Quantum Error Correction: A brief tutorial

Eddie Schoute[1,2]

2024

[1]Los Alamos National Laboratory

[2]IBM Research

eddie.schoute@ibm.com, LA-UR-24-25824

Classical error correction

Repetition code

Repetition code

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

Repetition code

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

We can detect 1–2 errors

$$001, \qquad 101.$$

Can correct at most 1 error.

Repetition code

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

We can detect 1–2 errors

$$001, \qquad 101.$$

Can correct at most 1 error.

Correction of i.i.d bit flips with probability p gives $O(p^2)$ error probability.

Encoding and decoding

For protection, we encode our logical information in a code.

$$001 \xrightarrow{\text{encoding}} (000)(000)(111)$$

Encoding and decoding

For protection, we encode our logical information in a code.

$$001 \xrightarrow{\text{encoding}} (000)(000)(111)$$

Now if errors occur, our decoding can correct them

$$(000)(000)(111) \xrightarrow{\text{error channel}} (100)(000)(101) \xrightarrow{\text{decoding}} 001$$

Success if 1 or less errors occur in each code block.

Code threshold

Threshold are generally computed numerically.
Tells us when it's worth using a code.

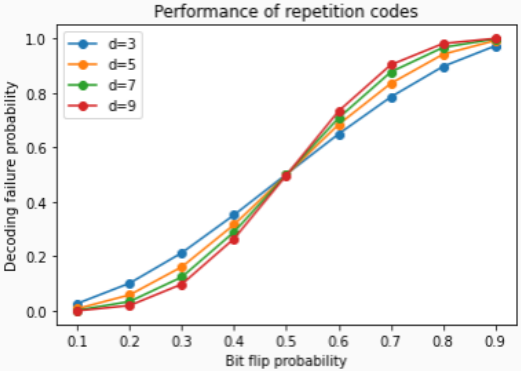


Figure: Threshold of p=0.5. Image [Del23]

Threshold are generally computed numerically.
Tells us when it's worth using a code.

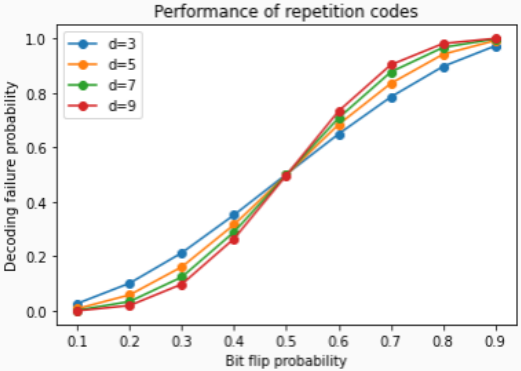


Figure: Threshold of p=0.5. Image [Del23]

Threshold are generally computed numerically.
Tells us when it's worth using a code.

Phase error threshold

Our simple repetition code cannot detect Z errors, so it has 0 threshold for general quantum error channels.

Some codes don't have a threshold (but are still useful)!

Classical error correction

Classical linear codes

Linear code

Given code words $C \subseteq \mathbb{Z}_2^n$, then C is a *linear code* if $x, y \in C \implies x + y \in C$.

Linear code

Given code words $C \subseteq \mathbb{Z}_2^n$, then C is a *linear code* if $x, y \in C \implies x + y \in C$.

Recall the repetition code has code words

$$C = \{000, 111\}$$

and we can verify C is linear. (Note: $1 + 1 = 0 \in \mathbb{Z}_2$.)

Linear code

Given code words $C \subseteq \mathbb{Z}_2^n$, then C is a *linear code* if $x, y \in C \implies x + y \in C$.

Recall the repetition code has code words

$$C = \{000, 111\}$$

and we can verify C is linear. (Note: $1 + 1 = 0 \in \mathbb{Z}_2$.)

Since the repetition code is linear, we can consider logical bits as a basis $\{x_i\}_{i=1}^k$ and construct code words through the *generator matrix*

$$G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix},$$

i.e., $\begin{pmatrix} 0 \end{pmatrix} G = 000$ and $\begin{pmatrix} 1 \end{pmatrix} G = 111$.

Generator and parity matrix

We have encoded $k = 1$ logical bits using $n = 3$ physical bits using the reptition code. Therefore, the generator matrix has rank 1 (out of 3)

$$G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix},$$

Generator and parity matrix

We have encoded $k = 1$ logical bits using $n = 3$ physical bits using the reptition code. Therefore, the generator matrix has rank 1 (out of 3)

$$G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix},$$

the remaining two dimensions are the orthogonal space spanned by a *parity check matrix*

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

We have encoded $k = 1$ logical bits using $n = 3$ physical bits using the reptition code. Therefore, the generator matrix has rank 1 (out of 3)

$$G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix},$$

the remaining two dimensions are the orthogonal space spanned by a *parity check matrix*

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

Now it is easy to check if we have a valid code word, since $H\vec{x} = 0$ for any valid code word, e.g.,

$$H \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 0, \qquad H \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 1. \tag{1}$$

[n,k,d] notation

We can summarily describe linear codes using the notation $[n, k, d]$:

- For n physical bits
- Encoding k logical bits
- With d bit flips between codewords ($|x - y| \geq d$ for all $x, y \in C$).

The repetition code is an $[n, 1, n]$ code.

Quantum error correction

Stabilizer codes

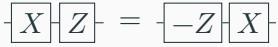
Recall a parity check matrix of the repetition code

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

Recall a parity check matrix of the repetition code

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

In a quantum computer, we can check for bit flips using a Z measurement. To see that, note



so if we measure Z its parity will flip when a bit has flipped.

Recall a parity check matrix of the repetition code

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

In a quantum computer, we can check for bit flips using a Z measurement. To see that, note

$$-\boxed{X}-\boxed{Z}- \;=\; -\boxed{-Z}-\boxed{X}-$$

so if we measure Z its parity will flip when a bit has flipped.

Now we can rewrite the classical repetition code as

$$H = \begin{pmatrix} Z & Z & 0 \\ 0 & Z & Z \end{pmatrix}.$$

Implementing a stabilizer measurement

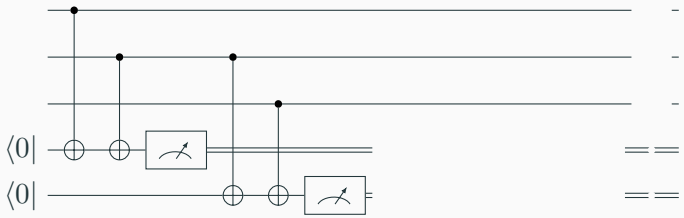
How do we implement

$$H = \begin{pmatrix} Z & Z & 0 \\ 0 & Z & Z \end{pmatrix} ?$$

How do we implement

$$H = \begin{pmatrix} Z & Z & 0 \\ 0 & Z & Z \end{pmatrix} ?$$

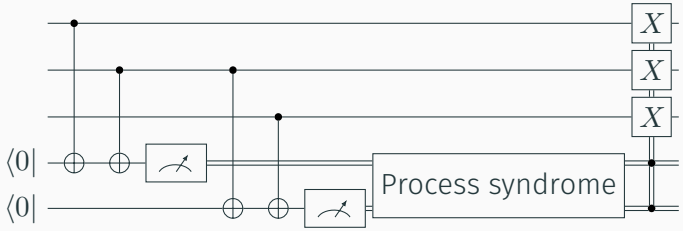
We can measure $Z_1 Z_2$ and $Z_2 Z_3$ by



How do we implement

$$H = \begin{pmatrix} Z & Z & 0 \\ 0 & Z & Z \end{pmatrix} ?$$

We can measure $Z_1 Z_2$ and $Z_2 Z_3$ by



We can correct any one X error.

[[7,1,3]] code

Time to construct your first quantum error-correcting code!

Let's consider the (classical) Hamming code (a $[7, 4, 3]$ code) with stabilizers

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

[[7,1,3]] code

Time to construct your first quantum error-correcting code!

Let's consider the (classical) Hamming code (a $[7, 4, 3]$ code) with stabilizers

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

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

Now we can correct any one X or Z or $XZ = Y$ error. This is a $[[7, 1, 3]]$ code (the Steane code).

Time to construct your first quantum error-correcting code!

Let's consider the (classical) Hamming code (a $[7, 4, 3]$ code) with stabilizers

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

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

Now we can correct any one X or Z or $XZ = Y$ error. This is a $[[7, 1, 3]]$ code (the Steane code).

Definition

A *CSS code* consists of solely Z and solely X stabilizers.

The stabilizer generators of the $[[7,1,3]]$ code are

$$\begin{pmatrix} Z & Z & Z & Z & 0 & 0 & 0 \\ Z & Z & 0 & 0 & Z & Z & 0 \\ Z & 0 & Z & 0 & Z & 0 & Z \\ X & X & X & X & 0 & 0 & 0 \\ X & X & 0 & 0 & X & X & 0 \\ X & 0 & X & 0 & X & 0 & X \end{pmatrix}$$

and they protect information from Pauli errors, but is there even a logical qubit here?

The stabilizer generators of the $[[7, 1, 3]]$ code are

$$\begin{pmatrix} Z & Z & Z & Z & 0 & 0 & 0 \\ Z & Z & 0 & 0 & Z & Z & 0 \\ Z & 0 & Z & 0 & Z & 0 & Z \\ X & X & X & X & 0 & 0 & 0 \\ X & X & 0 & 0 & X & X & 0 \\ X & 0 & X & 0 & X & 0 & X \end{pmatrix}$$

and they protect information from Pauli errors, but is there even a logical qubit here?

All stabilizers commute. So there is a 2^{7-6} dimensional subspace that is the common $+1$ eigenstate of all stabilizers. A qubit!

The stabilizer generators of the $[[7, 1, 3]]$ code are

$$\begin{pmatrix} Z & Z & Z & Z & 0 & 0 & 0 \\ Z & Z & 0 & 0 & Z & Z & 0 \\ Z & 0 & Z & 0 & Z & 0 & Z \\ X & X & X & X & 0 & 0 & 0 \\ X & X & 0 & 0 & X & X & 0 \\ X & 0 & X & 0 & X & 0 & X \end{pmatrix}$$

and they protect information from Pauli errors, but is there even a logical qubit here?

All stabilizers commute. So there is a 2^{7-6} dimensional subspace that is the common $+1$ eigenstate of all stabilizers. A qubit!

The logical operations commute with all stabilizers:

$$\bar{Z} = ZZZZZZZ, \qquad \bar{X} = XXXXXXX$$

We can compute what the logical $|\bar{0}\rangle$ state is by projecting into the $+1$ eigenspace of all stabilizers.

Let's start with

$$|0000000\rangle \xrightarrow{Z \text{ stabilizers}} |0000000\rangle.$$

We can compute what the logical $|\bar{0}\rangle$ state is by projecting into the $+1$ eigenspace of all stabilizers.

Let's start with

$$|0000000\rangle \xrightarrow{Z \text{ stabilizers}} |0000000\rangle.$$

Now let's measure $XXXXIII$

$$\begin{aligned} |0000000\rangle &\to \frac{1}{\sqrt{2}}(|0000000\rangle + (-1)^a XXXXIII|000000\rangle) \\ &\xrightarrow{a=0} \frac{1}{\sqrt{2}}(|0000000\rangle + |1111000\rangle) \end{aligned}$$

We can fix the $a = 1$ outcome by applying a Z_1.

We can compute what the logical $|\bar{0}\rangle$ state is by projecting into the $+1$ eigenspace of all stabilizers.

Let's start with

$$|0000000\rangle \xrightarrow{Z \text{ stabilizers}} |0000000\rangle.$$

Now let's measure $XXXXIII$

$$|0000000\rangle \to \frac{1}{\sqrt{2}}(|0000000\rangle + (-1)^a XXXXIII|000000\rangle)$$
$$\xrightarrow{a=0} \frac{1}{\sqrt{2}}(|0000000\rangle + |1111000\rangle)$$

We can fix the $a = 1$ outcome by applying a Z_1.

Question

Check that the resulting state still is $+1$ eigenstate of Z stabilizers.

We can compute what the logical $|\bar{0}\rangle$ state is by projecting into the $+1$ eigenspace of all stabilizers.

Let's start with

$$|0000000\rangle \xrightarrow{Z \text{ stabilizers}} |0000000\rangle.$$

Now let's measure $XXXXIII$

$$|0000000\rangle \to \frac{1}{\sqrt{2}}(|0000000\rangle + (-1)^a XXXXIII|000000\rangle)$$
$$\xrightarrow{a=0} \frac{1}{\sqrt{2}}(|0000000\rangle + |1111000\rangle)$$

Question

Check that the resulting state still is $+1$ eigenstate of Z stabilizers.

We can fix the $a = 1$ outcome by applying a Z_1.

Applying the remaining two X stabilizers gives us logical $|\bar{0}\rangle$. Then $|\bar{1}\rangle = \bar{X}|\bar{0}\rangle$.

Surface codes

Surface codes

The stabilizers of a surface code are given graphically.

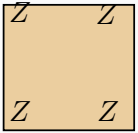


Figure: A $ZZZZ$ stabilizer

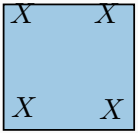


Figure: An $XXXX$ stabilizer

The stabilizers of a surface code are given graphically.

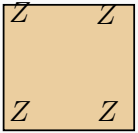


Figure: A $ZZZZ$ stabilizer

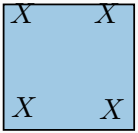


Figure: An $XXXX$ stabilizer

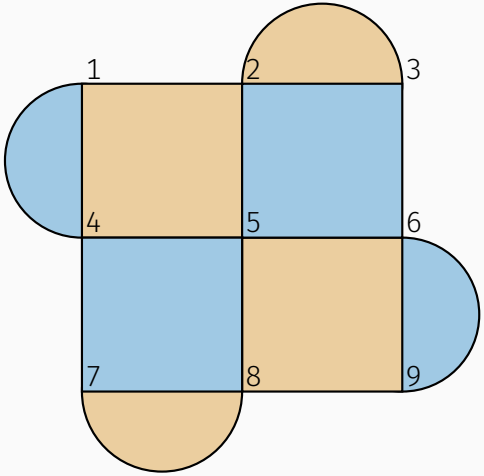


Figure: Surface code (a [[9,1,3]] code)

The stabilizers of a surface code are given graphically.

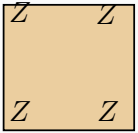


Figure: A $ZZZZ$ stabilizer

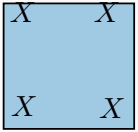


Figure: An $XXXX$ stabilizer

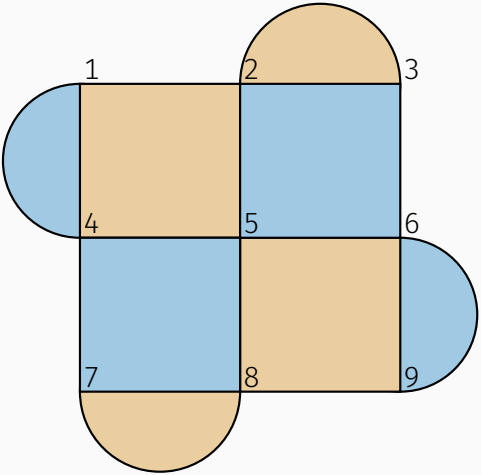


Figure: Surface code (a [[9,1,3]] code)

There are logical operations supported on 3 qubits:

1. $X_4 X_5 X_6$ is a logical $\bar{X}$.

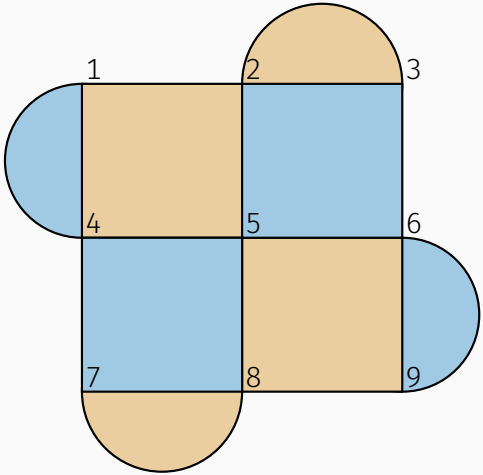


Figure: Surface code (a [[9,1,3]] code)

There are logical operations supported on 3 qubits:

1. $X_4X_5X_6$ is a logical $\bar{X}$.
2. $Z_2Z_5Z_8$ is a logical $\bar{Z}$.

Check for yourself that this works.

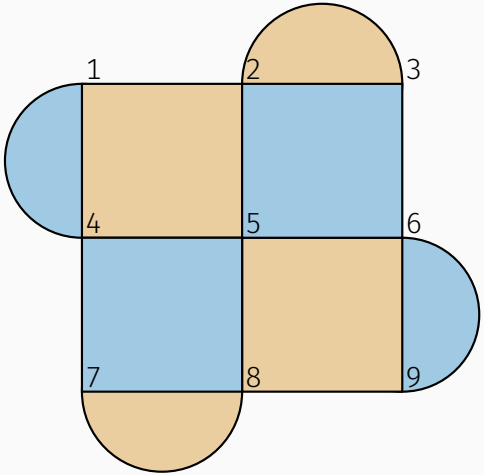


Figure: Surface code (a [[9,1,3]] code)

There are logical operations supported on 3 qubits:

1. $X_4X_5X_6$ is a logical $\bar{X}$.
2. $Z_2Z_5Z_8$ is a logical $\bar{Z}$.

Check for yourself that this works.

Question

Are there other logical operations that could work?

Scaling the surface code distance

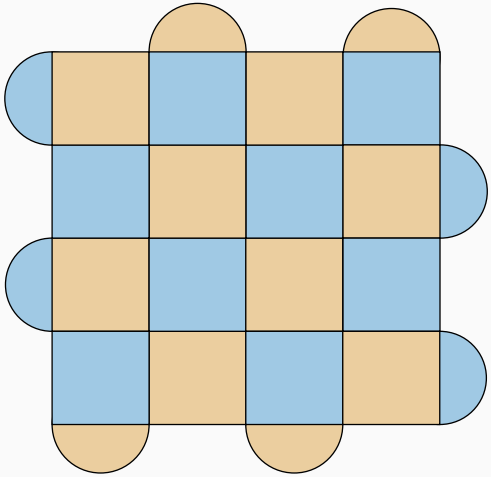


Figure: Distance 5 surface code. (What is [[n,k,d]] here?)

Scaling the surface code distance

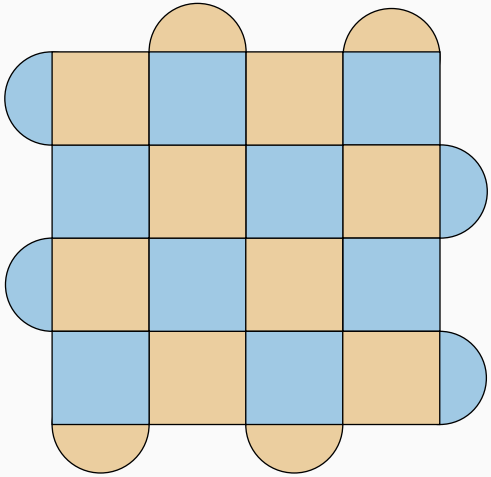


Figure: Distance 5 surface code. (What is [[n,k,d]] here?)

Surface codes have a high error-correction threshold of about 1%.

Scaling the surface code distance

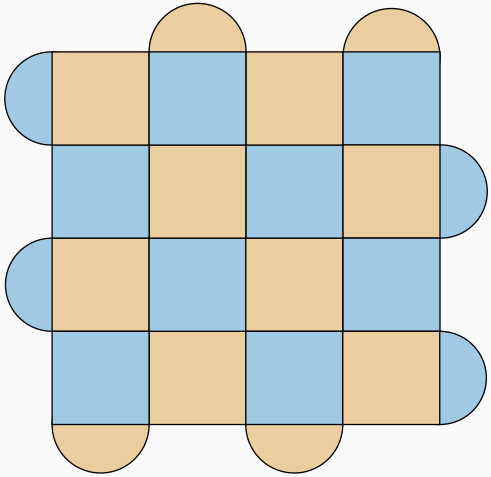


Figure: Distance 5 surface code. (What is [[n,k,d]] here?)

Surface codes have a high error-correction threshold of about 1%.
New *LDPC* codes have a better *encoding rate* than

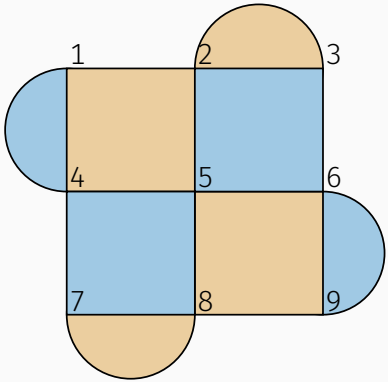
$$\frac{k}{n+c} = O\left(\frac{1}{d^2}\right)$$

for c ancilla measurement qubits.

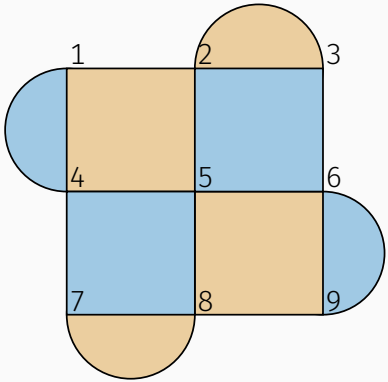
Surface codes

How to compute with surface codes

- We have seen how to perform logical $\bar{Z} = Z_2 Z_5 Z_8$ and $\bar{X} = X_4 X_5 X_6$.



- We have seen how to perform logical $\bar{Z} = Z_2 Z_5 Z_8$ and $\bar{X} = X_4 X_5 X_6$.
- Measuring the state in X/Z basis can be performed by physical X/Z measurements on each qubit.



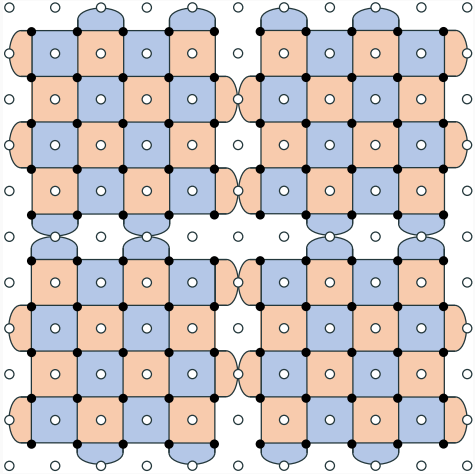


Figure: By tiling patches in the plane we can make an architecture out of surface codes.

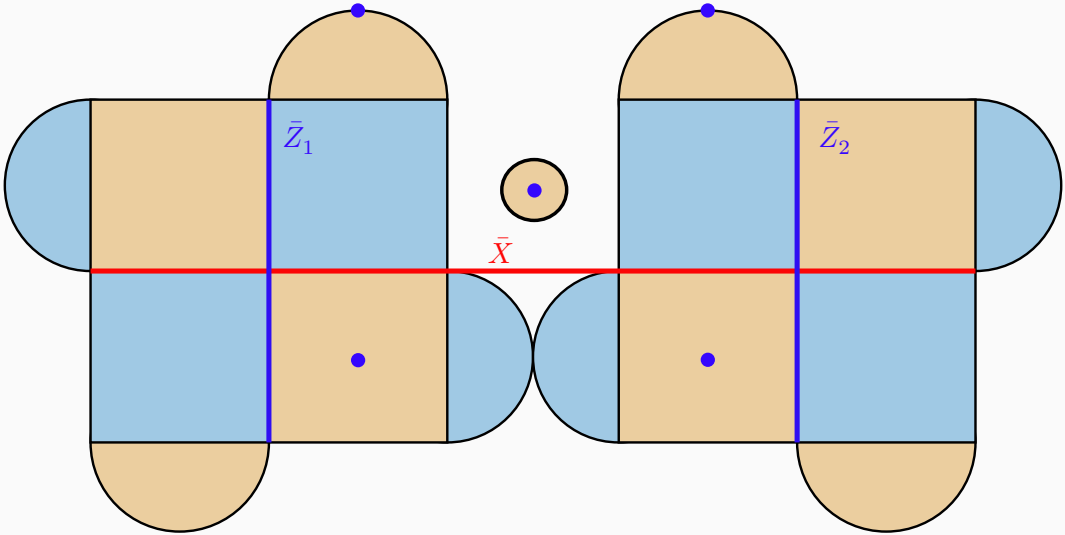


Figure: A joint ZZ measurement by horizontal merge.

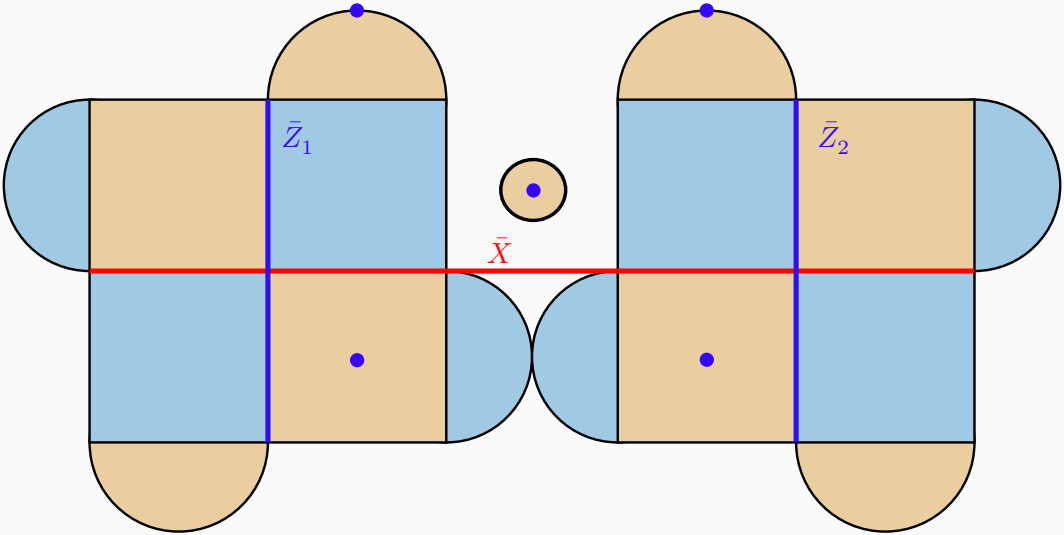


Figure: A joint ZZ measurement by horizontal merge. We can perform an XX measurement vertically.

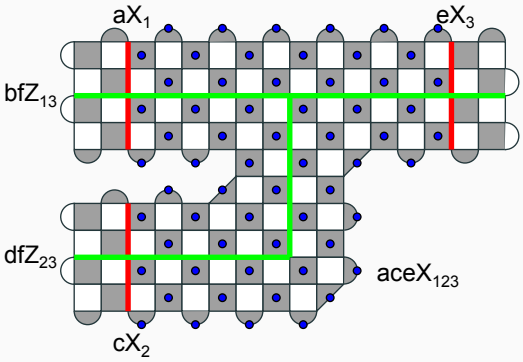


Figure: An XXX measurement by merging multiple patches [Aus19].

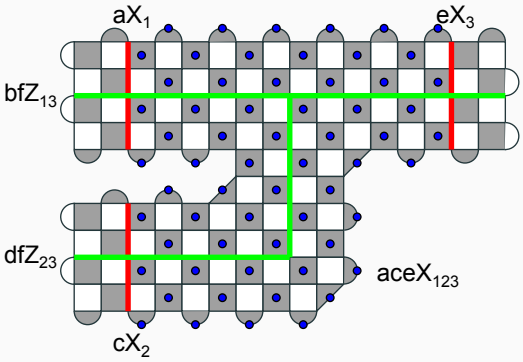


Figure: An XXX measurement by merging multiple patches [Aus19].

- We can do more complicated lattice surgery $\implies$ optimizations for particular operations!

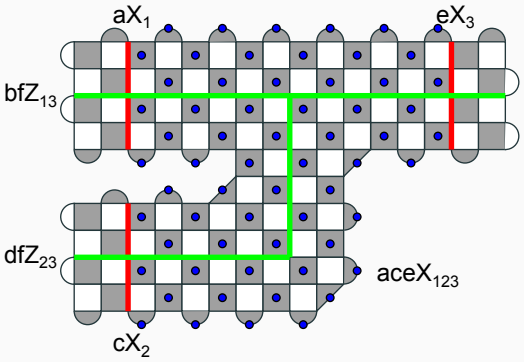


Figure: An XXX measurement by merging multiple patches [Aus19].

- We can do more complicated lattice surgery $\implies$ optimizations for particular operations!
- But two-qubit lattice surgery is sufficient to implement a CX gate (CNOT).

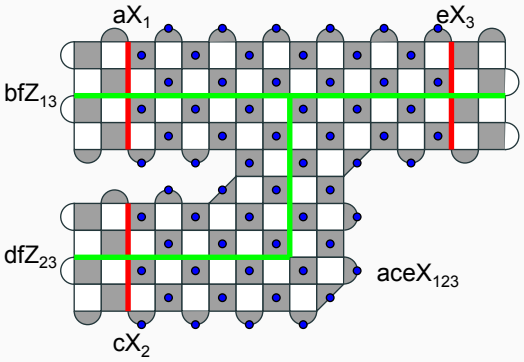


Figure: An XXX measurement by merging multiple patches [Aus19].

- We can do more complicated lattice surgery $\implies$ optimizations for particular operations!
- But two-qubit lattice surgery is sufficient to implement a CX gate (CNOT).
- With some additional tricks we can implement all Clifford gates.

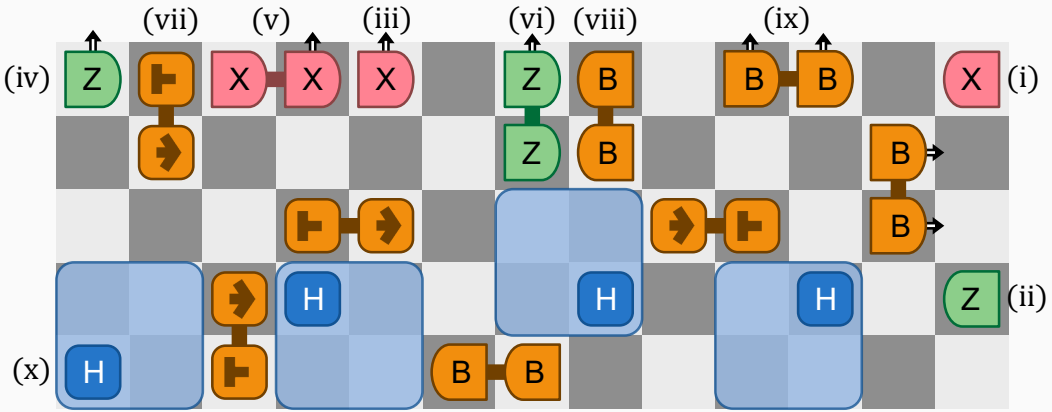


Figure: Elementary operations on a surface code architecture [BKS22].

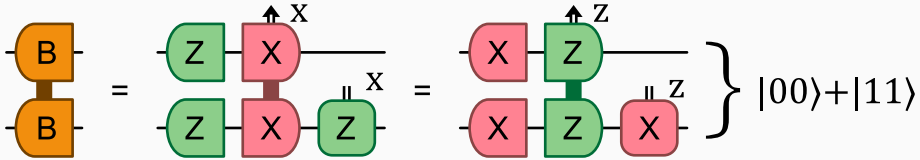


Figure: Bell state preparation using either ZZ or XX measurement.

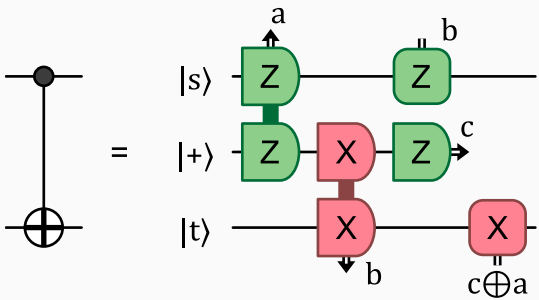


Figure: Using an ancilla in $|+\rangle$ we can perform a CX gate.

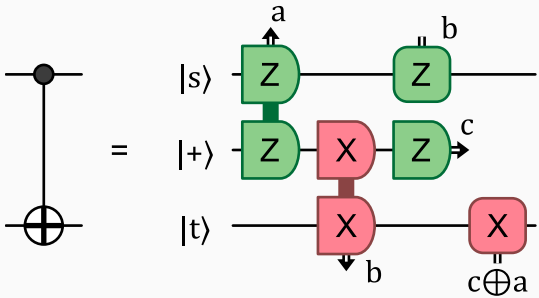
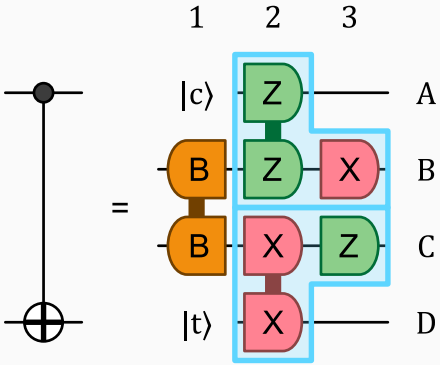


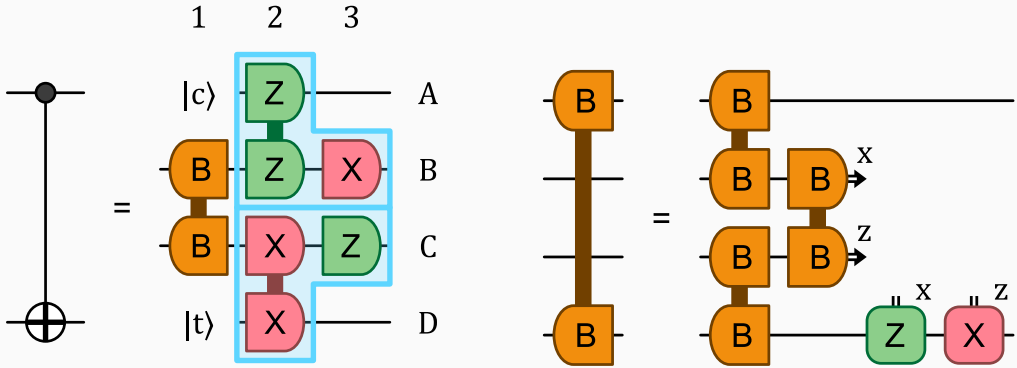
Figure: Using an ancilla in $|+\rangle$ we can perform a CX gate.

Question

How can the qubits be laid out to do the ZZ and XX measurements?



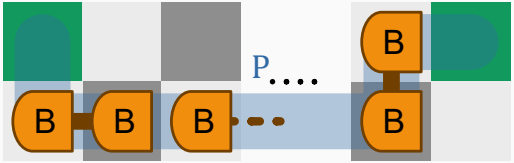
(a) CX via Bell state



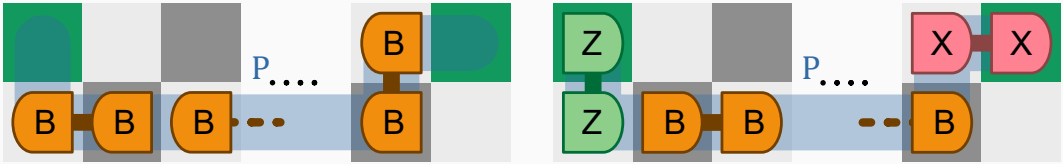
(a) CX via Bell state

(b) Longer-range Bell preparation

Figure: Iterating longer-range Bell preparation gives a construction for a long-range CX.



(a) Prepare Bell pairs



(a) Prepare Bell pairs

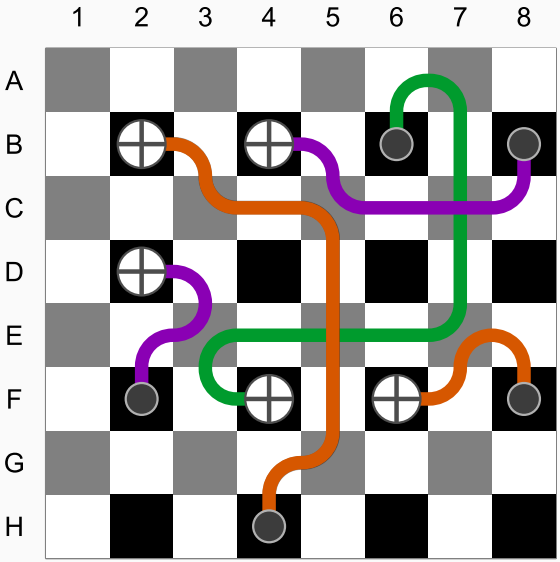
(b) Consume Bell pairs and apply CX

Figure: Long-range CX via iterated longer-range Bell preparation.

Assume $1:3$ data qubit (black) to ancilla (gray/white) ratio.

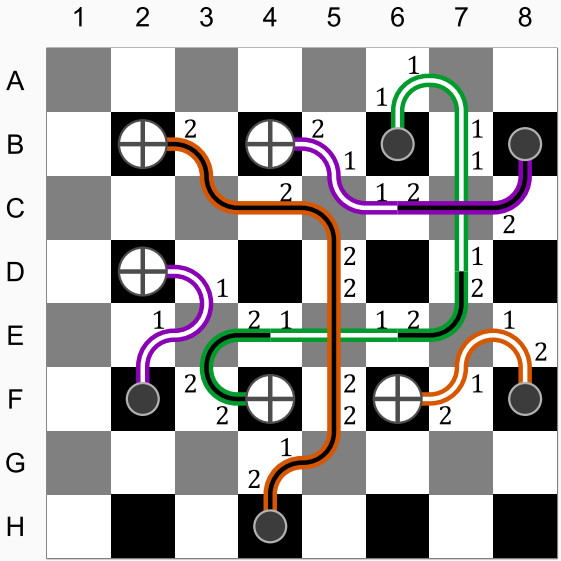
Theorem ([BKS22])

Execute all CNOTs connected via edge-disjoint paths in depth 4.



Fragment paths

- *Fragment* edge-disjoint paths into two sets of vertex-disjoint paths.
- Label each path at a crossing with $\{1, 2\}$.



Two-stage parallel cnot execution

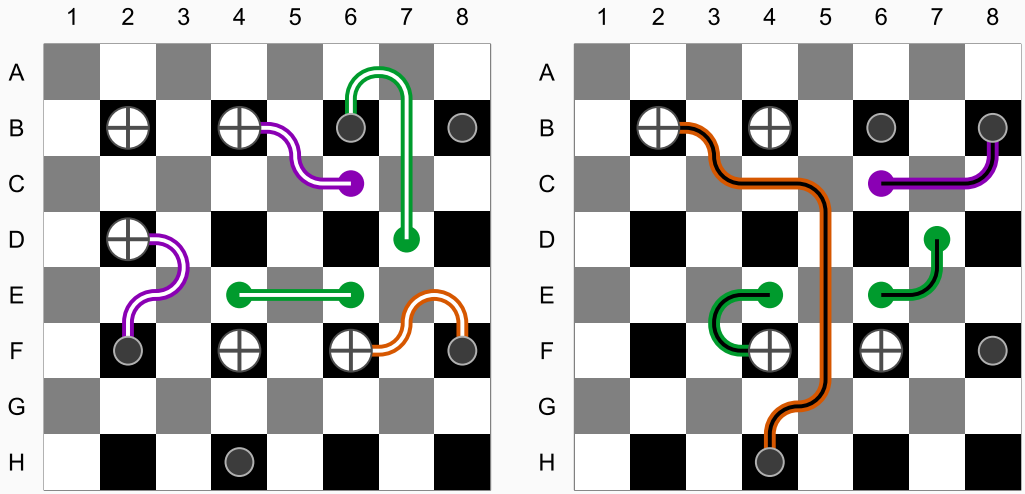


Figure: Edge-disjoint paths are fragmented into two sets of vertex-disjoint paths.

Surface codes

Decoding errors

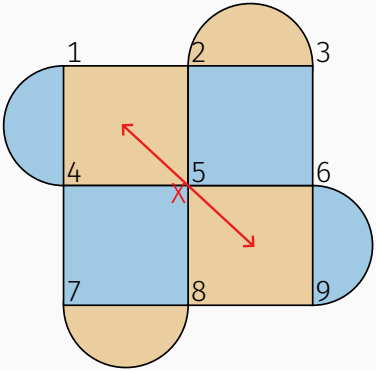


Figure: An X_5 error flips two Z-syndrome measurements.

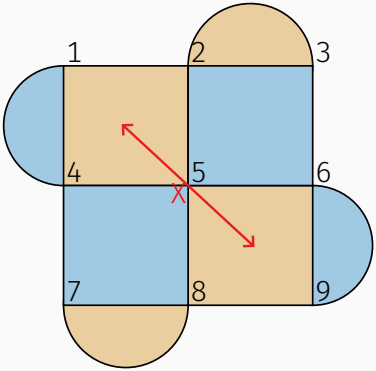


Figure: An X_5 error flips two Z-syndrome measurements.

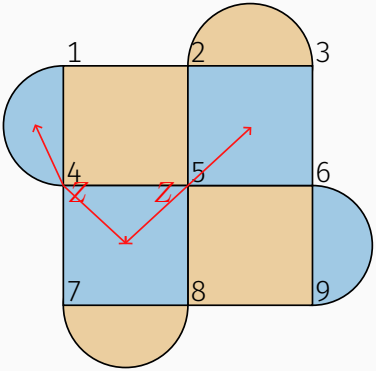


Figure: Two errors $Z_4 Z_5$ flip two syndromes and can be corrected!

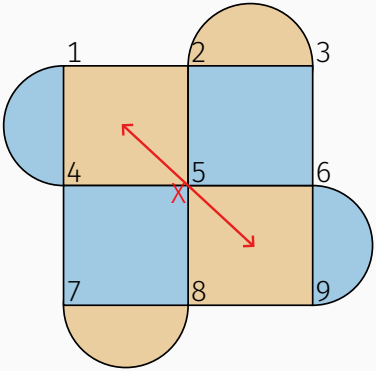


Figure: An X_5 error flips two Z-syndrome measurements.

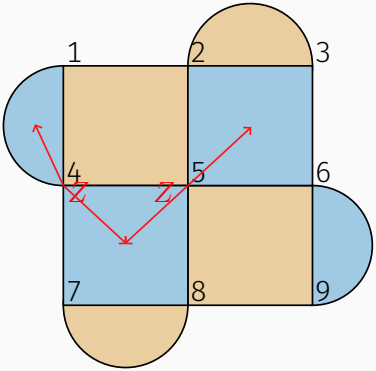


Figure: Two errors $Z_4 Z_5$ flip two syndromes and can be corrected!

Question

Why would applying $Z_1 Z_2$ just as well correct errors in $Z_4 Z_5$?

Conclusion

You have learned about:

1. Classical linear codes
2. Quantum stabilizer codes
3. Surface codes and operations
4. Decoding errors on the surface code

Conclusion

You have learned about:

1. Classical linear codes
2. Quantum stabilizer codes
3. Surface codes and operations
4. Decoding errors on the surface code

There's a lot more to quantum error correction!

- Fault-tolerance (how do we make sure errors do not propagate in bad ways)
- LDPC codes and other high-rate codes
- Decoding algorithms
- Compiling of quantum algorithms to a QEC architecture
- Universal computation using magic state distillation

Conclusion

You have learned about:

1. Classical linear codes
2. Quantum stabilizer codes
3. Surface codes and operations
4. Decoding errors on the surface code

There is a Google group (TQEC design automation) started by Austin Fowler to develop tools for compilation. It includes recordings of helpful lectures.

There's a lot more to quantum error correction!

- Fault-tolerance (how do we make sure errors do not propagate in bad ways)
- LDPC codes and other high-rate codes
- Decoding algorithms
- Compiling of quantum algorithms to a QEC architecture
- Universal computation using magic state distillation

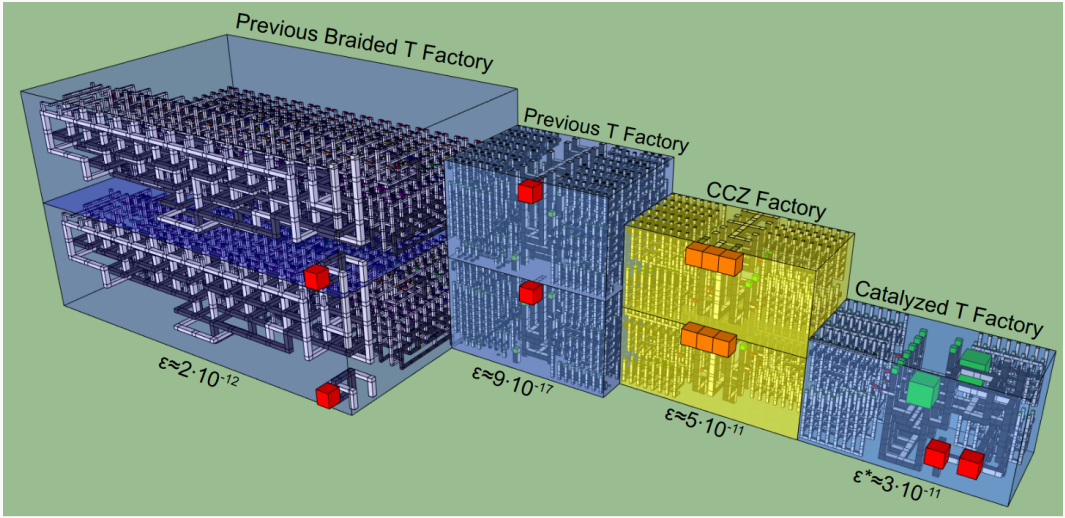


Figure: Evolution of magic state factories [Cra19].

Resources

[Aus19] Craig Gidney Austin G. Fowler. **“Low overhead quantum computation using lattice surgery”.** Aug. 30, 2019. arXiv: 1808.06709v4 [quant-ph].

[BKS22] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. **“Surface Code Compilation via Edge-Disjoint Paths”.** In: *PRX Quantum* 3 (2 May 2022), p. 020342. DOI: 10.1103/PRXQuantum.3.020342.

[Cra19] Austin G. Fowler Craig Gidney. **“Efficient magic state factories with a catalyzed |CCZ> to 2|T> transformation”.** Mar. 26, 2019.

[Del23] Nicholas Delfosse. **“Tutorials on quantum error correction”.** 2023.

[Fow] Austin Fowler. ***TQEC design automation (Google group).*** URL: https://groups.google.com/g/tqec-design-automation.

[Got24] Daniel Gottesman. **“Surviving as a Quantum Computer in a Classical World”.** May 7, 2024.