two vertices in $N(v)$ are toggled. That is, if the edge existed before the local complementation, it is removed; if it did not exist before, it is added.

III. SURFACE AND TORIC CODES

To find the canonical forms of CSS codes based on the ZX normal forms, we first consider Toric codes, introduced in [24], and surface codes, as explained in [23]. The canonical forms presented here have 0 internal nodes, so that each node corresponds to an input or output edge.

We begin with a definition of Toric codes.

Definition C. A *Toric code* is a quantum error-correcting code that can be represented on a three-dimensional torus $\mathcal{T}$. For a $m \times n$ Toric code, $\mathcal{T}$ is wrapped by $m-1$ circles parallel to the plane of the major circle and $n-1$ circles perpendicular to the plane of the major circle. The stabilizers of the code are defined by first labeling each disjoint edge with a qubit. Then, every disjoint four-sided face formed by the wrapped circles gives a Z-check (stabilizer with only Z's and I's) consisting of

Z's on the four edges' qubits, and every vertex formed by the intersection of the wrapped circles gives an X-check (stabilizer with only X's and I's) consisting of X's on the four edges' qubits adjacent to the vertex.

For concreteness, we explain the stabilizers for the case of the 3-by-3 Toric code.
Shown in Figure 3 is a representation of the 3-by-3 Toric code (note that this is not a ZX diagram). The edges labeled 3, 6, 9, 16, 17, and 18 would wrap around the torus if the diagram were put in three dimensions. In this representation, each edge (numbered 1 through 18) represents a physical qubit, with the Z-check stabilizers represented by the four edges surrounding a face and the X-check stabilizers represented by the four edges surrounding a vertex. There are 9 disjoint four-sided faces on the torus; two examples include the faces representing $Z_1Z_4Z_{10}Z_{11}$ and $Z_3Z_9Z_{16}Z_{18}$. Similarly, there are 9 vertices from which to get X-checks; two examples include the vertices representing $X_1X_3X_{10}X_{16}$ and $X_8X_9X_{15}X_{18}$. From this diagram, we can read off the stabilizers of the 3-by-3 Toric code. To convert this into the ZX diagram form, we place a Z node on each edge of Figure 3 to represent the physical qubits so that these Z nodes become the output nodes. We also place two Z nodes at the bottom of the diagram to be the input nodes. This results in 20 nodes, and we add no more nodes from here.

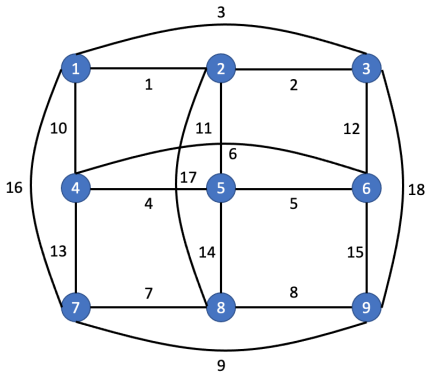


FIG. 3: The 3-by-3 Toric code.

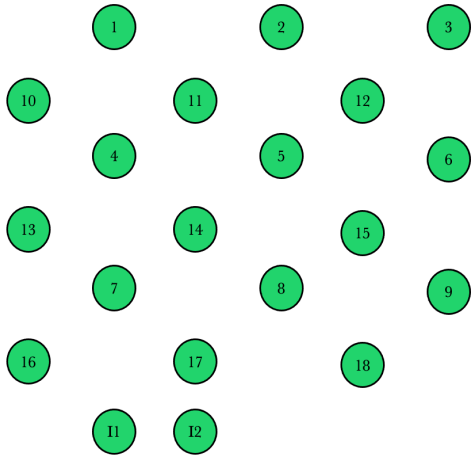


FIG. 4: The nodes in the 3-by-3 Toric code. The two input nodes are labelled I1 and I2, and the output nodes are labelled with integers 1 through 18. Note that these labels are not the phases of the nodes.

Instead of placing nodes 3, 6, and 9 directly on the edges shown in Figure 3, we place them to the right of nodes 2, 5, and 8, respectively. Similarly for nodes 16, 17, and 18.
As shown in the appendix, we determine the structure of the 3-by-3 Toric code's ZX diagram by deducing the placements of Hadamards on output edges and using stabi-

lizers to force the connections between output nodes.

By similar methods, we can also present the ZX calculus form of the 2-by-2 Toric code. The diagram is shown in Figure 5. This has an arrow-like structure among the output-output edges, as seen by nodes 1, 3, 5, and 6, as well as nodes 2, 4, 5, and 6.

The final 3-by-3 Toric code is shown in Figure 6. Note that, by wrapping this pattern around a torus, it would be horizontally periodic. The edges among vertices 1, 4, 10, and 11 form an upward-arrow-like figure. Similarly, nodes 2, 5, 11, and 12 form this figure and, on a torus, nodes 3, 6, 10, and 12 for this figure as well. By the simplicity of this diagram, it is relatively easy to read off the stabilizers by noting how the π-copy rule causes certain stabilizers to exist. Not much more work has to be done to convert this to the KLS form of the encoder. The grid-like structure of the codes in Figure 5 and Figure 6 are easy for anyone to see clear patterns, making this form canonical.

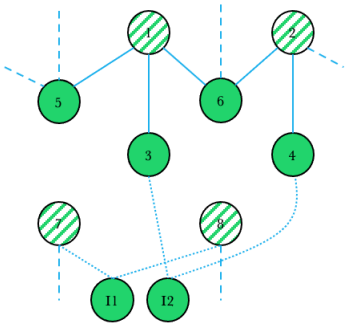


FIG. 5: The 2-by-2 Toric code in ZX calculus. The output nodes with the diagonal patterns are Hadamarded outputs while the solid nodes are not Hadamarded. Dashed edges represent edges that wrap around the torus. For example, the vertical dashed edge coming from node 5 meets node 7 and the dashed edge from node 2 meets node 5. The dotted edges are input-output edges.

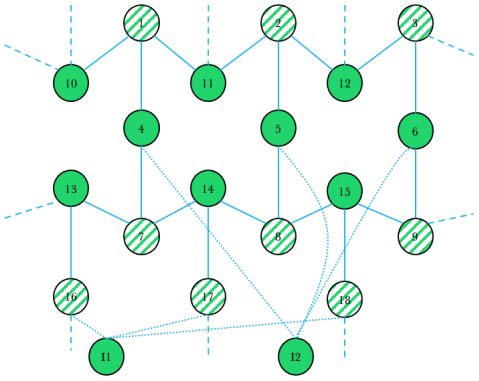


FIG. 6: The 3-by-3 Toric code in ZX calculus. The output nodes with the diagonal patterns are Hadamarded outputs while the solid nodes are not Hadamarded. Dashed edges represent edges that wrap around the torus. For example, the vertical dashed edge coming from node 10 meets node 16 and the dashed edge from node 3 meets node 10. The dotted edges are input-output edges.

For larger Toric codes and the surface codes, we use the ZX calculus software Quantomatic [25] to run through a simplification procedure (simproc) on the known ZX normal form [23] of a code. In this simproc, the main focus is on performing the bialgebra rule on internal nodes, so that, after running the simproc, all internal nodes will be removed from the diagram, leaving only input and output nodes. To write this simproc, we first make diagrammatically defined rules that would apply the bialgebra rule on an internal node. See the appendix

for an example of one of these rules.
Having defined these rules, the simproc can be written relatively simply as the following algorithm.

1. Use basic simplifications, by merging reds, merging greens, applying the red-copy rule, applying the green-copy rule, applying the Hopf rule, removing scalars, removing loops, or combining two Hadamards into the identity [14].
2. Change all reds into greens (with the appropriate Hadamard gates).
3. Apply one iteration of the bialgebra rule (any variation).
4. Apply step 1 again.
5. Loop through steps 3 and 4 until the diagram stops changing.

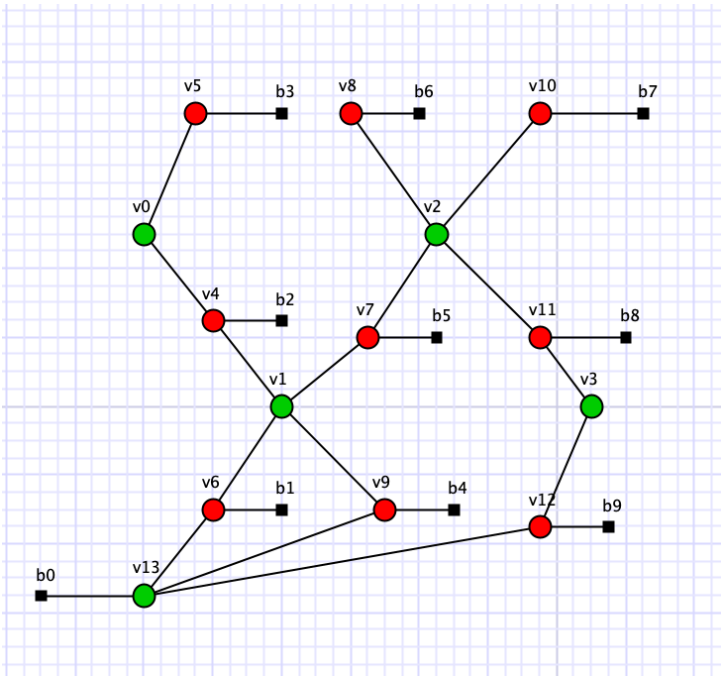
To illustrate the effectiveness of this simproc, we apply it on Eq. (12) from [23], as shown in Figure 7.
In diagram (b) of Figure 7, the graph's structure looks simpler than diagram (a), and it is easy to read off stabilizers from the graph using the same method of examining whether the output edges have Hadamards or not. By doing so, we can determine the stabilizers by examining each corner of the large rectangle in the figure.
Also, in diagram (a) of Figure 8, we have a more simplified version of (b) of Figure 7. And, in diagram (b) of Figure 8, we rearranged the boundary nodes into a grid-like structure, with each boundary node corresponding to the same one from the original ZX normal form. With all the internal nodes removed from the diagram, this new representation of the surface code can be quickly turned into KLS canonical form. Also, it is clearer how each output node is entangled with other output nodes in this diagram.

IV. ANOTHER DEFINITION OF EQUIVALENCE

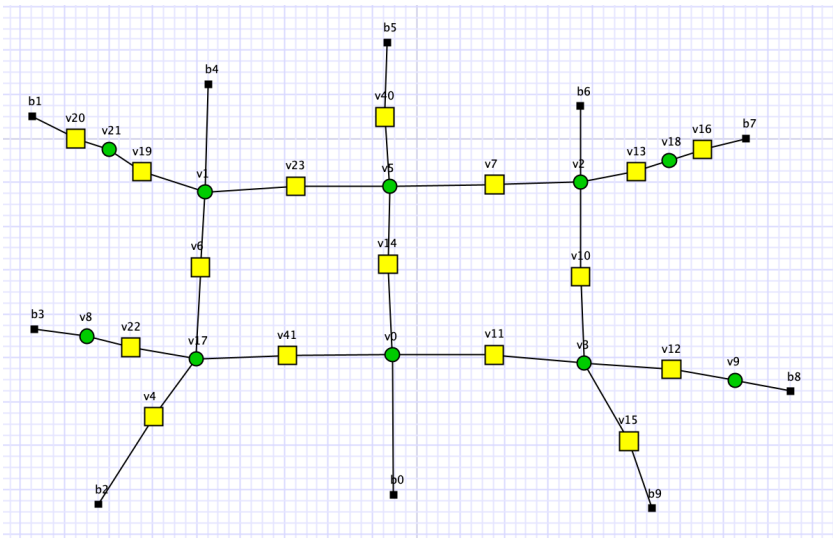
Previous works have examined the equivalence classes of graphs under local complementation. In Clifford codes, the presence of designated input and output vertices makes the definition of equivalence more exotic.
We now list five different operations which keep encoder graphs equivalent.

Definition D. The ZX diagrams for two Clifford codes $\mathcal{C}_1$ and $\mathcal{C}_2$ are *equivalent* if and only if the diagram for one of the codes can be reached by the other after a sequence of operations consisting of the following operations:

1. Local complementing about any vertex of the graph.
2. Permuting the output vertices.
3. Permuting the input vertices.
4. Performing linear operations on the adjacency matrix of input to output edges

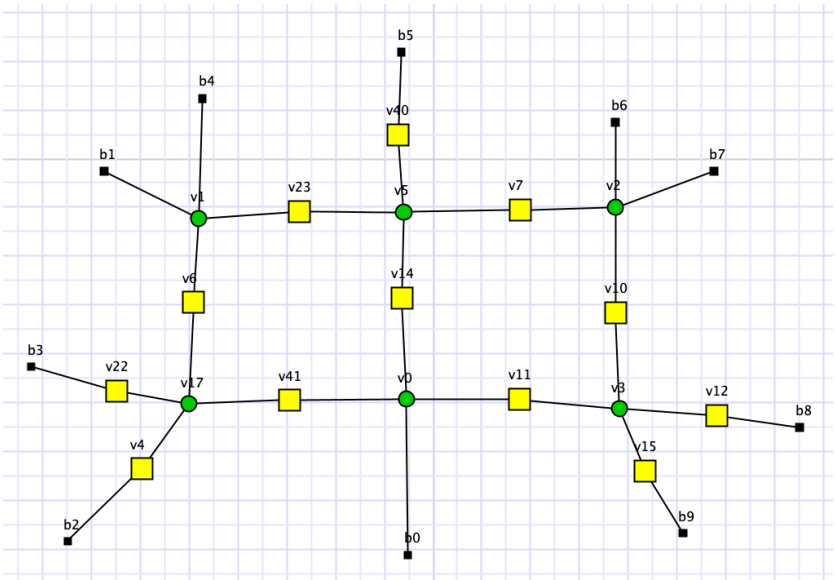


(a) The 3-by-3 surface code in ZX normal form.

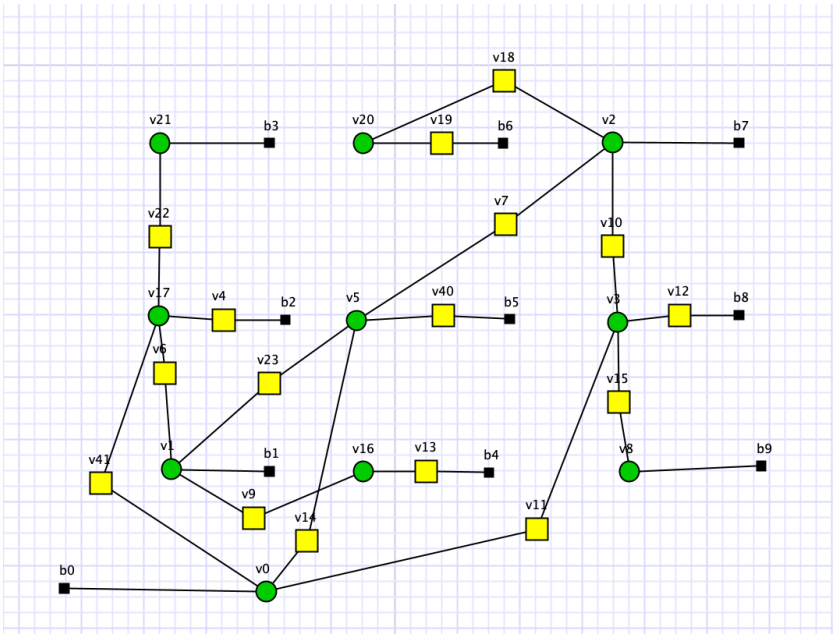


(b) The 3-by-3 surface code in a more canonical ZX form.

FIG. 7: The simproc, as described in the paper, is applied to the 3-by-3 surface code in ZX normal form in (a) to get to (b), after some rearranging of vertices and edges. The black boundary nodes in both parts represent inputs or outputs. The input node for diagram (b) is at the middle bottom.



(a) The 3-by-3 surface code in another form found by removing nodes using simplification rules.



(b) The 3-by-3 surface code in a grid-like form similar to (a) from Figure 7.

FIG. 8: More rearrangements of the 3-by-3 surface code. (a) is a slight variation on diagram (b) from Figure 7 while (b) has the output nodes arranged in a grid-like structure, with the boundary nodes corresponding to the same boundary nodes as the original ZX normal form diagram from Figure 7.

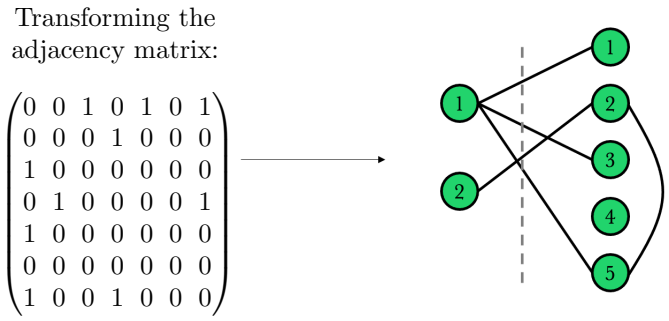


FIG. 9: In the adjacency matrix, the first row corresponds to the first input, the second row corresponds to the second input, and so on. After the inputs, the following row corresponds to the first output, and the other outputs follow.

5. Removing an input-input edge.

All of the operations in Definition D are reversible, so, if code $\mathcal{C}_1$ can be made equivalent to $\mathcal{C}_2$, the reverse is also true.

Operation 1, local complementation as in Definition B, is included to account for equivalence of encoder graphs based on their entanglement [26].

Two encoder diagrams should also be equivalent if the information they produce can be ordered differently to become the same. In this way, operations 2 and 3 reflect this, since connections among the vertices of the graph remain the same and these operations only change the order in which the information is inputted or outputted.

Operation 4, which consists of adding rows of the adjacency matrix between input and output vertices in modulo 2, preserves equivalence, by [21]. Lastly, operation 5 takes away a unitary operation from the input vertices, and doing so keeps the equivalence.

Note that this definition of equivalence does not allow two encoders to be in the same equivalence class if they only differ by an extra output (that is not connected to anything else). That is, if the two encoders differ by a quantum state, this definition of equivalence marks them as different. Therefore, this implies we focus on section Y of Figure 1, instead of section X.

As an example of these extra outputs/states, Figure 9. The output vertex labeled 4 is not connected to any input or output. It does not provide any more entanglement than if it was not present. For this reason, section X of Figure 1 is more useful for practical purposes.

V. SIMPLIFICATIONS

There are some simplifications that can be made using the above operations so we consider only encoder graphs that could possibly be non-equivalent.

Operations 3 and 4 of Definition D allow the input-to-output portion of the encoder diagram to be expressed in row-reduced echelon form (RREF). All encoder diagrams considered from here on are expressed in RREF form, as in the RREF rule from [21]. Therefore, the input vertices have corresponding output vertices that represent the pivots of the RREF. These particular output vertices are called pivot vertices.

Continuing from the RREF form of the encoder graph from the above paragraph, operation 2 from Definition D can be used to move the pivot vertices to be at the front of the line of the sequence of output vertices. In this way, the first output vertex can be made into the pivot vertex corresponding to the first input vertex, the second output vertex can be made into the pivot vertex corresponding to the second input vertex, and so on. Thus, these $n-k$ pivot vertices are fixed among the top of the output vertices. For brevity, the other k non-pivot output vertices are called free outputs.

In Definition D, no operation was included that affected local Clifford gates at the vertices. Therefore, this definition of equivalence neglects the presence of phase changing gates at vertices of the encoder's graph. This is because local Clifford gates change the qubits using a unitary operation but does not contribute to changes in entanglement of the qubits in any way. Therefore, for our purposes, we remove all local Clifford gates present at the vertices of the ZX diagram for the encoder.

Furthermore, to simplify the diagrams we draw, the incoming and outgoing edges (i.e. free edges from [21]) will be omitted. They are implied, since the diagrams for $(n-k)$-to-n encoders will be drawn with $n-k$ input vertices on the left side and n output vertices on the right side. Furthermore, for the internal edges that are included in the diagram, all edges have Hadamard gates as detailed in the Edge Rule from [21], but these gates will be omitted in our diagrams.

A further simplification, carried over from [21], is that graphs with pivot-pivot edges are omitted, since they can always be transformed into a graph without pivot-pivot edges using a sequence of local complementations.

As an example of the simplifications on the diagrams, as well as how the adjacency matrices transform into encoders, see Figure 9.

VI. TABULATIONS FROM CODE

To find patterns in the equivalence classes, we wrote code to take encoder graphs and assign them into disjoint sets.

In the written code, each encoder graph is turned into an integer based on the variable edges present in the graph. The variable edges of such a graph are the input-free edges, pivot-free edges, and free-free edges. These vari-

$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
1	3	11	40	185

(a) Number of equivalence classes for 2-to-n codes.

Rep:	0	1	3
Size:	1	6	9

(b) 2-to-3 codes equivalence classes and sizes.

Rep:	0	1	2	3	181	6	10	11	12	13	14
Size:	1	1	12	42	36	6	18	99	18	234	45

(c) 2-to-4 codes equivalence classes and sizes.

Rep:	0	1	3	8	72	9	73	11	75	12	76	13	77
Size:	1	3	4	18	27	126	297	60	144	18	27	396	972

Rep:	80	81	4753	82	530	594	83	4755	597	86	534	598	4758
Size:	54	702	108	378	108	540	2268	1080	2484	5184	168	5832	4896
Rep:	24	88	26	90	92	93	742	743	232	236	112	113	56	120
Size:	18	135	90	486	459	1188	1620	1152	108	414	54	1080	6	63

(d) 2-to-5 codes equivalence classes and sizes.

FIG. 11: (a) shows the number of equivalence classes for 2-to-n encoder graphs. (b-d) show the lexicographically smallest element (Rep) in each equivalence class, as well as the size of each of these classes, for the 2-to-3, 2-to-4, and 2-to-5 encoders.

able edges are those that can be changed, so it does not include input-pivot edges, input-input edges, or pivot-pivot edges. To assign a graph's adjacency matrix to an integer, each of the entries corresponding to a variable edge is assigned a distinct value of the form 2^i for a nonnegative integer i. Note that this adjacency matrix includes all vertices, so it is an $(2n-k)\times(2n-k)$ matrix. If the edge corresponding to 2^i is present, 2^i is added into the integer. Otherwise, it is not added in.

For example, in the 2-to-5 codes, the 7×7 adjacency matrix would look like the following:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & a_{14} & a_{13} & a_{12} \\ 0 & 0 & 0 & 1 & a_{11} & a_{10} & a_9 \\ 1 & 0 & 0 & 0 & a_8 & a_7 & a_6 \\ 0 & 1 & 0 & 0 & a_5 & a_4 & a_3 \\ a_{14} & a_{11} & a_8 & a_5 & 0 & a_2 & a_1 \\ a_{13} & a_{10} & a_7 & a_4 & a_2 & 0 & a_0 \\ a_{12} & a_9 & a_6 & a_3 & a_1 & a_0 & 0 \end{pmatrix}$$

The top-left 4×4 submatrix reflects the input-pivot edges, as well as the lack of input-input edges and pivot-pivot edges.

The entry labeled a_i is then weighted with the

value 2^i. So, if the only additional edges (besides the input-pivot edges) of the graph consisted of the edge between the second input and fifth output, the matrix would have $a_9 = 1$ and all other $a_i = 0$. Then, the integer representation is

$$\sum_{i=0}^{14} 2^i a_i = 512.$$

And, in the example from Figure 9, the integer representation would be $2^{14} + 2^{12} + 2^3 = 20488$.

We decide to put a_0 closest to the bottom right, the expand upward from there. The analogous 6×6 adjacency matrix for 2-to-4 codes would look like:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & a_8 & a_7 \\ 0 & 0 & 0 & 1 & a_6 & a_5 \\ 1 & 0 & 0 & 0 & a_4 & a_3 \\ 0 & 1 & 0 & 0 & a_2 & a_1 \\ a_8 & a_6 & a_4 & a_2 & 0 & a_0 \\ a_7 & a_5 & a_3 & a_1 & a_0 & 0 \end{pmatrix}$$

The disjoint set algorithm is useful for separating the whole set of possible encoder graphs into equivalence classes. In our code, we made the representative of each equivalence class, or disjoint set, the smallest integer representation among the encoder graphs. Note that this is not necessarily the canonical form of the equivalence class.

The code takes an integer representation, say n, of an encoder graph, performs one operation from Definition D on the encoder graph, then merges the disjoint sets of n and the integer representing the resulting encoder graph. Any possible operation is applied and merged with n's disjoint set.

When a local complementation is performed on a free output, it is possible that an input-pivot edge is removed. Furthermore, some input-input edges could be added. To fix this, we first employ operation 5 from Definition D to set all input-input edges to 0. Then, operations 3 and 4 are used to turn the submatrix representing the input-to-output adjacency matrix into RREF form. Operation 2 is used to put the pivots back into their fixed positions, so they once again correspond to their input vertices.

By looping the code to run through all integers in the set of representatives of the possible encoder graphs, all the encoder graphs are grouped into an equivalence class.

On the next page, the results of the code are shown. As shown in Figure 11(a), the number of equivalence classes increases quickly as the number of output vertices increases. Furthermore, in Figure 11(b-d), the equivalence classes show a variety of sizes, with many of the sizes in (c-d) divisible by 9.

Due to the nature of the weights of the edges in the adjacency matrix, it is also possible to find out which equivalence classes have a bi-

partite form. For example, in the 2-to-5 codes, since a bipartite form would need $a_0, a_1, \ldots, a_8$ to all equal 0, we look for the integer representations that are 0 mod 512.

By running a conditional statement in this way, we find the equivalence classes that have bipartite forms and write down the bipartite forms (as integer representations) for these classes.

For the 2-to-3 codes, we tabulate the following classes and the bipartite forms in these classes.

Rep.	Bipartite forms
0	0
1	4, 8
3	12

Similarly, for the 2-to-4 codes, we have the following table:

Rep.	Bipartite forms
0	0
2	32, 64, 128, 256
6	96, 384
10	160, 320
12	192, 288
14	224, 352, 416, 448, 480

And, for the 2-to-5 codes, we have the table shown in Figure 12. Note that, for 2-to-n encoders, the total number of bipartite encoders is 2^{2n-4}.

The reason we separated out the equivalence classes that have bipartite forms from those that do not is that bipartite forms are deemed more canonical. For the equivalence classes that have bipartite forms, a canonical form will be picked out from among these bipartite forms. The advantage of choosing a bipartite form is its simplicity. We can avoid including edges among output vertices in this way.

VII. EQUIVALENCE CLASSES WITH BIPARTITE FORM(S)

Consider an equivalence class containing a bipartite graph, in which the only edges are input-output edges. In these specific classes, one of the bipartite graphs is chosen as the canonical form due to its simplicity. The task now is to determine which one of these bipartite forms to pick.

First, consider a small case of 2-to-4 codes. The adjacency matrix of a bipartite form, taking into account the RREF and pivot simplifications from section IV, would look like:

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

By changing the red entries between 0s and 1s, there are 16 possible bipartite forms among all 2-to-4 codes.

Rep.	Bipartite forms
0	0
8	512, 1024, 2048, 4096, 8192, 16384
72	4608, 9216, 18432
80	5120, 6144, 8704, 10240, 16896, 17408
24	1536, 2560, 3072, 12288, 20480, 24576
88	5632, 6656, 9728, 11264, 12800, 13312, 13824, 18944, 19456, 20992, 22528, 23040, 25600, 26624, 27648
232	14848, 15360, 15872, 22016, 23552, 24064, 26112, 27136, 28160, 30208, 31232, 31744
112	7168, 10752, 14336, 17920, 21504, 25088
56	3584, 28672
120	7680, 11776, 19968, 29184, 29696, 30720, 32256

FIG. 12: Table of the representatives of all equivalence classes among 2-to-5 encoders that have bipartite forms, as well as a listing of the integer representation of these bipartite forms.

In some encoder diagrams, the graph can be split into separate parts; that is, some vertices are not entangled in any way with some other vertices.

Definition E. Suppose a graph $G = (V, E)$ can be separated into two groups of vertices V_1 and V_2 so that, for any vertex $a \in V_1$ and any vertex $b \in V_2$, there exists no path from a to b using edges in G. Then, graph G is a *disjoint* graph.

In the form of an adjacency matrix for 2-to-n, the graph is not disjoint if and only if there exists a column in which both entries are 1 and there exists no column in which both entries are 0. This is clear, since this means the two parts of the graph (one containing all input-output edges from the first input, the other containing all input-output edges from the second input) are entangled at some vertex, and there is no output vertex not connected to anything.

Claim F. Among all non-disjoint 2-to-4 bipartite graphs, there is only 1 distinct graph, up to equivalence through operations 2 through 4 from Definition D.

Proof. There are only 5 possible input-to-output adjacency matrices in this case:

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

Consider the first matrix above. Switching the third and fourth output vertices (corresponding to the third and fourth columns of the matrix) results in

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix},$$

which is the second matrix. From here, use operation 4 to replace the first row with the sum of the first and second rows modulo 2:

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

Permuting the outputs achieves the third and fourth matrices. Lastly, starting from the above matrix, use operation 4 to replace the second row with the sum of the current first and second rows modulo 2 to find

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

This can be permuted to give the fifth matrix. Analogous sequences of operations can bring any of the other matrices to another, so all 5 of the graphs are equivalent, showing the claim. □

Using Claim F, the canonical form for the equivalence class that contains these 5 adjacency matrices can be chosen to be

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

First, this graph shares the property of having the least number of edges. Next, to distinguish between the four matrices with the least number of edges, we take the one that has more edges reserved for the first input and first free output, which would be the one shown above. By writing one of these input-to-output adjacency matrices into the full 6×6 matrix, we can deduce the integer representation (as described in Section V) of the chosen matrix to be $2^8 + 2^7 + 2^6 = 448$. From the table of bipartite forms for 2-to-4 codes (in Section V), this means these matrices are in the same equivalence class as 14. Also, the fifth matrix from the proof of Claim F has representation 480. See Figure 13 for the graphs of the elements 14, 448, and 480.

Now, we present a general method of simplifying a bipartite $2 \times n$ input-to-output adjacency matrix.

Claim G. Consider a 2-to-n encoder graph equivalent to some bipartite form. It is also equivalent to a bipartite form where the two inputs are both connected to at most $\left\lfloor \frac{n}{2} - 1 \right\rfloor$ of the same free outputs.

Proof. In this proof, we only consider encoders that are equivalent to a bipartite form.

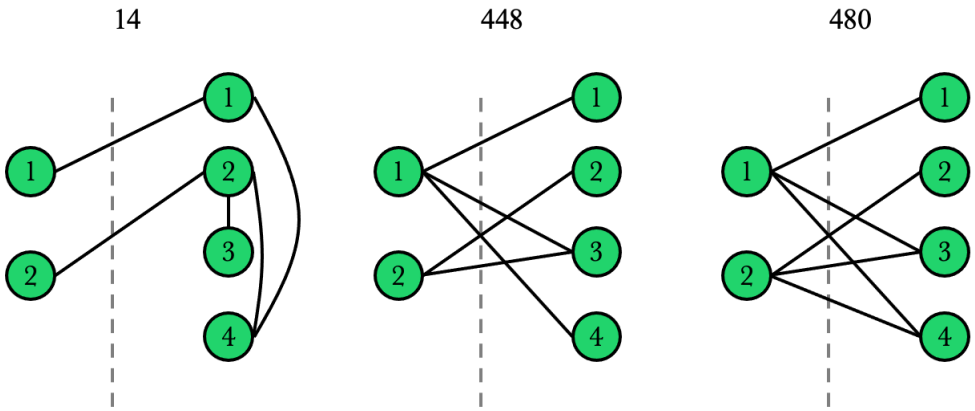


FIG. 13: In the 2-to-4 equivalence class with lexicographically smallest element of 14, 448 and 480 are possible bipartite forms with all edges being input-output edges. 448 is chosen as canonical due to it having fewer edges. Note that 14 is also bipartite if it is partitioned into one group with outputs 1 and 2 and one group with the inputs and the remaining outputs.

For the sake of contradiction, suppose all bipartite forms of this equivalence class have at least $\lfloor \frac{n}{2} \rfloor$ shared free outputs. In an input-to-output adjacency matrix, this would look like

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

The first two columns are fixed to be input-pivot edges, as usual. If there are at least $\lfloor \frac{n}{2} \rfloor$ shared free outputs, the other $n-2$ columns must have a majority of columns containing two 1's. In this example, 3 out of 5 columns contain two 1's.

However, using operation 4 from Definition D, the top row can be replaced with the sum of the top and bottom row modulo 2.

Note that this means all the free outputs that were shared by both inputs have their edges with the first input disconnected, so at least $\lfloor \frac{n}{2} \rfloor$ columns do not have two 1's.

Furthermore, after the operation, the first column cannot possibly have two 1's, so one additional column does not have two 1's. The example matrix above turns into

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

We can rearrange output vertices to bring back the pivots. The following is thus equivalent

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

Thus, the maximum number of columns with

two 1's is now $n-1-\lfloor \frac{n}{2} \rfloor$. However,

$$n-1-\left\lfloor \frac{n}{2} \right\rfloor < \left\lfloor \frac{n}{2} \right\rfloor,$$

so we reach a contradiction, since there are now less than $\lfloor \frac{n}{2} \rfloor$ shared free outputs in an equivalent bipartite form.
Therefore, the claim holds. □

Claim G demonstrates that we can choose a bipartite form that has a relatively small number of shared free outputs. In fact, if the top row is the horizontal vector $\mathbf{a}$ and the bottom row is the horizontal vector $\mathbf{b}$, by linear operations, there are only 3 possibilities of unordered combinations of two vectors in the rows. It could be $(\mathbf{a}, \mathbf{b}), (\mathbf{a}+\mathbf{b}, \mathbf{b}), (\mathbf{a}+\mathbf{b}, \mathbf{a})$. Then, we can choose which of these bipartite forms has the least number of shared free outputs and thus minimize this number.

Now, we classify operations 2, 3, and 4 from Definition D as *easy* operations, while operations 1 and 5 are *hard* operations.
In an attempt to show the uniqueness of the bipartite forms in an equivalence class, we conjecture the following:

Conjecture H. In an equivalence class with bipartite forms, all such bipartite forms can be transformed from one to another using only easy operations.

One straightforward approach starts by assuming for the sake of contradiction that two bipartite graphs, G_1 and G_2, are equivalent even though they cannot be transformed from one to another using only easy operations. Then, we can write down partial traces between all pairs of vertices of one bipartite graph. This quantifies the entanglement of the whole graph. If the partial traces between all pairs of vertices of the other bipartite graph are different, then the entanglement is evidently different and we would reach a contradiction.

VIII. OTHER EQUIVALENCE CLASSES

There are also equivalence classes with no bipartite graphs, and it is less intuitive to determine which graphs to call canonical forms. Call such equivalence classes *lacking*. From Definition E, disjoint encoder graphs cannot be transformed into a non-disjoint encoder graph using operations from Definition D. Therefore, if there exists a disjoint encoder graph in an equivalence class, all graphs in the equivalence class are disjoint.
In the 2-to-4 codes, the lacking equivalence classes, using the tables from Section V, have representatives 1, 3, 181, 11, and 13.
The diagrams for these classes reveal that 181, 11, and 13 are non-disjoint, which are shown in Figure 14.
Using the code from Section V, we were able to determine that the equivalence class of 181 does not contain any elements with 0 output-

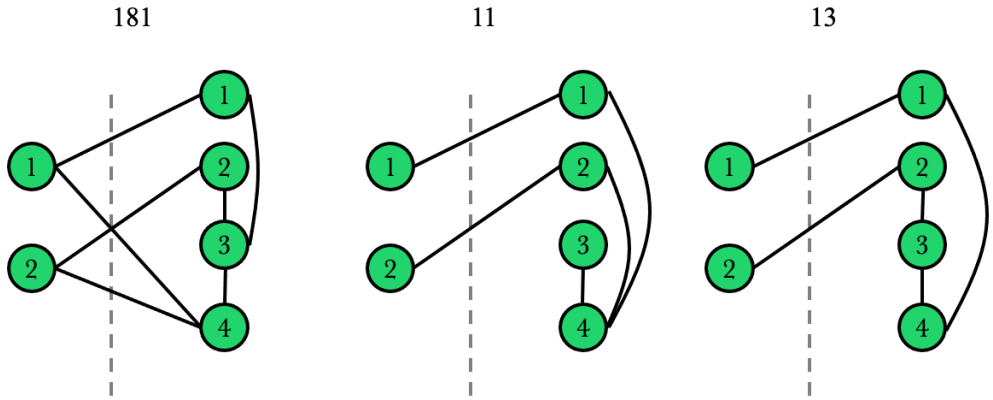
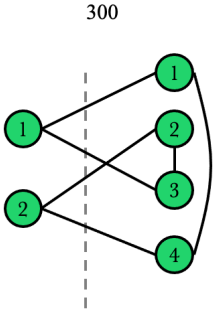


FIG. 14: The lexicographically smallest representatives of the 2-to-4 non-disjoint and lacking equivalence classes.

output edges or 1 output-output edge. Instead, the smallest number of output-output edges is 2, and an example of such a graph is 300, shown below.



The symmetry of 300's graph suggests the graph for 181 is also symmetrical. Indeed, the inputs in 181's graph share symmetrical connections, and their pivots share symmetrical connections. Both pivots, outputs 1 and 2, have an edge with output 3. Both inputs 1 and 2 share an edge with output 4, and they also both have an input-pivot edge.
Since 300 makes the symmetrical connections more apparent, and it has less output-output edges, it is deemed more canonical.

Similarly, from Figure 14, we see that 11 and 13 also have such symmetries, and their outputs can be rearranged to make it more apparent.

IX. CONCLUSION

This paper presented our work on detailing the canonical forms and structures of select Toric and surface codes and tabulated results from Java code that determined the representatives sizes of equivalence classes for ZX diagrams within sector X of Figure 1. Among these equivalence classes, we also analyzed

those that contained bipartite forms among the 2-to-4 codes.

The work done on Toric and surface codes ties in with the larger goal of finding explainable canonical forms given the stabilizers of an error-correcting code. The advantage of the Toric and surface codes presented in this paper are their structural simplicity. As seen by the 3-by-3 surface code, the number of internal nodes in the ZX normal form is reduced to 0 using the simproc we provided. Future works can extend these simplifications (i.e. of removing internal nodes) to other CSS codes and find patterns among the structures of the codes to see if newer definitions of canonicity could yield efficient transformations from the stabilizer formalism to the ZX calculus, as well as from the ZX calculus to the stabilizer formalism.

Furthermore, by extending the scope of the research to consider permuting outputs as giving equivalent codes, the tabulated results in this paper will provide future works a basis to determining patterns among equivalence class sizes and representatives, as well as a method for determining such sizes and representatives. Extending the definition of equivalence to allow for permutation of outputs is physically significant as these permutations do not affect the amount of entanglement the input qubits go through, and thus the ordering of output qubits could be considered as a non-fundamental aspect of a quantum error-correcting code.

X. ACKNOWLEDGMENTS

The work described in this paper and the writing of this paper were done by Alexander M. Li.

We would like to thank the MIT PRIMES-USA program for the opportunity to conduct this research. Thank you to my mentor Andrey Boris Khesin for providing valuable advice and guidance throughout this research. Thank you to Jonathan Lu for support in creating the diagrams.

[1] H. Xu, C. Li, G. Wang, H. Wang, H. Tang, A. R. Barr, P. Cappellaro, and J. Li, Two-photon interface of nuclear spins based on the optonuclear quadrupolar effect, Physical Review X **13**, 011017 (2023).

[2] A. K. Frederiksen, The quantum computer exists, but is not all that powerful (2023), accessed: August 15, 2023.

[3] D. L. Chandler, Engineers discover a new way to control atomic nuclei as "qubits" (2023), accessed: August 15, 2023.

[4] M. A. Nielsen and I. Chuang, Quantum computation and quantum information (2002).

[5] P. W. Shor, Scheme for reducing decoherence in quantum computer memory, Physical review A **52**, R2493 (1995).

[6] A. Steane, Multiple-particle interference and quantum error correction, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **452**, 2551 (1996).

[7] E. Knill, R. Laflamme, R. Martinez, and C. Negrevergne, Benchmarking quantum computers: The five-qubit error correcting code, Physical Review Letters **86**, 5811 (2001).

[8] A. Y. Kitaev, Quantum error correction with imperfect gates, in *Quantum communication, computing, and measurement* (Springer, 1997) pp. 181–188.

[9] D. Gottesman, *Stabilizer codes and quantum error correction* (California Institute of Technology, 1997).

[10] M. Van den Nest, J. Dehaene, and B. De Moor, Graphical description of the action of local clifford transformations on graph states, Physical Review A **69**, 022316 (2004).

[11] A. T. Hu and A. B. Khesin, Improved graph formalism for quantum circuit simulation, Physical Review A **105**, 022432 (2022).

[12] B. Coecke and R. Duncan, Interacting quantum observables, in *International Colloquium on Automata, Languages, and Programming* (Springer, 2008) pp. 298–310.

[13] B. Coecke and R. Duncan, Interacting quantum observables: categorical algebra and diagrammatics, New Journal of Physics **13(4)**, 043016 (2011).

[14] M. Backens, The zx-calculus is complete for stabilizer quantum mechanics, New Journal of Physics **16**, 093021 (2014).

[15] T. Peham, L. Burgholzer, and R. Wille, Equivalence checking of quantum circuits with the zx-calculus, IEEE Journal on Emerging and Selected Topics in Circuits and Systems **12**, 662 (2022).

[16] A. Cowtan and S. Majid, Quantum double aspects of surface code models, Journal of Mathematical Physics **63**, 042202 (2022).

[17] J. van de Wetering, Constructing quantum circuits with global gates, New Journal of Physics **23**, 043015 (2021).

[18] R. D. 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).

[19] N. de Beaudrap and D. Horsman, The zx calculus is a language for surface code lattice surgery, Quantum **4**, 218 (2020).

[20] A. Kissinger and J. van de Wetering, Reducing the number of non-clifford gates in quantum circuits, Physical Review A **102**, 022406

(2020).

[21] A. B. Khesin, J. Z. Lu, and P. W. Shor, Graphical quantum clifford-encoder compilers from the zx calculus (2023), arXiv:2301.02356 [quant-ph].

[22] J. Huang, S. M. Li, L. Yeh, A. Kissinger, M. Mosca, and M. Vasmer, Graphical css code transformation using zx calculus, arXiv preprint arXiv:2307.02437 (2023).

[23] A. Kissinger, Phase-free zx diagrams are css codes (... or how to graphically grok the surface code), arXiv preprint arXiv:2204.14038 (2022).

[24] A. Y. Kitaev, Fault-tolerant quantum computation by anyons, Annals of physics **303**, 2 (2003).

[25] Quantomatic, https://quantomatic.github.io/, accessed: August 5, 2023.

[26] J. C. Adcock, S. Morley-Short, A. Dahlberg, and J. W. Silverstone, Mapping graph state orbits under local complementation, Quantum **4**, 305 (2020).

Appendix

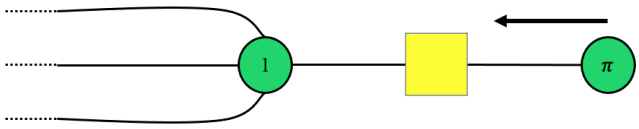
In the main text, we provided the 2-by-2 and 3-by-3 Toric codes in ZX calculus. Here, we provide a more detailed description of the methodology used to determine the structure of the 3-by-3 Toric code.

After placing down the output nodes as in Figure 4, we expect some of the output edges to contain Hadamard gates. Suppose the output edge onto vertex 1 has a Hadamard. This implies that applying the stabilizer $Z_1Z_4Z_{10}Z_{11}$ would result in sliding a Z gate from the end of the output edge, through the Hadamard (which converts the Z gate to an X gate), then through vertex 1 itself. By the π-copy rule [14], the X gate, which is an X node with phase π, copies itself onto the edges (excluding the output edge) connected to vertex 1. Diagrammatically, applying Z_1 onto the vertex 1 is shown in Figure 15.

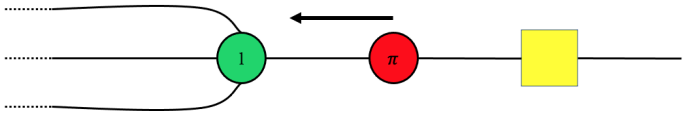
Similarly, while continuing the assumption that output node 1 has a Hadamard, if we instead applied the stabilizer $X_1X_3X_{10}X_{16}$, we slide an X gate from the end of the output edge, through the Hadamard (which converts the X gate to a Z gate), then onto vertex 1. By the spider rule [14], the Z gate, which is a phase π green node, merges with vertex 1, a phase 0 green node. This results in vertex 1 gaining a phase of π.

By the preceding paragraphs, the behavior of the Z and X gates on a Hadamarded output node is understood. The analogous behavior occurs on a non-Hadamarded output node by switching all the colors used in the processes above.

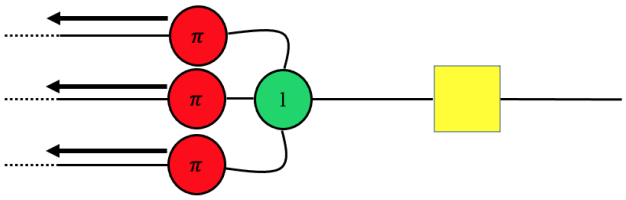
To determine all of the edges in the 3-by-3 Toric code in Figure 3 and Figure 4, we consider these processes of applying the stabilizers onto the output nodes. Note that all of the internal edges among nodes in the diagram must be Hadamarded edges so that the spider rule cannot be applied to merge mul-



(a) A Z gate, denoted by the green π is slid onto the output edge containing a Hadamard gate, denoted by the yellow square.



(b) Passing through the Hadamard gate, the Z gate turns into an X gate, or red π.



(c) By the π-copy rule, the X gate passes through vertex 1 and copies itself onto each of the other edges connected to vertex 1.

FIG. 15: The process by which a Z gate passes through a node (vertex 1) with phase 0. The Z gate is pushed through the output edge connected to output node 1. Note that an analogous diagram holds depending on the number of non-output edges connected to node 1.

tiple nodes into one. To simplify our work, we set the Hadamarded output nodes to be $1, 2, 3, 7, 8, 9$, and $16, 17, 18$. Then, by stabilizer $Z_1 Z_4 Z_{10} Z_{11}$, the nodes 4, 10, and 11 gain phase π from their Z gates while node 1 will cause a π-copy rule to move X gates onto the internal edges connected node 1. Since all internal edges are Hadamarded, moving the X gates through the Hadamards will result in Z gates. If these Z gates went to any nodes other than nodes 4, 10, and 11, the stabilizer would not have kept the configuration the same. Therefore, the Z gates must arrive at only nodes 4, 10, and 11. This works because the πs from these Z gates cancel with the πs already at the nodes. Thus, the only internal edges to node 1 are from nodes 4, 10, and 11.

Using similar reasoning, we can deduce the rest of the internal edges among the output nodes. Furthermore, to determine the logical

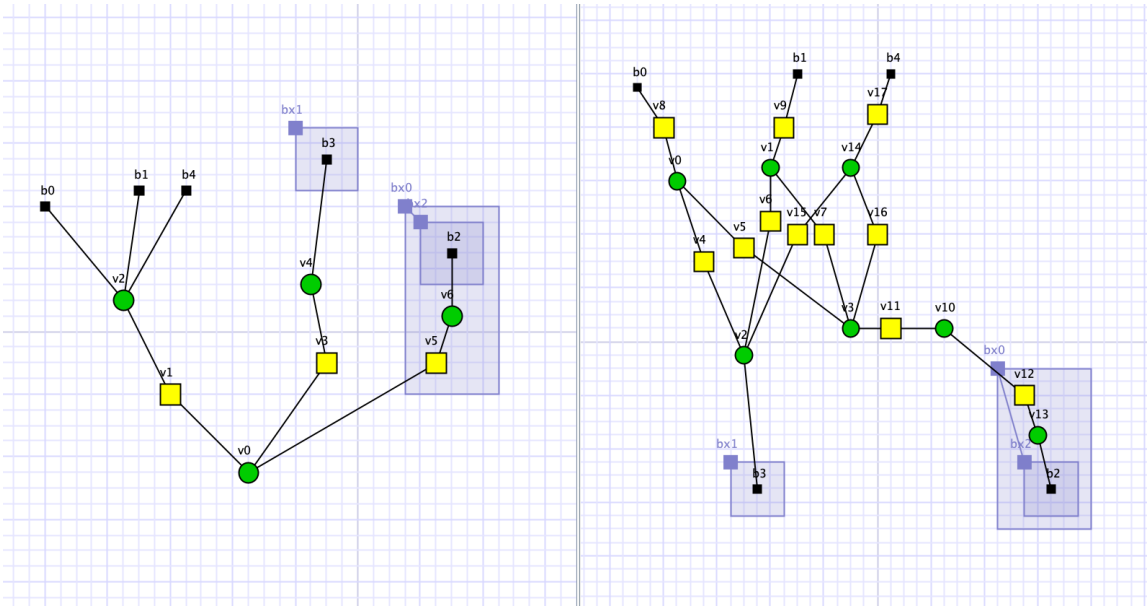


FIG. 16: Version of the bialgebra rule on an internal node v0 (the bottom-most node in the left side diagram) in the case with 3 boundary nodes attached to v2 (as shown in the left side diagram).

operators (to connect the input nodes to), we look for sets of nodes that, when any stabilizer is applied, keep the input node at phase 0.

In the simplification of the 3-by-3 surface code, we used a simproc based on multiple variations of the bialgebra rule. One of these rules is shown in Figure 16. The variations on the rule shown in Figure 16 have different numbers of boundary nodes attached to the vertex v2 in the left diagram. Note that the purple boxes are !-boxes, which denote any nonnegative integer number of copies of the objects inside the !-box. For our purposes, we will only be applying the bialgebra rule from left to right. The motivation behind this is that the resultant form will be able to make simplifications when v2 (the green node connected to three black boundary vertices) from the left diagram is split into multiple nodes in the right diagram.

Declaration of Academic Integrity

The participating team declares that the paper submitted is comprised of original research and results obtained under the guidance of the instructor. To the team's best knowledge, the paper does not contain research results, published or not, from a person who is not a team member, except for the content listed in the references and the acknowledgment. If there is any misinformation, we are willing to take all the related responsibilities.

Names of team members:
Alexander Li

Signatures of team members:


Name of the instructor:
Andrey Boris Khesin

Signature of the instructor:


Date:
August 19, 2023