GNarsil: Splitting Stabilizers into Gauges

Oskar Novak
Wyant College of Optical Sciences
University of Arizona
Tucson, USA 85721
E-mail: onovak@arizona.edu

Narayanan Rengaswamy
Department of Electrical and Computer Engineering
University of Arizona
Tucson, USA 85721
E-mail: narayananr@arizona.edu

Abstract*—Quantum subsystem codes have been shown to improve error-correction performance, ease the implementation of logical operations on codes, and make stabilizer measurements easier by decomposing stabilizers into smaller-weight gauge operators. In this paper, we present two algorithms that produce new subsystem codes from a "seed" CSS code. They replace some stabilizers of a given CSS code with smaller-weight gauge operators that split the remaining stabilizers, while being compatible with the logical Pauli operators of the code. The algorithms recover the well-known Bacon-Shor code computationally as well as produce a new $[\![9,1,2,2]\!]$ rotated surface subsystem code with weight-3 gauges and weight-4 stabilizers. We illustrate using a $[\![100,25,3]\!]$ subsystem hypergraph product (SHP) code that the algorithms can produce more efficient gauge operators than the closed-form expressions of the SHP construction. However, we observe that the stabilizers of the lifted product quantum LDPC codes are more challenging to split into small-weight gauge operators. Hence, we introduce the subsystem lifted product (SLP) code construction and develop a new $[\![775,124,20]\!]$ code from Tanner's classical quasi-cyclic LDPC code. The code has high-weight stabilizers but all gauge operators that split stabilizers have weight 5, except one. In contrast, the LP stabilizer code from Tanner's code has parameters $[\![1054,124,20]\!]$. This serves as a novel example of new subsystem codes that outperform stabilizer versions of them. Finally, based on our experiments, we share some general insights about non-locality's effects on the performance of splitting stabilizers into small-weight gauges.

I. Introduction

Quantum error correction is vital for quantum computers to achieve their full potential. The technique requires identifying the location of errors on a quantum computer without disturbing the delicate superposition states of the qubits involved in the computation. This is done by measuring stabilizers, which are quantum parity checks on different subsets of qubits, that help elucidate the locations of errors through an error syndrome. However, if measuring the stabilizers involves a high number of qubits, then the entangling measurements of the process pose the risk of creating additional errors.

Subsystem codes may help alleviate this issue [1], [2]. These are codes with gauge operators or operators on virtual qubits that do not carry quantum information. If designed well, then the eigenvalue of a high-weight stabilizer can be obtained as a product of the eigenvalues of several lower-weight gauge operators that can be measured more easily. Generally, these gauge operators form a non-abelian group, meaning that the order of measurement matters. The ordering can be chosen carefully during the process of *gauge fixing*, where changes to the code can be made mid-computation to help ease measuring stabilizers or even implementing logical operations [3]. Recent work has even translated these gauge-fixing insights of subsystem codes into the ZX calculus [4]. There are several examples of subsystem codes in the literature. The prototypical example of a subsystem code is the Bacon-Shor code [2]. Most constructions have topological or geometrically local properties, which makes finding the gauge group an intuitive process [5]. Closed-form expressions exist for constructing generalized Bravyi-Bacon-Shor or Subsystem Hypergraph Product (SHP) Codes [6]. There are also some computational methods for forming a set of gauge generators out of a set of Pauli operators via Gram-Schmidt orthogonalization [7], and computational search methods for finding the optimal subsystem code from a set of two-qubit measurement operators [8].

However, there does not exist an algorithm that allows one to input a stabilizer code and derive a subsystem code directly from it, especially one that determines gauge operators that compose to form stabilizers of the input code. In recent years, there has been tremendous progress in constructing quantum low-density parity-check (QLDPC) stabilizer codes with optimal code parameters. Such an algorithm can leverage these advances in stabilizer codes and potentially decompose their stabilizers into smaller-weight *local* gauge checks. For example, a good Lifted Product (LP) code can have stabilizer weights of 8 or even larger than 10, which involve long-range connections between qubits. Hence, the LDPC property alone does not make these constructions practical.

In this work, we present two algorithms capable of deriving subsystem codes from a "seed" CSS stabilizer code and present non-trivial examples of subsystem codes found by our algorithms. The algorithms identify a $[\![9,1,2,2]\!]$ rotated surface subsystem code whose stabilizer weights are still 4 but gauge weights are 3, albeit with some non-locality. In contrast, the subsystem surface code known in the literature [9] uses more qubits and has stabilizers of weight 6. Next, we demonstrate a modified SHP code that reduces the weight of gauge operators needed to produce a stabilizer compared to the closed-form expressions in [6]. We observe that it is in general difficult to decompose stabilizers of the LP construction into small-weight gauge operators. Hence, we introduce an extension to the SHP construction that we call the *Subsystem Lifted-Product (SLP)* codes, which can have

superior parameters compared to the Lifted Product stabilizer code constructed from the same base matrix. As an illustrative example, we produce a $[\![775, 124, 20]\!]$ SLP code from Tanner's classical quasi-cyclic LDPC code, whereas the corresponding LP code has parameters $[\![1054, 124, 20]\!]$. The code has high-weight stabilizers but the gauge operators to produce the stabilizer are weight-5, except one of high weight. It remains to be seen if the high-weight nature of stabilizers or some of the gauges is intrinsic to the SLP construction.

The paper is organized as follows: Section II introduces the necessary background to discuss the results in this work. Section III reviews the binary symplectic representation of stabilizer and subsystem codes. Section IV provides our algorithms for splitting stabilizers into gauges. Section V discusses some non-trivial examples found using our algorithms, including the SHP and SLP constructions. Section VI concludes the paper with thoughts about future work.

II. Background

A. Representing Pauli Operators as Binary Vectors

Given the vectors $a = [a_1, ..., a_n]$, $b = [b_1, ..., b_n] \in \mathbb{F}_2^n$, we define the Hermitian operator

$$E(a,b) \coloneqq i^{a_1 b_1} X^{a_1} Z^{b_1} \otimes ... \otimes i^{a_n b_n} X^{a_n} Z^{b_n}, \tag{1}$$

where X and Z are the standard Pauli operators and $i \coloneqq \sqrt{-1}$. We can define the n-qubit Pauli group $\mathcal{P}_n$ as

$$\mathcal{P}_n \coloneqq \langle i^\alpha E(a,b) \ : \ a, b \in \mathbb{F}_2^n; \alpha \in \{0,1,2,3\} \rangle . \tag{2}$$

This is also known as the Heisenberg-Weyl Group $HW_N, \ N = 2^n$. The standard symplectic inner product in $\mathbb{F}_2^{2n}$ is defined as [10]:

$$\begin{aligned} \langle [a,b], [a',b'] \rangle_s &\coloneqq a'b^T + b'a^T \pmod 2 \\ &= [a,b] \ \mathbf{\Omega} \ [a',b']^T \pmod 2, \end{aligned} \tag{3}$$

where the matrix $\mathbf{\Omega} = \begin{bmatrix} \mathbf{0} & \boldsymbol{I}_n \\ \boldsymbol{I}_n & \mathbf{0} \end{bmatrix}$ is the symplectic form in $\mathbb{F}_2^{2n}$. We notice that two operators $E(a,b)$, $E(a',b')$ commute if and only if $\langle [a,b], [a',b'] \rangle_s = 0 \pmod 2$. Thus, we see that a vector $[a,b] \in \mathbb{F}_2^{2n}$ is isomorphic to an operator $E(a,b)$ via the map $\gamma : \mathcal{P}_n / \langle i^\alpha \boldsymbol{I}_{2^n} \rangle \to \mathbb{F}_2^{2n}$ defined by

$$\gamma(E(a,b)) \coloneqq [a,b]. \tag{4}$$

Thus, without loss of generality, we can represent n-qubit Pauli operators as binary vectors. The *weight* of an n-qubit Pauli operator is the number of qubits on which it applies a non-identity Pauli operator, e.g., the operator $X_1 \otimes X_2 \otimes I_3$ can be described as $[\,1\,1\,0,\,0\,0\,0\,]$, and its weight is 2.

B. Quantum Stabilizer Codes

A stabilizer group $\mathcal{S} \in \mathcal{P}_n$ is an abelian subgroup of the Pauli group that does not contain $-I$. The corresponding *stabilizer code* is defined by:

$$\mathcal{Q} \coloneqq \left\{ |\psi\rangle \in \mathbb{C}^{2^n} : S|\psi\rangle = |\psi\rangle \ \ \forall \ S \in \mathcal{S} \right\}. \tag{5}$$

A stabilizer code with n physical qubits and m independent stabilizer generators can encode $k = n - m$ logical qubits. The logical Pauli operators of the code come from the normalizer of $\mathcal{S}$ in $\mathcal{P}_n$, which is also its centralizer, defined as

$$\mathcal{N}(\mathcal{S}) \coloneqq \{ U \in \mathcal{P}_n : [U,S] = 0 \ \forall \ S \in \mathcal{S} \}, \tag{6}$$

where $[U,S] \coloneqq US - SU$ is the commutator of U and S. Here, notice that $\mathcal{S} \subset \mathcal{N}(\mathcal{S})$. Finally, the *minimum distance*, d, of the code is given by the minimum weight of any Pauli operator in $\mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$, or the lowest weight of any logical operator. Thus, we denote the parameters of a quantum stabilizer code as $[\![n,k,d]\!]$. Finally, a CSS code is a stabilizer code whose stabilizer $\mathcal{S} = \langle \gamma^{-1}(\boldsymbol{H}_X), \gamma^{-1}(\boldsymbol{H}_Z) \rangle$, $\boldsymbol{H}_X \boldsymbol{H}_Z^T = \mathbf{0}$, where $\boldsymbol{H}_X$, $\boldsymbol{H}_Z$ are classical binary parity check matrices, and the map γ^{-1} is applied to each row of $\boldsymbol{H}_X$, $\boldsymbol{H}_Z$.

C. Subsystem Codes

A subsystem code is a quantum error-correcting code that splits the code space $\mathcal{C} = A \otimes B$ into the logical subspace, A, and a gauge subspace, B, that does not carry any logical information [1]. The Hilbert space of a subsystem code can be written as $\mathcal{H} = \mathcal{C} \oplus \mathcal{C}^\perp = A \otimes B \oplus \mathcal{C}^\perp$. The gauge subspace is supported on r gauge qubits such that, given n physical qubits and m independent stabilizers, the number of logical qubits is $k = n - m - r$. Thus, a subsystem code with these parameters and code distance d can be written as an $[\![n,k,r,d]\!]$ subsystem code. A subsystem code is defined by its gauge group

$$\mathcal{G} \coloneqq \langle iI, \mathcal{S}, X_1', Z_1', \ldots, X_r', Z_r' \rangle \subset \mathcal{P}_n, \tag{7}$$

where X_i', Z_i' are the Pauli operators for the i-th gauge qubit, and $\mathcal{S}$ is the stabilizer group of the code.

We can use $\mathcal{G}$ to define the stabilizers of the code:

$$\mathcal{S} \coloneqq \mathcal{Z}(\mathcal{G}) = \mathcal{C}(\mathcal{G}) \cap \mathcal{G}, \tag{8}$$

where $\mathcal{Z}(\mathcal{G})$ is the center of $\mathcal{G}$, and $\mathcal{C}(\mathcal{G})$ is the centralizer of $\mathcal{G}$ in $\mathcal{P}_n$. We define the bare logical operators $\mathcal{L}_b$ as:

$$\mathcal{L}_b \coloneqq \mathcal{C}(\mathcal{G}) \setminus \mathcal{G} = \mathcal{C}(\mathcal{G}) \setminus \mathcal{S}. \tag{9}$$

These are generated by k pairs of anti-commuting Pauli operators such that $[\mathcal{G}, \mathcal{L}_b] = 0$. These operators only act non-trivially on the subspace A. We also have the dressed logical operators $\mathcal{L}$, $\mathcal{L}_b \subset \mathcal{L}$, where:

$$\mathcal{L} = \mathcal{C}(\mathcal{S}) \setminus \langle i\mathcal{I} \rangle . \tag{10}$$

In other words, the set of dressed logical operators is the set of bare logical operators multiplied by an operator from $\mathcal{G} \setminus \langle i\mathcal{I} \rangle$.

Finally, we can define the code distance d such that

$$d \coloneqq \min_{P \in \mathcal{L}} |P|, \tag{11}$$

i.e., the minimum weight of a dressed logical operator.

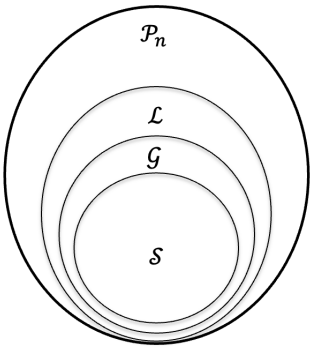

Fig. 1: A diagram describing the relation of the Gauge group $\mathcal{G}$ to the other important subsets of $\mathcal{P}_n$ for a subsystem code.

III. Binary Description of Subsystem Codes

A. Stabilizer Code Construction via Binary Matrices

We can describe a stabilizer code as a $2n \times 2n$ matrix $\boldsymbol{U}$ which has the following form [11], [12], [10]:

$$\boldsymbol{U} = \begin{bmatrix} \mathcal{L}_X \\ \mathcal{S} \\ \mathcal{L}_Z \\ \mathcal{S}' \end{bmatrix}, \tag{12}$$

where its rows $\langle \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{2n} \rangle$ span $\mathbb{F}_2^{2n}$. We are overloading notation to represent both the group as well as a binary matrix whose rows represent the generators of the group. Here, $\mathcal{L}_X$ and $\mathcal{L}_Z$ are the binary matrices that represent the generators of the logical Pauli operators. The sub-matrix $\mathcal{S}'$, otherwise known as the "destabilizer" [11], is added so that $\boldsymbol{U}$ has full rank. For a stabilizer code, we choose $\mathcal{S}'$ such that:

$$\boldsymbol{U}\boldsymbol{\Omega}\boldsymbol{U}^T = \boldsymbol{\Omega} \pmod 2. \tag{13}$$

This means that each row vector in $\mathcal{S}'$ is chosen so that it anti-commutes with only one stabilizer generator inside $\mathcal{S}$. In other words, $\boldsymbol{U}$ is a binary symplectic matrix [12].

Theorem 1. *Let $\boldsymbol{U}$ be a binary matrix constructed as in* (12)*. If $\boldsymbol{U}$ satisfies* (13)*, then $\boldsymbol{U}$ describes a valid stabilizer code.*

Proof: The L.H.S. of (13) is our definition of the symplectic inner product in (3) extended to a matrix. Since $\boldsymbol{\Omega}$ has a single non-zero element per row, each row of $\boldsymbol{U}$ anti-commutes with exactly one other row, which is n rows below (or above) it. Such pairs of rows are called *symplectic pairs*. Since an $[\![n, k, d]\!]$ stabilizer code has $n - k$ stabilizers, the symplectic pairs of rows in $\mathcal{L}_X$ must be in the rows of $\mathcal{L}_Z$. This means that the codes' logical operators commute with the operators represented as binary vectors in $\mathcal{S}$, $\mathcal{S}'$. Thus, satisfying (13) verifies all necessary conditions for $\boldsymbol{U}$ to represent a valid stabilizer code. ∎

B. Subsystem Code Construction via Binary Matrices

To construct a subsystem code using the methods of the previous section, we can take a binary matrix $\boldsymbol{U}$ constructed as in (12), and choose $2r$ rows of it to become our gauge generators, and $\mathcal{L}_X$, $\mathcal{L}_Z$ become our bare logical operators.

It should be noted that, generally, one can replace these $2r$ rows with other rows and still have a valid subsystem code, as long as $\boldsymbol{U}$ remains symplectic. Also, one is free to choose logical operators to become gauge operators.

Theorem 2. *Let $\boldsymbol{U}$ be a $2n \times 2n$ binary matrix that has the construction of* (12) *except that $2r$ of its rows have possibly been altered. We call this altered matrix $\boldsymbol{U}'$. Then $\boldsymbol{U}'$ represents a valid subsystem code if:*

$$\boldsymbol{U'}\boldsymbol{\Omega}\boldsymbol{U'}^T = \boldsymbol{\Omega} \pmod 2. \tag{14}$$

Proof: Since only the $2r$ rows that have been altered are those promoted to gauge operators, the rest of the rows of the matrix on the R.H.S. of (14) are identical to the rows in the same positions in (12). This means that the bare logical operators only anti-commute with their respective symplectic pairs and commute with the operators that generate the gauge group. Furthermore, this also means that since the leftover stabilizers in $\mathcal{S}$ only anti-commute with the leftover operators in $\mathcal{S}'$, the operators of $\mathcal{S}$ indeed form $\mathcal{Z}(\mathcal{G})$. Finally, as long as the previous conditions are met, then (14) guarantees that $\boldsymbol{U}'$ represents a valid subsystem code. ∎

Appendix A proves our algorithm's ability to find all suitable representations for the gauge generators.

IV. GNarsil Algorithms to Find Gauge Operators

Here we will use *gauge generators* to describe the non-stabilizer generators of the gauge group. In practice, a subsystem code's utility comes from measuring the product of a set of lower-weight gauge operators that yields the same information as measuring a higher-weight stabilizer [5]. Using the binary matrix constructions of the previous section, we can find low-weight gauge operators for CSS codes whose composition produces the code's stabilizers up to a remaining gauge operator that is not a gauge generator. Let us call this remaining gauge operator the *residual operator* for that stabilizer. We refer to the minimum (Pauli) weight over all residual operators as the *residual weight*. For a gauge decomposition to be useful, the residual weight must be less than the weight of the decomposed stabilizer. To achieve this, we propose two algorithms. The first algorithm finds the set of r gauge generators that decompose stabilizers with the least residual weight and then adds back stabilizers so that $n = k - r - m$ is still satisfied. We describe the first *GNarsil* algorithm, which "cuts" stabilizers into gauges, in Algorithm 1.

We also note that by bypassing the anti-commutation conditions, gauge operators that are not necessarily gauge generators may be found such that they decompose a stabilizer with lower residual weight. We also provide a second *GNarsil* algorithm for this case in Algorithm 2.

V. Examples Found by Our Algorithms

A. $[\![9, 1, 4, 3]\!]$ Bacon-Shor Code

Algorithm (1) was able to find the $[\![9, 1, 4, 3]\!]$ Bacon-Shor code from the $[\![9, 1, 3]\!]$ Shor (stabilizer) code. This required

Algorithm 1 *GNarsil* 1: Gauge Generators to Split Stabilizers

Input: $\boldsymbol{U} = \begin{bmatrix} \mathcal{L}_X \\ \mathcal{S} \\ \mathcal{L}_Z \\ \mathcal{S}' \end{bmatrix}$, $\{i_1, i_2, \ldots, i_r\} \subseteq \{k+1, k+2, \ldots, n\}$: indices of rows of $\boldsymbol{U}$ to be replaced with X-type gauge generators, desired Pauli weight w for gauge generators

Initialization:
- $\mathcal{G}_X \leftarrow \emptyset,\ \mathcal{G}_Z \leftarrow \emptyset$
- validXGauges $\leftarrow \emptyset,$ validZGauges $\leftarrow \emptyset$
- maxSize $\leftarrow$ max. number of gauge candidates to consider;
- $S_{\text{Xtargets}} \leftarrow$ indices of rows of $\boldsymbol{U}$ that are X-stabilizers;
- $S_{\text{Ztargets}} \leftarrow$ indices of rows of $\boldsymbol{U}$ that are Z-stabilizers;
- gaugesPerStab $\leftarrow$ # of gauge generators per stabilizer;
- Xops $\leftarrow$ weight-w (row) vectors in $\{0,1\}^n$ appended with $\mathbf{0}$ at the end to make length $2n$ (for X-gauges);
- Zops $\leftarrow$ weight-w (row) vectors in $\{0,1\}^n$ appended with $\mathbf{0}$ at the front to make length $2n$ (for Z-gauges);
- $\boldsymbol{V} \leftarrow \boldsymbol{U}(1:(n+k))$ (i.e., remove $\mathcal{S}'$ from $\boldsymbol{U}$)
- ▷ Find X, Z gauge generators

for i in 1:size(Xops) **do**
  if size(validXGauges) $\geq$ maxSize **then**
    break
  end if
  if $\text{Xops}(i)\,\boldsymbol{\Omega}\,\boldsymbol{V}^T = \mathbf{0} \pmod 2$ and
    $\text{rank}\left(\begin{bmatrix} \boldsymbol{V} \\ \text{Xops}(i) \end{bmatrix}\right) > \text{rank}(\boldsymbol{V})$ and
    $\text{rank}\left(\begin{bmatrix} \text{validXGauges} \\ \text{Xops}(i) \end{bmatrix}\right) > \text{rank}\,(\text{validXGauges})$ **then**
    validXGauges $\leftarrow$ validXGauges $\cup$ Xops(i)
  end if
end for
if validXGauges $== \emptyset$ **then**
  $w \leftarrow w+1$
  Regenerate Xops and Zops
  if $w \geq n$ **then**
    stop ▷ Algorithm Fails
  else
    go to Line 3
  end if
end if
XgChoices $\leftarrow$ NCHOOSEK(validXGauges , gaugesPerStab)
  ▷ List of all combinations of gaugesPerStab elements in validXGauges
for i in $\mathcal{S}_{\text{Xtargets}}$ **do**
  for j in 1:size(XgChoices) **do**
    resultantGauge $\leftarrow \boldsymbol{U}(i) + \text{XgChoices}(j) \pmod 2$
    residualWeight$(j) \leftarrow$ HammingWeight(resultantGauge)
  end for
  minIndex $\leftarrow$ argmin(residualWeight)
  $\mathcal{G}_X \leftarrow \mathcal{G}_X \cup \text{XgChoices(minIndex)}$
  if size$(\mathcal{G}_X) \geq r$ **then**
    Drop all rows from $r+1$ (if they exist)
    break
  end if
end for
Clear the rows $\{i_1, i_2, \ldots, i_r\}$ and $\{n+i_1, n+i_2, \ldots, n+i_r\}$ from $\boldsymbol{U}$ to add gauge generators
$\boldsymbol{U}([i_1, i_2, \ldots, i_r]) \leftarrow \mathcal{G}_X$
Repeat from Line 3 for Zops by appropriately replacing variables; add Zops(i) to validZGauges only if $\text{HammingWeight}\big(\text{Zops}(i)\,\boldsymbol{\Omega}\,\mathcal{G}_X^T\big) = 1$
  ▷ Zops(i) anti-commutes with exactly one X-gauge
$\boldsymbol{U}([n+i_1, n+i_2, \ldots, n+i_r]) \leftarrow \mathcal{G}_Z$
Replace unused destabilizers in $\mathcal{S}'$ such that $\boldsymbol{U}$ is symplectic
Return: $\boldsymbol{U}$, codeDistance($\boldsymbol{U}$)

Algorithm 2 *GNarsil* 2: Gauge Operators to Split Stabilizers

Input: $\boldsymbol{U} = \begin{bmatrix} \mathcal{L}_X \\ \mathcal{S} \\ \mathcal{L}_Z \\ \mathcal{S}' \end{bmatrix}$, $\{i_1, i_2, \ldots, i_r\} \subseteq \{k+1, k+2, \ldots, n\}$: indices of rows of $\boldsymbol{U}$ that will be removed to add gauge operators

Initialization:
- $\mathcal{G}_X \leftarrow \emptyset,\ \mathcal{G}_Z \leftarrow \emptyset$
- validXGauges $\leftarrow \emptyset,$ validZGauges $\leftarrow \emptyset$
- maxSize $\leftarrow$ max. number of gauge candidates to consider;
- $S_{\text{Xtargets}} \leftarrow$ indices of rows of $\boldsymbol{U}$ that are X-stabilizers;
- $S_{\text{Ztargets}} \leftarrow$ indices of rows of $\boldsymbol{U}$ that are Z-stabilizers;
- gaugesPerStab $\leftarrow$ # of gauge operators per stabilizer;
- numXGauges $\leftarrow$ gaugesPerStab $\times$ # of X stabilizers;
- numZGauges $\leftarrow$ gaugesPerStab$\times$# of Z stabilizers;
- Xops $\leftarrow$ weight-w vectors in $\{0,1\}^n$ appended with $\mathbf{0}$ at the end to make length $2n$ (for X-gauges);
- Zops $\leftarrow$ weight-w vectors in $\{0,1\}^n$ appended with $\mathbf{0}$ at the beginning to make length $2n$ (for Z-gauges);
- $\boldsymbol{V} \leftarrow \boldsymbol{U}(1:(n+k))$ (i.e., remove $\mathcal{S}'$ from $\boldsymbol{U}$)
- ▷ Find X, Z gauge operators

for i in 1:size(Xops) **do**
  if size(validXGauges) $\geq$ maxSize **then**
    break
  end if
  if $\text{Xops}(i)\,\boldsymbol{\Omega}\,\boldsymbol{V}^T = \mathbf{0} \pmod 2$ and
    $\text{Xops}(i) \notin \mathcal{L}_X$ **then**
    validXGauges $\leftarrow$ validXGauges $\cup$ Xops(i)
  end if
end for
if validXGauges $== \emptyset$ **then**
  $w \leftarrow w+1$
  if $w \geq n$ **then**
    stop ▷ Algorithm Fails
  else
    go to Line 3
  end if
end if
XgChoices $\leftarrow$ NCHOOSEK(validXGauges , gaugesPerStab)
  ▷ List of all combinations of gaugesPerStab elements in validXGauges
for i in $\mathcal{S}_{\text{Xtargets}}$ **do**
  for j in 1:size(XgChoices) **do**
    resultantGauge $\leftarrow \boldsymbol{U}(i) + \text{XgChoices}(j) \pmod 2$
    residualWeight$(j) \leftarrow$ HammingWeight(resultantGauge)
  end for
  minIndex $\leftarrow$ argmin(residualWeight)
  $\mathcal{G}_X \leftarrow \mathcal{G}_X \cup \text{XgChoices(minIndex)}$
end for
Remove rows $\{i_1, i_2, \ldots, i_r\}$ and $\{n+i_1, n+i_2, \ldots, n+i_r\}$ from $\boldsymbol{U}$; replace with numXGauges and numZGauges empty rows, respectively
$\boldsymbol{U}([i_1, i_2, \ldots, i_{\text{numXGauges}}]) \leftarrow \mathcal{G}_X$
Repeat from Line 3 for Zops by appropriately replacing variables; add Zops(i) to validZGauges only if $\text{HammingWeight}\big(\text{Zops}(i)\,\boldsymbol{\Omega}\,\mathcal{G}_X^T\big) \geq 1$
  ▷ Zops(i) may anti-commute with more than one X-gauge
$\boldsymbol{U}([n+i_1, n+i_2, \ldots, n+i_{\text{numZGauges}}]) \leftarrow \mathcal{G}_Z$
Return: $\boldsymbol{U}$, codeDistance($\boldsymbol{U}$)

first replacing two of the weight-2 Z-stabilizers with linearly independent weight-6 Z-stabilizers from the span of the weight-2 Z-stabilizers before running the algorithm. Finding this prototypical subsystem code example with our algorithm

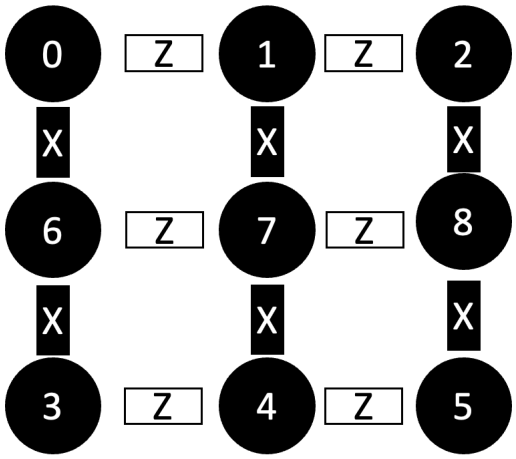

Fig. 2: The Bacon-Shor code and its weight-2 gauge operators.

points to its validity. We note here that although six gauge operators are shown in Fig. 2, not all are linearly independent. The last two are the product of the linearly independent gauge operators and stabilizers. The specific pre-processing of the stabilizers above is a key step that can naturally be generalized to other CSS concatenated codes. Specifically, the inner code stabilizers can be multiplied to produce large-weight stabilizers, thereby allowing us to make low-weight gauges from the original (inner code) stabilizers.

B. $[\![9, 1, 2, 2]\!]$ *Rotated Surface Subsystem Code*

In this example, we present a novel subsystem version of the $[\![9, 1, 3]\!]$ rotated surface code found by Algorithm (1). The code has weight-4 stabilizer generators and weight-3 gauge operators, as shown in Fig. 3. We summarize the set of gauge operators, both generators and dependent ones, in Table I. Using the dependent gauges and gauge generators GX_1, GZ_1, we can measure all of the weight 4 stabilizers shown above at the cost of the dependent gauge operators being not fully local and a small loss of distance. The well-known subsystem version of the surface code in the literature [5], [9] has weight-6 stabilizers instead of the usual weight-4 and requires significantly more qubits, e.g., for lattice size $L = 3$ the code uses $3L^2 + 4L + 1 = 40$ qubits. It is not unreasonable to consider the code in Fig. 3, but it will be interesting to explore whether there is an intermediate subsystem surface code between these two solutions that still has distance 3.

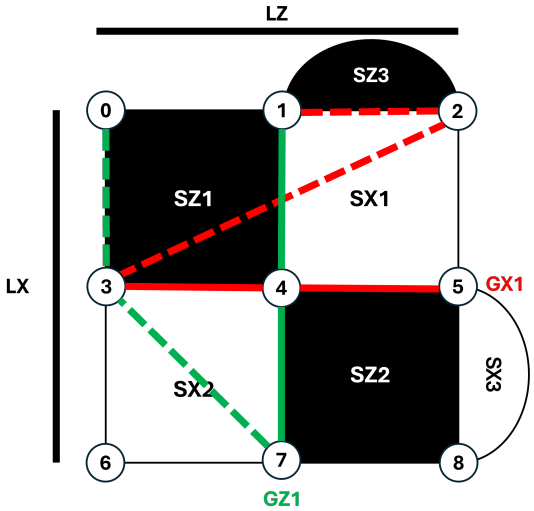

Fig. 3: The rotated surface subsystem code found by Algorithm 1. The two red (resp. green) operators are the X-gauges (resp. Z-gauges), all weight-3 (also see Table I). The product of the pair of red gauge operators, $X_1X_2X_3$ and $X_3X_4X_5$, gives SX_1, and the product of the pair of green operators gives SZ_1. Stabilizers SX_2, SZ_2 can be obtained by multiplying GX_1, GZ_1 by the respective stabilizers to find the respective dependent gauges. GX_2, GZ_2 are not shown here.

C. $[\![100, 25, 3]\!]$ *Subsystem Hypergraph Product (SHP) Code*

Using Algorithm (2), we present a $[\![100, 25, 3]\!]$ SHP code built from a $[10, 5]$ linear code [13] with parity-check matrix

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{15}$$

We construct the SHP code with the following definitions [6]:

$$\begin{aligned} \mathcal{G}_X &:= (\boldsymbol{H} \otimes \boldsymbol{I}_n), \\ \mathcal{G}_Z &:= (\boldsymbol{I}_n \otimes \boldsymbol{H}), \\ \mathcal{L}_X &:= (\boldsymbol{I}_n \otimes \boldsymbol{G}), \\ \mathcal{L}_Z &:= (\boldsymbol{G} \otimes \boldsymbol{I}_n), \\ \mathcal{S}_X &:= (\boldsymbol{H} \otimes \boldsymbol{G}), \\ \mathcal{S}_Z &:= (\boldsymbol{G} \otimes \boldsymbol{H}), \end{aligned} \tag{16}$$

where $\boldsymbol{G}$ is the generator matrix of the code defined by the parity check matrix $\boldsymbol{H}$, i.e, $\boldsymbol{G}\boldsymbol{H}^T = 0$. Thus, in general, the SHP construction in this form gives an $[\![n^2, k^2, d]\!]$ code with n, k, d being the respective values for the classical code defined by the parity check matrix $\boldsymbol{H}$. Here, the key observation is that although the stabilizers for this code are weight-12, our algorithm finds a decomposition that only requires weight-4 operators with a residual weight of 0 (resp. 5) for

List of $[\![9, 1, 2, 2]\!]$ subsystem code gauge operators	
Gauge Identifier	Gauge Operator
GX_1	$X_3X_4X_5$
GX_2	$X_3X_4X_7$
GX_{1d}	$X_1X_2X_3$
GX_{2d}	$X_5X_6X_7$
GZ_1	$Z_1Z_4Z_7$
GZ_2	$Z_4Z_5Z_8$
GZ_{1d}	$Z_0Z_3Z_7$
GZ_{2d}	$Z_1Z_5Z_8$

TABLE I: Gauge operators of the $[\![9, 1, 2, 2]\!]$ rotated surface subsystem code. GX_1, GX_2, GZ_1, GZ_2 are the independent gauge generators. The letter 'd' in subscripts indicates dependent gauge operators that are products of the gauge generators and stabilizers. The stabilizers are shown in Fig. 3.

the X-stabilizers (resp. Z-stabilizers). This is in contrast to the gauge operators found from the closed-form expression in (16), which are all weight-4, requiring residual weights of 6 and 16 for X- and Z-stabilizers respectively. Thus, not only does our algorithm find more gauge operators than the closed-form expression, it also finds gauge operators that decompose the given stabilizers more efficiently than the closed-form operators. This decomposition makes this high-rate code far easier to implement. We also note that high-weight stabilizers are common for the SHP construction, as shown by the $[\![49, 16, 3]\!]$ SHP code [6] built from the $[7, 4, 3]$ Hamming code with stabilizer weights 12 and 16, respectively. The parity check matrix for the $[7, 4, 3]$ Hamming code is given below.

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{17}$$

Hence, our algorithm is likely useful to determine more efficient implementations of SHP codes in general.

D. Subsystem Lifted-Product (SLP) Codes

As a natural extension of hypergraph product codes, lifted-product (LP) CSS codes [14] are constructed from the hypergraph product of base parity-check matrices $\boldsymbol{A}$ and $\boldsymbol{A}^*$, which is then lifted over the ring $R = \mathbb{F}_q(G)$, where G is some group. The resulting parity-check matrices are as follows:

$$\begin{aligned} \boldsymbol{H}_X &= [\boldsymbol{A} \otimes \boldsymbol{I}, \boldsymbol{I} \otimes \boldsymbol{A}], \\ \boldsymbol{H}_Z &= [\boldsymbol{I} \otimes \boldsymbol{A}^*, \boldsymbol{A}^* \otimes \boldsymbol{I}]. \end{aligned} \tag{18}$$

where $\boldsymbol{A}^*$ is the conjugate transpose of $\boldsymbol{A}$. This construction is typically to obtain good quantum LDPC codes, but its sparsity ends up producing non-local stabilizer operators. When LDPC codes are input into (18), and the resulting $\boldsymbol{H}_X, \boldsymbol{H}_Z$ are used as input for Algorithm 2, the algorithm is able to find several gauge operators for these codes. However, none of them are useful since the residual weight will greatly exceed the weight of the stabilizers themselves. This fact seems to stem from the sparse nature of the stabilizers: in order to commute with them, the gauge operators must also be sparse, which in turn requires a larger amount of residual weight to cover the spread of the stabilizers. The precise understanding of this situation for LP codes is an interesting future direction.

The SLP Construction: To find a lifted code with a better gauge decomposition, one is tempted to use the SHP construction in (16) as a natural extension of LP codes. We will refer to this as the SLP construction. By employing the base parity check-matrix $\boldsymbol{A}$ along with its (base) generator matrix $\boldsymbol{G}_{\boldsymbol{A}}$ into (16) and then lifting the construction by a circulant of size L, we obtain an $[\![Ln^2, Lk^2, D]\!]$ subsystem code, where n, k are the respective values defined by the base parity-check matrix $\boldsymbol{A}$. Here, $\boldsymbol{A}\boldsymbol{G}_{\boldsymbol{A}}^* = 0$, which implies that $\mathcal{G}_X \mathcal{S}_Z^* = 0$. This lifting procedure yields codes with potentially larger distances than a typical SHP code.

1) Comparing the SHP and SLP Constructions: For the most direct comparison, we choose a binary base matrix:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \tag{19}$$

where the generator matrix of (19) is given by:

$$\boldsymbol{G}_{\boldsymbol{A}} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}. \tag{20}$$

Placing into (16), this yields a $[\![9, 1, 3]\!]$ SHP code, which is simply the Bacon-Shor code. For the SLP construction, we interpret (19) as the matrix of powers of monomials corresponding to $L = 2$ circulant matrices. Thus, we define our new base matrix as:

$$\boldsymbol{B} = \begin{bmatrix} 1 & x & x \\ x & x & 1 \end{bmatrix}. \tag{21}$$

Clearly, Eqn. (20) interpreted in the same fashion is no longer a valid solution, so we must use another generator matrix. It can be shown that the following matrix is a valid generator matrix for the base matrix $\boldsymbol{B}$:

$$\boldsymbol{G}_{\boldsymbol{B}} = \begin{bmatrix} 1+x & 1+x & 0 \\ 1+x & 0 & 1+x \\ x & 0 & 1 \end{bmatrix}. \tag{22}$$

This yields a $[\![18, 2, 2]\!]$ SLP code. The SLP code here has the same rate as the SHP code but with more physical qubits and a small loss in distance. However, we will see other cases where the distance fairs favorably compared to an LP code with the same base matrix. We now look at a few non-trivial examples of the SLP construction and their performance.

2) $[\![27, 12, 2]\!]$ SLP code: Given $L = 3$, the $[\![27, 12, 2]\!]$ SLP code can be constructed by the following base matrix:

$$\boldsymbol{A} = \begin{bmatrix} 1 + x + x^2 & 1 + x & x \end{bmatrix}, \tag{23}$$

along with the base generator matrix of the code:

$$\boldsymbol{G}_{\boldsymbol{A}} = \begin{bmatrix} x^2 & x & 1 \\ x & x^2 & x \\ 1 & 0 & 1 + x + x^2 \end{bmatrix}. \tag{24}$$

After inserting these into (16) and lifting the construction, we find that the Pauli weight of the code's stabilizers is 18, and the Pauli weight of the gauge operators is 6. We also find that the gauge operators in the closed-form construction decompose the stabilizers with a residual weight of 9, given three weight-6 gauge operators as input. However, our algorithm is able to decompose the stabilizers with residual weights of $4, 6, 8$ for the X- and Z-stabilizers, an improvement over the closed-form gauge operator decomposition. Finally, we also note that the rate of the SLP construction is $\frac{4}{9}$, which is greater than the rate of the $[\![39, 12, 2]\!]$ LP code constructed from the same base matrix, $\frac{4}{13}$. Note that this is also while preserving the distance of the code.

3) $[\![775, 124, 20]\!]$ *SLP code:* For our final example, we give the SLP code constructed by Tanner's $(3, 5)$ QC-LDPC code with $L = 31$. Here, we use the following base matrix [15]:

$$\boldsymbol{B} = \begin{bmatrix} x & x^2 & x^4 & x^8 & x^{16} \\ x^5 & x^{10} & x^{20} & x^9 & x^{18} \\ x^{25} & x^{19} & x^7 & x^{14} & x^{28} \end{bmatrix}. \tag{25}$$

As shown by Smarandache, and then Chimal-Dzul, Lieb, and Rosenthal [16], [17], a generator matrix for (25) can be constructed using the matrix:

$$\boldsymbol{G_B} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & 0 \\ u_{21} & u_{22} & u_{23} & 0 & u_{25} \\ f & f & 0 & 0 & 0 \\ f & 0 & f & 0 & 0 \\ f & 0 & 0 & f & 0 \\ f & 0 & 0 & 0 & f \end{bmatrix},$$

$$\begin{aligned} u_{11} &= x^{28} + x^{25} + x^{18} + x^{16} + x^5 + x, \\ u_{12} &= x^{23} + x^{22} + x^{20} + x^{17} + x^7 + x^4, \\ u_{13} &= x^{29} + x^{25} + x^{21} + x^{12} + x^5 + x, \\ u_{14} &= u_{25} = x^{28} + x^{18} + x^{16} + x^{14} + x^9 + x^8, \\ u_{21} &= x^{27} + x^{24} + x^{19} + x^{11} + x^{10} + x^2, \\ u_{22} &= x^{30} + x^{28} + x^{26} + x^{18} + x^{16} + x^6, \\ u_{23} &= x^{20} + x^{14} + x^9 + x^8 + x^7 + x^4, \\ f &= \frac{x^{31} - 1}{x - 1} = x^{30} + \cdots + x + 1. \end{aligned} \tag{26}$$

We find that the code has stabilizer weights of 120, 310, and 465 and gauge operator weights of 5. Due to the large number of physical qubits, this code is beyond the practical scale that our algorithm can handle. Our algorithm is suited for small- to medium-sized codes due to its exponential memory complexity, but this decomposition by GNarsil may be achievable with a sufficiently large amount of memory. Our SLP example demonstrates that the SLP construction can be quite advantageous if a generator matrix can be found for a certain base matrix. For the same base matrix (25), an LP($\boldsymbol{B}$,$\boldsymbol{B}^*$) construction yields a $[\![1054, 124, 20]\!]$ code, which trails the SLP version both in rate. If a circulant form generator matrix is found for a base matrix, then the resulting SLP code will have a higher rate than its LP counterpart with the same code dimension. No relationship between the distance of the two constructions has been formalized, but it seems that the SLP code will at least have the same distance as its LP counterpart. This makes the SLP construction for a given base matrix a very attractive construction whenever possible.

VI. Conclusion

In this paper, we have introduced a new set of algorithms, which we call *GNarsil*, for deriving subsystem codes from an input "seed" stabilizer code. We have demonstrated that these algorithms not only recover well-known examples of subsystem codes but also find interesting new ones, such as a novel version of the rotated surface subsystem code and a new SHP code that is more efficient than the closed-form construction in [6]. We also reported that we could not find useful gauge decompositions of LP codes due to the highly non-local structure of the stabilizers. However, by using our new SLP construction, we can construct codes with excellent parameters that also have promising stabilizer decomposition properties, which our algorithms can improve. As noted, this also depends on finding a generator matrix for the base matrix in circulant form, which is not always guaranteed to exist.

We foresee these algorithms becoming useful tools for finding subsystem versions of stabilizer codes that may prove easier to implement due to the stabilizers' decomposition into smaller-weight gauge operators. However, it is clear that this problem remains complex, as the solution spaces for useful gauge generators are sparse, making it likely that our algorithms are optimal for this case, even though they are exponentially complex in memory. Thus, our algorithms work best for small- to medium-sized codes. Improving our algorithms' performance will most likely require specializing them for certain code constructions. This would allow us to exploit code symmetries to find optimal gauge operators.

We also foresee that the measure of residual weight can be useful for understanding the properties of good subsystem codes and the symmetries that they may possess. Finally, we wish to extend this work into looking at the relationship between fault-tolerant operations on a stabilizer code and its derived subsystem code(s) by extending tools such as the LCS algorithm [10]. This can potentially be approached by observing how new gauge operators found by our algorithm change the structure of logical gates for our code from the seed stabilizer code. Insights into this relationship may be useful for the development of logical operations on Floquet codes [18], at least the ones with parent subsystem codes [19].

Acknowledgements

We thank Nithin Raveendran and Bane Vasić at the University of Arizona for their insights into LP and quasi-cyclic (QC) LDPC codes, respectively, especially about finding circulant form generator matrices for QC-LDPC codes. Finally, we would like to thank M. Sohaib Alam of the Quantum Artificial Intelligence Laboratory at NASA Ames Research Center for his insights on subsystem codes and our algorithms.

The work of N. R. was partially supported by the National Science Foundation under Grant no. 2106189.

References

[1] D. Kribs, R. Laflamme, and D. Poulin, "Unified and generalized approach to quantum error correction," *Physical review letters*, vol. 94, no. 18, p. 180501, 2005. [Online]. Available: https://arxiv.org/abs/quant-ph/0412076

[2] D. Bacon, "Operator quantum error-correcting subsystems for self-correcting quantum memories," *Physical Review A*, vol. 73, no. 1, p. 012340, 2006. [Online]. Available: https://arxiv.org/abs/quant-ph/0506023

[3] N. P. Breuckmann, "Quantum subsystem codes: Their theory and use," 2011.

[4] 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. [Online]. Available: https://arxiv.org/abs/2307.02437

[5] O. Higgott and N. P. Breuckmann, "Subsystem codes with high thresholds by gauge fixing and reduced qubit overhead," *Physical Review X*, vol. 11, no. 3, p. 031039, 2021. [Online]. Available: https://arxiv.org/abs/2010.09626
[6] M. Li and T. J. Yoder, "A numerical study of Bravyi-Bacon-Shor and subsystem hypergraph product codes," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2020, pp. 109–119. [Online]. Available: https://arxiv.org/abs/2002.06257
[7] M. M. Wilde, "Logical operators of quantum codes," *Physical Review A*, vol. 79, no. 6, p. 062322, 2009. [Online]. Available: https://arxiv.org/abs/0903.5256
[8] G. M. Crosswhite and D. Bacon, "Automated searching for quantum subsystem codes," *Physical Review A*, vol. 83, no. 2, p. 022307, 2011. [Online]. Available: https://arxiv.org/abs/1009.2203
[9] S. Bravyi, G. Duclos-Cianci, D. Poulin, and M. Suchara, "Subsystem surface codes with three-qubit check operators," *Quant. Inf. Comp.*, vol. 13, pp. 0963–0985, 2013. [Online]. Available: https://arxiv.org/abs/1207.1443
[10] N. Rengaswamy, R. Calderbank, S. Kadhe, and H. D. Pfister, "Logical Clifford synthesis for stabilizer codes," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–17, 2020. [Online]. Available: https://arxiv.org/abs/1907.00310
[11] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Physical Review A*, vol. 70, no. 5, p. 052328, 2004. [Online]. Available: https://arxiv.org/abs/quant-ph/0406196
[12] J. Dehaene and B. De Moor, "Clifford group, stabilizer states, and linear and quadratic operations over GF(2)," *Physical Review A*, vol. 68, no. 4, p. 042318, 2003. [Online]. Available: https://arxiv.org/abs/quant-ph/0304125
[13] W. E. Ryan *et al.*, "An introduction to LDPC codes," *CRC Handbook for Coding and Signal Processing for Recording Systems*, vol. 5, no. 2, pp. 1–23, 2004.
[14] P. Panteleev and G. Kalachev, "Quantum LDPC codes with almost linear minimum distance," *IEEE Transactions on Information Theory*, vol. 68, no. 1, pp. 213–229, 2021. [Online]. Available: https://arxiv.org/abs/2012.04068
[15] N. Raveendran, N. Rengaswamy, F. Rozpędek, A. Raina, L. Jiang, and B. Vasić, "Finite rate QLDPC-GKP coding scheme that surpasses the CSS hamming bound," *Quantum*, vol. 6, p. 767, 2022. [Online]. Available: https://arxiv.org/abs/2111.07029
[16] R. Smarandache, A. Gómez-Fonseca, and D. G. Mitchell, "Using minors to construct generator matrices for quasi-cyclic ldpc codes," in *2022 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2022, pp. 548–553.
[17] H. Chimal-Dzul, J. Lieb, and J. Rosenthal, "Generator matrices of quasi-cyclic codes over extension fields obtained from Gröbner basis," *IFAC-PapersOnLine*, vol. 55, no. 30, pp. 61–66, 2022.
[18] M. B. Hastings and J. Haah, "Dynamically generated logical qubits," *Quantum*, vol. 5, p. 564, 2021. [Online]. Available: https://arxiv.org/abs/2107.02194
[19] M. Davydova, N. Tantivasadakarn, and S. Balasubramanian, "Floquet codes without parent subsystem codes," *PRX Quantum*, vol. 4, no. 2, p. 020341, 2023. [Online]. Available: https://arxiv.org/abs/2210.02468

APPENDIX A
PROOF OF ALGORITHM'S ABILITY TO FIND ALL POSSIBLE REPRESENTATIONS OF A SUBSYSTEM CODE USING BINARY SYMPLECTIC MATRICES

Theorem 3. *Given an $[\![n, k, d]\!]$ CSS code with logical operators $\boldsymbol{X_i}, \boldsymbol{Z_i}$ for each logical qubit, the GNarsil algorithms can find all representations for the $2r$ gauge generators of the derived $[\![n, k, r, d]\!]$ subsystem code, where $2n-2r$ of the rows of the matrix $\boldsymbol{U}$ representing the code* (12) *are fixed.*

Proof: Let r be the number of gauge qubits chosen from the input code. The input code is described by a $2n \times 2n$ matrix $\boldsymbol{U}$, a symplectic matrix that describes the code's logical operators and stabilizers. The row vectors $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{2n}\}$ of $\boldsymbol{U}$ then span the space $\mathcal{U} = \mathbb{F}_2^{2n}$. We choose $2r$ vectors $\{\boldsymbol{u}_i : i \in \mathcal{I}\}$, $\mathcal{I} = \{i_1, i_2, \ldots, i_r, n+i_1, n+i_2, \ldots, n+i_r\}$, from $\boldsymbol{U}$ which correspond to a submatrix $\boldsymbol{U}_r$ of $\boldsymbol{U}$. These vectors from $\mathcal{I}$ form a symplectic basis for a 2^{2r}-dimensional subspace $\mathcal{U}_r \subseteq \mathcal{U}$. Since the subspace is symplectic, we see that any vector $\boldsymbol{u}_i, i \in \mathcal{I}$, has a symplectic product $\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle_s = 0 \pmod 2$ with any vector $\boldsymbol{u}_j, j \notin \mathcal{I}$.

We now replace the vectors from $\boldsymbol{U}_r$ with vectors $\tilde{\boldsymbol{u}} \in \mathcal{U}_r$ such that $\langle \tilde{\boldsymbol{u}}_i, \tilde{\boldsymbol{u}}_{i+r} \rangle_s = 1 \pmod 2$. To see which pairs of vectors in $\mathcal{U}_r$ form valid symplectic pairs we arrange all of the symplectic products of vectors in $\mathcal{U}_r$ into a matrix $\boldsymbol{P} = \sum_{ij} \langle \boldsymbol{u'}_i, \boldsymbol{u'}_j \rangle_s |i\rangle \langle j| \pmod 2$, where $\boldsymbol{u'}_i, \boldsymbol{u'}_j \in \mathcal{U}_r$. We use $\boldsymbol{P}$ to count the number of possible representations generated of symplectic pairs of $\mathcal{U}_r$. The number of representations is defined as the number of unique matrices $\boldsymbol{V}$ built from $\boldsymbol{U}$ by replacing the vectors $\{\boldsymbol{u}_i : i \in \mathcal{I}\}$ with vectors $\tilde{\boldsymbol{u}}$ such that $\boldsymbol{V}\boldsymbol{\Omega}\boldsymbol{V}^T = \boldsymbol{\Omega}$, where $\boldsymbol{\Omega} = \left[\begin{smallmatrix} \boldsymbol{0} & \boldsymbol{I}_n \\ \boldsymbol{I}_n & \boldsymbol{0} \end{smallmatrix}\right]$.

This condition is equivalent to all constraints needed to be satisfied by a valid subsystem code. The number of representations for a given r can be found by examining $\boldsymbol{P}$. For instance, in the case of $r = 2$ we see that:

$$\boldsymbol{P} = \left[\begin{smallmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0\\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1\\
0&1&0&1&1&0&1&0&0&1&0&1&1&0&1&0\\
0&1&0&1&0&1&0&1&1&0&1&0&1&0&1&0\\
0&1&0&1&1&0&1&0&1&0&1&0&0&1&0&1\\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\
0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0\\
0&0&1&1&0&0&1&1&1&1&0&0&1&1&0&0\\
0&0&1&1&1&1&0&0&1&1&0&0&0&0&1&1\\
0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0\\
0&1&1&0&1&0&0&1&0&1&1&0&1&0&0&1\\
0&1&1&0&0&1&1&0&1&0&0&1&1&0&0&1\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0
\end{smallmatrix}\right]. \tag{27}$$

By examining $\boldsymbol{P}$, we first see that there are $2^{2r} - 1 = 15$ non-zero vectors of $\mathcal{U}_r$ that can be chosen as the first vector. Each of these vectors has $2^{2r-1} = 8$ available symplectic partners. For finding the third vector to add to $\boldsymbol{V}$, out of $2r$ needed vectors, we look for possible candidates by searching for the columns of $\boldsymbol{P}$ where the first two vectors' rows are 0, i.e., the third vector is symplectically orthogonal to the first two vectors added to $\boldsymbol{V}$. We find 4 such columns but note that one is the column corresponding to the trivial zero vector. Therefore, we have $2^{2r-2} - 1 = 3$ possible nontrivial choices for the third vector. Finally, we search for the compatible symplectic pairs of the third vector, which gives us $2^{2r-3} = 2$. Thus, our total number of subsystem codes for $r = 2$ is 720.

However, this number includes the multiplicity of codes due to the possible permutations of the $2r$ rows that still allow $\boldsymbol{V}$ to satisfy $\boldsymbol{V}\boldsymbol{\Omega}\boldsymbol{V}^T = \boldsymbol{\Omega}$. By looking at all possible permutations of the rows such that we preserve the symplectic pairs between the row vectors, we find that the multiplicity is 8 for $r = 2$. Thus, the total number of unique representations for $r = 2$ is 90. We can extrapolate this procedure to arbitrary r and find that the number of representations for a given $r \geq 2$ is

$$\frac{1}{\mathcal{M}} \left(\prod_{l \in \{0,1,2,\ldots,r-1\}} 2^{2r-2l} - 1 \right) \left(\prod_{m \in \{1,3,5,\ldots,2r-1\}} 2^{2r-m} \right), \tag{28}$$

where $\mathcal{M}$ is the multiplicity of the $2r$ rows. We conclude that these form all possible representations of the subsystem code as the gauge generators must necessarily come from $\mathcal{U}_r$. ∎