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

Lecture 15: Product Codes

March 13, 2024

Lecturer: John Wright *Scribe: Samyak Surti*

1 Recap of Topological Codes

Let us recall the toric code [Kit97, Kit03]. Defined on a cellulated torus, the toric code is a good candidate for a quantum error correcting code as all of its stabilizer checks are local. This has significance, not only from the perspective of implementing such codes on physical devices, but also from the perspective of quantum coding theory. In particular, the Toric code is an example of a **Quantum Low-Density Parity Check (QLDPC)** code, as each qubit is involved in at most a constant number of checks and each parity check involves at most a constant number of qubits. Hence, the term *"low-density"* is appropriate.

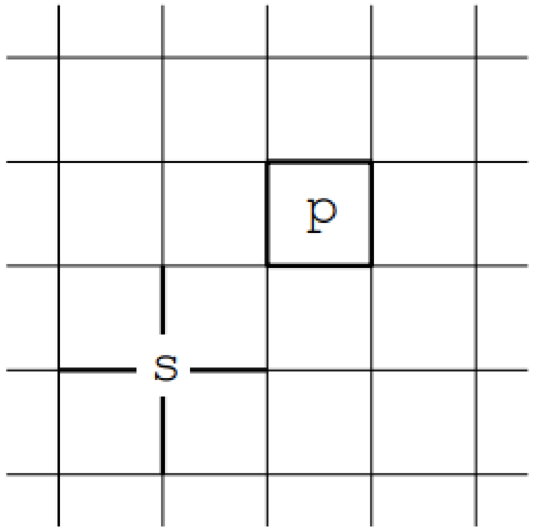


Figure 1: A quantum code defined on a cellulated torus. We associate the top and bottom edges and the left and right edges. [Kit03]

Of particular interest is seeing how far this idea of locality can be pushed to construct so-called "good" QLDPC codes where the number of encoded logical qubits and the code distance grow linearly with respect to the number of physical qubits in the system (Its important to reiterate here that the term "locality" can be interpreted in two different ways; one in reference to physical locality and another in terms of the number of checks each qubit is involved in and the number of qubits each check involves). Naturally, the next step lead to trying out cellulations of more general topological surfaces by varying both genus and dimension. Unfortunately, this lead to codes that only traded encoding rate for distance and vice versa, without making progress towards "good" QLDPC codes. In fact, when restricting ourselves to 2D topological surfaces, Bravyi, Poulin, and Terhal set forth the following bound:

Theorem 1.1 (Bravyi-Poulin-Terhal Bound [BPT10])**.** *For an* $[[n, k, d]]$ *quantum error correcting code defined on a 2D topological surface*

$$kd^2 = \mathcal{O}(n)$$

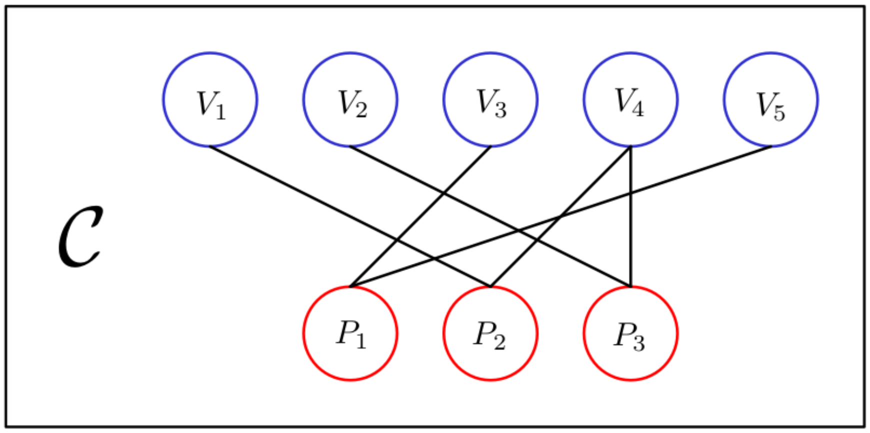
Since the toric code saturates this bound, it is the best code in this regime. Thus we must look elsewhere, perhaps abandoning physical locality of our checks, but retaining the second interpretation of locality, with respect to the relation of parity checks and qubits.

2 Building up to Hypergraph Product Codes

The road to "good" QLDPC codes begins with the Hypergraph Product Code due to Tillich and Zemor in 2009 [TZ14]. To build up to these codes, we begin by drawing inspiration from a related procedure for classical linear codes—tensor products of classical linear codes. To understand this construction, we first need to consider an alternate graphical perspective of classical linear codes.

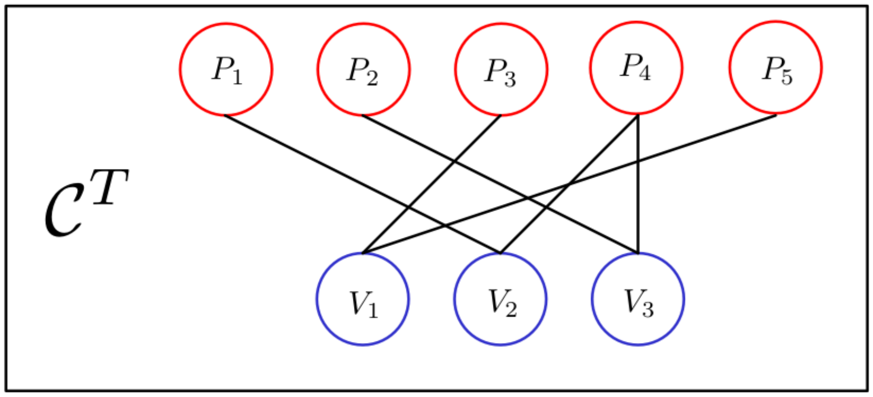
2.1 An Alternate Representation - The Tanner Graph

Definition 2.1 (Tanner Graph)**.** Let C be an $[n, k, d]$ classical linear code with parity checks given by $P_1, ..., P_l$.To this code, we can associate a bipartite graph called the **Tanner Graph**. One side of this graph has nodes corresponding to our l parity checks and the other has nodes corresponding to bits of our code. Pictorially, we have the following:



The Tanner graph admits a natural symmetry along the two sets of nodes in the graph. That is, we can swap the roles of the variables and the parity checks, admitting a different code. In this new "transpose" code, the top blue nodes are now the parity checks and the bottom red nodes are the variables or bits. Let us formally define this code:

Definition 2.2 (Transpose Code)**.** For an $[n, k, d]$ code C with parity checks $P_1, ..., P_l$ acting on bits $V_1, ..., V_n$, we can define the code C^T in terms of the parity checks for code C. In particular, C^T has parity checks $P'_1, ..., P'_n$ and variable nodes $V'_1, ..., V'_l$. Again, pictorially we have



As it is defined, C^T is not necessarily uniquely associated with our original code C, as it is a property of C's Tanner graph. We could conceivably add redundant parity checks to C, which would result in a code C^T defined over a larger number of bits. Since our parity check matrices essentially serve as adjacency matrices for our Tanner graphs, we can define them as the following:

$$H_C = \begin{bmatrix} — & P_1 & — \\ — & P_2 & — \\ \vdots & \vdots & \vdots \\ — & P_l & — \end{bmatrix}$$

and the definition of the parity check matirx for C^T follows naturally as the transpose of H_C:

$$H_{C^T} = H_C^T = \begin{bmatrix} | & | & \cdots & | \\ P_1 & P_2 & \cdots & P_l \\ | & | & \cdots & | \end{bmatrix}$$

Given H_C and H_{C^T}, let us now define the subspaces corresponding to codes C and C^T, as well as their dimensionality. For a general classical linear code, the null space (or kernel) of the parity check matrix corresponds to our set of codewords. Thus

$$C = \ker(H_C), \quad \boxed{\dim(C)} = \dim(\ker(H_C)) = \boxed{n - \text{rank}(H_C)}$$

$$C^T = \ker(H_{C^T}) \quad \boxed{dim(C^T)} = \dim(\ker(H_{C^T})) = l - \text{rank}(H_{C^T}) = \boxed{l - \text{rank}(H_C)}$$

We can relate the dimensionality of these codes with the rank of H_C. In particular

$$\text{rank}(H_C) = n - \dim(C) = l - \dim(C^T)$$

$$\therefore \dim(C^T) = (l - n) + \dim(C)$$

To further understand this transpose code, let us consider the following example:

Example 2.3. *Let C be an $[n, k, d]$ code such that its parity checks are all linearly independent. Therefore, the number of parity checks is $l = n - k$. By our equation above, for code C^T, we have that*

$$\boxed{\dim(C^T)} = (l - n) + k = (n - k) - n + k \boxed{= 0}$$

Let's understand why we end up with a transpose code of dimension 0. We know that for a codeword $v \in C^T$ we have $H_{C^T} v = H_C^T v = \overrightarrow{0}$. However, under the assumption that H_{C^T} has full column rank, by the rank-nullity theorem, we know that $\dim(Null(H_{C^T})) = 0$. Therefore, the dimension of the transpose code is 0.

2.2 Tensor Products of Classical Linear Codes

Definition 2.4 (Tensor Product Code)**.** Let C_1 be an $[n_1, k_1, d_1]$ classical linear error correcting code with generator matrix G_1 and let C_2 be an $[n_2, k_2, d_2]$ code with generator matirx G_2. We can specify C_1 and C_2 by their encoding maps:

$$C_1 : m \in \{0,1\}^{k_1} \xrightarrow{E_1} G_1 \cdot m$$

$$C_2 : m \in \{0,1\}^{k_2} \xrightarrow{E_2} G_2 \cdot m$$

Then the **tensor product code $\mathbf{C_1} \otimes \mathbf{C_2}$** is given by the following two-stage encoding map:

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1k_2} \\ x_{21} & x_{22} & \dots & x_{2k_2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k_1 1} & x_{k_1 2} & \dots & x_{k_1 k_2} \end{bmatrix}$$

The idea behind defining an input bitstring as a matrix is that each column corresponds to an input for the encoding map for C_1 and each row corresponds to an input for C_2's encoding map. Naturally, we perform the encoding map by sequentially applying E_1 to each column of X and E_2 to each row of the resulting matrix:

$$\boxed{X} \xrightarrow{E_1} \boxed{Y} = \begin{bmatrix} y_{11} & y_{12} & \dots & y_{1k_2} \\ y_{21} & y_{22} & \dots & y_{2k_2} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n_1 1} & y_{n_1 2} & \dots & y_{n_1 k_2} \end{bmatrix} \xrightarrow{E_2} \boxed{Z} = \begin{bmatrix} z_{11} & z_{12} & \dots & z_{1n_2} \\ z_{21} & z_{22} & \dots & z_{2n_2} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n_1 1} & z_{n_1 2} & \dots & z_{n_1 n_2} \end{bmatrix}$$

This sequential encoding can equivalently be stated in terms of multiplication by generator matrices of our constituent codes. Therefore, more succinctly

$$\boxed{Z = G_1 X G_2^T}$$

Interestingly, the above expression tells us that we didn't necessarily have to encode with G_1 followed by G_2. We just as easily could have applied the encoding maps in the opposite order. The result of this sequential encoding in terms of the matrix picture is the following: After applying G_1, every column is now a codeword of C_1 and after applying G_2, every row is a codeword of C_2. Let us now understand how this matrix is constructed from message bitstring m_1 of length k_1 and message bitstring m_2 of length k_2 being encoded by E_1 and E_2 respectively. Let $X = m_1 m_2^T$. Then

$$X = m_1 m_2^T = \begin{bmatrix} \square & \square & \cdots & \square \\ \square & \square & \cdots & \square \\ \vdots & \vdots & \boxed{m_1^{(i)} m_2^{(j)}} & \square \\ \square & \square & \cdots & \square \end{bmatrix}$$

where $m_1^{(i)} m_2^{(j)} = 1$ if $m_1^{(i)} = m_2^{(j)} = 1$, and 0 otherwise. Therefore, applying the generator matrices G_1 and G_2, we have

$$\begin{aligned} &G_1 X G_2^T \\ &= G_1(m_1 m_2^T)G_2 \\ &= (G_1 m_1)(G_2 m_2)^T \\ &= c_1 c_2^T \end{aligned}$$

for codewords $c_1 \in C_1$ and $c_2 \in C_2$. In general, however, the X matrix can be any binary matrix and isn't necessarily the outer product of two message vectors. The above construction simply provides intuition when starting from two linear codes C_1 and C_2.

Fact 2.5. *The tensor product code $C_1 \otimes C_2$ is an $[n_1 n_2, k_1 k_2, d_1 d_2]$ linear error correcting code.*

Proof. The code $C_1 \otimes C_2$ must be linear as we have specified a linear map that takes us from message matrix X to codeword $G_1 X G_2^T$ for generator matrices G_1 and G_2 corresponding to the constituent codes C_1 and C_2.

Suppose $X \neq 0$. When flattened, X has length $k_1 k_2$ and the encoding map takes it to matrix Z which when flattened has length $n_1 n_2$. This gives us the first two parameters of our code. For linear codes, we know that the minimum Hamming weight of any codeword is the distance of our code. Therefore, for each of C_1 and C_2, let c_1 and c_2 be codewords with Hamming weight d_1 and d_2. Taking the outer product of these codewords, we get that the resulting matrix $c_1 c_2^T$ must have a 1 at position (i, j) if and only if $c_1^{(i)} = c_2^{(j)} = 1$. Therefore, the resulting codeword of $C_1 \otimes C_2$ must have Hamming weight of at least $d_1 d_2$, as $c_1^{(i)} = c_2^{(j)} = 1$ at at least $d_1 d_2$ positions.

Let us consider an alternate proof. Suppose again we start with $X \neq 0$. Then encoding each column of X via encoding map E_1 will leave us with columns of length n_1 that each have Hamming weight at least d_1. Subsequently encoding each row of the resulting matrix $Y = G_1 X$ will result in rows of length n_2, each with Hamming weight at least d_2. Therefore, we have d_1 rows that are encoded to encoded bitstrings of Hamming weight at least d_2, resulting in an encoded matrix Z with Hamming weight at least $d_1 d_2$. $\square$

Let us consider an alternate characterization of the tensor product code

Fact 2.6. $C_1 \otimes C_2 = \{Z \in \{0,1\}^{n_1 \times n_2} : \textit{cols} \in C_1 \quad \textit{rows} \in C_2\}$

Proof. Let us begin by considering the forward inclusion. This follows from the fact that we are applying two sequential encodings on columns and rows of X and the order in which we apply these encodings doesn't matter. Our last step can be to apply E_2 to every row of $G_1 X$, which gives us that every row of $Z = G_1 X G_2^T$ is a codeword of C_2. Equivalently, the last step can be to apply E_1 to every column of $X G_2^T$, such that every resulting column of $Z = G_1 X G_2^T$ is a codeword of C_1. It therefore follows that $C_1 \otimes C_2 \subseteq \{Z \in \{0,1\}^{n_1 \times n_2} : \text{cols} \in C_1 \quad \text{rows} \in C_2\}$

Let us now prove the reverse inclusion. Suppose we have some matrix Z that satisfies the above property. Then, to prove that $Z \in C_1 \otimes C_2$, we must show that Z passes the parity checks of our tensor product code. From Figure 2, we have that in our new Tanner graph, we have an edge between a check node $(P_1^{(l)}, V_2^{(j)})$ and variable node $(V_1^{(i)}, V_2^{(j)})$ if there existed an edge between $P_1^{(i)}$ and $V_1^{(i)}$ in our original code (There is a slight abuse of notation here but the first superscript i should be different from the second i). In matrix form, if we are fixing the index j for our variable node's second element, then it implies that our check is running across a column of Z. This corresponds to checking if a column of Z is in the code C_1. This, however, is true by definition of Z. Similarly, we also have an edge between a check node $(V_1^{(i)}, P_2^{(m)})$ and $(V_1^{(i)}, V_2^{(j)})$. For this check, we are fixing the row index of Z given by i and performing a check across a row of Z with parity check $P_2^{(m)}$. Once again, by definition, the rows of Z are codewords of C_2, all of these checks pass. Therefore, such a matrix Z must be a codeword for $C_1 \otimes C_2$. $\square$

This is a very nice fact because we now have a sense of the parity checks of our tensor product code. Suppose code C_1 has variables V_1 and parity checks P_1 and code C_2 has variables V_2 and parity checks C_2. For some $Z \in \{0,1\}^{n_1 \times n_2}$, if we want to know if Z is in $C_1 \otimes C_2$, this requires performing each of P_1 on each column of Z and each of P_2 on each row of Z. We can visualize these checks in the following way:

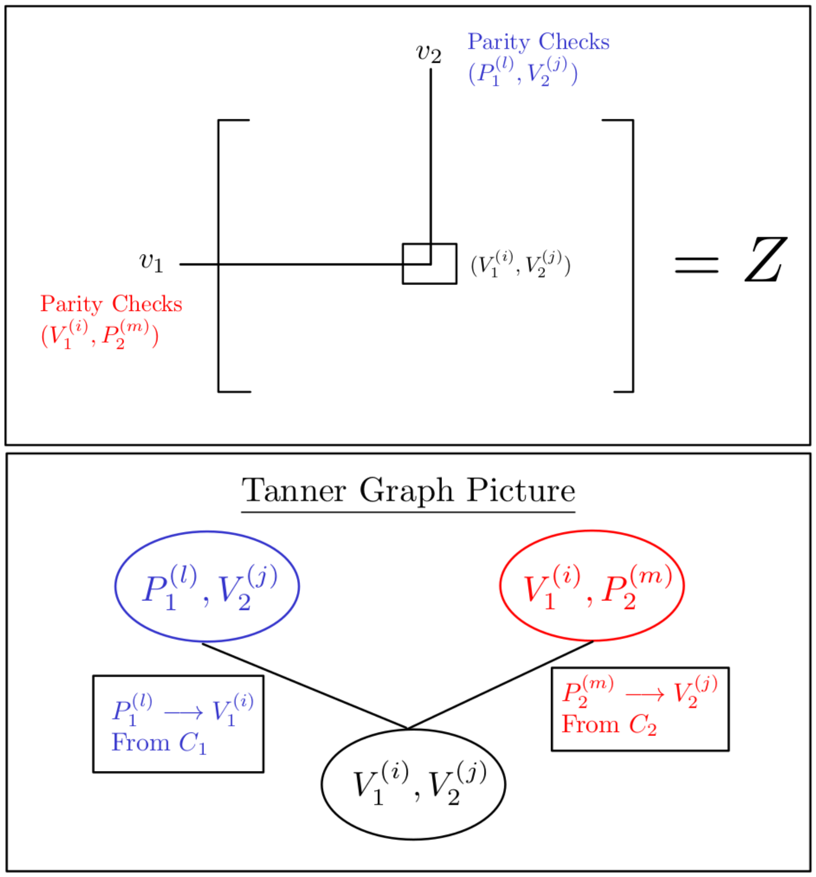


Figure 2: Parity checks corresponding to each variable node of tensor product code $C_1 \otimes C_2$ and corresponding Tanner graph structure.

Claim 2.7 (Generator Matrix for $C_1 \otimes C_2$ is $G_1 \otimes G_2$). *Proof.* Consider a matrix $X \in \{0,1\}^{k_1 \times k_2} = m_1 m_2^T$, for $m_1 \in \{0,1\}^{k_1}$ and $m_2 \in \{0,1\}^{k_2}$. Then, viewing X as a flattened vector as opposed to a matrix, we can instead write $X_{\text{flat}} = m_1 \otimes m_2$, as a flattened vector. This is just the tensor product of our two message vectors. Therefore, the natural picture for encoding X_{flat} via the encoding for our tensor product code $C_1 \otimes C_2$ is to apply generator matrix G_1 to the first tensor factor m_1 and G_2 to th esecond tensor factor m_2.

$$(G_1 \otimes G_2)(m_1 \otimes m_2)$$

$$= (G_1 m_1 \otimes G_2 m_2)$$

Re-expressed in the matrix picture, we get that the above expression is equal to

$$G_1 m_1 (G_2 m_2)^T$$

$$= G_1 m_1 m_2^T G_2^T$$

$$= G_1 X G_2^T$$

which is equivalent to the matrix codeword Z we had derived above. $\square$

3 Quantum Product Codes

Now that we've established tensor product codes in the classical setting, let us work towards defining an analogous structure in the quantum setting. In fact, we'll see the classical tensor product code encapsulated in this quantum construction.

The key idea here is arises from the following observation: For topological quantum codes, we were working with 2-dimensional objects (faces of the lattice), 1-dimensional objects (edges of the lattice), and 0-dimensional objects (vertices of the lattice). In this picture, our Z checks and X checks corresponded to faces and vertices of these lattices, respectively, while our qubits corresponded to edges of this lattice. Pulling this idea back to the classical setting, we now are working with collections of only 1 and 0-dimensional objects, as we only have one type of error. The driving question here is then **how can we take classical codes, which are these 1-dimensional objects, and generalize them to quantum codes, which can be represented by 2-dimensional objects?** To illustrate this idea, let us begin with the classical repetition code:

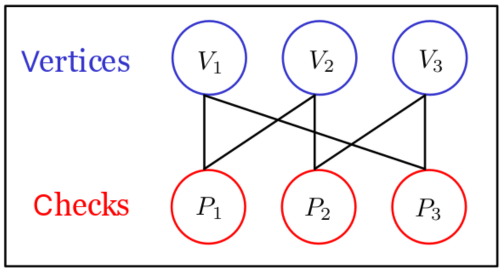
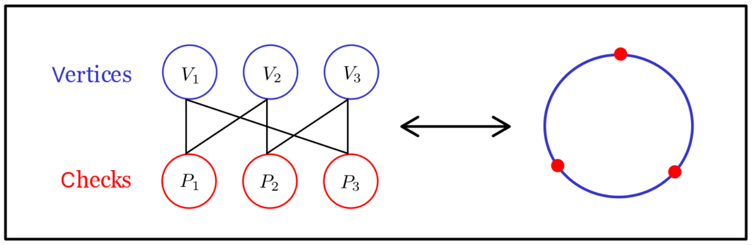


Figure 3: **Classical 3-Bit Repetition Code** Each check is connected to variable nodes in a cyclic fashion. Notice that the last check is not really needed, as it is dependent on the first two checks. However, in transitioning to the quantum picture we hang on to it.

Let us now associate this with a 1-dimensional object as we were claiming:



The red vertices can be thought of as our 0-dimensional objects or our checks and the 1-dimensional objects are our edges or bits. A nice property of this particular code is that it really doesn't matter which object we denote to be our variables or our checks. Looking at the Tanner graph, if we were to instead consider the transpose code, we get the exact same code back. This is evident from the cellulation of the circle as well.

Let's propose using these 1-dimensional objects as building blocks for a 2-dimensional object that corresponds to a quantum code. One natural transformation to do is to take a product of our 1-dimensional objects. In this case, it turns out the correct product to apply is a Cartesian product between two of these one-dimensional objects (See Figure 4 below).

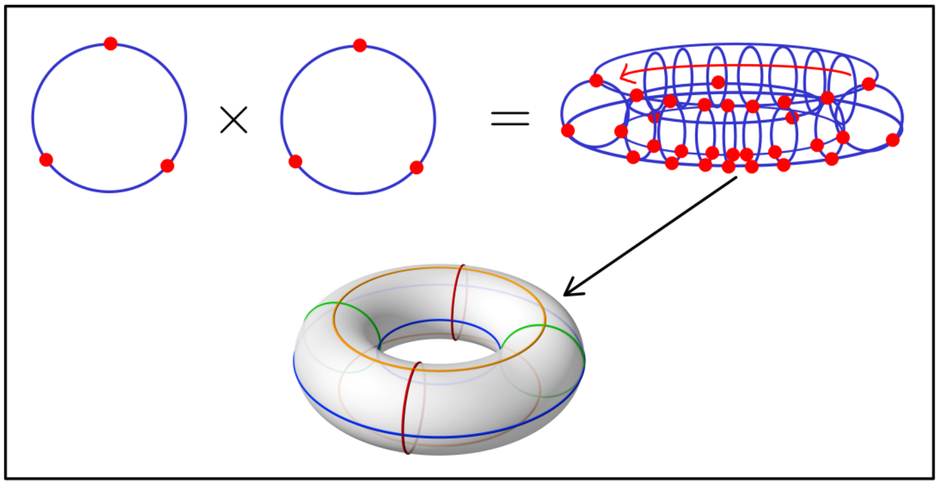
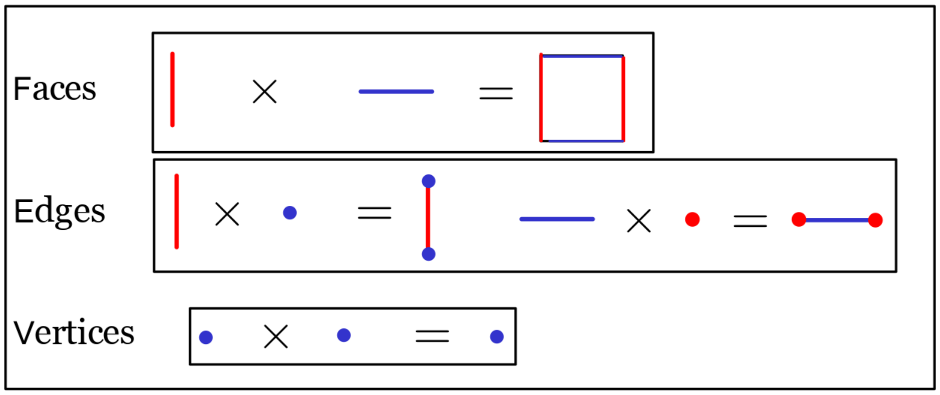


Figure 4: **Construction of Toric Code**

We can generalize the cellulation of our one-dimensional objects to the 2-dimensional setting. To do so, let us first define some product rules between 1-chains and 0-chains:



With these rules established, let us work through taking the product of the 1-dimensional objects' cellulations. Since we're working with topological surfaces here, it is valid to consider deformed versions of our 1D cellulations for the purposes of this calculation.

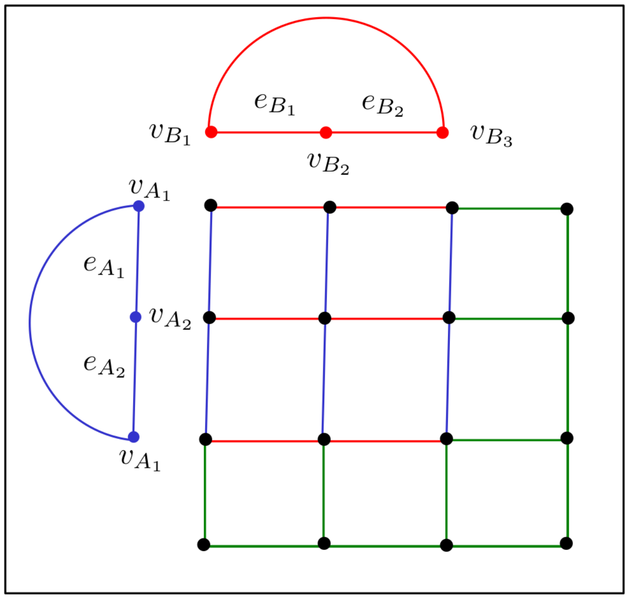


Figure 5: Using the product rules above what results is the cellulation of a torus. An important feature to note is that the green edges here are a result of multiplying the semi-circle edges (1-chains) with other 1-chains and 0-chains.

Ultimately, we get the cellulation of a torus (we are restricting ourselves to a 2D plane so it doesn't look like a torus, however we associate the top edge of the lattice with the bottom edge, and associate the left edge with the right edge). A nice feature of this picture is that the individual parity check relationships from each of our repetition codes is maintained in the toric code picture. Naturally, this toric code picture carries over other aspects of our constituent classical repetition codes as well. Say we wanted to figure out the distance of our resulting toric code. Looking back at the repetition code, the distance corresponds to the minimum Hamming weight of a non-zero codeword, which in this case is 3 for codeword `111`. Recall for the toric code that error operators that correspond to the distance of the code look like non-contractible loops around the torus (See the red and blue loops on Torus at the bottom of Figure 4). We can see this as arising from the codewords `111` from each of our repetition code, resulting in non-trivial loops in the toric code picture. This construction—taking the product of two classical codes—is one example of, and general formulation for **Hypergraph Product Codes**

3.1 The Hypergraph Product Code

We've already seen the Shor-9 Qubit code as a way to generalize from classical to quantum error correction via concatenation of encoding maps. By comparison, the hypergraph product construction allows us to choose any two classical linear codes, take their product, and obtain a quantum code. More importantly, as we'll see, we actually end up with quantum CSS codes. This is crucial because we no longer have to search for classical codes whose parity check matrices satisfy the CSS orthogonality conditions. It is an aspect that is baked into our resulting quantum code.

Definition 3.1 (The Hypergraph Product Code)**.** Let (A_1, A_0) be a chain complex with boundary map $\partial_A : A_1 \to A_0$. Here A_1 contains 1-dimensional objects (edges) denoted e_A and A_0 contains 0-dimensional objects (vertices) denoted v_A. We define an analogous chain complex (B_1, B_0) with boundary map $\partial_B : B_1 \to B_0$ with edges e_B and vertices v_B. Therefore, the hypergraph product code is given by the following length-2 chain complex:

$$\begin{tikzcd}
\text{2-Dim} & & (e_A, e_B) \arrow[dl, dash, "\partial_A \otimes \mathbb{I}"'] \arrow[dr, dash, "\mathbb{I} \otimes \partial_B"] & & \text{Z Checks} \\
\text{1-Dim} & (v_A, e_B) \arrow[dr, dash, "\mathbb{I} \otimes \partial_B"'] & \oplus & (e_A, v_B) \arrow[dl, dash, "\partial_A \otimes \mathbb{I}"] & \text{Qubits} \\
\text{0-Dim} & & (v_A, v_B) & & \text{X Checks}
\end{tikzcd}$$

Figure 6: **Hypergraph Product Code Construction** Length-2 chain complex corresponding to hypergraph product of two length-1 chain complexes of classical linear codes.

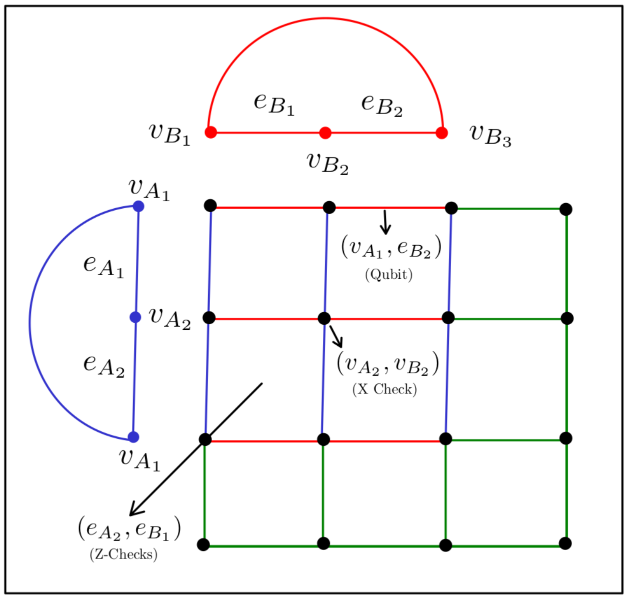


Figure 7: We can use the hypergraph product construction of the toric code to see how checks (vertices) and variables (edges) from the repetition codes relate to our two types of checks (faces and vertices) and variables (edges) of our resulting quantum code:

What's interesting to see in Figure 6 is that the tensor product code from the classical setting (See Figure 2) is encased in this hypergraph product code (specifically the transpose of the tensor product code as the role of the variable nodes and the check nodes have been reversed). One important difference to note is that Figure 2 is a Tanner graph of our classical code while Figure 6 is a chain complex of our quantum code. To make the connection with the classical setting even stronger, an alternate picture of the hypergraph product code is an aspect of it's name—hypergraph. In fact, Tillich and Zemor prescribed the construction of these quantum codes by taking Cartesian graph products of Tanner graphs of classical codes, which results in a Tanner graph that has an analagous structure to the chain complex in Figure 6. This can also be viewed as a tripartite variant of the Tanner graph if we consider the X checks and Z checks separately.

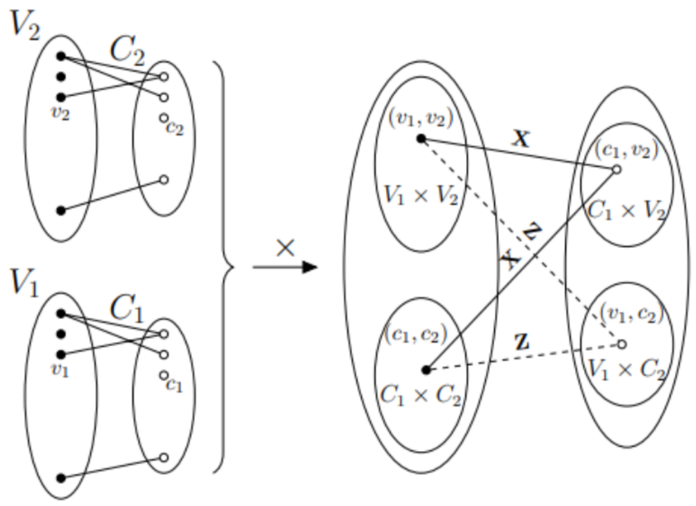


Figure 8: **Cartesian Product of Tanner Graphs - Hypergraph Product Construction**

Let us now consider what happens if our input classical codes are **low-density parity check** codes. For an LDPC code, we have a degree l Tanner graph if each check node is connected to at most l variable nodes and each variable node is connected to at most l check nodes. Examining Figure 6, we have that each of the variable nodes will be connected to at most l of the Z checks and at most l of the X checks. Likewise, each of the X and Z checks will be connected to l qubits on the left summand of our direct sum and l qubit on the right summand. Therefore, the resulting Tanner graph for our Quantum LDPC code will be of degree $2l$.

3.2 Proving CSS Orthogonality of Hypergraph Product Codes

Let us prove that the codes resulting from this hypergraph product do indeed satisfy the CSS orthogonality condition. Recall, for a quantum code specified by X parity check matrix

H_X and Z parity check matrix H_Z, it must be the case that

$$H_X H_Z^T = 0$$

Equivalently, we have that the rows of H_Z are codewords of the code specified by H_X and vice versa.

To understand what our parity checks are, let's compute the boundary operators for our length-2 chain complex in Figure 6, starting with ∂_2. Going from (e_A, e_B) to (v_A, e_B), we are applying the boundary operator ∂_A on the first element and leaving the second element unchanged, giving us $\partial_A \otimes \mathbb{I}$. Likewise, from (e_A, e_B) to (e_A, v_B), we are applying an identity to our first element and boundary operator ∂_B to the second, giving us $\mathbb{I} \otimes \partial_B$. We can use this same reasoning to define ∂_1 going from the 1D objects to the 0D objects, using the fact that maps from edges (1D objects) to vertices (0D objects) of type A or B invoke the boundary operator ∂_A or ∂_B respectively. Else, if the element is left unchanged going from the 1D layer to the 0D layer, we apply an identity. In totality, we can therefore define ∂_2 and ∂_1 as

$$\partial_2 = \begin{array}{c} \\ (v_A, e_B) \\ (e_A, v_B) \end{array} \begin{array}{c} (e_A, e_B) \\ \left[\begin{array}{c} \partial_A \otimes \mathbb{I} \\ \hdashline \mathbb{I} \otimes \partial_B \end{array}\right] \end{array}$$

$$\partial_1 = \begin{array}{cc} & \begin{array}{cc} (v_A, e_B) & (e_A, v_B) \end{array} \\ (v_A, v_B) & \left[\begin{array}{c:c} \mathbb{I} \otimes \partial_B & \partial_A \otimes \mathbb{I} \end{array}\right] \end{array}$$

Given that these are contiguous boundary operators of chain complexes, a property that necessarily holds true is

$$\partial_i \partial_{i-1} = 0$$

In this particular case, we have

$$\partial_2 \partial_1$$

$$= (\partial_A \otimes \mathbb{I})(\mathbb{I} \otimes \partial_B) + (\mathbb{I} \otimes \partial_B)(\partial_A \otimes \mathbb{I})$$

following from the fact that we are multiplying block matrices.

$$= (\partial_A \otimes \partial_B) + (\partial_A \otimes \partial_B)$$

$$= 0$$

as all of our addition is being done modulo 2. Thus, these boundary operators satisfy the CSS orthogonality condition. In particular, letting

$$H_X = \partial_2$$

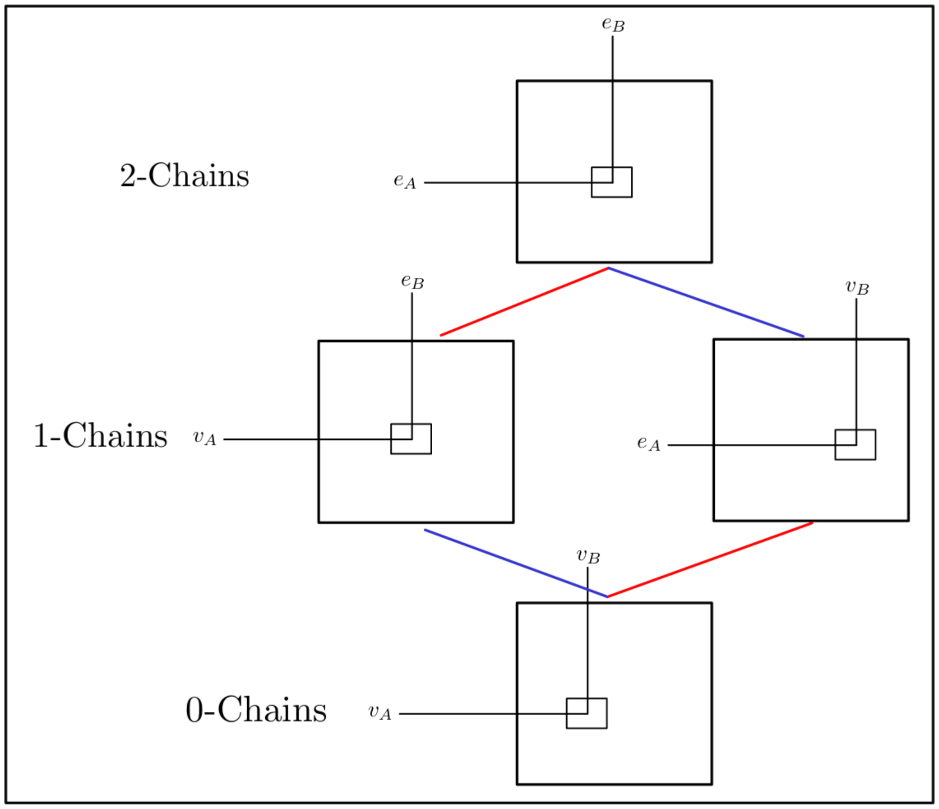
and

$$H_Z = \partial_1^T$$

we get

$$H_X H_Z^T = \partial_2 \partial_1 = 0$$

One final informative picture to understand the action of these boundary operators is the following:



At each level, we can imagine that each of the black boxes represent matrices where elements of these matrices tell us whether a particular pair of vertices and edges from A and B are included in our 0 and 1-chains, respectively. The red and blue edges going between these boxes represent the boundary operators, whereby the red edges take us from 1-chains to boundaries of these 1-chains in A and blue edges take us from 1-chains to boundaries of these 1-chains in B. This picture is very reminiscent to the tensor product of classical codes (See Figure 2). Furthermore, the boundary operators ∂_2 and ∂_1 we wrote down above tell us exactly how 2-chains are mapped to their boundary 1-chains and how 1-chains are mapped to their boundary 0-chains.

References

[BPT10] Sergey Bravyi, David Poulin, and Barbara Terhal. Tradeoffs for reliable quantum information storage in 2d systems. *Phys. Rev. Lett.*, 104:050503, Feb 2010. 1.1

[Kit97] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, dec 1997. 1

[Kit03] Alexei Kitaev. Fault-tolerant quantum computation by anyons. *Annals of physics*, 303(1):2–30, 2003. 1, 1

[TZ14] Jean-Pierre Tillich and Gilles Zémor. Quantum ldpc codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2014. 2