
- necessary to start the computation

- key subroutine in Steane-style \( (\frac{1}{2} \times + \frac{1}{2} \times) \) and Knill-style \( (\frac{1}{2} \times) \) error correction

- often dominant in overhead of a scheme

- for a random/typical \( n \)-qubit CSS code, transversal gates &-meas cost \( n \); preparing state costs \( \frac{n^2}{4} \) \( (\frac{1}{2} \times \text{stabilizers}, \text{each of act. } \approx \frac{n}{2}) \).

\[ \text{1st Example: } \Sigma - E + (E + ) = (E + ) \]

Method 1: Shor-style prep.

- apply EC to an arbitrary (eg. random) state
  - circular, except for Shor-style EC
  - moves into the codespace, but w/ unknown codeword

- ok if think of \( 107 \) as an \[ \text{[n,0,dI]} \] QECC

- extract syndrome of each of the \( \frac{n - 1}{2} \) \( \times \) and \( \frac{n + 1}{2} \) \( \times \) stabilizers

- works for non-CSS too, using appropriate ort states

- disadvantage: works poorly for large \( n \)

Method 2: Steane-style prep. (for CSS codes)

- prepare & verify \[ \text{Steane } 020\overline{2}036 \]

Preparation

- Gaussian-eliminate \( X \) (or \( Z \)) stabilizers into the form \( (I \ A) \)

\[ \text{eg. } \begin{pmatrix} 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \end{pmatrix} \]

\[ \text{prepare } 1 \oplus \text{num } X \text{stabs} \quad \text{on that} \]

\[ \text{apply CNOTs from controls to targets to create } X \text{stabs. } \]

\[ \text{eg. } \]

\[ \text{state necessarily has correct } Z \text{stabs } \implies \text{all } 107 \]

Optimized preparation

- don't prepare qubit until needed

- parallelize gates into as few rounds as possible
Lemma: Consider a matrix a subset of whose positions have been marked \( \ast \).

Let \( n = \max \) # of \( \ast \)s in any row or col.

\( \Rightarrow \) \( \ast \)e\( d \) positions can be colored w/ 51, 3, \( \ldots \), \( n \), \( \ast \)s st. no number appears twice in any row or column.

Remark: A completely naive greedy algorithm can easily get in trouble

\[ \begin{array}{c}
0 & 1 & 0 & 0 & 0 \\
1 & 2 & 3 & 0 & 0 \\
0 & 0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 1
\end{array} \]

Proof: Induction in \( n \). \( n = 0 \) trivial.

Suffices to place \( n \)'s \( 1 \)s in remaining positions, \( \leq n-1 \) \( \ast \)s in any row or col.

"Worst case" (????) is every row \& col has exactly \( n \) \( \ast \)s.

Then consider the bipartite graph (the adjacency matrix is the \( \ast \)s)

\[ \begin{array}{c}
\text{rows} \\
\vdots \\
\text{cols}
\end{array} \]

\[ \text{edge if they overlap} \]

This is a regular graph with degree \( n \) on both sides.

\( \therefore \) \( \exists \) a perfect matching, ie. a coloring that colors everybody by one \( \ast \).

(Proof Hall's Matching Thm: For a set \( S \) of rows, the \# of incident edges is \( n \cdot |S| \). This is \( \leq n \cdot |N(S)| \), the \# of incident edges to \( N(S) \).

Thus \( |S| \leq |N(S)| \) \( \ast \)s, so matching thm. applies.)

This is true b/c we can always embed the bipartite graph in a larger one in which every vertex has degree \( n \), and the restriction of a matching on the larger graph in particular cases every vertex of the smaller graph.

Here’s how:

\[ \begin{array}{c}
\text{row1} \\
\vdots \\
\text{rown}
\end{array} \]

\[ \begin{array}{c}
\text{col1} \\
\vdots \\
\text{coln}
\end{array} \]

Repeat over \( \ast \) over again, up to \( n-1 \) times, etc.

This can be suboptimal:

- (eg. 10000+11111)

- using teleportation, multiple CNOTs can be applied in one round

(example)

Open: Come up w/ a faster encoding method for general CSS states
1. Verifying ancillas

- Precede same stabilizer at a time (error detection or correction)
- All at once (error detection)

For a $d=3$ perfect CSS code:

```
Error orders:
0 1 2 3 0 1

Ancilla:
0 1 1 1 0 1/
0 1 2 2 0 1
0 1 2 3 0 1/
```

For a $d=7$ perfect CSS code:

```
Error orders:
0 1 2 3 4 0 1 2 3
0 1 1 1 0 1 1 1
0 1 2 2 2 0 1 2 2
0 1 2 3 3 0 1 2 3
```

---

```
FT against $X \otimes Z$ errors
```

(can apply a random cut to reduce $X/2$ asymmetry)

-can be optimized: use different prep cuts to create different correlated errors that can't cancel out

-do not use full error rejection

2. Start by encoding into an error-detecting code

```
XXX
1 2 3 4 5 6
```

-then apply Steane's prep cut

- finally, or occasionally, check for errors

- decode

3. Use the "slow means" decoding trick from last time to avoid any/some verification

- reduces overhead, also threshold

(massive ancilla verification probably gives best thresholds)