Quantum Coding Theory **(UC Berkeley CS294, Spring 2024)**

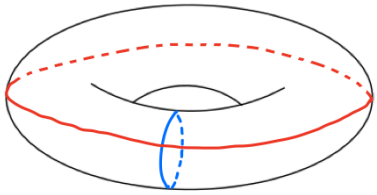
Lecture 12: The Toric Code

March 1, 2024

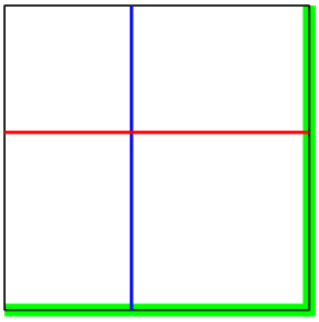
Lecturer: John Wright *Scribe: Sandy Irani*

1 Introduction to the Toric Code

Kitaev's toric code [Kit03] is a remarkably original construction, especially considering that it was first intoduced only two years after Shor's original 9-qubit code. To begin with the description of the toric code, we start with a diagram of the torus and notice that there are two different types of loops that go around the torus as shown below in red and blue.



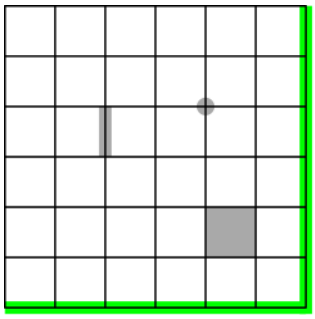
We will draw the torus in 2D with the understanding that the right edgeof the square (highlighted in green) is identified with the left edge and the bottom edge (also highlighted in green) is identified with the top edge. The red loop is shown then from left to right and the blue loop extends from top to bottom. In traveling upwards on the blue loop, when the top edge is reached, the loop continues from the bottom. This is just like the game of Pac-Man: when the Pac-Man disappears off the top side of the screen, it appears instantaneously on the bottom edge at the same horizontal position.



We will now cellulate the torus by laying a square grid on the surface.
There are three types of objects of interest:

- Vertices: where two lines intersect.

- Edges: the segment connecting two neighboring vertices.
- Plaquettes: the squares formed by four neighboring edges.

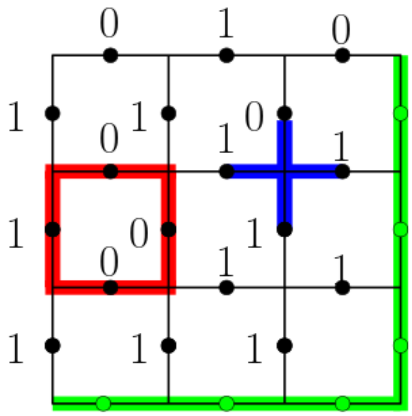


The torus shown above is a 6×6 torus since the edges and vertices on the right are identified with the edges and vertices on the left. Also, the edges and vertices on the bottom are identified with the edges and vertices on the top.

In an $L \times L$ torus, there are L^2 vertices since there is a vertex where every horizontal and every vertical line intersect. The number of edges is $2L^2$. This can be seen by associating each vertex with the two edges that extend to the right and below that vertex. Since each edge is assigned to a unique vertex and there are two edges assigned to each vertex, the number of edges must be twice the number of vertices. There are also L^2 plaquettes in an $L \times L$ torus because each vertex can be uniuely identified with the plauette that is just below and to the right of that vertex.

2 Classical codes defined on the torus

The toric code is a CSS defined on the torus. We start by defining the two classical codes that are used in the construction of the toric code. In the figures, we will use blue for the classical code C_x and red for the classical code C_z. The bits are assigned to the edges of the torus. The figure below shows a 3×3 code on the torus. The bitstring for a codeword is read from left to right, so the string associated with the torus below is 010,110,011,101,011,111. The commas are included to make it easier to read.



There will be a parity check in C_x for each vertex in the torus. The bits checked will be the bits on the four edges incident to that vertex. The string associated with the parity check for vertex v will have a 1 for each bit that lies on an edge incident to v and a 0 everywhere else. The parity check enforces the condition that the number of 1's on the edges incident to v must be even. The string corresponding to the blue parity check in the figure above is 000,001,011,001,000,000.

Th parity checks for C_z correspond to the plaquettes. The parity check enforces the condition that the number of 1's on the edges bordering that plaquette must be even. The string associated with a parity check for plaquette p has a 1 for each edge bordering the plaquette and a 0 everywhere else. The string corresponding to the red parity check in the figure above is 000,000,100,110,100,000.

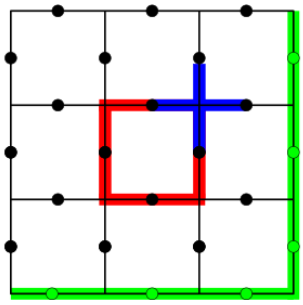
Now that C_x and C_z have been defined, the fact below establishes that the two codes can be used together to form a CSS code.

Fact 2.1. *For each $h_x \in C_x^\perp$ and each $h_z \in C_z^\perp$, $h_x \cdot h_z = 0 \pmod 2$.*

Proof. We only need to check the fact for a basis of elements in $C_x^\perp$ and a basis of elements in $C_x^\perp$. The vertex parity checks form a basis for $C_x^\perp$ and the plaquette parity checks form a basis for $C_z^\perp$. Let p be a plaquette and $h_{z,p} \in C_z^\perp$ be the string associated with the parity check for p. Let v be a vertex and $h_{x,v} \in C_x^\perp$ be the string associated with the parity check for v. We will consider two cases:

Case 1: Plaquette p is disjoint from the parity check for vertex v. This situation is pictured in the figure above. In this case, there are no locations where $h_{z,p}$ and $h_{x,v}$ both have a 1, which implies that $h_{z,p} \cdot h_{x,v} = 0$.

Case 2: Plaquette p and the parity check for vertex v share a common edge. This situation is pictured in the figure below.



In this case, edges incident to v have exactly two edges in common with the edges bordering p. This means that there are exactly two locations where $h_{z,p}$ and $h_{x,v}$ both have a 1, which also implies that $h_{z,p} \cdot h_{x,v} = 0$. □

Corollary 2.2. *C_x and C_z can be used to form a CSS code.*

How good is the CSS code formed from C_x and C_z? In order tow answer this question, we start by first describing the set of strings which satisfy all the parity checks in C_x. First of all, the all 0's string is always in C_x. Now imaging flipping a bit on some edge $e = \{v, w\}$.

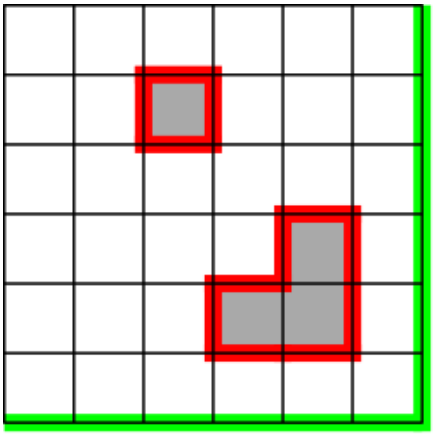
The two partiy checks centered at v and w are now violated. In order to restore those parity checks, we need to flip another bit at an edge incident to each of v and w. This restores the parity checks at v and w but now two new vertex parity checks have been violated and we need to flip a bit at edges incident to those two vertices. This process continues forming a path of 1's in the grid. At any point in this process, the parity checks at the endpoints of the path are violated. The only way then to satisfy all the parity checks is for the two endpoints of the path to meet, forming a closed loop. Therefore, the strings in C_x correspond to edges of closed loops in the grid and all linear combinations of those loops.

Fact 2.3. *The distance of C_x is at most* 4

Proof. The distance of C_x is the minimum weight of a codeword in the code. If you pick a plaquette p and put 1's on the edges bordering the plaquette and 0s everywhere else, then all of the vertex parity checks are satisfied. The associated string is in C_x and has weight 4. □

Another way to establish the fact that the distainc of C_x is at most 4 is to observe that $C_z^\perp \subseteq C_x$. Therefore the parity checks corresponding to plaquettes (which have weight 4 and are in $C_z^\perp$) are also in C_x.

The parity checks in $C_z^\perp$ are described by borders of sets of plaquettes: pick any subset of the plaquettes and take the border of this set of plaquettes. The resulting edges are always a set of closed loops. See the figure below for an example.



Note that the set of all plaquette parity checks in $C_z^\perp$ is not independent because every edge borders exactly two plaquettes, so

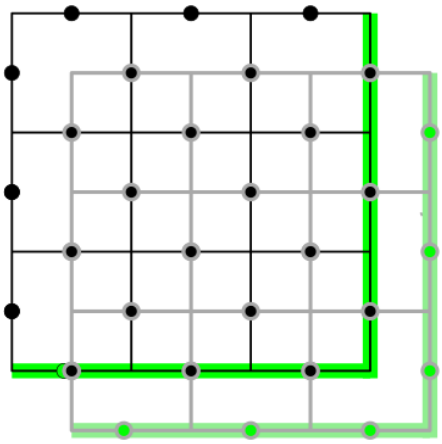
$$\sum_p h_{z,p} = 0.$$

However, if we take any proper subset of the set of plaquettes and sum over those parity checks, the result will be non-zero. This is because a proper subset of the plaquettes must contain a plaquete with a neighboring plaquette that is not included in the subset. The edge between the included plaquette and the excluded plaquette will only be included once in

the sum. The resulting sum will be a string that is not all 0's. Therefore, we can remove any plaquette term from the set of all plaquettes and the resulting set will correspond to an independent set of parity checks. In other words, if we identify a particular plaquette p^*, the set $\{h_{z,p}\}_{p \neq p^*}$ forms a linearly independent basis for $C_z^\perp$. The number of checks in a linearly independent basis for $C_z^\perp$ is the number of plaquettes minus 1, which is $L^2 - 1$.

We have now given descriptions for the codewords in C_x and and the parity checks in $C_z^\perp$. The codewords in C_z and parity checks in $C_x^\perp$ are not as intuitive to describe in the original lattice. However, they do have clean definition in the dual lattice.

To obtain the dual lattice, move the entire lattics down and to the right by half a square length. The picture below shows the original lattice in bold and the dual lattice in grey.



Vertices in the primal lattice correspond to plaquettes in the dual lattice, and plaquettes in the primal lattice correspond to vertices in the dual lattice. The set of bits on edges is exactly the same. Moreover, the relationship of the edges to the vertices/plaquettes remains the same, so the four edges that border a plaquette in the primal lattice are exactly the four edges that are incident to the corresponding vertex in the dual lattice. Similarly, the four edges incident to a vertex in the primal lattice are exactly the four edges bordering the corresponding plaquette in the dual lattice. Therefore, we can apply exactly the same reasoning in the dual lattice to understand the structure of C_z and $C_x^\perp$. Fortunately, the dual lattice is also an $L \times L$ grid, so the descriptions of these sets and their parameters are exactly the same.

3 Putting C_x and C_z together to form the Toric Code

We are now well positioned to describe the quantum CSS code resulting from C_x and C_z. The stabilizers for the quantum code are generated by Paulis oof the form X^v (which denotes an X operator appied to the qubits on the edges incident to vertex v) and Z^p (which denotes a Z operator applied to the qubits on the edges bordering plaquette p).

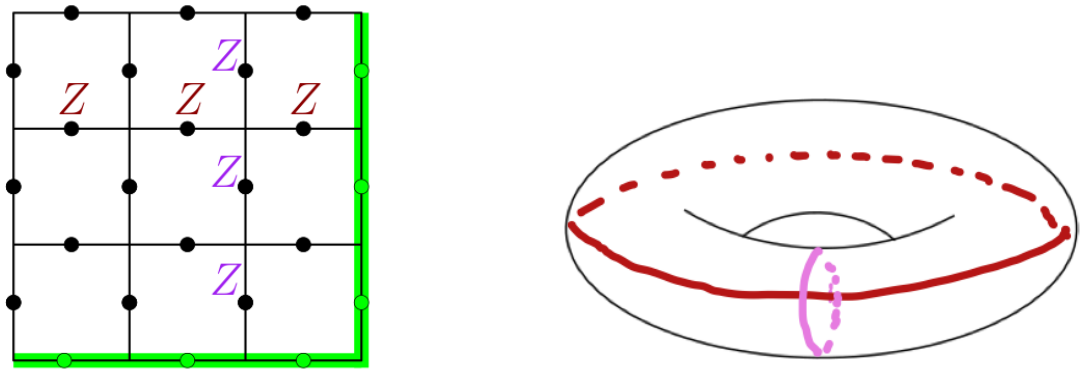
The number of physical qubits is the number of edges, which is $2L^2$. The number of logical qubits is $2L^2$ minus the number of indepdent parity checks in $C_x^\perp$ and $C_z^\perp$. We have

argued that the number of indepedent parity checks in $C_x^\perp$ is $L^2 - 1$, and the same holds for $C_z^\perp$. Therefore the number of logical qubits is:

$$2L^2 - (L^2 - 1) - (L^2 - 1) = 2.$$

It remains to determine the distance $d^+ = \min\{d_x^+, d_z^+\}$. Since the two codes are isomorphic, we only need to determine $d_x^+ = \min_{c \in C_x \backslash C_z^\perp} |c|$. Recall that we have argued that d_x and d_z are at most 4. However, we can get a much better bound by considering d_x^+.

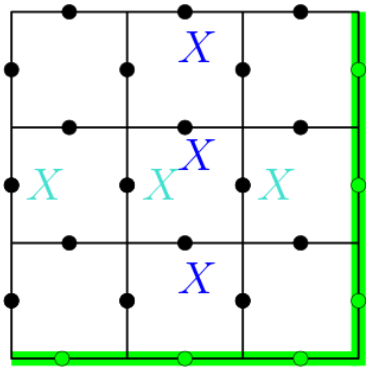
The elements of the classical code C_x are the set of all closed loops in the torus, and $C_z^\perp$ is the set of call closed loops that are boundaries of plaquettes. The set $C_x \backslash C_z^\perp$ is then the set of all loops that go around the torus, either from left to right or from top to bottom, or both. In the quantum setting, these correspond to Pauli strings that commute with all the X^v operators but are not products of Z^p operators. These are Pauli strings consisting of Z operations applied to the qubits on the edges on one of these non-contractible loop. (A contractible loop corresponds to a set of Z operators applied to the boundary of a set of plaquettes. These loops are called *contractible* because they can be transformed to the identity operation by multiplying by X^p operators.) The figure below shows the two generators of $C_x \backslash C_z^\perp$ on the grid and how they correspond to the two loops around the torus.



It's possible to shift the loops by multiplying by Z^p operators, but it is impossible to transform the red operator into the pink operator by multiplying by Z^p's. The operator shown in red and pink above are distinct in the code. If we multiply the red operation by Z^p's, the resulting operation is effectively the same error.

Since the smallest non-contractible loop has to go all the way around the torus in one direction or the other, it will have length at least L. Therefore $d_x^+ = L$. Note that this is much better than the distance of 4 achieved by the two classical codes on their own. The large difference between d_x^+ and d_x indicates that the code is highly degenerate. All of the small (contractible) loops are indeed errors in the classical codes but they are all equivalent to no error in the quantum code.

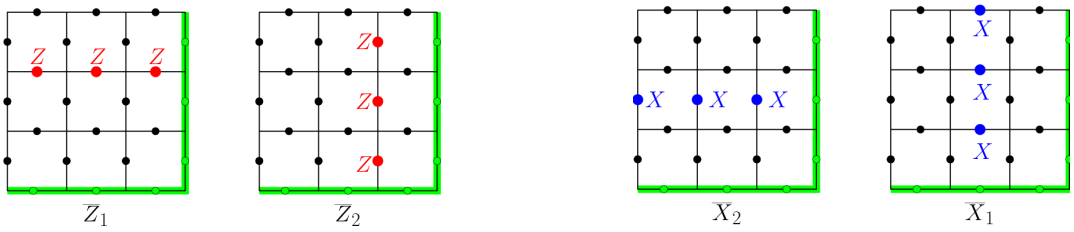
Note that $d_z^+ = d_x^+$ because the same argument can be made for the dual lattice which is identical to the primal lattice. The picture below shows the X errors in the primal lattice:



The parameters toric code are: $[[2L^2, 2, L]]$.

As we shall see in a later lecture, codes that use a shape with larger genus, it's possible to have a larger k (number of logical qubits), but this benefit comes at the expense of distance. We shall also see a code resulting from a grid with aperiodic boundary conditions. This is more practical because it can be more easily embedded into the flat architecture of a quantum computer. However, the resulting code can only encode a single logical qubit.

The logical operations on the encoded qubits in the toric code are pictured below:



Let's verify that the logical operators obey the commutation relationships that we would expect. Note that $\overline{X}_1$ and $\overline{Z}_2$ operate on disjoint sets of qubits, so they commte. The same holds for $\overline{X}_2$ and $\overline{Z}_1$. Also observe that there is exactly one qubit that both $\overline{X}_1$ and $\overline{Z}_1$ act on non-trivially, so $\overline{X}_1\overline{Z}_1 = -\overline{Z}_1\overline{X}_1$. The same holds for $\overline{X}_2$ and $\overline{Z}_2$.

The final step is to show that $\overline{Z}_1$ and $\overline{Z}_2$ (and similarly $\overline{X}_1$ and $\overline{X}_2$) correspond to distinct operations modulo the stabilizer group for the code. In other words, there is no Pauli P in the stabilizer group such that $\overline{Z}_1 = \overline{Z}_2 \cdot P$. Intuitively, this is essentially showing that the two loops $\overline{Z}_1$ and $\overline{Z}_2$ shown in the figure above are not *topologically equivalent*, meaning that it is impossible to obtain one from the other by multiplying by plaqutte operators of the form Z^p. The action of multiplying by plaquette operators has the effect of dragging and stretching the loop but never cutting it. Recall that the generators of the stabilizer for the torus are all Paulis of the form X^v or Z^p, where v is any vertex and p is any plaquette.

Fact 3.1. *There is no stabilizer P for the toric code such that $\overline{Z}_1 = \overline{Z}_2 \cdot P$.*

Proof. Suppose by contradiction that there is a stabilizer P such that $\overline{Z}_1 = \overline{Z}_2 \cdot P$. Then it must also be the case that $\overline{X}_1 \cdot \overline{Z}_1 = \overline{X}_1 \cdot \overline{Z}_2 \cdot P$. However, then we would have that

$$\overline{X}_1 \cdot \overline{Z}_1 = \overline{X}_1 \cdot \overline{Z}_2 \cdot P = \overline{Z}_2 \cdot \overline{X}_1 \cdot P = \overline{Z}_2 \cdot P \cdot \overline{X}_1 = \overline{Z}_1 \cdot \overline{X}_1.$$

The second equality uses the fact that $\overline{Z}_2$ and $\overline{X}_1$ commute (as argued above). The third equality uses the fact that since P is in the stabilizer group, it must commute with all the logical operators. The resulting equality contradicts the fact (also argued above) that $\overline{X}_1$ and $\overline{Z}_1$ anti-commute. □

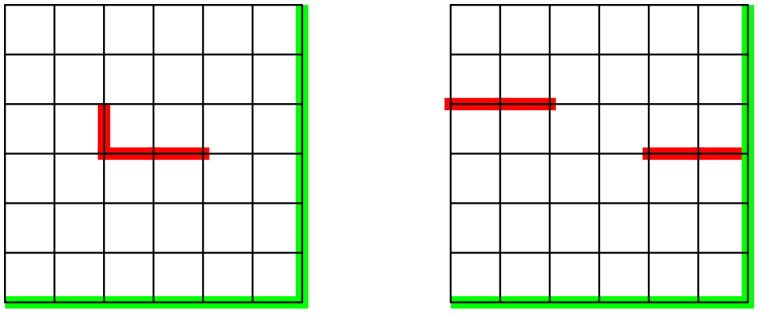
To frame the parameters of the toric code as a function of the number of physical qubits, let $n = 2L^2$. The toric code is a $[[2L^2, 2, L]]$ code, which means that it is a $[[n, 2, \sqrt{n/2}]]$ code.

One nice property of the two classical codes C_x and C_z used in toric code is that every parity check only involves a 4 bits and each bit is involved in 4 parity checks. Thus, C_x and C_z are LDPC (or *low-denisty parity check*) codes. More generally, the requirement for an LDPC code C is that it has a basis of parity checks $h_1, \ldots, h_l$ such that each qubit occurs in $O(1)$ of the h_j's and $|h_j| = O(1)$. Note that this property also holds for the toric code constructed from the two classical LDPC codes.

LDPC codes are nice because the checks can be implemented in parallel. However, it is more challenging to construct LDPC codes with good parameters.

3.1 Decoding errors

A general error will be of the form $E = X^a Z^b$. The X-parity checks are used to decode the Z errors, so we will focus on Z^b. The analysis for X^a is similar. The set of edges denoted by string b will form a set of disjoint paths. The errors are only detected at endpoints of the path because if Z operations are applied to the qubits on two edges incident to vertex v, the v parity check will commute with the error. For example, the two errors shown below (applying a Z operation to the qubits on the red edges) will yield the same syndrome:



Given a particular syndrome consisting of a set of vertices where the X-checks are violated, we want to find a way to correct the errors that applies a Pauli of minimum weight. This is achieved by the following decoding algorithm:

DECODING ALGORITHM FOR THE TORIC CODE

Let $G = (V, E, w)$ be a weighted graph, where:

V is the set of vertices in the torus such that

X^v has a $+1$ eigenvalue on the given state.

E is the complete graph on V

$w(v, w)$ is the shortest path from v to w in the torus.

Find the minimum weight matching on G

(using, for example, Edmunds' Blossom algorithm).

This decoding algorithm is guaranteed to find the actual error if for errors of weight at most $L/2$. Moreover, under random error models, many more errors can be corrected.

References

[Kit03] A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):230, January 2003. 1