Decoding

The simplest problem in error correction is the following

Step 1: Logical qubits are prepared in a state of the stabilizer space

$$|\psi\rangle = \sum_{j,k\in\{0,1\}} c_{jk} |jk\rangle_L$$

$$A_v |jk\rangle_L = B_p |jk\rangle_L \quad \forall v, p, j, k \;, \quad X_1 |jk\rangle_L = |j+1, k\rangle_L \;, \quad \text{etc}$$

Step 2: Time passes while we go and have a cup of tea. Errors happen during this time

$$\mathcal{E}(\rho) = \sum_j \sum_{\alpha\in\{0,x,y,z\}} p_\alpha(\Delta t)\, \sigma_\alpha^j \rho \sigma_\alpha^j$$

Let's focus on phase flip errors. These happen on each qubit with a probability

$$p = p_z(\Delta t) + p_y(\Delta t)$$

This generates a random pattern of phase flip errors, E_z

Step 3: We measure the vertex operators to get some clues about what phase flip errors occurred (equivalently and independently, we also measure plaquette operators for bit flip errors). These measurements are noiseless.

This gives us a configuration of e anyons: the syndrome, $S(E_z)$

Step 4: Using the syndrome, we try to find a correction operator E_z^c

Ideally we would like $E_z^c = E_z$, but this is not possible in general. The same syndrome results from many different Pauli errors

$$\underbrace{E_z,\ E_z B_{p_j},\ E_z B_{p_j} B_{p_k},\ \dots}_{\text{right class}},\ \underbrace{ZE_z,\ ZE_z B_{p_j},\ ZE_z B_{p_j} B_{p_k},\ \dots}_{\text{wrong class}}\ (\text{similarly for } E_x)$$

They form two classes: equivalent to E_z up to stabilizers

equivalent to ZE_z up to stabilizers

Apply any in the correct class and you correct the errrors

$$E_z B_{p_j} E_z |\psi\rangle_L = E_z^2 B_{p_j} |\psi\rangle_L = |\psi\rangle_L$$

Using the wrong class causes a logical error

$$Z E_z B_{p_j} E_z |\psi\rangle_L = Z E_z^2 B_{p_j} |\psi\rangle_L = Z |\psi\rangle_L$$

Note: If E applied a σ_z to a qubit j, we don't need to physicall apply another to undo it. Instead we could just relabel the σ_z basis states for j

$$|+\rangle \rightarrow |-\rangle, \; |-\rangle \rightarrow |+\rangle$$

This leads us to interpret future measurement results differently, getting the same results as if σ_z had been applied

Also, if we are just using the code to store information, we can leave all processing of the syndrome information until just before we use it

Decoding Algorithms

Step 4 uses a 'decoding algorithm', This is a method to determine a good E_z^c to use for any given S .

Best results would come from looking at all possible E_z that could cause the syndrome.

These can be split into four classes, such that those within the same class differ only by stabilizers. Those in different classes differ by logical operators.

The probability that the error came from each class can then be calculated. The operator E_z^c can then be chosen from the most likely class

This is the best error correction we can hope to do. But the required calculations mean summing over a number of error configurations that is exponential in L^2 !

Something more efficient is required, even if it is a bit worse.

Decoding is a complicated optimization problem.

For ideas on how to do it efficiently, we can look at solutions that have been found to other such problems

Thresholds

When we get decoding wrong, and $E_z^c E_z$ turns out to contain a logical operator, we end up disturbing our stored qubit. This is called a logical error.

The probability that we get a logical error, given a noise model and decoding algorithm, is called the logical error rate. This typically looks something like

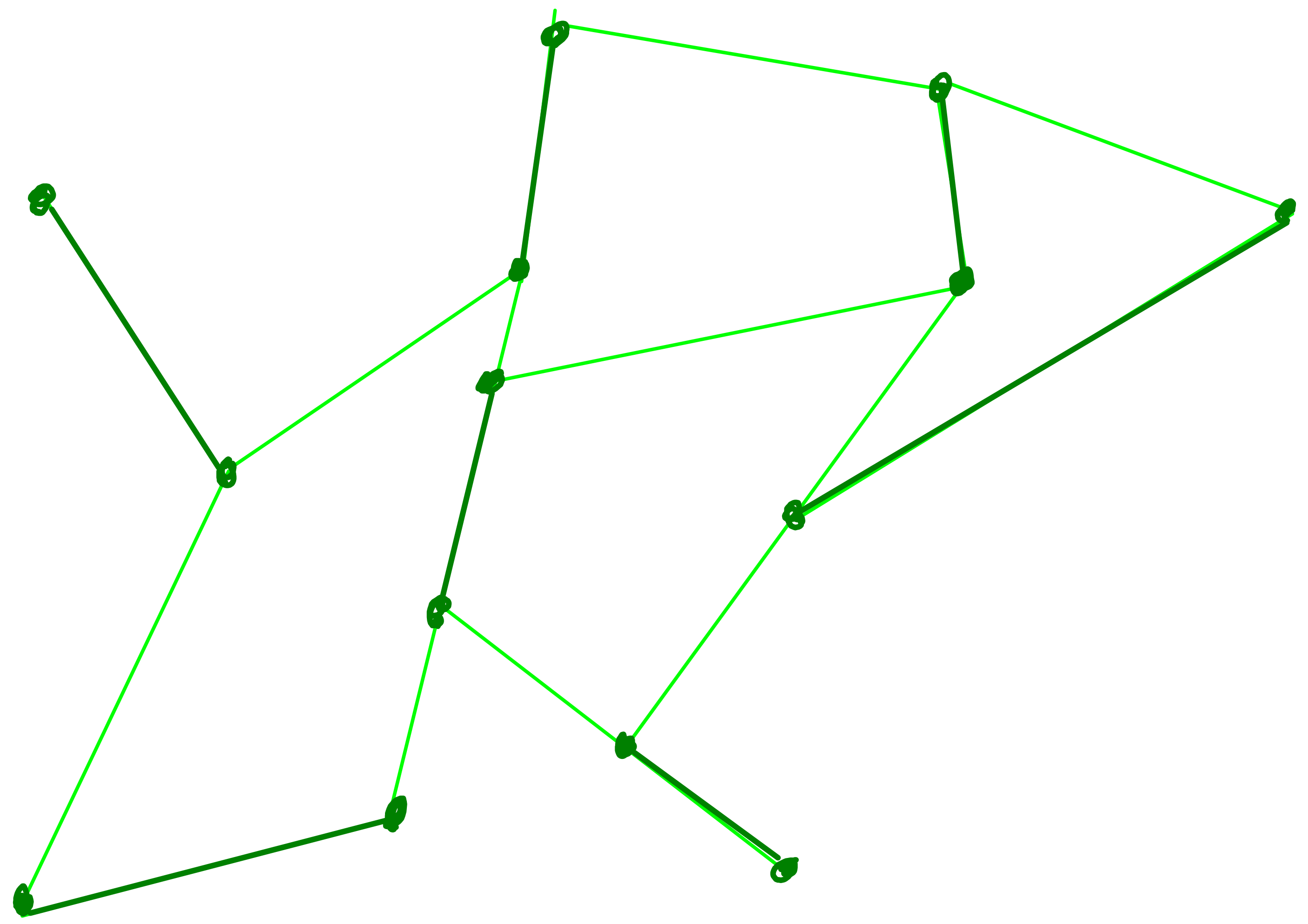
$$P = \begin{cases} \frac{1}{2} & p > p_c \\ e^{-\alpha(p) L^{\beta}} & p < p_c \end{cases} \qquad \alpha(p) > 0 \,, \quad 1 \geqslant \beta > 0$$

Logical error rate is exponentially suppressed, and so correction is good, as long as p is below a threshold value p_c

This depends on decoder, and needs to be finite

MWPM

One of the best algorithms we can use for the planar code is based on the problem of minimum weight perfect matching (MWPM) from graph theory



A graph is a collection of vertices j and edges (j,k)

A matching is a subset of edges so that no two edges meet at the same vertex

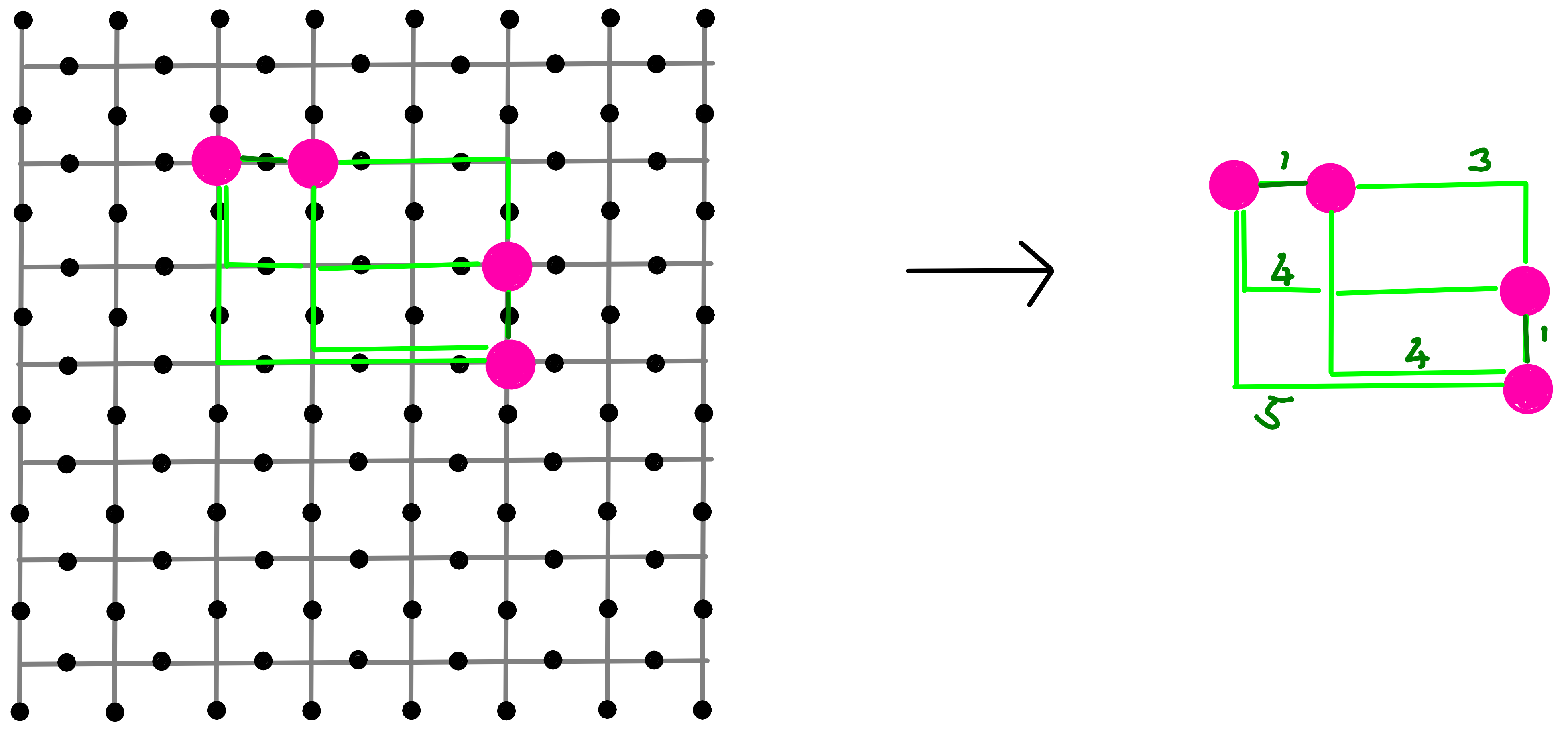
A perfect matching involves every vertex

If the edges have weights, we can calculate the total weight of a matching

How can we find a perfect matching of minimum weight?

An efficient method to solve this kind of problem was found by Edmonds in 1961

We can use this as the basis of a decoding algorithm

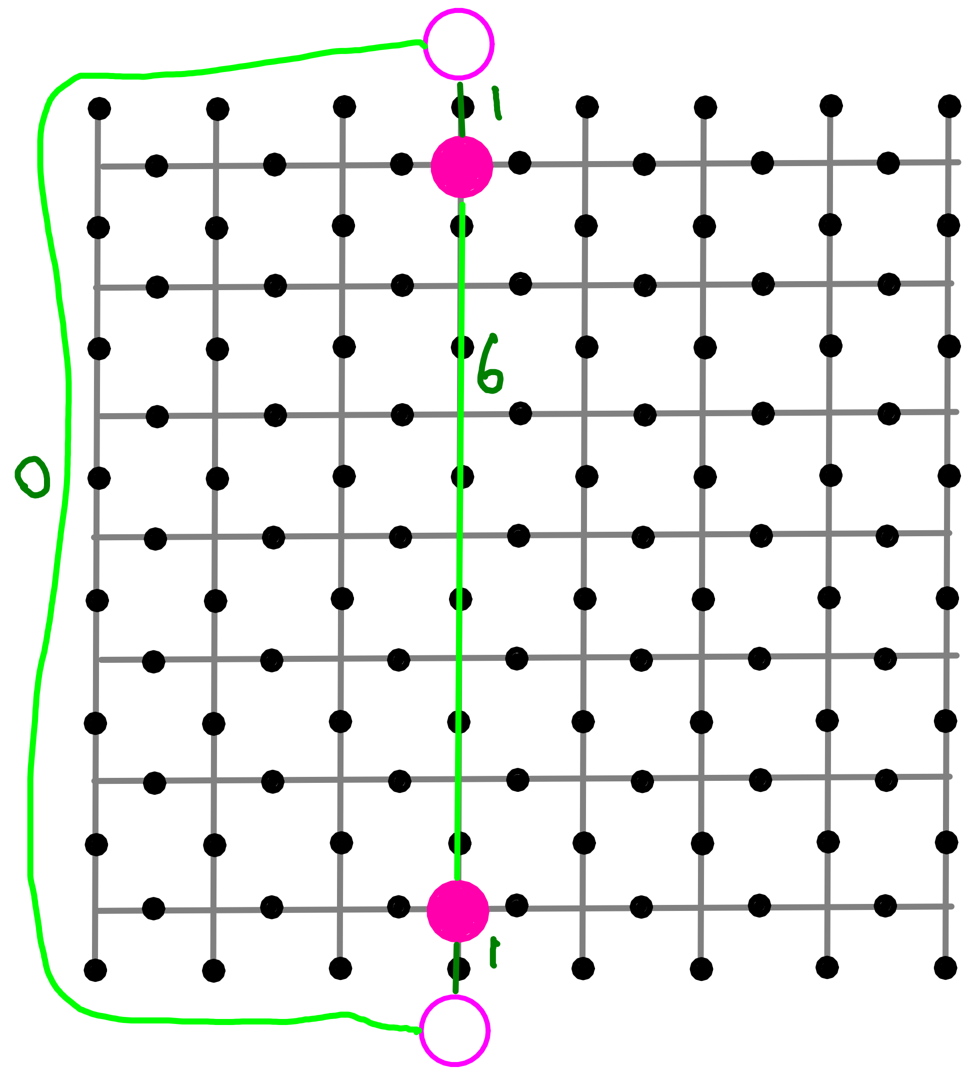


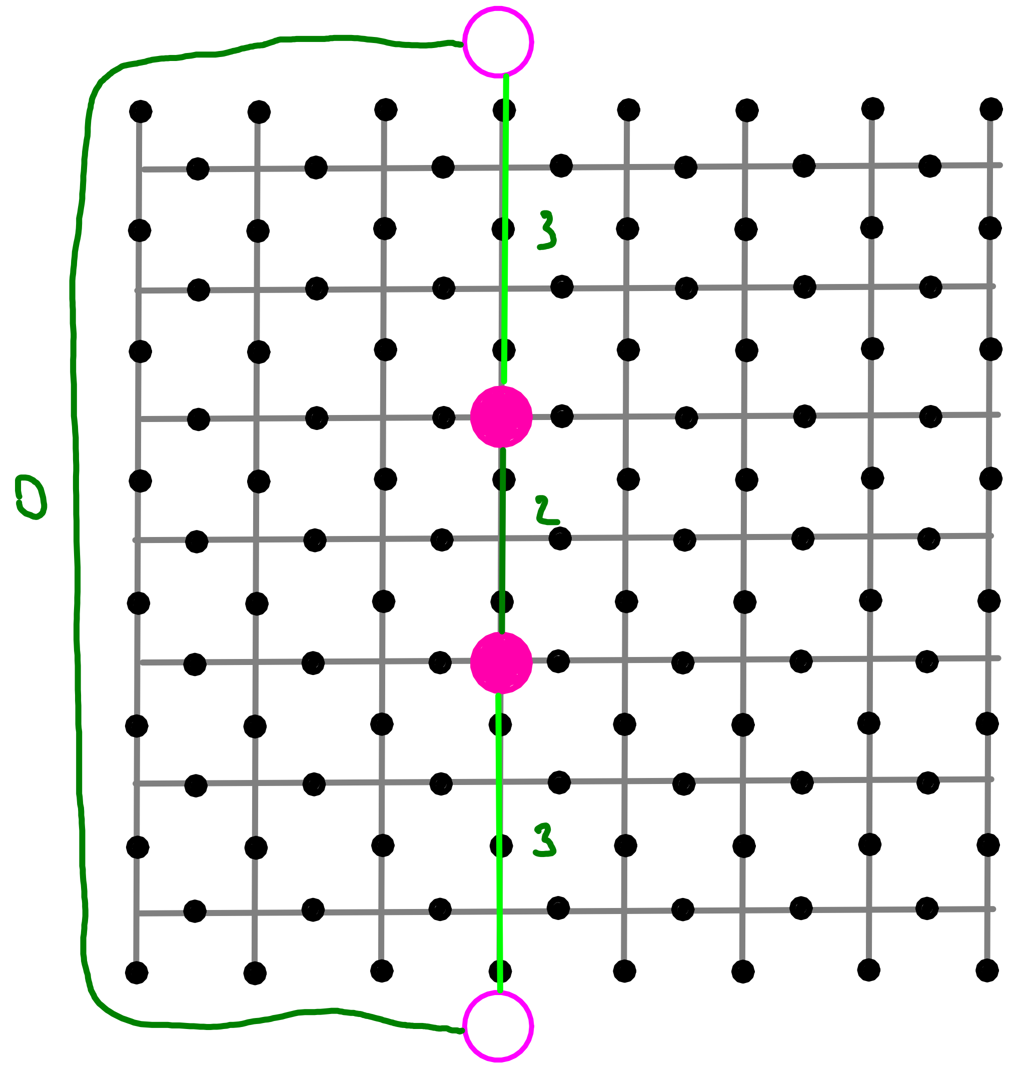
We construct a graph whose vertices are the anyons

There is an edge between all pairs of anyons

The weight of the edge is the number of errors needed to make that pair

What about the edges of the code?





Each anyon has a 'virtual anyon' living on the nearest edge.

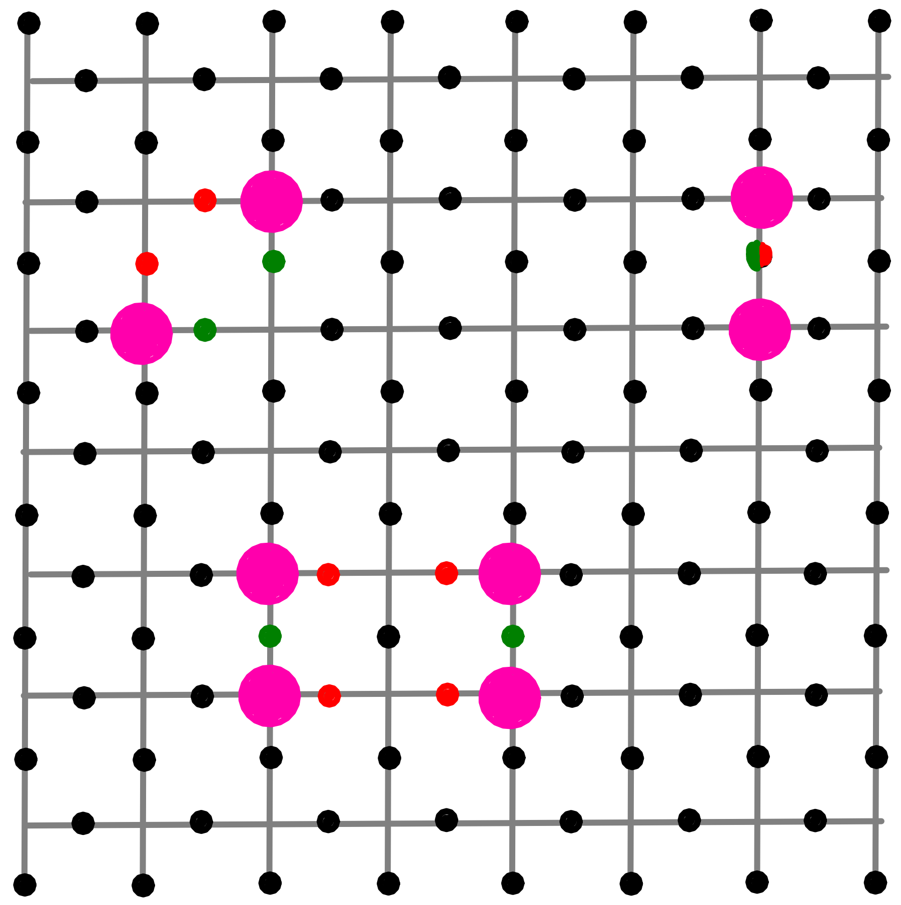
This is connected only to its twin (weight=distance to edge) and to all other virtuals with weight zero

The minimum weight matching then gives us the E_z^c with the least number of errors

For $p < 0.5$, this means it is also the most likely single error chain

We can expect that the most likely single error chain will belong to the most likely equivalence class (most likely) so this should provide good error correction

Threshold Proof

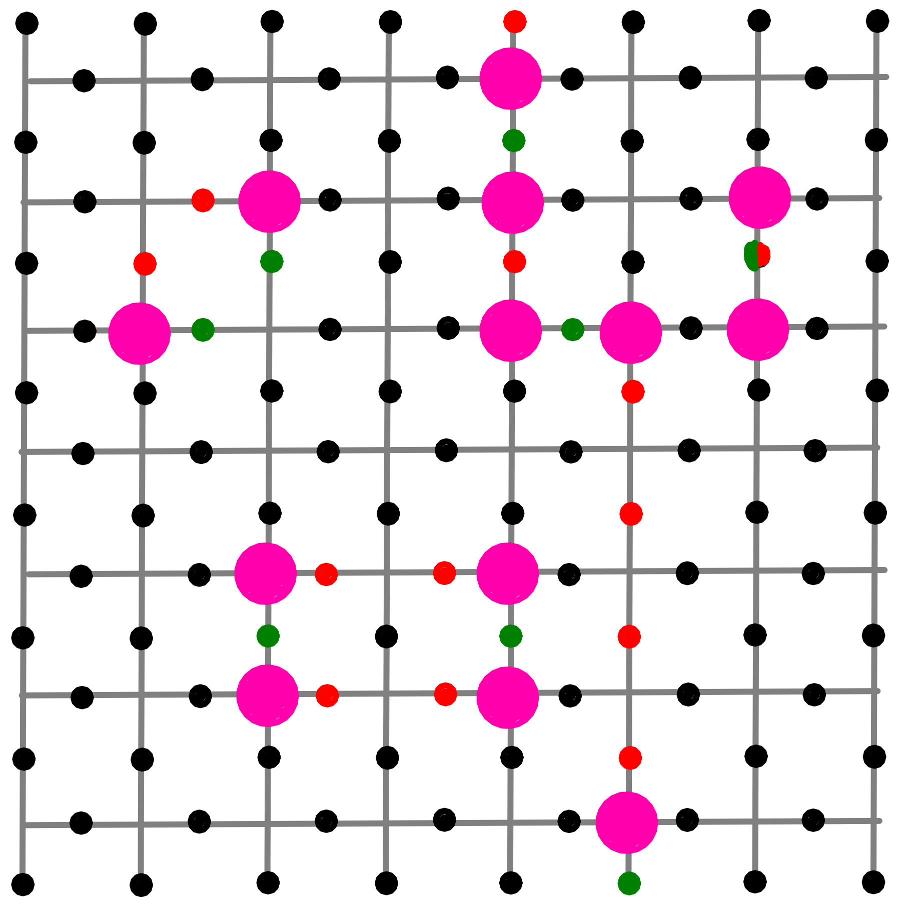


If successful, the effect of $E_z^c E_z$ is loops of phase flips

In each loop

$\#\text{errors} \geq \#\text{corrections}$

This is because MWPM always corrects with the minimal number. If there were less errors, the correction would have done that instead.



If unsuccessful, there is at least one line across the code for which

$\#\text{errors} \geq \#\text{corrections}$

Which means

$\#\text{errors} \geq \frac{\#\text{qubits in line}}{2}$

$\therefore \#\text{errors} \geq L/2$

For p<0.5, the probability of errors on more than half of a given set of L qubits is very small. The Chernoff bound gives us

$$P \leqslant e^{-\frac{1}{2(1-p)}\left(p-\frac{1}{2}\right)^2 L}$$

But the number of possible routes across the grid is exponentially large

Lets look at the probability of n errors on a path of length l , and then use that to upper bound the logical error rate

An upper bound for the number of paths of length l is

$$\text{starts at any point} \longrightarrow 2L^2 \times \underset{\substack{\uparrow \\ \text{can then go in 4 directions}}}{4} \times 3^{l-1} \longleftarrow \text{thereafter in 3 directions without going back}$$

Let's just use 3^l for simplity. Not an upper bound, but it'll do

There are many ways to put n errors on l qubits. But would require a binomial coefficient. Much easier just to upper bound with the total number of ways you can put any number of errors on l qubits: 2^l

So the total number of ways that you can get n errors on a path (loop or string) of l qubits is upper bounded by

$$2^l \times 3^l = 6^l$$

The probability for any given chain of such errors is

$$p^n (1-p)^{l-n} < p^n$$

So the total probability for n errors on any path of l qubits is upper bounded by

$$6^l \; p^n$$

The only cases that can cause a logical error are

$$l \geq L \qquad n \geq \frac{l}{2}$$

For a given l , the probability is upper bounded by using $n = \frac{l}{2}$

$$6^l p^{l/2} = \left(6\sqrt{p}\right)^l$$

For the total probability, we sum over all $l \geq L$

$$P \geq \sum_{l=\frac{L}{2}}^{\infty} \left(6\sqrt{p}\right)^l$$

This will clearly vanish for

$$6\sqrt{p} < 1 \quad \therefore \quad p < \frac{1}{36}$$

This shows us that there is indeed a finite threshold when using MWPM, and we have a lower bound for its value

$$p_c \geq \frac{1}{36}$$

Continuous error correction

Ideally we would like to protect a logical qubit state for long time periods

This could be while we wait to use it for some quantum communication protocol, or while it is being used in a quantum computation

Previously we looked at the one-shot case: we prepare the state, then errors happen, then we decode

To preserve for long time, we need to continuously perform error correction

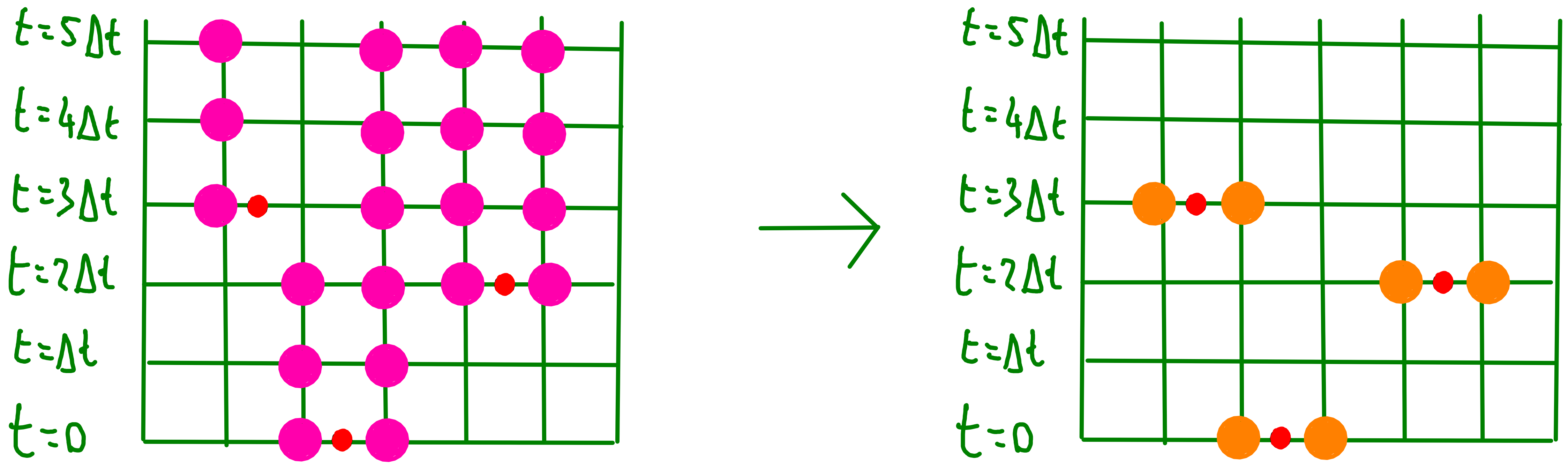
The entire syndrome is measured periodically, with a time Δt between measurements

Perfect syndrome measurements

At each time slice, the syndrome tells us the anyon config at that time

By looking at the anyon config, and comparing it for different times, we want to be able to determine which errors occured

To do this we look at the change in the anyon config



The time slices then correspond to independent decoding problems, each equivalent to the one shot case

Error model is (for charge syndrome)

$$\mathcal{E}(\rho) = \sum_j (1-p)\rho + p\ \sigma_z^j \rho \sigma_z^j, \qquad p = p_z(\Delta t) + p_y(\Delta t)$$

The exponential suppression of the logical error rate in the one-shot case then become an exponentially long lifetime for continuous error correction

$$P(\Delta t) = e^{-\alpha(p)L^\beta} \Rightarrow P(T) < \frac{T}{\Delta t} e^{-\alpha(p)L^\beta} \qquad \therefore T(P) = P\Delta t\, e^{\alpha(p)L^\beta}$$

The time before the logical error rate reaches some unacceptable value can be made arbitrarily long by increasing the code size