Classical Decoding for Topological Codes

October 19, 2022

1 Introduction

Decoding algorithms for error correcting codes can be extremely complex computationally, even in the classical setting, so much work has been done over the years to optimize these procedures. For many quantum codes, decoding algorithms for their classical analogues can be used, taking advantage of this history of work and optimization. Topological codes such as Kitaev's toric code however have no direct classical analogue – possibly the closest being low density parity check codes – and so much unique work has been done to decode them.

Due in part to both locality of the stabilizer generators and high simulated thresholds, topological codes are becoming increasingly attractive for realistic quantum computing. By providing stabilizer generators with only geometrically local checks, the code can be implemented in a linear nearest neighbour grid without the need for long range interactions. However, for a code to be practically applicable to general quantum computing, there must be efficient decoding schemes that can be performed between quantum processing steps. Fortunately, algorithms have been found for accurately decoding the toric family of codes that run in polynomial, and thus tractable, time.

This paper will examine some of these classical algorithms proposed for decoding the toric code, as well as the closely related planar code, comparing their computational complexity and simulated thresholds. In particular, I will examine approaches based on maximum graph matchings, renormalization group methods, as well as a recent algorithm based on parallel tempering.

2 The toric code

We begin by reviewing Kitaev's toric code, as described in [8].

Suppose we have a square $l \times l$ lattice with periodic boundary conditions and $2l^2$ (data) qubits placed on the edges. For a vertex s on the lattice, $\text{star}(s)$ denotes the qubits on the edges adjacent to s, and for a face p, $\text{plaquette}(p)$ denotes the qubits on the edges of p. Before we continue, it will be useful to note that every qubit on the lattice is contained in

the star and plaquette of exactly two vertices and faces, respectively. We can now define

$$A_s = \prod_{i \in \text{star}(s)} X_i, \qquad B_p = \prod_{i \in \text{plaquette}(p)} Z_i,$$

where $X_i \in \mathcal{P}_n$, the Pauli group on n qubits, denotes the n-fold tensor product with the Pauli-X gate on the ith qubit, and the identity on all remaining qubits; Z_i is defined similarly, with the Pauli-Z gate instead of X.

These star and plaquette operators are all mutually commuting, since X and Z anti-commute, and $|\text{star}(s) \cap \text{plaquette}(p)| \in \{0, 2\}$ for all vertices s and faces p. As a result, they generate a stabilizer S and associated stabilizer code on the $2l^2$ (data) qubits, called the *toric code*.

An important point to note is that each qubit is acted on non-trivially by exactly two star operators and two plaquette operators. From this fact, we can both ascertain that the product of every star (plaquette) operator is equal to the identity (note that $X^2 = Z^2 = I$), and the fact that the product of at most $l^2 - 1$ star (plaquette) operators cannot be equal to the identity, as there will be at least 2 qubits acted on non-trivially. This informs us that there are maximally $2l^2 - 2$ independent generators, and so 2 logical qubits are encoded by the toric code.

The fact that each qubit is acted on by exactly two star and plaquette operators also hints at how errors will look when one measures the syndrome. For example, a single X error on a qubit i in the lattice will commute with the two star operators acting on i, and anti commute with the two plaquette operators acting on it. If another X error is placed on one of the boundaries of these plaquette operators, the resulting state is one that commutes with the plaquette operator containing both errors, and anti commutes with the two plaquette operators, each containing one of the errors. In general, only an even number of stabilizer generators can have -1 eigenvalues at the same time, and are the result of some set of error chains between pairs of negative eigenvalues.

Knowing this, we can characterize the logical operations of the code. Since X and/or Z operations on inidividual qubits correspond to pairs of anti-commuting generators, the only non-trivial operations commuting with all stabilizer generators are loops of X and/or Z operations. Loops that don't wrap around the boundary, are products of stabilizer generators (specifically the generators contained inside or outside the loop) and thus contractible. The logical operations then are the 4 X or Z loops around the opposite boundaries.

In general, we will want our error correction to form contractible loops with the actual errors. Formally, we want to identify the correct *homotopy class* of a chain (or chains) of errors. Two chains of errors on the lattice are homotopically equivalent if they are the product of stabilizer generators, with the homotopy class being defined by the equivalence relation. If we correctly guess the homotopy class of the syndrome, then applying any error correction in the same homotopy class will create a set of closed, contractible loops. On the other hand, applying error correction corresponding to a different homotopy class will cause some non-trivial loops around the torus to appear.

As an important related code, the planar code corresponds to removing the periodicity of the toric code. Smooth (rough) defects are introduced by not enforcing a contiguous

region of plaquette (star) stabilizers. If these contiguous regions wrap around the torus in homologically[1] distinct ways, the effect is to produce a toric-like code with two rough, two smooth boundaries, and 1 logical qubit. Since the boundaries correspond to stabilizers not enforced, a chain of X (Z) errors between two smooth (rough) boundaries commutes with all the stabilizer generators. If the two boundaries happen to be the same, the loop is contractible, otherwise it corresponds to one of the 2 logical operations.

3 Matching-based

A simple yet versatile classical decoding of the surface code is based on minimum weight graph matching. The idea is to formulate the syndrome information as a weighted graph and then perform a minimum-weight maximum matching algorithm; with high probability, the error chains corresponding to this matching should be homotopically equivalent to the actual set of errors.

Efficient (in the sense of polynomial time) algorithms are already known for maximum graph matchings, and simulations using matching-based decoding have shown high thresholds and potentially scalable performance [2], making this an attractive paradigm for topological decoding.

3.1 Formulation as a graph matching problem

Given a graph $G = (V, E)$, a *matching* in G is a subset of edges $M \subseteq E$ such that no two edges in M are adjacent (ie. end on the same vertex). A matching M is *maximum* if for any $e \in E \setminus M$, $M \cup \{e\}$ is not a matching, and a maximum matching is *perfect* if every $v \in V$ is an endpoint of an edge in M.

For a syndrome measurement of a toric code, we can define a (complete) graph G with vertices corresponding to the non-zero bits of the syndrome (ie. stabilizer generators with eigenvalue -1), and placing edges between each vertex. It will be helpful later to notice that in general, this graph can have $O(l^2)$ vertices, and $O(l^4)$ edges on a lattice of linear size l. Any set of errors that can generate the given syndrome is then a perfect matching in this graph, where the edges correspond to chains of X or Z errors. To correct these errors, it suffices (but is not required) to pick the correct matching and the correct homotopy class for each edge within that matching; this is done by choosing the most probable matching and correcting along the most probable homotopy class for each edge. These algorithms typically simplify the problem by letting each edge correspond to only one error chain, usually the most probable (in most cases the shortest) error chain between the syndrome bits, so that each matching corresponds to exactly one homotopy class.

It is important to note, however, that the correct matching is not required to correct the error. For example, one matching may pair up the correct syndrome bits but cause a logical operation by applying correction along the (homotopically) wrong paths between

[1] Distinct from homotopic equivalence

those bits, while another matching may pair up the wrong syndrome bits, but still result in a contractible loop when the corrections are applied.

A few different methods of assigning weights to the edges in the graph have been proposed. The simplest such weighting assigns weight to the edges by the Manhattan distance – for two vertices or faces at coordinates (x, y) and (x', y'), the weight is given as $|x-x'|+|y-y'|$. The Manhattan distance between two syndrome bits gives the number of errors in the shortest error chain between them, and so if every qubit in the lattice is subjected to the same channel, a smaller Manhattan distance will correspond to a higher probability of being connected by a chain of errors. However, the probability of an error chain between two syndrome bits decreases exponentially in the separation, and so a more accurate edge weighting will use the probabilities of the chains between vertices. In [3], simulations were performed using $wt_{(u,v)} = -\ln(P_{\max}(u, v))$ as the edge weights, where $P_{\max}(u, v)$ is the maximum probability of any error chain between u and v, and a higher threshold was observed than in simulations using the Manhattan distance.

One of the strengths of matching based decoding is that it can easily be extended to deal with syndrome errors as well [4]. A syndrome error at a single timestep will cause the syndrome to change in that timestep, then change again in the subsequent timestep. By recording only when a syndrome bit changes and then compiling syndrome information across multiple time steps into one graph by adding edges between syndrome changes on the same bit, we can match these syndrome changes to each other, indicating a syndrome error. Graphs for each set of syndrome measurements are created as before; edges are then added between vertices in different graphs corresponding to the same syndrome bit. If the same syndrome changes more than twice, only edges between adjacent changes in terms of the time ordering are created (ie. if a syndrome bit changes 3 times, no edge is created between first and third vertex). By placing weights on these edges corresponding to the probability of syndrome errors, the regular minimum weight matching algorithm can then be run on this graph, and a single syndrome bit errors will likely be matched to itself in the next timestep.

The inclusion of syndrome errors poses a problem in that vertices may be better matched to some syndrome information in the future. To solve this problem, a time boundary is created, where a vertex with no forward temporal edge is given an edge to the time boundary.

As one additional point on the construction of graphs, in the planar code any error chain may begin at a boundary corresponding to the type of error, resulting in only one syndrome bit flip. This is handled simply by adding an edge between each vertex and a boundary, weighted according to the distance to the closest corresponding boundary. To facilitate the use of standard graph matching algorithms, edges of 0 weight are added between each pair of boundary vertices, such that they can be matched at no cost after the syndrome vertices have been matched.

3.2 Maximum matching algorithms

By formulating error correction as a graph matching problem, classic algorithms for maximum graph matching can now be used for decoding. Specifically, Edmond's matching

algorithm finds a maximum matching in time polynomial in the number of vertices[2]. While this algorithm itself is not weighted, a simple modification that finds a minimum weight matching exists [9].

To describe Edmund's maximum matching algorithm, we need a few definitions from graph theory. Given a matching M in a graph $G = (V, E)$, we say a vertex $v \in V$ is *free* if for any $(u, v) \in E, (u, v) \notin M$; v is *matched* if it is not free. We then find that a perfect matching is a matching with no free vertices.

An *alternating path* is a sequence of vertices $v_1, ..., v_n$ in V such that $(v_{2k}, v_{sk+1}) \in M, (v_{2k+1}, v_{2k+2}) \in E \setminus M$ for $k = 0, ..., \lfloor n/2 \rfloor - 1$. In other words, an alternating path is a chain of adjacent vertices where the edges of the path alternate between in M and not in M. A path is *augmenting* if it is an alternating path beginning and ending at free vertices, and a *blossom* is an odd length alternating path beginning and ending on the same vertex. *Augmenting* an alternating path P in a matching M refers to exchanging the matching edges in P for the unmatched edges. It will be useful to note that a blossom will always begin and end with edges from $E \setminus M$, since the path is odd length beginning and ending on the same vertex. We say a blossom is free if one of the vertices contained in it is free.

Additionally, an *alternating tree* is defined as a tree rooted at a free vertex, in which every path from the root to a leaf is alternating. In such a tree, the vertices of even depth are called *outer* nodes and branch on edges from $E \setminus M$, while vertices of odd depth are called *inner* nodes and branch on edges from M.

The correctness of the algorithm consists in two propositions:

Proposition 1. *(Berge's lemma) A matching M is maximum in G if and only if M admits no augmenting path in G*

As a standard result in graph theory, the proof of Berge's lemma is left out.

Proposition 2. *Given a matching M, suppose G has some blossom at an outer node in an alternating tree. The graph G' created by contracting the blossom has an augmenting path if and only if G does.*

Proof. Since the blossom begins at an outer node, it is connected by an alternating, even length path to the root of the tree, a free node. We define M' to be the matching after augmenting that path; M' thus has the same number of edges as M, so M' is maximum if and only if M is. In particular, if M admits an augmenting path so does M'. Also, after augmenting this path, the base of the blossom is now a free node, and the contracted blossom node is free.

Now suppose G' has some augmenting path. If it does not end at the blossom, then G trivially has the same augmenting path. Otherwise, expanding out the blossom, the augmenting path can continue around the blossom in the direction producing an alternating path, until it reaches the base, giving an augmenting path in G.

[2]While the original algorithm runs in time $O(n^2m)$ for a graph with n vertices and m edges, improvements have been made that decrease this time to as low as $O(nm + n \log n)$ [9]

```
function EDMONDS(G, M)
    while there is some free vertex v do
        Root alternating tree at v
        while there are unexamined edges in G do
            Pick some edge (v, w) from an outer vertex v
            if w is matched and (w, x) ∈ M then
                Add (v, w) and (w, x) to tree
            else if w is free then
                Let M' be the matching M after augmenting the path from the root to w
                return Edmonds(G, M')
            else if w is an inner node of some alternating tree then
                Do nothing
            else if w is an outer node in the tree containing v then
                Contract the blossom from w to w
            else if w is an outer node in some other tree then
                Let M' be the result of augmenting the path between tree roots
                return Edmonds(G, M')
            end if
        end while
    end while
end function
```

On the other hand, suppose G has an augmenting path, ignoring the trivial case when it does not intersect the blossom. If it hits the blossom, neither edge entering the blossom can be in the matching (every vertex in the blossom is either matched to another vertex in the blossom or is free), so the path from either end of the augmenting path to the blossom is an augmenting path in G' □

The algorithm examines each edge of the graph and so always finds the augmenting path, if it exists. By contracting the blossoms augmenting paths are preserved, but the size of the graph shrinks – more detailed proof will be left to [9].

To extend this algorithm to find a minimum weight matching, each node is assigned a variable y_k, initially assigned to 0 when added to the tree (or created, as in the case of contracted blossoms). An edge (u, v) is called *tight* if its weight $w_{(u,v)}$ is such that $w_{(u,v)} - y_u - y_v - \sum_k y_k = 0$ where the sum is over all blossom nodes that contain exactly one of u and v. The only change to the algorithm is to require that each edge (v, w) chosen must be tight – if there are no tight edges, then all outer nodes have their variable y_k increased, and all inner nodes are decreased. Additionally, if a blossom node is decreased to 0, it is expanded again.

Formally, this modification involves the formulation as a linear programming instance, and is beyond the scope of this paper. In simulations of the toric and surface code, the implementation by Vladimir Kolmogorov [9], believed to run in $O(n^3m)$ time, is commonly

used [1].

3.3 Improvements

A number of heuristics can be employed to improve the performance of graph matching algorithms, most of which deal with pruning the graph.

One scheme for pruning the graph is described in [1]. A vertex u is in *shadow* with respect to v if a minimum length path (on the lattice) between u and v passes through another vertex. A vertex u is then *deeply shadowed* with respect to v if all of the other neighbours of u are shadowed. Two vertices that are deeply shadowed with respect to one another will never be matched in a minimum weight matching where the probability of a chain is proportional to the length of the chain, and thus they can be removed. It is claimed in [1] that this method reduces the number of edges in the syndrome graph from $O(n^4)$ to $O(n^2)$, and timing analysis appears to imply this scaling [2].

However, it is also noted that when syndrome errors are included in the graph, it becomes difficult to find deeply shadowed vertices, and so this method does not scale to the case of imperfect syndrome measurement. Progress can be made by observing that at low error rates, only local matchings should need to be performed, as long error chains and thus spatially separated syndrome changes are infrequent. In [1], the authors describe a scheme by which only a small initial radius of vertices is matched, and they suggest that the probability of requiring to increase the radius should decrease exponentially with the size of the radius.

If we look at the planar code, more optimizations can be found. While the addition of boundaries complicates the problem, it also allows some significant improvements. For example, consider two vertices u, v in the graph and suppose M is a maximum matching and $(u, v) \in M$ (without loss of generality we assume their boundaries are matched to each other). If the sum of their weight to the boundary, $w_{(u,b)} + w_{(v,b)}$, is greater than or equal to the weight $w_{(u,v)}$ of (u, v), then $M \cup \{(u, b), (v, b)\} \setminus \{(u, v), (b, b)\}$ is a maximum matching with at most the same weight. It follows that this edge can be removed from the graph, as any minimum weight matching can use the boundary edges instead. Additionally, this method applies to both spatial and temporal boundary edges.

As a useful consequence of this pruning, at low error rates disconnected components start to appear in the graph. These disconnected components can be matched in parallel, as they are completely independent from one another. As well, since these components never have to be considered again, the matching can be completed, then all of the related syndrome information can be discarded from future computations.

3.4 Experimental results

Many simulations of the planar code using maximum matching algorithms have been performed to date. As one simulation result, the toric code was simulated subject to the depolarizing channel and perfect syndrome measurement for distances up to 11, and the threshold was found to be 0.155 [4]; under non-ideal syndrome measurement the threshold was found to drop to 0.0078. More recently, the planar code was simulated under the bit

flip channel with distances up to 1024, and a threshold was found at 0.1025 in the case of perfect syndrome extraction; again when syndrome errors are added, the observed threshold drops to 0.9 (note that distances only up to 55 were simulated for non-ideal syndrome measurement) [1].

Graph matching thus appears to both produce good error correction, while at the same time being a flexible paradigm with opportunities for optimization. While an $O(l^6)$ runtime [5] is most likely too slow to perform actively in a realistic quantum computer, recent timing analysis [2] suggests that maximum matching decoding could feasibly run fast enough to keep up with quantum circuits.

4 Renormalization Group Algorithms

While classical decoding algorithms using graph matchings are both simple and powerful, in choosing the lowest probability error chains, they optimize for distance rather than the most likely coset of S. To address this, a renormalization group[3] method for decoding the toric code has been developed in [5] and [6]. It will turn out that while more closely resembling a maximum likelihood decoding algorithm, it achieves a threshold reaching that of perfect matchings, at best. Additionally, the scheme as described in the aforementioned papers does not deal with syndrome measurement errors, limiting its practicality. However, the renormalization group algorithm operates with a much faster proven complexity than the matching method, and can be extended to other topological codes where no previous decoding algorithm was previously known [5], making it a useful tool for topological decoding.

4.1 Construction

We begin with a lattice of linear size $l = 2^c$ for some integer c. The idea is to approximate the toric code as c levels of concatenation of a topological code on a 2×2 lattice. Specifically, a *unit cell* is a collection of 12 qubits in a lattice with a surface code defined by three A_s and B_p generators. In the full lattice, these unit cells contain overlapping qubits, so the decoding of different cells can't be completely separated, hence this is truly an approximation scheme.

From the stabilizer code formalism this surface code must encode $12 - 6 = 6$ logical qubits and have 6 logical X and Z operators. It will be convenient to split these operators into two types: 2 pairs of logical operators, denoted X_i^L, Z_i^L, and 4 pairs of edge operators, denoted X_i^E, Z_i^E.

The logical operators are directly analogous to logical operators in the toric code or a surface code with modified boundaries – they are defined as error chains between opposing boundaries. In fact, they will define the probability distribution of errors at the next level of concatenation. The edge operators, on the other hand, will later facilitate the combining of probability distributions on neighbouring unit cells.

[3] A technique in statistical physics for viewing a system at different scales, where different scales are self-similar up to renormalization of the parameters

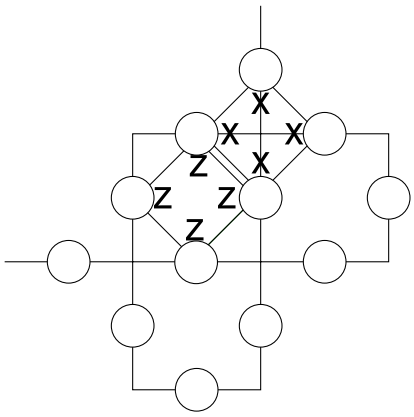

Figure 1: Unit cell with Star and Plaquette operators

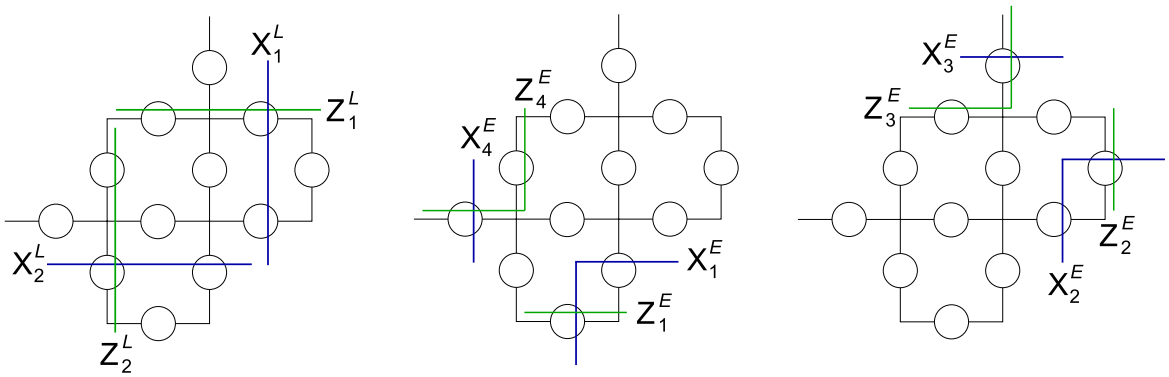

Figure 2: Logical and Edge operations: green and blue lines represent Z and X operators applied to the corresponding qubits

We also assign canonical errors Q_c to each of the 2^{12} possible syndromes on a unit cell. The canonical errors are concisely defined as products of error chains that cause a change in exactly one stabilizer generator. Specifically, with each stabilizer generator S, we define a *pure error* S' such that $\{S, S'\} = 0$, $[T, S'] = 0$ for any stabilizer generator $T \neq S$. Then, the canonical error with syndrome c is given by

$$Q_c = \prod (S')^{c(S)},$$

where $c(S)$ is the bit of the syndrome corresponding to stabilizer generator S, and the product is over all 6 stabilizer generators.

4.2 Renormalization algorithm

With all the definitions in place, the error decoding scheme can be described. The algorithm follows by taking a probability distribution over the errors and a syndrome on a $l \times l$ lattice, then for every unit cell in the lattice computing the total probabilities of each of the 4 logical operations. These logical error probabilities defined on two (logical) qubits for each unit

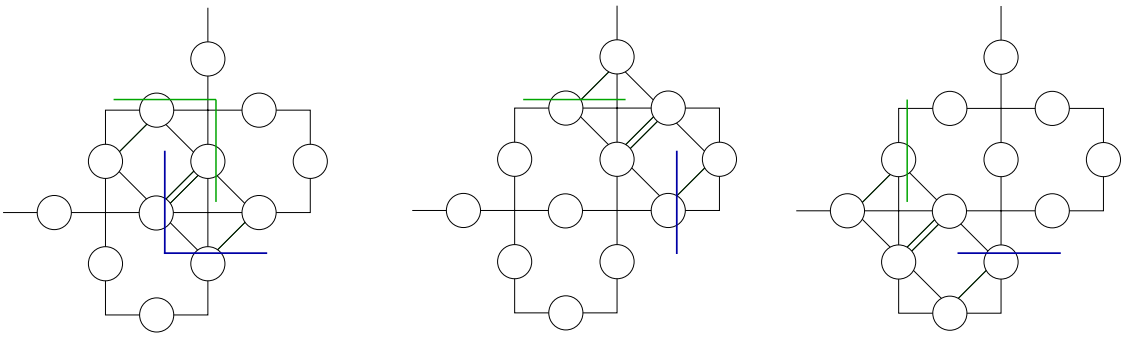

Figure 3: Pure errors with corresponding stabilizer generators

cell are then used to define the (renormalized) error probabilities on two qubits of a lattice of size $\frac{l}{2} \times \frac{l}{2}$. The form this lattice takes in relation to a specific unit cell is shown below; the syndromes for each stabilizer generator in this lattice are given by multiplying the 4 containing syndromes at the lower level lattice.

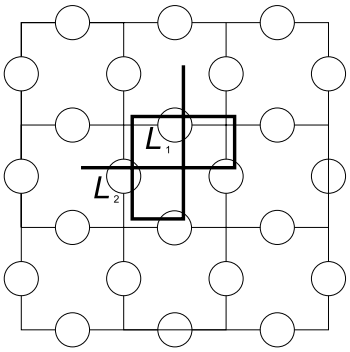

Figure 4: Lattice with probability distributions on two qubits given by the logicals one level down

By repeating this procedure for each level of concatenation, we obtain a probability distribution over the logical operators of the topological code on the encoded qubits.

We can elucidate this algorithm with the following pseudo-code: at the first level, $\mathcal{L}$ is the full lattice, P is the probability distribution defined on qubits of the physical lattice, and c is the syndrome obtained by measuring the stabilizer generators of the toric code.

Given an initial lattice of size $l = 2^c$, there are $c = \log l$ levels of concatenation (and thus recursion), where each lattice has at most l^2 unit cells, each of which take constant time to compute their distributions. So, the algorithm has a clear complexity bound of $O(l^2 \log l)$. However, it can be observed that the computation of the probability distribution on each unit cell is performed independent of the rest, so they can be parallelized, bringing the complexity down to $O(\log l)$.

A few subtle points still need to be examined. First, two qubits with probability distribution given by some $P_{U,c}(L)$ can have correlations between them, but each unit cell contains 4 "halves" of these distributions (ie. only one of the two correlated qubits are contained in the unit cell). The solution to this problem used in [5] was to ignore the correlations for

function RENORMALIZE($\mathcal{L}, P, c$)
  Divide $\mathcal{L}$ into unit squares
  for each unit square U **do**
    Compute joint probability over all logical & edge operators:
      $P_{U,c}(L, E) = \frac{1}{\mathcal{N}} \sum_{F \in Q_c S} P(LEF)$ ▷ $\mathcal{N}$ is the normalization factor
    Compute marginal probability over all logical operators:
      $P_{U,c}(L) = \sum_{E \in \text{Edge operators}} P_{U,c}(L, E)$
  end for
  Define $\mathcal{L}'$ of size $|\mathcal{L}|/2^2$
  Define P' using the marginal distributions $P_{U,c}(L)$
  Compute c' by multiplying each group of 4 adjacent syndrome bits in c
  return Renormalize($\mathcal{L}', P', c'$)
end function

these "halves" and approximate $P_{U,c}(L) = P_{U,c}(L_1)P_{U,c}(L_2)$, where $P_{U,c}(L_i)$ is the marginal distribution on X_i and Z_i.

The other issue has a more involved solution. Since some qubits are shared between unit cells, they can't be considered as independent when considering a unit cell, without incurring significant inaccuracies in decoding. The solution presented in [5] is to use a belief propagation algorithm to converge on a dsitribution for the shared qubits.

The key point to notice is that the edge operators modulo the pure errors are single qubit errors on shared qubits. For a specific syndrome c, the pure errors are fixed, so the marginal probabilities $P_{U,c}(E) = \sum_{L \in \text{Logical operators}} P_{U,c}(L, E)$ of the edge operators in a unit cell form a probability distribution of errors on the shared qubits alone. Using belief propagation, these marginal distributions can be "agreed" upon between neighbouring unit squares.

For concreteness, at each step, every unit cell exchanges a pair of messages with each of its 8 neighbours corresponding sharing a qubit. Initially, every unit cell sends the uniform distribution, and once it receives all 8 messages it updates its outgoing messages $M_{out,q}$, corresponding to each shared qubit q, with the rule

$$M_{out,q} = \frac{1}{P(E_q)} \sum_{F \in LQ_cS} P(\prod_{q'} E_{q'} F) \prod_{q' \neq q} M_{in,q'},$$

E_q is the edge operator acting on q, and LQ_cS is the set of logical operator cosets of Q_cS. After the messages converge, or alternately after a certain number of iterations, the unit cells can then reweigh the probabilities on the shared qubits and proceed with the renormalization algorithm independently.

4.3 Experimental results

In practice, the renormalization group decoding method achieves both good error rates as well as efficient performance. In [5] simulations of the toric code subject to the depolarizing channel on each qubit we run, additionally assuming perfect syndrome measurements. Data

was collected for lattice sizes up to 258, and a threshold error rate of 0.152 was observed, rivaling the results of [4] for the toric code.

It appears as though this could be a useful decoding algorithm in practice. It avoids the potentially huge and complicated graphs that matching algorithms have to contend with, countering with a strict, small complexity bound and parallelization. However, assuming perfect stabilizer measurements is a major obstacle that needs to be surpassed to make a practical decoding scheme.

5 Other decoding algorithms

Monte Carlo methods are used to simulate many complex systems, and in particular parallel tempering has found applications in many fields, including quantum information [10]. In [7], the authors develop a decoding algorithm for the planar code based on the parallel tempering idea; while it achieves extremely high thresholds, the complexity of this process makes it impractical for general quantum computing. What follows is only a cursory description of their algorithm, and the interested reader is directed to [7] for the full details.

Given some syndrome measurement c, to best correct the error one could find the (conditional) probability $P(E|c)$ of a given logical error E by adding the (conditional) probabilities $P(e|c)$ of each set of physical errors e consistent with the syndrome and logical error. While the individual probabilities $P(e|c)$ are easy to compute, the size of this set is exponential in the number of stabilizer generators, and thus intractably large. The solution proposed in [7] is to randomly sample error configurations from the distribution $P(e|c)$ and use them to approximate $P(E|c)$.

The algorithm follows by generating a random set of physical errors corresponding to the syndrome. At each step a new error e' is generated by applying a random change to the previous error e, and the new error replaces the old one with probability given by their ratio $\frac{P(e'|c)}{P(e|c)}$, or simply probability 1 if the ratio is greater than 1. Eventually, this sequence converges – but, if the new error is allowed to sample across different logical errors, the ratio will be exponential in $-L$, causing convergence to take time exponential in L, where L is the linear size of the lattice.

To mitigate this exponential complexity, parallel tempering is employed, whereby some number N of Markov chains as described above are run in parallel. Each Markov chain uses a slightly higher error rate p, such that the N^{th} chain has a ratio of 1 for all errors, allowing the errors to be sampled from all logical errors without incurring any cost to this chain. However, we need the distribution of the first chain to have these unrestricted errors. After a given number of iterations of the Markov chains, the current error e in each chain is swapped with the neighbouring chain according to probability exponential in the difference in the number of physical errors. Once enough errors have propagated to the original chain, we start checking for convergence and stop when the original chain converges.

While this hardly constitutes a precise mathematical description of the algorithm, the details are beyond the scope of this paper and left to [7]. The complexity of this algorithm is given as $O(e^{(\ln N)^{1.4}} L \log L)$ where l is the linear size of the lattice and N is the num-

ber of chains. In practice, the authors suggest that $N = O(L)$ works, giving complexity subexponential in L.

Simulations of this error decoding algorithm have been performed on the surface code, under the depolarizing channel and perfect syndrome measurements, for lattice sizes up to 65 [7]. A remarkable threshold value of 0.185 was observed, offering a significant improvement over the thresholds obtained by both maximum matchings and renormalization. While the algorithm itself is most likely not tractable for the timescales of practical quantum computing, the high threshold suggests that it may be possible to find efficient algorithms, possibly using some of these techniques, that achieve better error correction than any of the current efficient decoding algorithms.

6 Conclusion

While significant work has bee done to make efficient, accurate decoding procedures for the toric and related codes, it appears there is still a lot of room for improvements. Both maximum matching and renormalization group methods admit decoding procedures with potentially high thresholds compared to other codes, while remaining relatively efficient and tractable. While the renormalization group algorithm has excellent complexity, a realistic decoding procedure should account for syndrome errors, and so maximum matching appears to be the most practical decoding procedure currently.

By demonstrating thresholds as high as 0.185 in the planar code on the other hand, one might wonder if there exist efficient algorithms reaching these high thresholds. Given that a simple change from using the Manhattan distance to the ln of the probability created a profound difference in the observed threshold [3], there may still be room for improvement in matching-based algorithms by tweaking edge weighting. Either way, it is clear that high simulated thresholds are possible, and future work should attempt to attain such thresholds in a computationally tractable way.

References

[1] A. Fowler, A. Whiteside, L. Hollenberg, *Towards practical classical processing for the surface code.* arXiv:1110.5133v1 (2011).

[2] A. Fowler, A. Whiteside, L. Hollenberg, *Towards practical classical processing for the surface code: timing analysis.* arXiv:1202.5602v1 (2012).

[3] D. Wang, A. Fowler, L. Hollenberg, *Quantum computing with nearest neighbor interactions and error rates over 1%.* arXiv:1009.3686v1 (2010).

[4] D. Wang, A. Fowler, A. Stephens, L. Hollenberg, *Threshold error rates for the toric and surface codes.* arXiv:0905.0531v1 (2009).

[5] G. Duclos-Cianci, D. Poulin, *A renormalization group decoding algorithm for topological quantum codes*. arXiv:1006.1362v1 (2010).

[6] G. Duclos-Cianci, D. Poulin, *Fast Decoders for Topological Quantum Codes*. arXiv:0911.0581v2 (2010).

[7] J. Wootton, D. Loss, *High threshold error correction for the surface code*. arXiv:1202.4316v2 (2012).

[8] A. Kitaev, *Fault-tolerant quantum computation by anyons*. arXiv:quant-ph/9707021v1 (1997).

[9] V. Kolmogorov, *Blossom V: A new implementation of a minimum cost perfect matching algorithm*. Mathematical Programming Computation 1 (2009).

[10] H. Bombin, R. Andrist, M. Ohzeki, H, Katzgraber, M. Martin-Delgado, *Strong resilience of topological codes to depolarization*. arXiv:1202.1852v1 (2012).