Quantum Error Correction: A Brief Walk Through Stabilizer Codes

ANAGHA G .

October 2023

Contents

1 Introduction

Quantum Error Correction is a core concept in Quantum Information Theory. Efficient Quantum Algorithms exploit large scale quantum interference, which is fragile due to the nature of qubits. This has the potential to restrict the kinds of computations we can perform. There are several error correcting codes, but we shall be focusing on the Stabilizer Code, which corresponds to its classical analogue: linear codes. We will also take a look at a and attempt to implement and analyze a few Stabilizer codes on Qiskit.

Classical error correction is well studied, and there have been quite a few advances. However, in the quantum realm, we are bound by a few restrictions. These include

- **No cloning theorem:** This prohibits us from copying quantum states. This greatly restricts the *types* of operations we can perform on qubits.
- Measurement of quantum states in superposition leads to their destruction.
- Apart from bit flip errors, we also need to deal with phase flip errors.

These undoubtedly add another layer of complexity to error correction.

2 Literature Survey

The primary literature survey involved reading the basic concepts of error correction both classically and in the quantum world. This helped us build motivation for the evolution of quantum error correction techniques [2, 13]. Then, we understood the basics of Stabilizer formalism using [1, 9, 12]. The rest of the report comprises of several interesting results pertaining to Quantum Stabilizer Codes and Computational Complexity theory, which have implications for both classical and quantum computation. Finally, we demonstrate encoding and decoding stabilizer codes using one particular example: the Steane code.

3 Methodology

3.1 Motivation for Stabilizer Formalism

Consider the classic case of the three bit repetition code. This maps the binary values 0 and 1 as follows:

$$0 \mapsto 000$$
$$1 \mapsto 111$$

Suppose we are in the classical world. A single bit flip might result in 1 being encoded as 101. In this case, the actually bit sent is inferred using a minority vote. Of course, our inference would be wrong in case of 2 bit flips. Another way to do this would be through a parity check. We check the parity of all pairs of bits. If some have an odd parity, then an error has occurred. We define the **distance** of a code is the number of positions at which two codes might differ from one another. Mathematically, this is given by $d = 2t+1$ where t represents the number of errors the code can correct. We now move to the quantum world.

Here, things are a bit more complicated because bits are no longer in a binary state but can in fact assume an infinite number of states, and hence are subject to infinitely many errors. **Digitization of errors** is the process of decomposing any continuous error into X and Z errors. This is fairly straight forward to do: in the quantum world, we are subject to bit flips and phase flips. Bit flips correspond to X errors, and phase flips to Z errors. We now return back to our 3 bit repetition code, but in the quantum world. Consider the following mapping:

$$|0\rangle_L = |000\rangle \ |1\rangle_L = |111\rangle$$

and we have the quantum state

$$|\Psi\rangle = a\,|0\rangle + b\,|1\rangle$$
$$|\Psi\rangle_L = a\,|0\rangle_L + b\,|1\rangle_L$$
$$|\Psi\rangle = a\,|000\rangle + b\,|111\rangle$$

Suppose there is an X error on the first qubit:

$$\left|\bar{\Psi}\right\rangle = a\,|100\rangle + b\,|011\rangle$$

We cannot take a majority vote here, because 'reading' the message will read in a total collapse of the quantum state. We resort to the parity check method. We formally introduce the notion of parity checks for qubits. This is nothing but the observable $Z_i Z_j$. If there is a bit flip, it returns an eigenstate of -1, otherwise returns a state of 1. Z errors are more complicated. Suppose there is a Z error on the second qubit, this won't work, because we will still be in the subspace spanned by $|000\rangle\, and\, |111\rangle$. We modify the previous code to detect only Z errors now: we propose a change of basis from $\{|000\rangle\,, |111\rangle\}$ to $\{|+++\rangle\,, |---\rangle\}$, and the parity check measuremenets from $Z_i Z_j$ to $X_i X_j$.

3.1.1 Shor's Code

Combining these two, we obtain the Shor's code. I will give a quick exposition of what it entails here:

$$|0\rangle_{L_2} == \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)$$
$$|1\rangle_{L_2} == \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)$$

There is no X error if and only if for all pairs of qubits i and j belonging to the same block $Z_i Z_j |\Psi\rangle_{L_2} = |\Psi\rangle_{L_2}$

For detecting Z error, define:

$$\bar{X} = XXX$$
$$|\bar{0}\rangle_{L_1} = |1\rangle_{L_1} \; \bar{X}_1 = X_1X_2X_3, \bar{X}_2 = X_4X_5X_6, \bar{X}_3 = X_7X_8X_9$$

Measure the parity $\bar{X}_i\bar{X}_j \; \forall i,j \in \{1,2,3\}, i \neq j$. Then $X_iX_j |\Psi\rangle_{L_2} = |\Psi\rangle_{L_2}$ in case there is no error.

The stabilizer formalism is an attempt to generalize these observations to come up with new quantum error correcting codes. Before that, we will take a quick detour into using code concatenation to correct multiple qubit errors.

3.2 Code Concatenation

To correct errors in multiple qubits, we generally employ code concatenation. For example, Shor's code is a $[[9,1,3]]$ code. On concatenating with itself, we get a $[81,1,9]]$ code. Theoretical results tell us that we should be able to correct upto $\lfloor \frac{9-1}{2} \rfloor = 4$ errors. An interesting observation is that a worst case two-qubit or three qubit bit flip only causes a phase error. A four qubit bit flip doesn't even cause an error. We take a sample error:

Example: $X_1X_2X_{10}X_{11}$

- There are 81 qubits, divided into 9 level 1 blocks
- Each block is encoded via Shor's code. Thus there are 9 encoded level 1 qubits
- Encode these once more to get 1 level 2 encoded qubit.

The basis states of this level 2 encoded qubit are:

$$\left|\bar{\bar{0}}\right\rangle = (|\overline{000}\rangle + |\overline{111}\rangle)(|\overline{000}\rangle + |\overline{111}\rangle)(|\overline{000}\rangle + |111\rangle)$$
$$\left|\bar{\bar{1}}\right\rangle = (|\overline{000}\rangle - |\overline{111}\rangle)(|\overline{000}\rangle - |\overline{111}\rangle)(|\overline{000}\rangle - |\overline{111}\rangle)$$

The single bar states are the level 1 encoded qubits. We have the error $X_1X_2X_{10}X_{11}$. These are the two-qubit errors on the 1st and 2nd level 1 qubits. From our observations, these lead to only a phase error (after error detection and correction at level 1). More particularly,

- X_1X_2 creates a phase error after correction on the first barred qubit
- $X_{10}X_{11}$ creates a phase error after correction on the second barred qubit

After level 1 correction, basis states look like

$$\left|\bar{\bar{0}}\right\rangle \rightarrow (|0\bar{0}0\rangle + (-1)^2) |1\bar{1}1\rangle)(|0\bar{0}0\rangle + |1\bar{1}1\rangle)(|0\bar{0}0\rangle + |1\bar{1}1\rangle)$$
$$\left|\bar{\bar{1}}\right\rangle \rightarrow (|0\bar{0}0\rangle - (-1)^2 |1\bar{1}1\rangle)(|0\bar{0}0\rangle - |1\bar{1}1\rangle)(|0\bar{0}0\rangle - |1\bar{1}1\rangle)$$

So in this case, the four qubit error got corrected in the first level itself. For some other errors, the entire error might not get corrected at the first level itself. Sometimes, a phase error will lead to distortion. Thus one needs to come up with smart level one and level two corrections. There is no general algorithm to do this kind of error correction for concatenated codes. There have been some recent advances in quantum annealing regarding the same, however. An optimal algorithm for this still remains an open problem.

3.3 Preliminaries

3.3.1 Some terminology:

An [[**n, k, d**]] code is a quantum error correction code . It encodes k qubits in an n-qubit state such that any operation that maps any encoded state to another acts on atleast d qubits. In other words, the code distance must be d. Moreover, d=2t+1, where t is the maximum number of errors that can be corrected.

3.3.2 Group Theory

We first lay down some basic terminology:

A group is a set equipped with an operation, denoted by $(G, \cdot)$ such that for elements $a, b, c \in G$, the following hold:

1. $a \cdot b \in G$ (closure)
2. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ (associativity)

3. $\exists e \in G$ such that $a \cdot e = e \cdot a = a$

4. For every element $g_1 \in G, \exists g_2 \in G$ such that $g_1 \cdot g_2 = e$

3.3.3 Pauli Group

The Pauli operators on a single qubit are $\{I, X, Y, Z\}$. Then define the Pauli group $\mathcal{P}_n$.

$$\mathcal{P}_n = \{\omega P_1 \bigotimes P_2 \bigotimes \cdots \bigotimes P_n : P_i \in \{I, X, Y, Z\}, \omega \in \{\pm 1, \pm i\}\}$$

One can observe that this group has 4^{n+1} elements. The elements of this group satisfy the following properties:

1. $\forall P \in \mathcal{P}_n, P^2 = \pm I$

2. All the elements either commute or anti-commute. i.e., for any two elements $P, Q \in \mathcal{P}_n$, either of the two always hold: $PQ = QP$ or $PQ = -QP$

3. The elements are all unitary

3.3.4 Clifford Groups:

The Clifford Group is a group generated by Hadamard, phase and C-NOT gates. These gates normalize $\mathcal{P}_n$. (Recall that the normalizer of a set (in a group) is a set of elements that leave the elements fixed under conjugation).

3.4 An introduction to Stabilizer Codes

3.4.1 Stabilizer Group

A subset of a group that satisfies all the group axioms forms a subgroup. The subgroup of the Pauli group that has

1. all elements that commute with each other

2. and that does not contain $-I$

, forms the stabilizer group. Any group can be represented by its generators. By generator, we shall refer to the minimal set of elements that together generate the entire group. Any group of size G can be represented by $log(G)$ generators. Because it is a minimal set, any element cannot be written as a product of two different pairs of generators. Consider the following example:

$$S = \{III, ZZI, ZIZ, IZZ\}$$

S forms a stabilizer group. This can also be denoted by $S = \langle ZZI, ZIZ \rangle$ By virtue of property 2, The stabilizer group is an abelian (commutative) group.

3.4.2 Stabilizer States

Given $P \in \mathcal{P}_n$, a state $|\psi\rangle$ such that $P|\psi\rangle = |\psi\rangle$ is called a stabilizer state. All states that satisfy this condition for all elements $P \in \mathcal{P}_n$ form a subspace on n qubits called the stabilizer space V_s. Note the following:

1. Any linear combination of two(or more, for that matter) elements of $V_s \in V_s$. So V_s is a subspace of the n-qubit state space
2. Moreover, V_s is the intersection of the subspaces fixed by each operator in S. In other words, it is the +1 eigenspace of all operators in a stabilizer group.

3.4.3 Check matrix

Suppose S= $\langle g_1, g_2, \cdots, g_m \rangle$. One can define an $m \times 2n$ matrix where there are m generators of the code, and n qubit codes. The LHS of the matrix contains 1s to indicate which generators contain X, same with the RHS to indicate Z. Consider the construction of the ith row:

- If g_i contains an I on the kth qubit, then the kth and n+kth column elements are 0.
- If g_i contains an X on the kth qubit, then the kth column element is a 1, and the n+kth column element is a 0
- If g_i contains a Z on the kth qubit, then the kth column element is a 0, and the n+kth column element is a 1
- If g_i contains a Y on the kth qubit, then the kth column element is a 1, and the n+kth column element is also a 1

This doesn't provide much new information, but is a simpler way of representation. This is a compact form, because in general it is very tedious to write something like $X \otimes X \otimes Z$ etc

Proposition: The generators $g_1, g_2, \cdots g_m$ are independent if and only if the rows of the correspondent check matrix are linearly independent.

The Check matrix representation also helps us prove this very important result:

Proposition: If there are m=n-k generators, then V_s is 2^k dimensional.

3.4.4 Unitary dynamics:

Consider a vector space V_s stabilized by a group S. Let $|\psi\rangle \in V_s$. $\forall g \in S$, we have

$$U|\psi\rangle = Ug|\psi\rangle = UgU^\dagger U|\psi\rangle$$

So, $U|\psi\rangle$ is stabilized by $gU^\dagger \implies UV_s$ is stabilized by $USU^\dagger = \{UGU^\dagger : g \in S\}$ Moreover, if $S = \langle g_1, g_2, \cdots, g_m\rangle$, then $USU^\dagger = \langle Ug_1Y^\dagger, Ug_2U^\dagger, \cdots, Ug_mU^\dagger\rangle$. The change is stabilizer is nothing but the change in generators of the stabilizer. Consider the case of the H gate:

$$HXH^\dagger = Z, HYH^\dagger = -Y, HZH^\dagger = X$$

On applying H gate to a quantum gate stabilized by $Z(|0\rangle)$, the resultant state is stabilized by $X(|+\rangle)$. You can see how this formalism greatly simplifies things for us when we are working with a larger number of qubits.

3.4.5 Logical Operators

We take a quick detour and define the logical X and Z operators. The operators $\bar{X} and \bar{Z}$ are found by requiring two operators that commute with all the stabilizers, but cannot be expressed as a product of the stabilizers. These two will anticommute with each others. A way of computationally finding this would be to draw out an $M \times 2N$ binary matrix, where M is the number of generators and N is the number of qubits they act on. Each row corresponds to a different generator, and we fill in 0/1 depending on whether X (or Z) are present in that qubit for the first (second) N. After calculating this, simply find the null space of this matrix modulo 2. These are different from the generators.

3.5 Connections between Stabilizer Codes and Classical Linear Codes

It is very natural to notice the similarities between classical linear codes and stabilizer codes. The same way most codes in the classical world are linear codes, most codes in the quantum world are stabilizer codes. Quantum stabilizer codes can be considered very closely related to binary linear codes (linear codes over $\mathbb{F}_2$. Another striking similarity is the structure of generators of stabilzier codes and the parity check matrices for classical linear codes. We introduced the notion of check matrix for quantum codes because of the same.

3.6 Gottesman-Knill Theorem

Theorem 1 *A quantum circuit using only the following elements can be simulated efficiently on a classical computer:*

1. *Qubits prepared in their computational basis states*
2. *Clifford gates (Hadamard gates, CNOT gates, phase gate), and*
3. *Measurements in the computational basis.*

This theorem is testament to the power of quantum computation. Quantum systems with even highly entangled states can then be simulated on classical computers. Therefore, a wide variety of error correcting codes can be specified using the Stabilizer formalism. But of course, there are codes, that *cannot* be specified using the same. We shall now outline a quick proof of the theorem:
Proof:

1. Start with a stabilizer state $|\psi\rangle$ with stabilizer group S_ψ generated by n generators $\langle S_1, S_2, \cdots, S_n\rangle$
2. Pick a generator of the Clifford Group, say u
3. Under the action of u, $|\psi\rangle$ stabilized by some S_ψ becomes $u\,|\psi\rangle$
4. Then $|\psi'\rangle = u\,|\psi\rangle$ can be described using Stabilizer group as: $S_{\psi'} = uS_\psi u^\dagger$
5. We now need to find $uS_1u^\dagger, uS_2u^\dagger, \cdots, uS_nu^\dagger$
6. The above step requires $\mathcal{O}(n^2)$ operations on a classical computer.
7. If we have m such operations, then the computation can be done on a classical computer in $\mathcal{O}(mn^2)$
8. If m=$O(n)$, then simulating a stabilizer circuit on a classical computer takes $O(n^3)$ time

It is also clear that preparation of qubits in computational basis states is essential for universal quantum computation.

3.7 On the complexity of decoding quantum stabilizer codes

In this investigation, we delve into the challenges of optimally decoding a quantum stabilizer code. It is established that errors can be identified by measuring specific operations, yielding an error syndrome. The objective is to ascertain the most probable recovery given the observed syndrome. The classical counterpart of this problem has been demonstrated to be NP-complete, and a similar complexity is known for generic quantum codes. However, the inadequacy of this approach lies in its failure to account for error degeneracy in the quantum realm—where distinct errors can produce the same effect on the code. The authors demonstrate that the truly optimal decoding strategy is significantly more intricate: it is #P-complete. (Here, #P denotes the problem of counting solutions to the decision problems in NP) Clearly, any #P problem is, at the very least, as challenging as the corresponding NP-complete problem.

3.8 Importance of (non) stabilizer states for quantum computation

The Gottesman-Knill theorem showed that stabilizer circuits can be simulated efficiently on classical computers. Aaronson and Gottesman have shown the following improvements:

1. They propose a speed-up in the algorithm by eliminating Gaussian Elimination (which has the same complexity as matrix multiplication) by introducing a blow up in the number of bits needed to represent a state by a factor of 2. Thus the time needed to simulate a stabilizer circuit is reduced from $O(n^3)$ to $O(n^2)$

2. This problem is $\bigoplus L$ complete. ($\bigoplus L$- parity-L is the set of languages accepted by a non-deterministic Turing machine that can (only) distinguish between an even and odd number of acceptance paths in log-space. From the Gottesman-Knill Theorem, it is clear that this problem was already $\bigoplus L$ hard. They then showed that his problem itself was in $\bigoplus L$, which can also be interpreted as the class of problems that reduce to simulating a polynomial-size CNOT circuit. Thus this result is strong evidence that stabilizer circuits are probably not even universal for classical computation, and that we need to shift focus on studying non stabilizer circuits as well.

An interesting follow up question to this result would be: Are there any sets of quantum gates that are neither universal for *quantum* computation, nor can be simulated classically? There is ongoing research in this direction. We can also think about the position of this problem if classical post-processing is allowed (in which case, it would no longer be even in $\bigoplus L$)

Thus, all these results help us to narrow down where exactly stabilizer circuits lie in the realm of computational complexity.

3.9 Examples:

3.9.1 3 qubit bit flip:

We first take an elementary case: The 3 qubit bit flip.

$$S = \langle ZZI, ZIZ \rangle$$

The generator table is given by This is clearly a much simpler way of represen-

Element	Operator
S_1	ZZI
S_2	ZIZ
$\bar{X}$	XXX
$\bar{Z}$	ZII

tation.

3.9.2 3 Qubit Phase Flip

Next, consider the 3 qubit phase flip code, used for correcting single bit phase flips. It is clear that we need a change of basis using Hadamard gates. Appropriately,

Element	Operator
S_1	XXI
S_2	XIX
$\bar{X}$	XII
$\bar{Z}$	ZZZ

3.9.3 Shor's Code

Recall our earlier example of Shor's code. It is a concatenation of the bit flip and phase flip codes. Using the notation introduced earlier, it is a $[[9, 1, 3]]$ code. The Stabilizer is given by

Element	Operator
S_1	ZZIIIIIII
S_2	ZIZIIIIII
S_3	IIIZZIIII
S_4	IIIZIZIII
S_5	IIIIIIZZI
S_6	IIIIIIZIZ
S_7	XXXXXXIII
S_8	XXXIIIXXX
$\bar{X}$	XXXXXXXXX
$\bar{Z}$	ZZZZZZZZZ

3.9.4 Steane Code:

The Steane Code is a $[[7, 1, 3]]$ code. It is the dual of the Hamming code, which is a classical code, and hence is very widely studied. It is used to correct arbitrary single qubit errors. It has the following stabilizers:

Element	Operator
S_1	IIIXXXX
S_2	IXXIIXX
S_3	XIXIXIX
S_4	IIIZZZZ
S_5	IZZIIZZ
S_6	ZIZIZIZ
$\bar{X}$	XXXXXXX
$\bar{Z}$	ZZZZZZZ

3.10 Implementation

We will be implementing certain stabilizer codes using the STAC library. This can also be done via Qiskit, but using STAC made the code look more compact and readable. For higher order qubits, the generation of circuits in Qiskit turned out to be rather messy. The generator matrices of several codes are stored inside STAC.

3.10.1 Construction of Encoding Circuit for Stabilizer Codes

We make use of Gottesman's Algorithm to construct logical zero state of stabilizer codes. This involves three broad steps:

1. Bring the generator matrix to the Standard Form
2. Construct logical operators of the code (as discussed earlier)
3. Determine the sequence of gates in the encoding circuit using Gottesman's algorithm

The first example we will consider here is of the Steane Code, which can be represented as the $[[7, 1, 3]]$ code.

```
import stac
import numpy as np
cd_stean = stac.CommonCodes.generate_code('[[7,1,3]]')
stac.print_matrix(cd_stean.generator_matrix, augmented=True)
```

This dispays the generator matrix for the Steane Code.

$$
\left(\begin{array}{ccccccc|ccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}\right)
$$

It has 6 generators, and 7 phsyical qubits. It encodes 7-6=1 logical qubit. Now, print the Pauli form of the generators using:

```
stac.print_paulis(cd_stean.generator_matrix)
```

$$
\begin{array}{c}
XXXXIII \\
XXIIXXI \\
XIXIXIX \\
ZZZZIII \\
ZZIIZZI \\
ZIZIZIZ
\end{array}
$$

Follow the following steps to create an encoding circuit. **Step 1:** Bring the generator matrix to the standard form. Because we are in the framework of group theory, we can manipulate generators and qubits using one(or more) of the following:

1. Replace one the generators g_i by $g_i g_j$. This corresponds to adding rows.
2. Reindex the generators. This corresponds to permuting rows of G
3. Reindex the physical qubits. This corresponds to permuting the columns.

First, bring the X part of the generator matrix to RREF, and then apply property 2. Then,

$$G = \begin{pmatrix} I & A & | & B & C \\ 0 & 0 & | & D & E \end{pmatrix}$$

Note that if the rank of the X part is r, then the blocks have

- r and m-r rows
- r and m-r columns

The next step is to bring E to RREF form, and then apply the properties listed above.

$$G = \begin{pmatrix} I & A_1 & A_2 & | & B & C_2 & C_2 \\ 0 & 0 & 0 & | & D & I & E_2 \end{pmatrix}$$

We can simply use a library function for this:

```
cd_stean.construct_standard_form()
stac.print_matrix(cd_stean.standard_generator_matrix, augmented=True)
```

The standard form is:

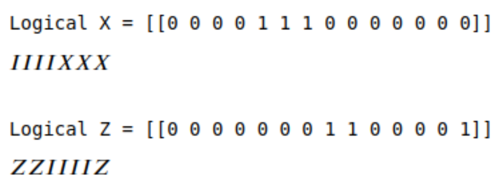
$$\left(\begin{array}{ccccccc|ccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{array}\right)$$

Step 2: Construct logical operators of the code. Recall that these are operators that can act directly on the encoded state.

Here is an algorithm given by Gottesman: The $|BarX's$ are the rows of the matrix $\bar{X} = \begin{pmatrix} 0 & E_2^T & I & | & (E_2^T C_1^T + C_2^T) & 0 & 0 \end{pmatrix}$ and the $\bar{Z}'s$ are the rows of the matrix $\bar{Z} = \begin{pmatrix} 0 & 0 & 0 & | & A_2^T & 0 & I \end{pmatrix}$

```
    cd_stean.construct_logical_operators()
    print("Logical X =", cd_stean.logical_xs)
    stac.print_paulis(cd_stean.logical_xs)

    print("\nLogical Z =", cd_stean.logical_zs)
    stac.print_paulis(cd_stean.logical_zs)
```



```
Logical X = [[0 0 0 0 1 1 1 0 0 0 0 0 0 0]]
IIIIXXX

Logical Z = [[0 0 0 0 0 0 0 1 1 0 0 0 0 1]]
ZZIIIIZ
```

Step 3: Use the Gottesman Algorithm to find the sequence of gates Suppose there are n qubits. Further let

- qubits 0 to n-k-1 be the ancilla (extra) bits in state $|0\rangle$
- the k qubits n-k to n-1 be in some state $|\psi\rangle$ (to be encoded)

The algorithm is as follows:

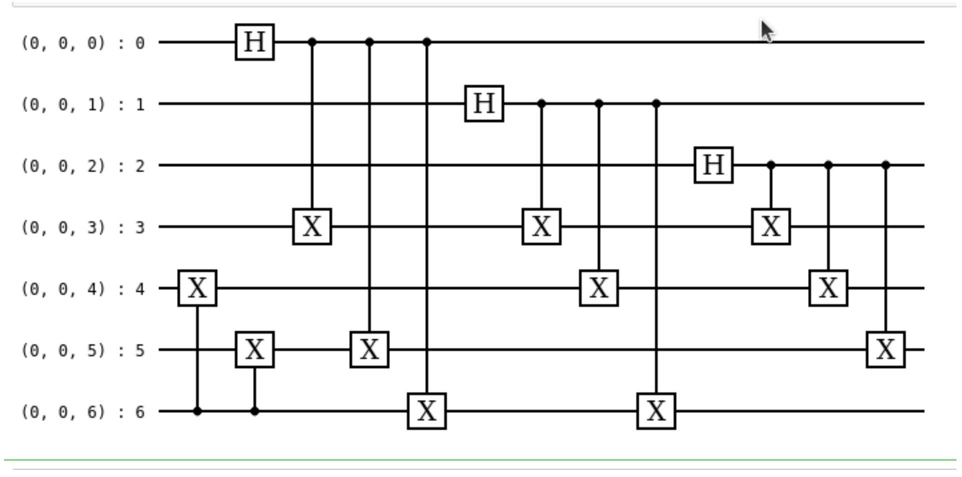
```
    encoding_circuit = stac.Circuit.simple(n)
for i in range(k):
    for j in range(r, n-k):
        if cd.logical_xs[i, j]:
            encoding_circuit.append("CX", n-k+i, j)

for i in range(r):
    encoding_circuit.append(["H", i])
    for j in range(n):
        if i == j:
            continue
        if cd.standard_generators_x[i, j] and
             cd.standard_generators_z[i, j]:
            encoding_circuit.append("CX", i, j)
            encoding_circuit.append("CZ", i, j)
        elif cd.standard_generators_x[i, j]:
            encoding_circuit.append("CX", i, j)
        elif cd.standard_generators_z[i, j]:
            encoding_circuit.append("CZ", i, j)
```

[12] Then, we apply the library function:

```
enc_circ = cd_stean.construct_encoding_circuit()
enc_circ.draw()
```

The circuit looks like:



The state produced is

$$|\bar{0}\rangle = |0000000\rangle + |1100110\rangle + |1111000\rangle + |0011110\rangle + |1010101\rangle + |0110011\rangle + |0101101\rangle + |1001011\rangle$$

Similarly, one can form logical 1 state by using the logical $\bar{X}$ to flip the zero state.

3.11 Decoding circuit

Now that we have the encoding circuit in hand, decoding is just running the encoding circuit in reverse. Another way is using the Gottesman Decoding algorithm. Here, the key is to add k extra qubits to our circuit to store our decoded state there, instead of modifying it 'in place'. The steps are:

1. Compute standard form
2. Find the logical $\bar{X}$ and $\bar{Z}$ operations for the generator
3. Employ the following algorithm:

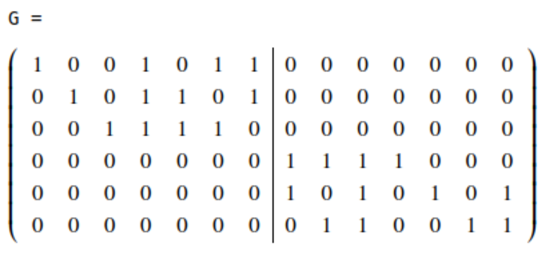
```
    decoding_circuit = []

for i in range(len(logical_zs)):
    for j in range(n):
        if logical_zs[i, n+j]:
            decoding_circuit.append(["cx", j, n+i])

for i in range(len(logical_xs)):
    for j in range(n):
        if logical_xs[i, j] and logical_xs[i, n+j]:
            decoding_circuit.append(["cz", n+i, j])
            decoding_circuit.append(["cx", n+i, j])
        elif logical_xs[i, j]:
            decoding_circuit.append(["cx", n+i, j])
        elif logical_xs[i, n+j]:
            decoding_circuit.append(["cz", n+i, j])
```

Using STAC,

```
    cd_steane.construct_standard_form()
print("G =")
stac.print_matrix(cd_steane.standard_generator_matrix, augmented=True)
```



G =

$$\left(\begin{array}{ccccccc|ccccccc}
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1
\end{array}\right)$$

```
    cd_steane.construct_logical_operators()
print("Logical X =")
stac.print_matrix(cd_steane.logical_xs, augmented=True)
```

```
print("Logical Z =")
stac.print_matrix(cd_steane.logical_zs, augmented=True)
```

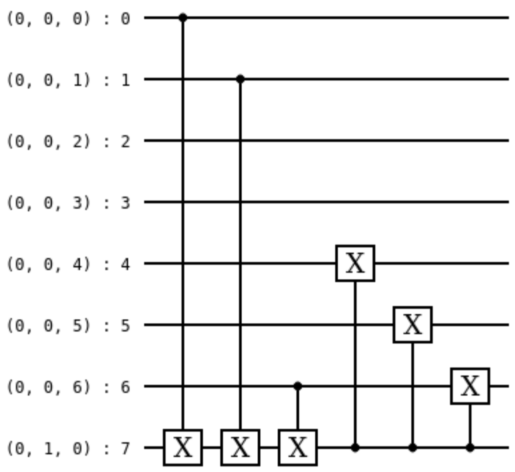
```
Logical X =
```

$$\left(\begin{array}{ccccccc|ccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

```
Logical Z =
```

$$\left(\begin{array}{ccccccc|ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

```
dec_circ = cd_steane.construct_decoding_circuit()
dec_circ.draw()
```



Then, we can check the input state using

```
dec_circ.simulate()
```

The result is the state $|0000000\rangle$ with an amplitude of 1, which is what we expected.

3.12 Conclusion

In this exposition, we familiarized ourselves with the basics of quantum error correction and stabilizer codes. We understood the convenience the stabilizer formalism bestowed upon us, as well as the nice properties given by the group structure for computations, as well as algorithms. We were also able to understand the position of Stabilizer Circuits in classical and quantum computational complexity. Finally, we studied classical algorithms given by Gottesman for encoding and decoding stabilizer circuits.

References

[1] Daniel Gottesman (1999) Stabilizer Codes and Quantum Error Correction

[2] Joschka Roffe (2019) Quantum error correction: an introductory guide Contemporary Physics, 60:3, 226-245, DOI: 10.1080/00107514.2019.1667078

[3] Pavithran Iyer, David Poulin. (2013). Hardness of decoding quantum stabilizer codes `https://arxiv.org/abs/1310.3235`

[4] Aaronson, S., Gottesman, D. (2004). Improved simulation of stabilizer circuits. Phys. Rev. A, 70, 052328

[5] Lecture notes by Dave Bacon

[6] Lecture Notes by Preskill

[7] A Short Introduction to Stabilizer Codes, Andreas Klappenecker

[8] Michael. E. Cuffaro, On the Significance of the Gottesman–Knill Theorem

[9] Arthur Pesah's Blog on Quantum Computing

[10] Exposition by Madhur Tulsiani

[11] `https://errorcorrectionzoo.org/`

[12] STAC

[13] Nielsen, M. A., Chuang, I. L. (2010, December 9). Quantum Computation and Quantum Information. Cambridge University Press.