Achieving Thresholds via Standalone Belief Propagation on Surface Codes

Pedro Hack,[1,*] Luca Menti,[1,†] Francisco Lázaro,[1,‡] and Alexandru Paler[2,§]

[1] *German Aerospace Center, Germany*
[2] *Aalto University, Finland*

The standard approach to Belief Propagation (BP) decoding passes messages on the Tanner graph of the quantum error-correcting (QEC) code, and is known to lack a threshold for the surface code. We propose novel BP decoders that exchange messages on the decoding graph and obtain thresholds (both for code capacity and circuit-level noise) via standalone BP for the surface code under depolarizing noise. Our approach, similarly to the minimum weight perfect matching (MWPM) decoder, is applicable to any graphlike QEC code. The thresholds observed with our decoders are close to those obtained by MWPM. This result opens the path towards scalable hardware-accelerated implementations of MWPM-compatible decoders.

I. INTRODUCTION

Standard belief propagation (BP) is known to lack a threshold for the surface code, a failure usually attributed to the highly loopy structure of the code's Tanner graph. In this work we show that this failure is not intrinsic to BP, but a consequence of the graph on which messages are passed. We move the message-passing from the code's Tanner graph to the decoding graph, the graph on which MWPM operates, and recover threshold behavior for the surface code, despite the presence of short cycles in the latter graph.

A. Motivation

For graphlike CSS codes, where each physical qubit participates in at most two X- and two Z-checks, the standard decoding algorithm is minimum-weight perfect matching (MWPM). Graphlike CSS codes include several important codes [1–6] such as the surface code. At the same time, graphlike approaches like MWPM can be adapted to more general codes [7–9].

The complexity of a naïve implementation of MWPM scales like $O(n^3 \log n)$, where n is the number of physical qubits [10]. In general, this is considered to be prohibitively slow for decoding large scale fault-tolerant quantum computations, which hence motivates the study of fast MWPM implementations whose performance does not degrade abruptly [11–13].

Herein, we explore the use of standalone BP-based approaches to MWPM as a decoder for quantum error correction. We present algorithms which save computational time when compared to MWPM by taking a BP approach to the problem, that is, by **running message-passing on the QEC decoding graph**, the same graph where MWPM operates. This contrasts with the standard BP approach to decoding [14, 15], which performs message-passing directly on the QEC code's Tanner graph.

Compared to the standard approach of running BP on the Tanner graph, our algorithms have slightly higher time complexity owing to the larger size of the graph on which they operate (Sec. II). However, they can significantly improve the performance of BP decoding, and in contrast to vanilla BP [16] which does not achieve threshold, our algorithms are achieving it (Sec. III).

While message-passing algorithms for computing matchings have been studied extensively [17–21], their application to quantum error correction has not been explored, and empirical evidence of their practical performance remains limited. Moreover, prior work primarily focused on approximating MWPM solutions, whereas our focus is on evaluating their performance under constrained time complexity.

B. MWPM on the surface code

Our BP-based method is applicable as a direct replacement of vanilla MWPM. For simplicity, we focus on uncorrelated decoding in the surface code, and decode X and Z Pauli errors separately. In the following, we fix the case of Z Pauli errors and refer to the X parity check matrix by $\boldsymbol{H}$. Moreover, we explain our method using the code capacity noise model. This means that what we call check refers to a detection event.

MWPM works on the **decoding graph** $\mathcal{G} = (\mathcal{V}_d, \mathcal{E}_d)$. For each error realization $\boldsymbol{e}$, the decoding graph consists of one node $i \in \mathcal{V}_d$ for each unsatisfied check, as well as one edge $(i, j) \in \mathcal{E}_d$ for each pair of unsatisfied checks. The edge (i, j) represents the fact that one can map the unsatisfied check associated to i to the one associated to j. Additionally, for each unsatisfied check associated with some vertex $i \in \mathcal{V}_d$, the decoding graph contains a **boundary node** $i_b \in \mathcal{V}_d$ and a boundary edge $(i, i_b) \in \mathcal{E}_d$. The boundary node and the edge represent the fact that one can map the unsatisfied check to the boundary.

For a given error realization $\boldsymbol{e}$, we will denote by s its associated number of unsatisfied checks. In this scenario, the decoding graphs consists of $|\mathcal{V}_d| = 2s$ nodes

* pedro.hack@dlr.de
† luca.menti@dlr.de
‡ francisco.lazaroblasco@dlr.de
§ alexandru.paler@aalto.fi

and $|\mathcal{E}_d| = s(s-1)/2 + s$ edges. For a given physical error rate p, and for large n, we expect to have pn faulty qubits. If that is the case, then the maximum expected number of unsatisfied checks is $2pn = O(n)$. This takes place whenever the faulty qubits do not share any check with each other, so each of them triggers two checks in a graph-like code.

As a result, taking into account the boundary vertices, the decoding graph has at most $|\mathcal{V}_d| = 2pn + 2pn = O(n)$ nodes and $|\mathcal{E}| = 2pn(2pn-1)/2 + 2pn = O(n^2)$ edges. Fig. 1 shows an error realization and its associated decoding graph.

MWPM associates a weight to each edge in the decoding graph and runs the Blossom algorithm [10, 22] in order to find an error $\boldsymbol{e'}$ with minimal weight that has the observed syndrome $\boldsymbol{He'} = \boldsymbol{He}$. Instead of doing this, we associate a **factor graph** $\mathcal{F}$ to the decoding graph and correct errors by running BP on $\mathcal{F}$.

II. METHODS

A. Bipartite Graph Representation

We define the bipartite graph $\mathcal{F}$ associated to the decoding graph $\mathcal{G}$ as $\mathcal{F} = (\mathcal{V}, \mathcal{C}, \mathcal{A})$ where

- $\mathcal{V} = \{\mathtt{v}_{1,1}, \ldots, \mathtt{v}_{1,s}, \ldots, \mathtt{v}_{s,1}, \ldots, \mathtt{v}_{s,s}\}$ is the set of variable nodes. Variable nodes in the bipartite graph are associated to edges in the decoding graph. In particular, variable node $\mathtt{v}_{i,j}$, $i \neq j$, is associated to edge (i,j) in the decoding graph $\mathcal{G}$, i.e., to an edge connecting the i-th and j-th unsatisfied syndromes. Similarly, variable node $\mathtt{v}_{i,i}$ is associated to the edge (i, i_b) in $\mathcal{G}$, i.e., to an edge connecting the i-th unsatisfied syndrome to the boundary. Although we have presented them independently, note that $\mathtt{v}_{i,j} = \mathtt{v}_{j,i}$, i.e. there are $s(s-1)/2 + s$ **independent** variables in $\mathcal{V}$.
- $\mathcal{C} = \{\mathtt{c}_1, \ldots, \mathtt{c}_s\}$ is the set of factor nodes associated to the unsatisfied syndromes.
- $\mathcal{A}$ is the set of (undirected) edges of the bipartite graph. The bipartite graph contains $s(s-1)/2 + s$ edges, one for each independent variable. In particular, we have that every factor node $\mathtt{c}_i$ is connected to the variable nodes $\mathtt{v}_{i,1}, \ldots, \mathtt{v}_{i,s}$, i.e., it has exactly s incident edges. That is, variable node $\mathtt{v}_{i,j}$, $i \neq j$, is connected to factor nodes $\mathtt{c}_i$ and $\mathtt{c}_j$, whereas variable node $\mathtt{v}_{i,i}$ is connected only to $\mathtt{c}_i$.

The random variables $v_{i,j}$ associated to the variable nodes $\mathtt{v}_{i,j}$ take binary values. In particular, we associate:

- the value $v_{i,j} = 1$ to the corresponding edge in the decoding graph $\mathcal{G}$ being included in the matching;
- and the value $v_{i,j} = 0$ to the corresponding edge in $\mathcal{G}$ not being included in the matching.

For the sake of notational simplicity, we will use $v_{i,j}$ to refer to the binary random variables, as well as their corresponding realizations.

Each factor node $\mathtt{c}_i \in \mathcal{C}$ in our bipartite graph $\mathcal{F}$ encodes a local compatibility constraint. In particular, the constraint is that only one out of the s variable nodes it is connected to can take value one, that is, that each unsatisfied syndrome is invovled in a single matching. Formally:

$$\mathtt{c}_i(v_{i,1}, \ldots, v_{i,s}) = \begin{cases} 1 & \text{if } \sum_{j=1}^{s} v_{i,j} = 1, \\ 0 & \text{else,} \end{cases} \tag{1}$$

The prior probability associated to random variable $v_{i,j}$, $\rho_{i,j}$, corresponds to

$$\rho_{ij} = \begin{cases} \left(\frac{p}{1-p}\right)^{w_{i,j}} & \text{if } v_{i,j} = 1, \\ 1 - \left(\frac{p}{1-p}\right)^{w_{i,j}} & \text{if } v_{i,j} = 0, \end{cases} \tag{2}$$

where $w_{i,j}$, $i \neq j$, is the weight of the lowest-weight error that matches syndromes i and j, whereas $w_{i,i}$ is the weight of the lowest-weight error that matches syndrome i to the boundary.

Overall, the bipartite graph $\mathcal{F}$ describes the decomposition of the probability distribution $\mathbb{P}_0$ into local factors defined over the configurations of $v_{1,1}, \ldots, v_{s,s}$:

$$\mathbb{P}_0(v_{1,1}, \ldots, v_{1,s}, \ldots, v_{s,1}, \ldots, v_{s,s}) \propto \prod_{1 \leq i \leq s} \mathtt{c}_i(v_{i,1}, \ldots, v_{i,s}) \prod_{1 \leq j \leq k \leq s} \rho_{jk}(v_{j,k}). \tag{3}$$

We have omitted a normalization constant in (3).

The factor nodes $\mathtt{c}_i$ strictly enforce the matching constraint by assigning a probability of zero to any invalid variable configuration. The remaining valid configurations, which correspond to perfect matchings, are then assigned probabilities proportional to the product of the independent prior probabilities $\rho_{i,j}$ of their constituent edges. Consequently, finding the minimum weight perfect matching is equivalent to identifying the configuration that maximizes the distribution $\mathbb{P}_0$.

B. Belief Propagation

Maximizing $\mathbb{P}_0$ is a computationally demanding task. Instead, in this paper, we aim at a low-complexity approximation of this solution. In particular, we apply the sum-product belief propagation algorithm over the factor graph $\mathcal{F}$. This yields the marginal probability of each variable in $\mathcal{F}$, which represents the likelihood of that variable being present in a valid matching, weighted by the overall likelihood of the matchings it belongs to. We then rely on these marginals to heuristically approximate (see Section II D) the specific configuration that maximizes $\mathbb{P}_0$.

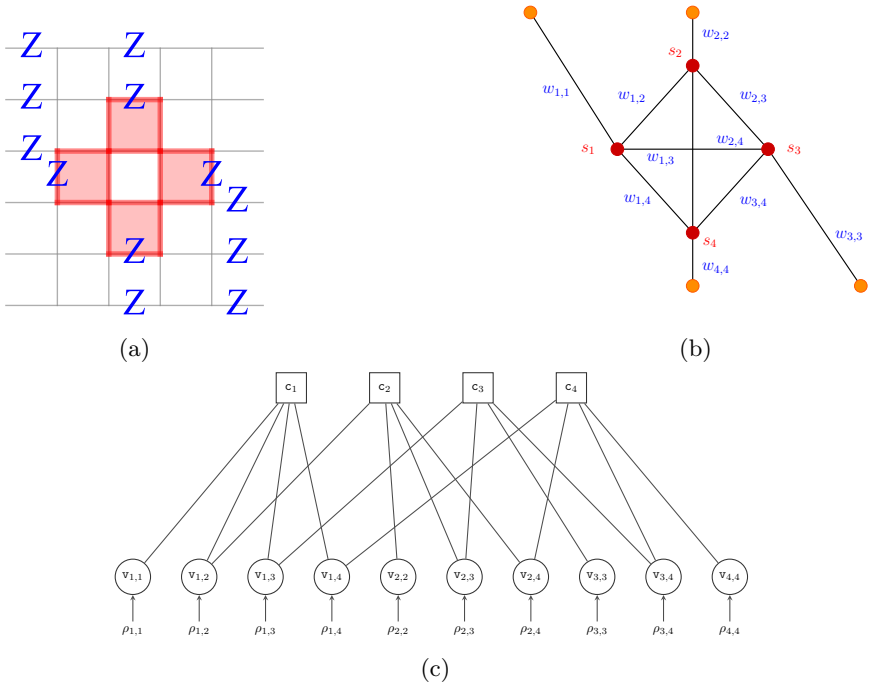


FIG. 1: Decoding and factor graphs. (a) Example error realization $\boldsymbol{e}$ in the $d = 5$ unrotated surface code. Z phase-flip errors are blue and unsatisfied syndrome plaquettes are red. (b) Decoding graph associated to $\boldsymbol{e}$. The unsatisfied syndromes are associated to red nodes and the boundary nodes are orange. We include one edge for each pair of unsatisfied syndromes and one edge that joins each unsatisfied syndrome to the boundary. Edge (i, j) has a weight $\omega_{i,j} \equiv (p/(1-p))^{w_{i,j}}$, where p is the physical error rate and $w_{i,j}$, $i \neq j$, is the weight of the shortest error path that satisfies the syndromes connected by the edge, whereas $w_{i,i}$ is the shortest path from s_i to the boundary. (c) Factor graph associated to $\boldsymbol{e}$. Each unsatisfied syndrome i is associated to a factor c_i (1) and a variable $\mathtt{v}_{i,i}$. Each pair of unsatisfied syndromes i, j $(i < j)$ is associated to a variable $\mathtt{v}_{i,j}$. Each variable $\mathtt{v}_{i,j}$, $i \leq j$, is associated to a prior probability ρ_{ij} (2).

In the sum-product algorithm, factor and variable nodes exchange messages over the edges $\mathcal{A}$ of the bipartite graph $\mathcal{F}$.For numerical stability and notational convenience, we provide a log-domain implementation of the sum-product algorithm. The log-likelihood ratio associated to the prior probability distribution (2) of a variable node $\mathtt{v} \in \mathcal{V}$ is denoted by $\ell_{\mathtt{v}}$. For $\mathtt{v} = \mathtt{v}_{i,j}$,

$$\ell_{\mathtt{v}_{i,j}} = \ln \frac{\rho_{i,j}(v_{i,j} = 1)}{\rho_{i,j}(v_{i,j} = 0)} \tag{4}$$

The message sent by factor node $\mathtt{c}$ to variable node $\mathtt{v}$ at the t-th iteration is denoted by $m^{(t)}_{\mathtt{c}\to\mathtt{v}}$, whereas the message sent from $\mathtt{v}$ to $\mathtt{c}$ is denoted by $m^{(t)}_{\mathtt{v}\to\mathtt{c}}$. Initially, the messages from variables to factors $m^{(0)}_{\mathtt{v}\to\mathtt{c}}$ are initialized to $\ell_{\mathtt{v}}$, whereas the messages from factors to variables $m^{(0)}_{\mathtt{c}\to\mathtt{v}}$ are initialized to zero. In the subsequent iterations, the messages from variable nodes to factor nodes are updated as

$$m^{(t+1)}_{\mathtt{v}\to\mathtt{c}} = \ell_{\mathtt{v}} + \sum_{\mathtt{c}'\in\mathcal{N}(\mathtt{v})\setminus\mathtt{c}} m^{(t)}_{\mathtt{c}'\to\mathtt{v}}$$

whereas the messages from factor to variable nodes are updated as

$$m^{(t+1)}_{\mathtt{c}\to\mathtt{v}} = -\max^{*}_{\mathtt{v}'\in\mathcal{N}(\mathtt{c})\setminus\mathtt{v}} \left(m^{(t)}_{\mathtt{v}'\to\mathtt{c}}\right)$$

where $\max^*$ is the soft-max operator, defined over a set of variables $x_1, \ldots, x_n$ as:

$$\max^{*}(x_1, \ldots, x_n) = \ln\left(\sum_{i=1}^{n} \exp(x_i)\right)$$

In practice, computing the multi-argument soft-max operator directly via exponentials can lead to numerical instability. Therefore, the soft-max operation is typically evaluated recursively using its two-argument form, sometimes known as the Jacobian logarithm:

$$\max^{*}(x, y) \approx \max(x, y) + \ln(1 + \exp(-|x - y|)).$$

Similarly, the posterior probability of the random variable associated with $\mathtt{v}$ after the t-th iteration is obtained as

$$\lambda^{(t+1)}_{\mathtt{v}} = \ell_{\mathtt{v}} + \sum_{\mathtt{c}\in\mathcal{N}(\mathtt{v})} m^{(t)}_{\mathtt{c}\to\mathtt{v}}. \tag{5}$$

for a predefined maximum number of iterations T.

Each variable node is connected to at most two factor nodes, so each of its outgoing messages can be updated in $O(1)$ time. There are $O(n^2)$ variable nodes, and the update of one is independent of the others. With $O(n^2)$

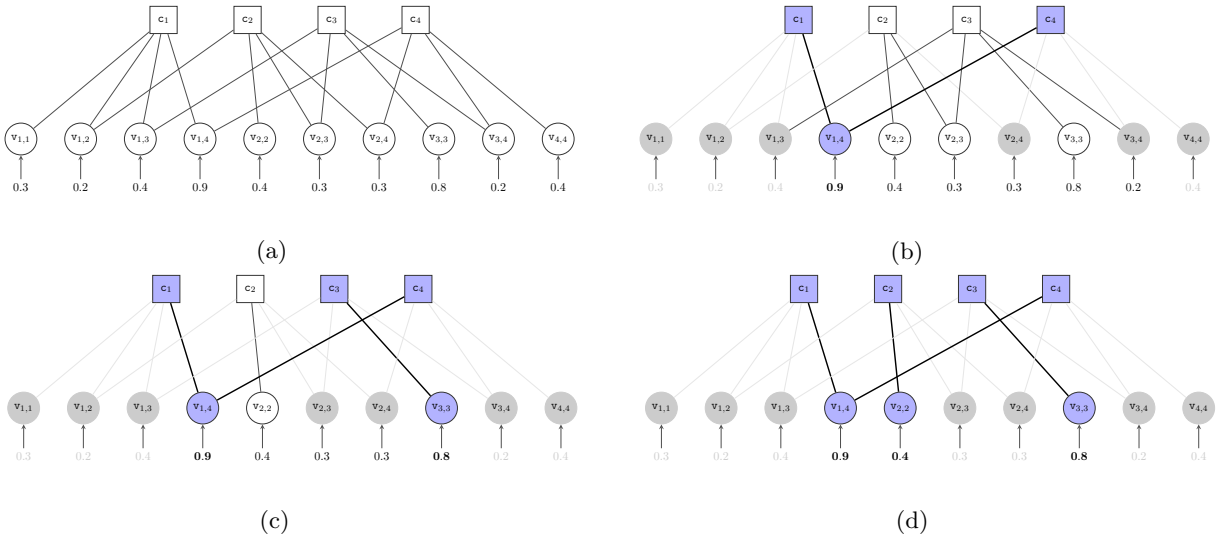


FIG. 2: Marginalization and forced convergence: (a) An example marginalization output for the error candidate in Fig. 1. We include the (probability) marginalization result after T iterations $\mathbb{P}^{(T)}(v = 1)$ for each variable $\mathtt{v} \in \mathcal{V}$. (b) The first decision made by forced convergence. We color in blue the chosen variable together with the unsatisfied syndromes it matches. We color in gray the variables and the edges that are eliminated for future steps because of the choice. (c) The second decision made by forced convergence. This decision, together with the previous one constitute the output of marginalization. Marginalization is not guaranteed to converge and, in fact, in this example it does not. (d) The third decision made by forced convergence. This decision, together with the previous ones, constitute the output of forced convergence. Forced convergence is guaranteed to converge and, in fact, in this example it does.

parallel resources, all of them can therefore be updated simultaneously in $O(1)$ time. With only $O(n)$ resources, each of them handles $O(n)$ variable nodes serially, and the update of all variable nodes takes $O(n)$ time. In contrast, the factors are adjacent to $O(n)$ variable nodes. Using the leave-one-out approach [23], all messages outgoing from a factor node can be computed in $O(n)$ time in total. Since all factor nodes can be updated in parallel (i.e. we assume we have access to $O(n)$ parallel resources), each iteration over them takes time $O(n)$. Lastly, saving the messages requires $O(n^2)$ space. Overall, each iteration can be done in $O(n)$ time and requires $O(n^2)$ space.

C. Checking convergence

Checking convergence in our algorithms is useful since there is no guarantee that message-passing will achieve it. Hence, if we check for it early, it can save us computational time.

Assuming that we have inferred an error candidate, checking for convergence requires for each factor node to sum over the inferred values of the variable nodes adjacent to it. If the sum is equal to one for all factor nodes, then we have converged. This can be parallelized for each factor node and, as a result, it requires $O(n)$ time.

D. Forced convergence

We infer an error candidate $\boldsymbol{e'}$ by using the set of messages from checks to variables. We consider two inference methods: **marginalization** and **forced convergence**.

In marginalization, we infer the posterior probability distribution $\lambda_{\mathtt{v}}^{(t)}$ for each individual variable $\mathtt{v} \in \mathcal{V}$ and include $\mathtt{v}$ in the matching provided $\lambda_{\mathtt{v}}^{(t)} > 0$.

The marginalization over each variable requires $O(1)$ time since each variable node is only connected to at most two factor nodes. Since we can marginalize each variable in parallel, we only require $O(1)$ time for marginalizing provided we use $O(n^2)$ space (and, as in the case of variable to factor updates, $O(n)$ time if we only use $O(n)$ space). Fig. 2 is an example where marginalization does not converge.

Given that marginalization decides locally on each variable, we are not guaranteed to converge. Fig. 8b provides data for the number of times we do not obtain (through marginalization) a converged error candidate in any of the $T = 25$ iteration steps.

On the other hand, forced convergence uses the posteriors computed by marginalization for ordering all variable nodes in terms of their probability of being part of the matching. We then recursively pick the variable with the highest probability until we obtain a matching. The recursive decision on the variables together with their ordering in terms of likelihood provide global information that always allows us to provide an error candidate matching the syndrome. Forced convergence is somewhat

Algorithm	Noise mod.	Iter. Comp.	#Iter. (T)	Inf. Comp.	Overall Comp.	Threshold
BP4M	Capacity	$O(n)$	25	$O(n \log n)$	$O(Tn + n \log n)$	0.1170
BP4MF	Capacity	$O(n \log n)$	25	-	$O(Tn \log n)$	0.1340
BP4M+M	Capacity	$O(n)$	25	$O(r_{nc} n^3 \log n)$	$O((1 - r_{nc})Tn + r_{nc} n^3 \log n)$	0.1570
B-BP4MF	Capacity	$O(n \log n)$	$25 + 25$	-	$O(Tn \log n)$	0.1500
MWPM	Capacity	-	-	-	$O(n^3 \log n)$	0.1550
BP4M	Circuit-level	$O(n)$	50	$O(n \log n)$	$O(Tn + n \log n)$	0.0050
BP4MF	Circuit-level	$O(n \log n)$	50	-	$O(Tn \log n)$	0.0055
B-BP4MF	Circuit-level	$O(n \log n)$	$75 + 75$	-	$O(Tn \log n)$	0.0061
MWPM	Circuit-level	-	-	-	$O(n^3 \log n)$	0.0077

TABLE I: Algorithms' worst case complexity and thresholds for the unrotated surface code. The second column is the noise model (circuit-level stands for memory-Z experiment); the third column is the complexity per iteration; the fourth column is the number of iterations; the fifth column is the complexity of inference after BP (a hyphen indicates that this step is not needed since BP always converges); the sixth column is the overall complexity; the seventh column is the observed threshold. We have taken the MWPM threshold from the literature [24] for code capacity, and we have simulated it through Pymatching [10] for circuit-level noise. The complexities reported here are for a parallel implementation (see Sections II and III).

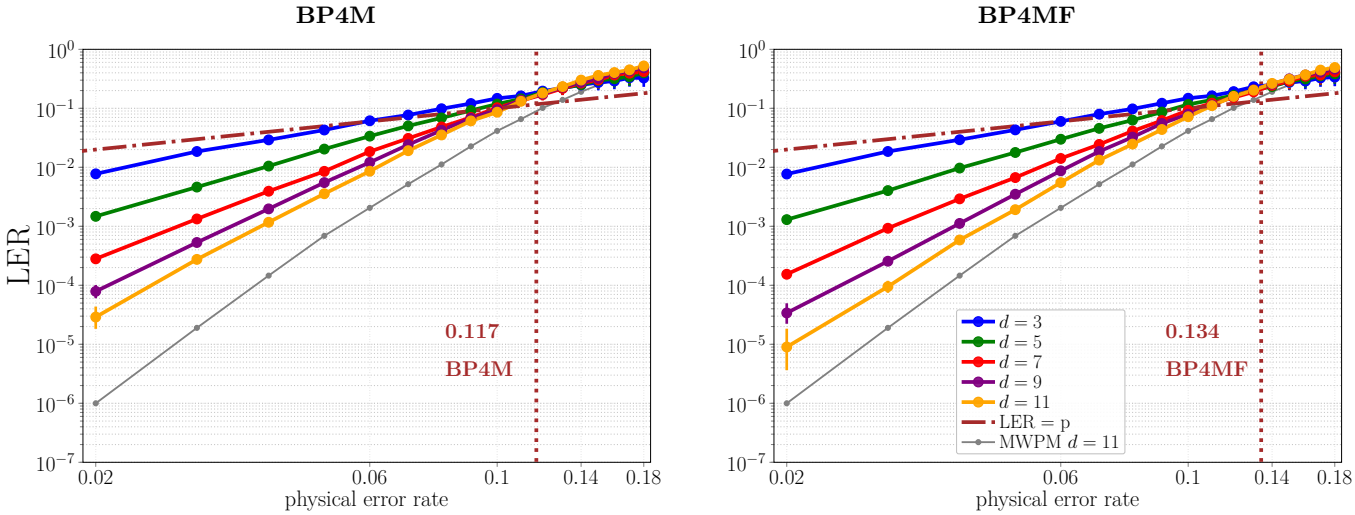


FIG. 3: Code capacity unrotated surface code. We ran our algorithms for 25 iterations. The horizontal red dotted line is the **pseudo-threshold** (i.e. the curve where the logical error rate equals the physical error rate) and the vertical dotted line indicates the achieved threshold. All subfigures include a gray LER curve that represents pure MWPM distance 11 decoding performance. This is for comparison reasons with the corresponding distance 11 LER curve of our methods (orange).

reminiscent of **guided decimation** [25].

To force the convergence, we associate to each $\mathsf{c} \in \mathcal{C}$ a subset of its adjacent variable nodes

$$\mathcal{E}_{\mathsf{c}} \subseteq \mathcal{N}(\mathsf{c})$$

chosen so that that these subsets partition the variable nodes,

$$\mathcal{E} = \cup_{\mathsf{c} \in \mathcal{C}} \mathcal{E}_{\mathsf{c}} \text{ and } \mathcal{E}_{\mathsf{c}} \cap \mathcal{E}_{\mathsf{c}'} = \emptyset.$$

This partition can be pre-computed assuming each syndrome is unsatisfied and obtaining a partition for the complete set of variable nodes that the decoding graph may be. Each $\mathcal{E}_{\mathsf{c}}$ can then be in parallel adapted to the observed syndrome for a total cost of $O(n)$.

Given the partition, we marginalize each variable in $O(1)$ and compute the block maxima

$$\mathsf{v}^{\mathsf{c}}_{\max} \equiv \arg\max_{\mathsf{v} \in \mathcal{E}_{\mathsf{c}}} \left\{ \lambda^{(t)}_{\mathsf{v}} \right\}$$

one for each $\mathsf{c} \in \mathcal{C}$. Since we can do this for each of them in parallel, this requires $O(n)$.

Next, we build the vector of maxima

$$\mathsf{v}_{\max} = (\mathsf{v}^{\mathsf{c}_1}_{\max}, \ldots, \mathsf{v}^{\mathsf{c}_s}_{\max})$$

and reorder it $\mathsf{v}^{\pi}_{\max}$ such that

$$\lambda^{(t)}_{(\mathsf{v}^{\pi}_{\max})_i = 1} \geq \lambda^{(t)}_{(\mathsf{v}^{\pi}_{\max})_{i+1}}$$

for $i = 1, \ldots, s-1$. When performed serially, reordering requires $O(n \log n)$, but there exist parallel hardware implementations [26] with even lower complexity.

We then process $\mathsf{v}^{\pi}_{\max}$ starting from its first entry, which carries the largest posterior. At each step, we take the first remaining entry, say this entry is associated to check node c_i, and proceed as follows. If its representative is the boundary variable $\mathsf{v}_{i,i}$, we match c_i to the boundary and remove the entry associated to c_i from

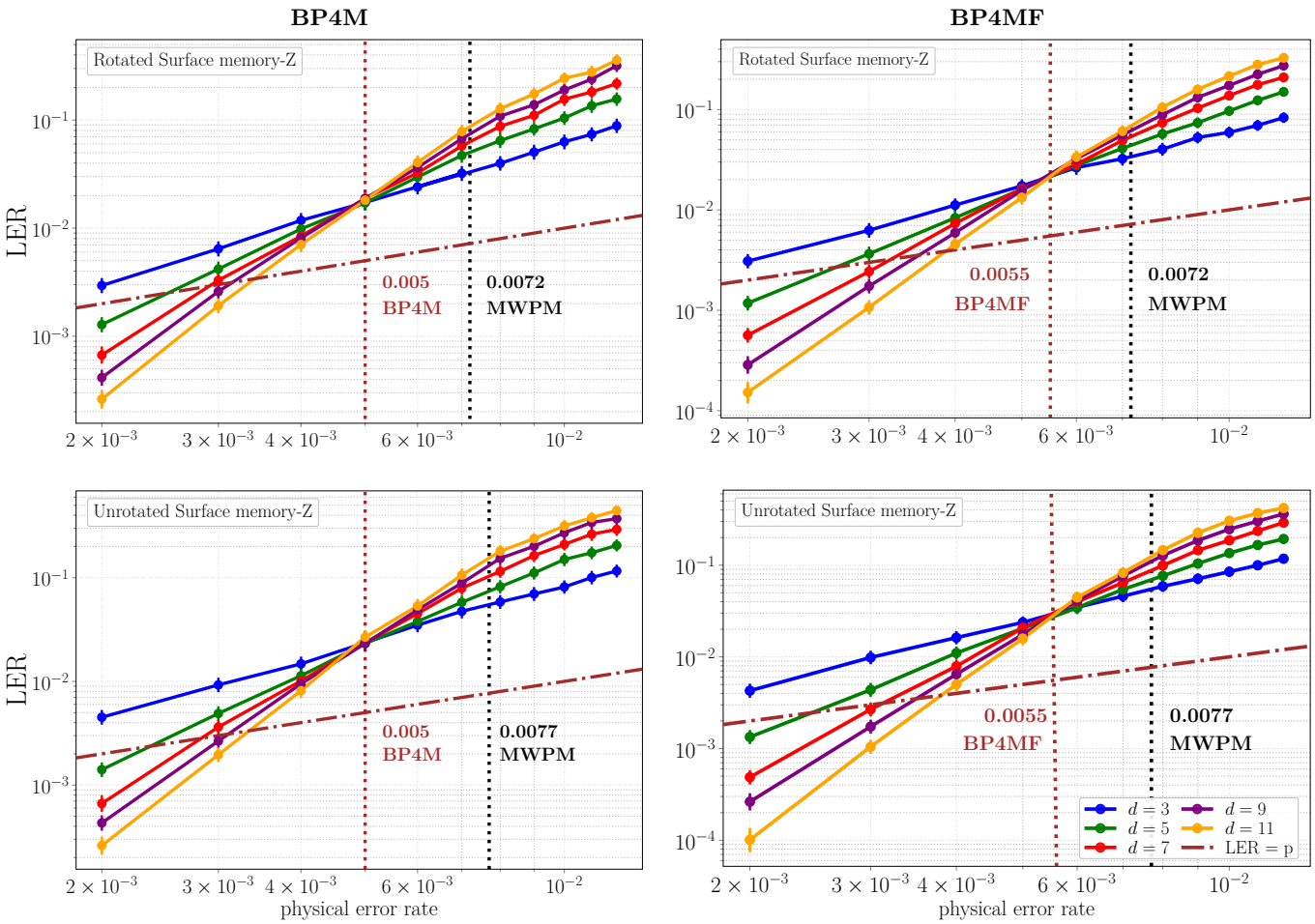


FIG. 4: Circuit-level noise memory-Z surface code (rotated and unrotated). We ran our algorithms for 50 iterations. The columns determine the algorithm, while the rows determine the code used. The horizontal red dotted line is the pseudo-threshold and the vertical dotted line indicates the achieved threshold.

$\mathtt{v}^{\pi}_{\max}$. Otherwise the representative is $\mathtt{v}_{i,k}$ for some $k \neq i$, in which case we match $\mathtt{c}_i$ to $\mathtt{c}_k$ and remove from $\mathtt{v}^{\pi}_{\max}$ both the entry associated to $\mathtt{c}_i$ and the one associated to $\mathtt{c}_k$. We proceed in this way until every entry has been processed.

Since an entry is removed from $\mathtt{v}^{\pi}_{\max}$ exactly when its check is matched, every check is matched exactly once, which yields a valid matching, i.e., an error candidate that reproduces the observed syndrome. Processing the entries costs $O(n)$, so forced convergence requires $O(n \log n)$ overall, dominated by the reordering.

It is worth noting that, in case marginalization converges, running forced convergence with the same BP messages will provide the same output as marginalization. Hence, at a given iteration t, it is convenient to check first whether marginalization converged, at a cost of $O(n)$, in which case one can skip forced convergence, and save its $O(n \log n)$ cost. For illustration, Fig. 2 is an example where marginalization does not converge and forced convergence does.

E. Correlated decoding

MWPM can approximate the joint decoding of X and Z Pauli errors by first running quaternary BP on the Tanner graph and then using the BP qubit posteriors in order to compute the input weights required by MWPM. This approach is known as Belief-matching [27].

We can emulate Belief-matching to incorporate correlations into our scheme. In order to compute the prior probabilities that we associate with the edges in the decoding graph, we:

1. Run quaternary BP on the full Tanner graph and compute its qubit posteriors.

2. Run Dijkstra's algorithm [28] in order to get the path of minimum weight (according to the qubit BP posteriors) that joins either two pairs of unsatisfied syndromes or one unsatisfied syndrome to the boundary.

The graph $\mathcal{G}_{\mathrm{D}} = (\mathcal{V}_{\mathrm{D}}, \mathcal{E}_{\mathrm{D}})$ on which Dijkstra's algorithm is run is quite close to the Tanner graph: it has one node $i \in \mathcal{V}_{\mathrm{D}}$ for each check. This includes virtual

checks, which are located at the check-free end of each qubit that is connected to a single check node. The graph has one edge $(i,j) \in \mathcal{E}_{\mathrm{D}}$ for each qubit, with $i,j \in \mathcal{V}_{\mathrm{D}}$ being the two check nodes adjacent to that qubit. One of these checks might be virtual.

There is some subtlety regarding the graph on which we run Dijkstra, depending on whether we allow corrections between pairs of syndromes to go through the boundary or not. If we do, then we can use a quotient graph $\mathcal{G}_{\mathrm{D}}/\!\sim = (\mathcal{V}_{\mathrm{D}}/\!\sim, \mathcal{E}_{\mathrm{D}}/\!\sim)$, where $\mathcal{V}_{\mathrm{D}}/\!\sim$ has one equivalence class for each non-virtual check and all the nodes associated to virtual checks are collapsed into a single equivalence class. Moreover, $i,j \in \mathcal{V}_{\mathrm{D}}/\!\sim$ are connected by an edge in $\mathcal{E}_{\mathrm{D}}/\!\sim$ provided one member of the equivalence class of i is connected to a member of that of j in $\mathcal{V}_{\mathrm{D}}$. We have given preference to $\mathcal{G}_{\mathrm{D}}$ over $\mathcal{G}_{\mathrm{D}}/\!\sim$ in order to help BP on the decoding graph: If we take $\mathcal{G}_{\mathrm{D}}/\!\sim$, we risk having some redundancy in the matchings. More specifically, a situation may arise where, given two unsatisfied syndromes, the solution that matches each of them separately to the boundary is equivalent to the one that matches them together (i.e. this matching also goes through the boundary). This symmetry may reduce BP's performance, hence we avoid it.

Regarding complexity, assuming we have $O(n)$ parallel resources with $O(1)$ space in each of them and hence $O(n)$ space in total, BP on the Tanner graph requires $O(Tn)$ time to do T iterations. Furthermore, given a starting node $i \in \mathcal{V}_{\mathrm{D}}$, Dijkstra's algorithm computes its distance to all other nodes in $O(|\mathcal{E}_{\mathrm{D}}| + |\mathcal{V}_{\mathrm{D}}| \log |\mathcal{V}_{\mathrm{D}}|)$. For the surface code, this requires $O(n \log n)$ time, assuming we can run $O(n)$ Dijkstra instances in parallel (spending $O(n)$ space in each Dijkstra and hence $O(n^2)$ space in total), i.e. one for each unsatisfied syndrome. This directly provides the weights associated to unsatisfied syndrome pairs that we need to initialize the BP approach to MWPM. In order to obtain the weights associated to matching an unsatisfied syndrome to the boundary, we need to minimize over all the Dijkstra paths joining its associated node to a virtual node. This adds an extra $O(n)$ time, assuming we can do it for all unsatisfied syndromes in parallel. Overall, adding correlations via this procedure requires $O(Tn + n \log n)$ time.

III. RESULTS

The core of our results are two BP-based algorithms: BP4M, BP4MF. Additionally, we build also a two-stage decoder BP4M+M(atching) and one that incorporates correlations B-BP4MF. Table I lists the time complexity of each algorithm. In the naive implementation we have adopted for simplicity, each algorithm requires $O(n^2)$ space in the worst case. We discuss space complexity further in Section IV.

A. Belief Propagation for Matching (BP4M)

Our first method is **belief propagation for matching (BP4M)**. Herein, we perform message-passing for a number of iterations T, performing marginalization at each iteration and, provided we converge in several iterations, we keep the error candidate with lowest weight among them. Once the iterations are over, and provided the last one did not converge, we perform forced convergence in order to get some error candidate for each syndrome. Lastly, if forced convergence was used, we compare the weight of the lowest-weight candidate coming from marginalization with that of forced convergence and output the error candidate with smallest weight between them. Otherwise, we output the lowest-weight candidate coming from marginalization.

The complexity of each iteration of BP4M is that of message-passing itself $O(n)$ plus that of the marginalization $O(1)$ for a total of $O(nT)$ when finishing the iterations. After that, forced convergence requires $O(n \log n)$, and we get an overall complexity of $O(nT + n \log n)$.

B. BP4M Forced (BP4MF)

The second method is called **belief propagation for matching forced (BP4MF)**. In this approach, we perform message-passing for a number of iterations T, performing forced convergence at each iteration and keeping the error candidate with lowest weight among them. For the same number of iterations T, we expect this approach to perform better than BP4M, since the weight of the final output will be upper bounded by that of BP4M. This is the case since, if marginalization converges, then forced convergence will output the same error candidate.

The complexity of each iteration of BP4MF is that of message-passing itself $O(n)$ plus that of forced convergence $O(n \log n)$ for a total of $O(Tn \log(n))$ when finishing the iterations. Before performing forced convergence, we can marginalize and check for convergence for $O(n)$, and avoid the $O(n \log n)$ cost of forced convergence. The number of times forced convergence cannot be avoided diminishes with the physical error rate (see Figure 8b).

C. BP4M + Matching (BP4M+M)

The third method is **belief propagation for matching plus matching (BP4M+M)**. This is a two-stage decoder, where the first stage is running BP4M and the second one vanilla MWPM. In this approach, we reproduce BP4M while we are performing iterations and, once we have reached T iterations, we check whether we have converged. If that is the case, then we output the converged error candidate with minimal weight like BP4M does. Otherwise, we perform MWPM. This method is somehow similar to [27], although the soft information from BP4M is not used to reweigh the MWPM graph.

The complexity of each iteration of BP4M+M is that of BP4M, that is, $O(nT)$ after the iterations have finished. If we do not obtain an error candidate that matches the observed syndrome after T iterations, then we use matching for an extra complexity of $O(n^3 \log n)$. As a result, if we call r_{nc} the **ratio of non-convergence**, i.e. the quotient of the number of times we do not converge and the number of errors we have generated, then the expected runtime is $O((1-r_{nc})nT + r_{nc}n^3 \log n)$. As we show in Fig. 8b, r_{nc} diminishes with the physical error rate and we get, for $p = 0.02$ and $d = 3, 5, 7, 9, 11$ under the code capacity noise model, that $r_{nc} \leq 0.1$.

D. Belief-BP4MF (B-BP4MF)

The fourth method is **quaternary belief propagation plus belief propagation for matching forced** or **belief-BP4MF (B-BP4MF)**. This is a two-stage decoder, where the first stage runs quaternary BP. If the first stage converges, we take its hard decision as output. Otherwise, we run Dijkstra's algorithm (see Section II E) in order for the BP4MF initial log-likelihoods to take into account correlations, and we then run BP4MF. One can substitute BP4MF by BP4M in order to get another decoder, although we only simulate the BP4MF case. This methods are the belief propagation for matching counterparts of belief-matching [27].

Since the time complexity of quaternary BP together with our Dijkstra subroutine is $O(Tn + n \log n)$ and that of BP4MF is $O(Tn \log n)$, the overall complexity of B-BP4MF is $O(Tn \log n)$. We use the same number of iterations $T = 75$ for both quaternary BP and BP4MF.

E. Numerical Results

We benchmark our decoding algorithms across code distances $d \in \{3, 5, 7, 9, 11\}$ using both code capacity and circuit-level noise models:

For code capacity noise, we evaluate the unrotated surface code subject to a standard depolarizing channel with physical error rates $p \in [0.02, 0.18]$.

For circuit-level noise, we consider both the unrotated and rotated memory-Z surface codes. The circuits are constructed using Stim's [29] standard `surface_code:rotated_memory_z` and `surface_code:unrotated_memory_z` generation, employing $R = d$ syndrome-extraction rounds, ending in a final data-qubit measurement. The noise model injects a depolarizing error (probability p) after every Clifford gate and onto data qubits before each syndrome extraction, with state preparation (reset) and measurement flip probabilities similarly set to p. We sweep the physical error rate from $p = 0.002$ to 0.012 for most configurations, while for the B+BP4MF decoder, we evaluate code distances $d \in \{5, 9, 13\}$ across rates from $p = 0.003$ to 0.012.

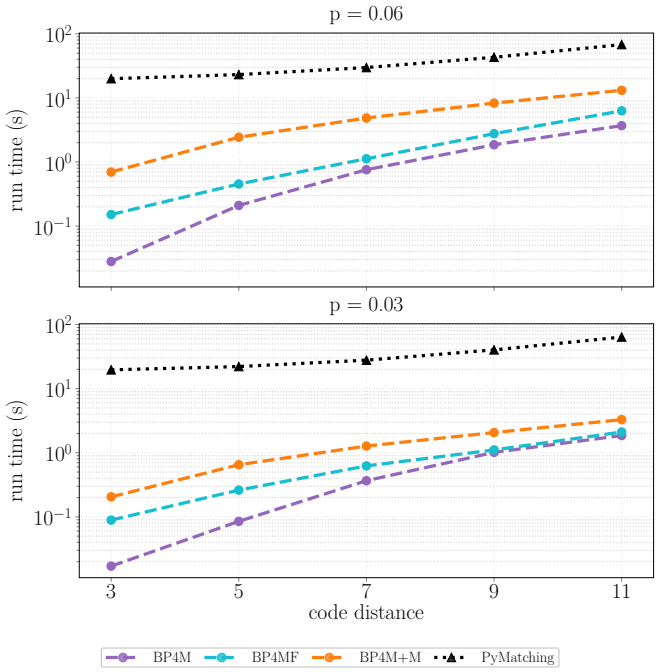


FIG. 5: Code capacity average runtimes (over 1 million errors) compared to MWPM. We use $T = 25$ iterations for our decoders.

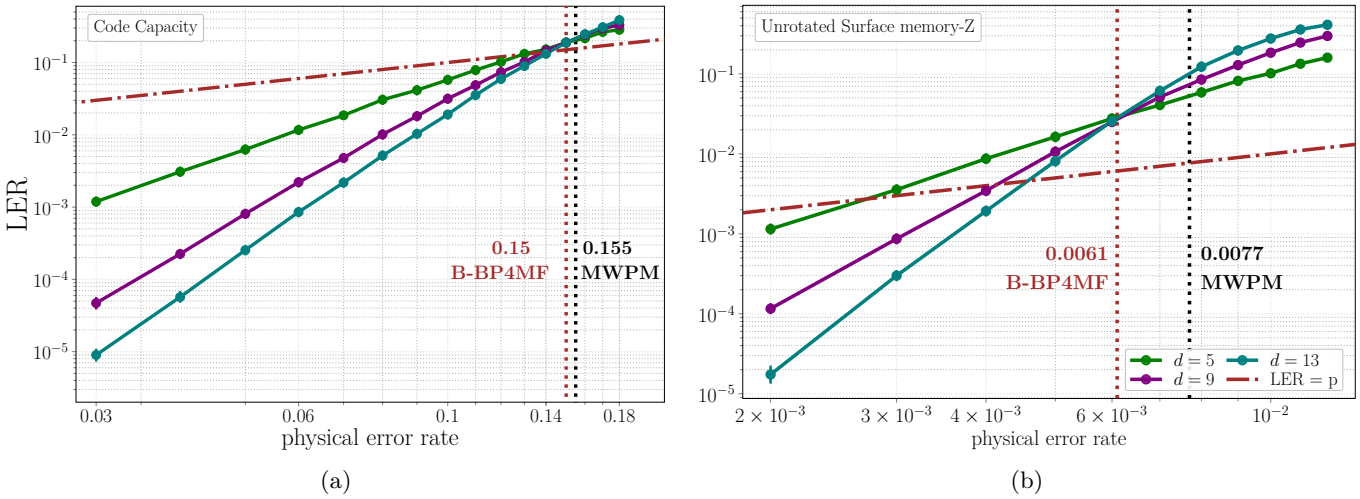
We implemented our code with Numpy and C++. Simulations were performed on a Linux workstation running Ubuntu 24.04.3 LTS (kernel 6.8.0-94-generic), equipped with an AMD EPYC 9274F (24 cores / 48 threads, single socket; up to 4.05 GHz) and 188 GB RAM, and 2× NVIDIA RTX 6000 Ada Generation GPUs (48 GB VRAM each). For comparison against vanilla MWPM, we utilize PyMatching [10]. This library also serves as the underlying MWPM subroutine within our BP4M+M decoder. We have also used the QECSIM library [30].

Since all our algorithms perform message-passing, their complexity and performance depend on the number of iterations T they use. We consider the following number of iterations: $T = 25$ for all code capacity simulations except for B-BP4MF, where we do $T = 25$ for quaternary BP and $T = 25$ for BP4MF. $T = 50$ for all circuit-level noise simulations except for B-BP4MF, where we do $T = 75$ for quaternary BP and $T = 75$ for BP4MF. Table I summarizes the number of iterations.

1. *Comparison to MWPM*

All of our algorithms have thresholds comparable to that of MWPM both for code capacity (Fig. 3, 8c and 6a) and for circuit-level noise (e.g. Fig. 4). See Table I for a summary of threshold across all algorithms and noise models. Furthermore, these comparable thresholds are achieved in a faster runtime than MWPM (Fig. 5).

All of our algorithms show good performance. One



(a) (b)

FIG. 6: B-BP4MF threshold results: Code capacity unrotated surface code with 25+25 iterations and circuit-level noise unrotated memory-Z surface code with 75+75 iterations. The horizontal red dotted line is the pseudo-threshold and the vertical dotted line indicates the achieved threshold.

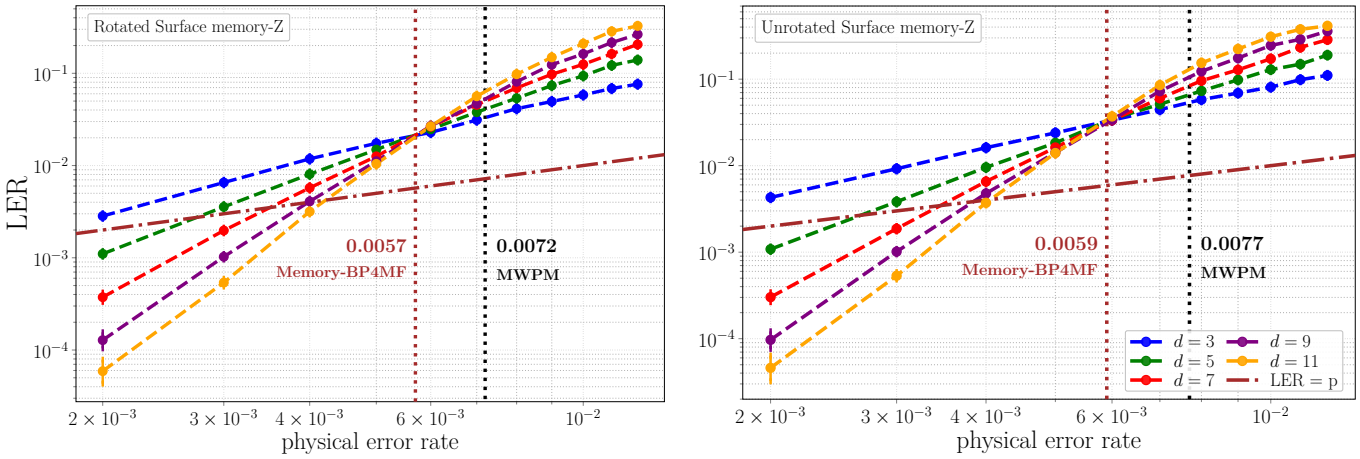


FIG. 7: Memory-BP4MF ($\alpha = 0.85$) circuit-level noise (rotated and unrotated) memory-Z surface code threshold plots with 50 iterations. The horizontal red dotted line is the pseudo-threshold and the vertical dotted line indicates the achieved threshold.

can see this in Fig. 9 for $d = 9, 11$. As expected, for the same number of iterations T, we get that BP4MF performs better than BP4M. When considering different values of T, we have noticed that, even in the intermediate iterations where we do not converge, forced convergence typically provides meaningful error candidates. In particular, we have concluded from our experiments that exploring more forced convergence candidates is better in terms of performance than running more iterations and only exploring the converged error candidates. Another conclusion that we draw from these experiments is that a low number of iterations already provide good results, with the performance improving slowly as T grows. Moreover, these improvements become more prominent as the physical error rate diminishes. Our experiments with T were performed under code capacity noise.

When we converge via marginalization, the performance is comparable to that of MWPM. The performance of BP4M+M (Fig. 8c) is almost indistinguishable from that of MWPM (Fig. 9). This indicates that, when our algorithms converge via marginalization, their performance is comparable to that of MWPM. To support this, we include in Fig. 8a the **converged logical error rate** or **converged LER**, which consists of the logical errors ignoring the cases where we do not converge. Moreover, the number of instances where we do not converge via marginalization diminishes with the physical error rate (Fig. 8b).

2. *BP Improvements*

Known improvements for BP on the Tanner graph can also be advantageous for our algorithms. An example of this is adding memory [31]. More specifically, when updating the messages from variable nodes to factor nodes,

we add a tunable parameter α, getting the following equation

$$m_{\mathtt{v}\to\mathtt{c}}^{(t+1)} = \ell_{\mathtt{v}} + \frac{1}{\alpha} \sum_{\mathtt{c}'\in\mathcal{N}(\mathtt{v})\setminus\mathtt{c}} m_{\mathtt{c}'\to\mathtt{v}}^{(t)}.$$

By doing a grid search with $\alpha = 0.6 + k \cdot 0.05$ and $k \in \{0, 1, \ldots, 20\}$, we manage to improve the performance of BP4MF (see Figure 7). We refer to this algorithm as Memory-BP4MF.

We have tried a couple more approaches that did not yield significant gains, at least in the code capacity setup. This includes serial scheduling of the messages and adding memory during marginalization.

IV. CONCLUSION

We have shown that the absence of a threshold for standard belief propagation (BP) on the surface code is not intrinsic to the algorithm, but a consequence of the graph on which messages are passed. Moving the message-passing to the decoding graph, the graph on which minimum weight perfect matching (MWPM) operates, recovers threshold behavior, despite the short cycles present in the latter graph.

Our algorithms achieve thresholds of 0.117 (BP4M) and 0.134 (BP4MF) under code capacity noise, against 0.155 for MWPM, and of 0.0050 and 0.0055 under circuit-level noise, against 0.0072 and 0.0077 for the rotated and unrotated memory-Z experiments. The forced convergence strategy of BP4MF, which guarantees a valid error candidate at every iteration, is what closes most of the gap at higher distances. Adding correlations in the style of belief-matching closes it further, raising the code capacity threshold to 0.150, at the cost of a second BP stage. Used instead as a pre-decoder that defers to matching only when marginalization fails, BP4M+M attains a threshold of 0.157 while calling MWPM on a small and rapidly decreasing fraction of syndromes, and is faster than PyMatching across the distances we considered. We also find that a small number of iterations suffices, and that heuristics developed for BP on the Tanner graph transfer to our setting: adding memory improves the circuit-level threshold of BP4MF to 0.0059.

A gap to MWPM remains. We attribute it primarily to the greedy nature of forced convergence, which fixes variables serially by reliability and therefore does not recover the globally optimal matching that Blossom provides, and to the independence assumption underlying the edge priors. Narrowing this gap without giving up the low per-iteration cost is, in our view, the main open problem raised by this work.

Several directions follow. The X and Z matchings are currently decoded independently, and improving their interaction is the natural route towards correlated decoding. The matching-based approaches to non-graphlike codes [7–9] may allow our algorithms to be applied beyond graphlike codes. Further BP heuristics are worth exploring, in particular concatenating several BP instances in series or using ensembles, as Relay BP does [32]. Our space complexity is $O(n^2)$ because we build the complete decoding graph. Sparse Blossom [13] shows that this can be avoided for matching, and doing so in our setting would be practically relevant. Finally, the per-iteration cost of our algorithms is $O(n)$ with $O(n)$ parallel resources and the message updates are local, which makes an FPGA or ASIC implementation a natural next step.

ACKNOWLEDGEMENTS

We thank Francisco García Herrero for feedback on the manuscript and Henry D. Pfister for useful discussions. This research was supported by the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. Luca Menti's research was funded by a scholarship awarded by the German Academic Exchange Service (DAAD).

We acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LUMI, hosted by CSC (Finland) and the LUMI consortium through a EuroHPC Regular Access call.

[1] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, Journal of Mathematical Physics **43**, 4452 (2002).

[2] N. P. Breuckmann and B. M. Terhal, Constructions and noise threshold of hyperbolic surface codes, IEEE transactions on Information Theory **62**, 3731 (2016).

[3] S. Bravyi, G. Duclos-Cianci, D. Poulin, and M. Suchara, Subsystem surface codes with three-qubit check operators, arXiv preprint arXiv:1207.1443 (2012).

[4] M. Suchara, S. Bravyi, and B. Terhal, Constructions and noise threshold of topological subsystem codes, Journal of Physics A: Mathematical and Theoretical **44**, 155301 (2011).

[5] D. Bacon, Operator quantum error-correcting subsystems for self-correcting quantum memories, Physical Review A—Atomic, Molecular, and Optical Physics **73**, 012340 (2006).

[6] M. B. Hastings and J. Haah, Dynamically generated logical qubits, Quantum **5**, 564 (2021).

[7] H. Bombin and M. A. Martin-Delgado, Topological quantum distillation, Physical review letters **97**, 180501 (2006).

[8] A. Kubica and N. Delfosse, Efficient color code decoders in $d \geq 2$ dimensions from toric code decoders, Quantum

7, 929 (2023).
[9] B. J. Brown, Conservation laws and quantum error correction: Toward a generalized matching decoder, IEEE BITS the Information Theory Magazine **2**, 5 (2023).
[10] O. Higgott, Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching, ACM Transactions on Quantum Computing **3**, 1 (2022).
[11] A. G. Fowler, Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $o(1)$ parallel time, arXiv preprint arXiv:1307.1740 (2013).
[12] N. Delfosse and N. H. Nickerson, Almost-linear time decoding algorithm for topological codes, Quantum **5**, 595 (2021).
[13] O. Higgott and C. Gidney, Sparse blossom: correcting a million errors per core second with minimum-weight matching, Quantum **9**, 1600 (2025).
[14] K.-Y. Kuo and C.-Y. Lai, Refined belief propagation decoding of sparse-graph quantum codes, IEEE Journal on Selected Areas in Information Theory **1**, 487 (2020).
[15] Y.-H. Liu and D. Poulin, Neural belief-propagation decoders for quantum error-correcting codes, Physical review letters **122**, 200501 (2019).
[16] A. deMarti iOlius, P. Fuentes, R. Orús, P. M. Crespo, and J. E. Martinez, Decoding algorithms for surface codes, Quantum **8**, 1498 (2024).
[17] M. Bayati, D. Shah, and M. Sharma, Maximum weight matching via max-product belief propagation, in *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.* (IEEE, 2005) pp. 1763–1767.
[18] S. Sanghavi, D. Malioutov, and A. Willsky, Linear programming analysis of loopy belief propagation for weighted matching, Advances in neural information processing systems **20** (2007).
[19] M. Bayati, D. Shah, and M. Sharma, Max-product for maximum weight matching: Convergence, correctness, and lp duality, IEEE Transactions on information theory **54**, 1241 (2008).
[20] A. E. Gelfand, J. Shin, and M. Chertkov, Belief propagation for linear programming, in *2013 IEEE International Symposium on Information Theory* (IEEE, 2013) pp. 2249–2253.
[21] S.-S. Ahn, S. Park, M. Chertkov, and J. Shin, Minimum weight perfect matching via blossom belief propagation, Advances in neural information processing systems **28** (2015).
[22] J. Edmonds, Paths, trees, and flowers, Canadian Journal of mathematics **17**, 449 (1965).
[23] M. Mezard and A. Montanari, *Information, physics, and computation* (Oxford University Press, 2009).
[24] D. S. Wang, A. G. Fowler, A. M. Stephens, and L. C. Hollenberg, Threshold error rates for the toric and surface codes, arXiv preprint arXiv:0905.0531 (2009).
[25] H. Yao, W. A. Laban, C. Häger, A. G. i Amat, and H. D. Pfister, Belief propagation decoding of quantum ldpc codes with guided decimation, in *2024 IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2024) pp. 2478–2483.
[26] D. Bascones and B. Morcillo, Hourglass sorting: A novel parallel sorting algorithm and its implementation, arXiv preprint arXiv:2507.16326 (2025).
[27] O. Higgott, T. C. Bohdanowicz, A. Kubica, S. T. Flammia, and E. T. Campbell, Improved decoding of circuit noise and fragile boundaries of tailored surface codes, Physical Review X **13**, 031007 (2023).
[28] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik **1**, 269 (1959).
[29] C. Gidney, Stim: a fast stabilizer circuit simulator, Quantum **5**, 497 (2021).
[30] D. K. Tuckett, *Tailoring surface codes: Improvements in quantum error correction with biased noise*, Ph.D. thesis, University of Sydney (2020), (qecsim: `https://github.com/qecsim/qecsim`).
[31] K.-Y. Kuo and C.-Y. Lai, Exploiting degeneracy in belief propagation decoding of quantum codes, npj Quantum Information **8**, 111 (2022).
[32] T. Müller, T. Alexander, M. E. Beverland, M. Bühler, B. R. Johnson, T. Maurer, and D. Vandeth, Improved belief propagation is sufficient for real-time decoding of quantum memory, arXiv preprint arXiv:2506.01779 (2025).

V. APPENDIX

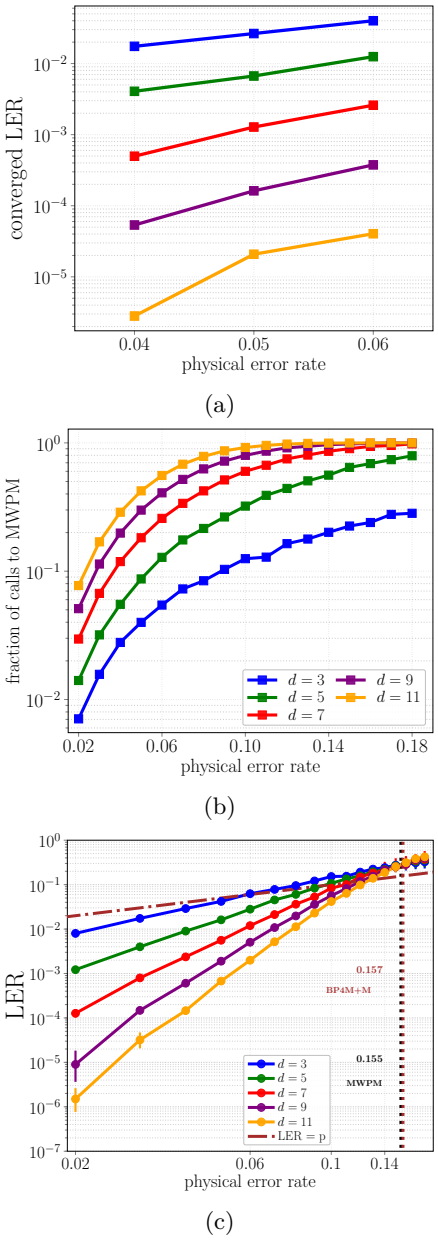


FIG. 8: Code capacity convergence performance and MWPM postprocessing: (a) Converged LER. (b) Convergence fraction. (c) Performance of BP4M+M with 25 iterations.

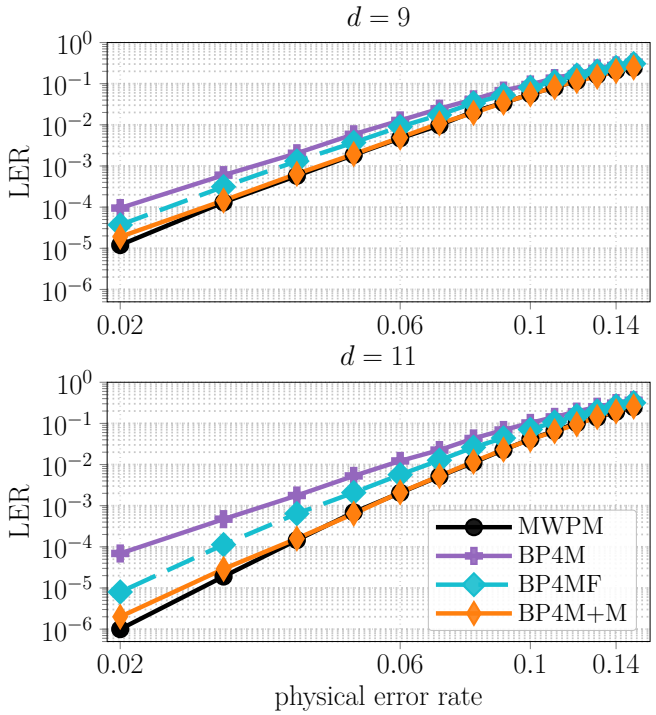


FIG. 9: Code capacity performance of our algorithms (BP4M, BP4MF, BP4M+M) with 25 iterations compared to MWPM for distances $d = 9, 11$.

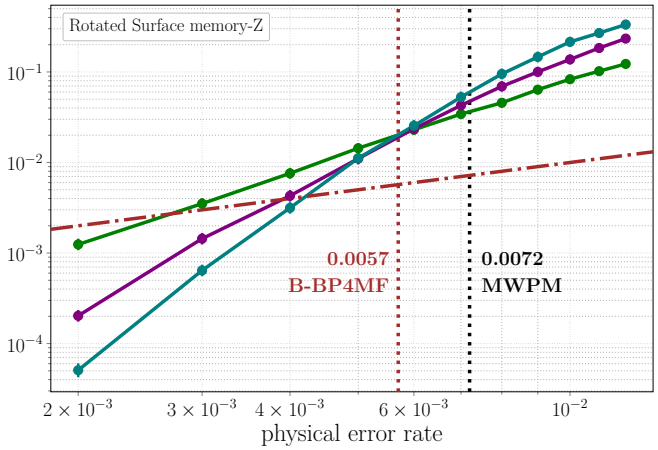


FIG. 10: B-BP4MF threshold result for circuit-level noise rotated memory-Z surface code with 75+75 iterations.