Quantum Error Correction

Correlated Decoding as Bilinear Programming

Ben Criger

December 11, 2016

1. Introduction

The toric code is a popular 2D nearest-neighbour LDPC code whose decoding can be efficiently accomplished by minimum-weight perfect matching. Qubits are placed on the edges of a square tiling of a 2D torus, and are subjected to one of three errors, with labels X, Y and Z:

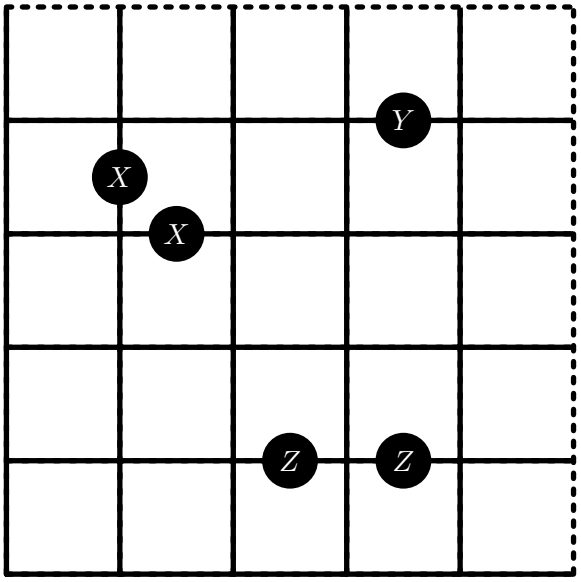


Figure 1: 5-by-5 toric code lattice with an error configuration. Qubits are supported on each edge, of which there are 50. Five of these qubits have been subject to an error, drawn from the set $\{X, Y, Z\}$. Edges with no marker are not subject to error. Dotted edges indicate 'wrapping around' the boundary to create a torus.

The parity checks of this code respond to X and Z errors, indicating when an odd number of X errors surround a tile, and an odd number of Z errors surround a vertex:

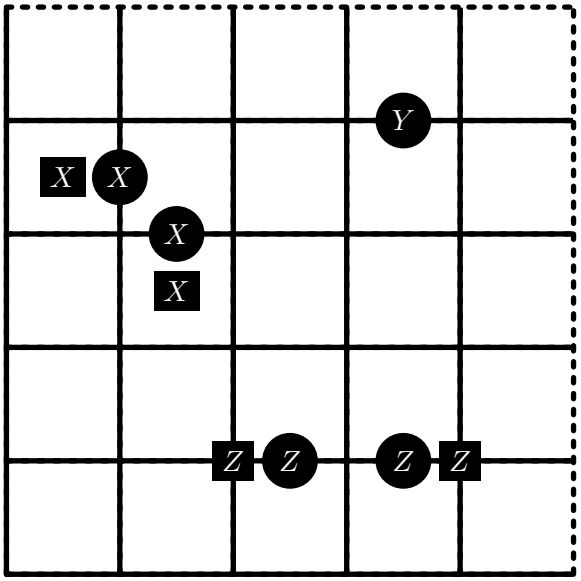


Figure 2: Syndromes corresponding to X and Z errors.

One thing to notice about these syndromes is that, if a continuous chain of errors of the either X or Z type appears on the lattice, two syndromes will appear at the endpoints of the chain. If only X and Z errors occur, then, the assignment of an error to a given syndrome in the toric code can be reduced to minimum-weight perfect matching, since the probability of an error chain can be expressed as a function of the distance between the syndromes.

If Y errors could be independently detected and decoded in the same fashion, we'd be done. The problem is that, due to Quantum Code Technicalities®, a Y error produces a pair of X *and* Z syndromes:

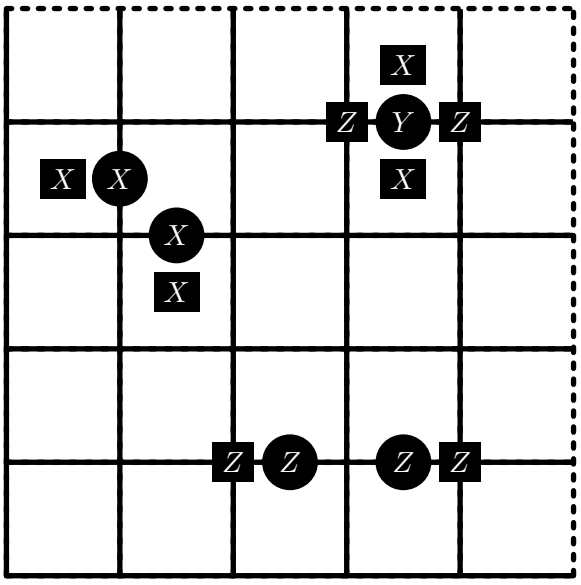


Figure 3: Syndromes corresponding to X, Y and Z errors.

Our decoding problem is made more difficult by this, though decent performance can still be obtained by assuming that Y errors occur with probability $\sim p^2$ (this works because a Y error is the result of an X and a Z error occurring on the same qubit, bu we don't need to get into it), and decoding the X and Z syndromes independently. This leaves a gap in the decoder's performance, that we're hoping to fix.

Existing approaches for this problem [3, 1] rely on *reweighting*, adjusting the edge weights in an X syndrome graph based on where Z errors have been inferred, or vice versa. These weights are derived

from *conditional* probabilities, treating $p(Y) = p\left(X \mid Z\right)$ or $p\left(Z \mid X\right)$.

I have beef with this:

- It multiplies runtime by a factor of at least two, since the matching problem must be solved for one graph before the result can be applied to the other. This factor may be larger if we iterate, using each matching to reweight the other graph until convergence.
- Performance can, in principle, depend on which graph we match first. It may be the case that both orders (XZ and ZX) have to be attempted in parallel to get a decent solution. Even then, we have no guarantee that Y errors can be accurately inferred given a decoding procedure that puts X and Z steps in order like this.
- Last, but certainly not least, there is a massive degeneracy which the re-weighting approach ignores. That is, there are multiple minimum-weight paths between syndromes which are equally valid, and a re-weighting decoder chooses one seemingly arbitrarily.

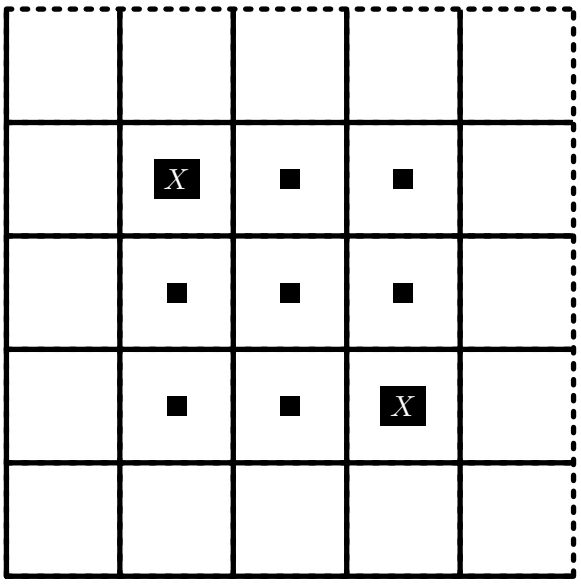


Figure 4: Possible X syndrome pair. If the decoder determines that a length-4 chain exists between the two, such a path can be restricted to a bounding box whose interior is marked with black squares above. Only if the two syndromes are collinear is there a unique path.

We'd prefer to have a solution that can handle Y errors 'out of the box', to try to put them on the same footing as X and Z. In the rest of this document, I loosely explore this kind of thing.

2. Weights

Let's first show how the linear program for decoding syndromes independently is derived (see [2] for a longer and stronger explanation). We begin by writing the probability of an X error that produces pairs of X syndromes, according to an independent error model:

$$p(E_X) = \prod_{C_X \in E_X} p^{|C_X|}(1-p)^{n-|C_X|}, \tag{1}$$

where E_X is the X error, C_X is a continuous chain of Xs that makes up part of the error, and n is the number of qubits. We can simplify this by taking the log:

$$p(E_X) = (1-p)^n \prod_{C_X \in E_X} \left(\frac{p}{1-p}\right)^{|C_X|} \tag{2}$$

$$\propto \sum_{C_X \in E_X} \log\left(\frac{p}{1-p}\right) |C_X| \propto - \sum_{C_X \in E_X} |C_X| \tag{3}$$

Maximizing the likelihood, then, is equivalent to minimizing the sum of lengths of the chains C_X whose endpoints are given by the X (or equivalently Z) syndromes.

We can immediately see the problem. A single Y error has an effective weight of two, since it is detected as a length-one X chain and a length-one Z chain in the absence of other errors. To fix this, let's make the unjustified assumption that $p_X = p_Y = p_Z$. They're going to be different, later (we'll actually have a very different error model later), but this is an easy place to start. This results in "free edges", since adding a Z error to an edge where there's already an X error results in a Y error, incurring no additional cost. We can always adjust the costs we derive to determine how to compensate for a lower p_Y. In case of a higher p_Y, we can probably do a trivial code redesign to ensure that the type of error that produces excess syndromes is the least likely.

3. Optimisation Problem

I'd like to have a nice structure for an optimisation problem, so I can look at slackness and adapt Blossom. It's not working, for the following reasons:

3.1. Cost Not Given By Single Intersection

In the crisp, clean picture of what these chains look like, we subtract one unit of weight whenever one chain crosses another:

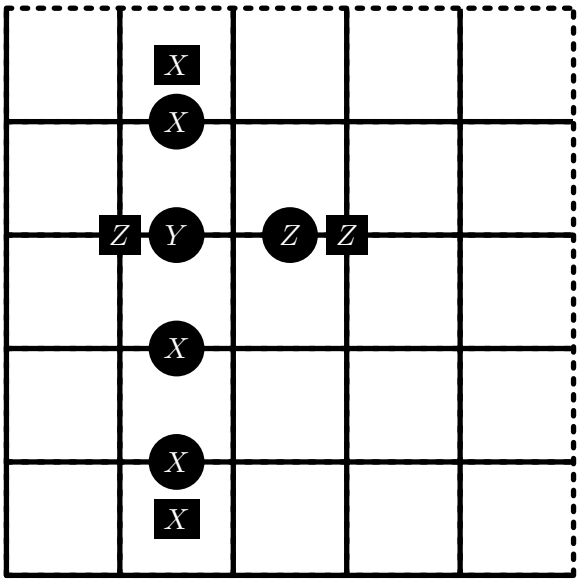


Figure 5: This is not a minimum-length path, but it would be on a larger lattice, so it's legitimate.

If all the chains intersected like this, we'd have a bilinear program on our hands. In this world, we would just record which minimum-length paths intersect in one position with which others, and evaluate the cost by pricing the X and Z chains separately, then evaluating a matrix element which counts the number of

intersections and subtract that off. The first problem is that minimum-length paths which aren't straight can intersect in weird ways: This is in principle not a problem, since we can associate the maximum change in

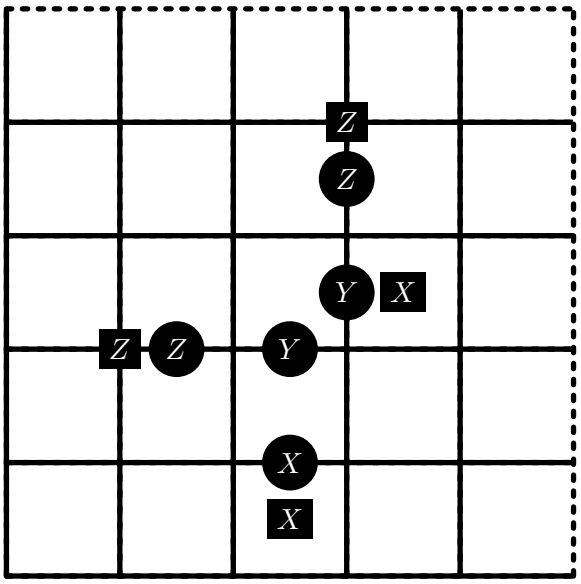


Figure 6: Bent X and Z paths that intersect on two qubits. Note that other minimum-length paths in the bounding boxes of these syndrome pairs can intersect on one or zero qubits.

cost with any edge pair. The real problem would be if a union of three paths (a, b, and c) had a cost that could not be described by a bilinear function.

3.2. Cost Not Given By Bounding Box

We're still trying to attach a cost to a syndrome configuration, and it looks like you can obtain this for an X/Z pair configuration by first getting the independent cost, then counting the number of qubits where the bounding boxes of the X/Z syndrome intersect and subtracting. This works for the chain in Figure 6 (the bounding boxes of those syndrome pairs intersect on exactly two qubits), but it doesn't work for an arbitrary pair:

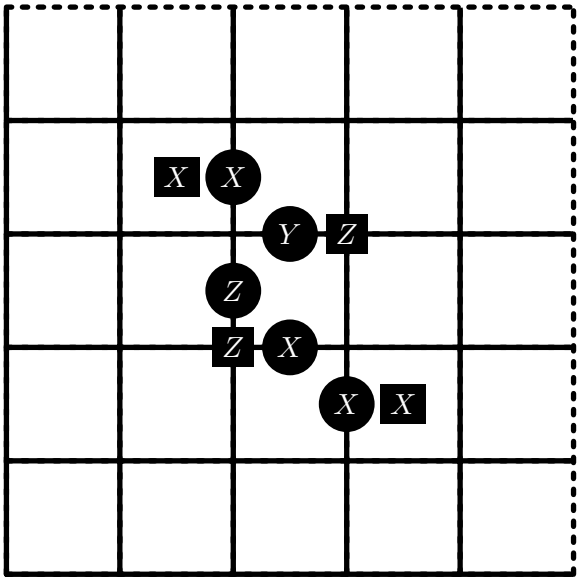


Figure 7: Syndrome pairs that can't be priced by bounding box intersection. The bounding box of the Z pair is contained entirely in that of the X pair, but the maximum intersection between these paths is one.

It is now no longer clear what the cost of a syndrome should be, let alone whether it can be represented by a bilinear function.

References

[1] N. Delfosse and J.-P. Tillich. A decoding algorithm for CSS codes using the X/Z correlations. *ArXiv e-prints*, January 2014.

[2] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[3] A. G. Fowler. Optimal complexity correction of correlated errors in the surface code. *ArXiv e-prints*, October 2013.

Appendices

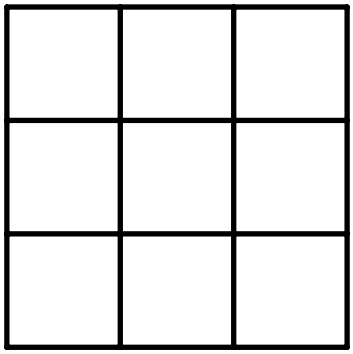
A. Measuring a Different Code Every Round

If a bilinear program-based decoder doesn't work for whatever reason, do we have the option to measure a different set of stabilisers at every round that commute with the previous round's (e.g. CSS codes with Y and X strings instead of X and Z) and reconstruct the error history from there? A program of research suggests itself, where we go back to DKLP and figure out which errors form chains in 3D given the cycle of stabilisers that we measure.

B. Re-Re-weighting

Can we solve the problem with re-weighting where a geodesic error chain is chosen arbitrarily? Maybe. We can try to evaluate the conditional probability of an X error on the qubit in a syndrome pair's bounding

box, given that the syndrome pair is in the matching produced by the decoder. We can even use message-passing to do this. Considering the syndrome pair in Figure 4, we take the subgraph of the dual lattice that corresponds to the bounding box:



To ensure that we restrict to minimum-length paths, we add directions to each edge. We then perform a forwards and backwards pass to obtain the probability that each vertex is involved in a path:

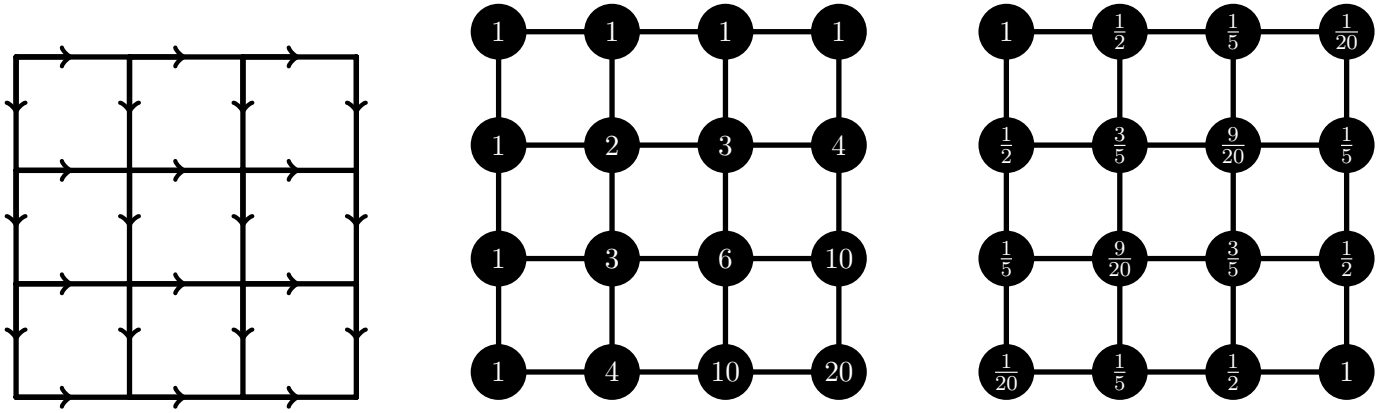


Figure 8: From left to right: Directions for which any valid path is minimum-length; number of paths from the top left to the highlighted vertex found by Pascal's triangle-type message-passing; probability of each vertex being involved in a randomly selected path.

The probability that an edge is involved in a given path is given as $p(e) = p(v_1, v_2) = p(v_1)p(v_2 \mid v_1)$, which we can evaluate by looking at subdiagrams of Figure 8:

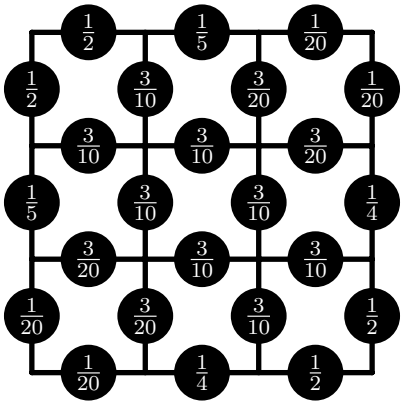


Figure 9: Edge probabilities obtained by evaluating conditional probabilities on sub-diagrams of Figure 8.

Though this is a little messy, it looks like this might be a better way to re-weight the edges given that a path between the top left and bottom right corners exists. If it isn't, I still suspect that it's related to some way to re-weight better.