

## BlockBP decoder for the surface code

Aviad Kaufmann <sup>\*</sup>*Physics Department, Technion—Israel Institute of Technology, Haifa 3200003, Israel*Itai Arad *Centre for Quantum Technologies, National University of Singapore, Singapore 117543, Singapore*

(Received 12 March 2025; accepted 27 August 2025; published 7 October 2025)

We present a new decoder for the surface code, which combines the accuracy of the tensor-network decoders with the efficiency and parallelism of the belief-propagation algorithm. Our main idea is to replace the expensive tensor-network contraction step in the tensor-network decoders with the blockBP algorithm—a recent approximate contraction algorithm, based on belief propagation. Our decoder is therefore a belief-propagation decoder that works in the degenerate maximal likelihood decoding framework. Unlike conventional tensor-network decoders, our algorithm can run efficiently in parallel, and may therefore be suitable for real-time decoding. We numerically test our decoder and show that for a large range of lattice sizes and noise levels it delivers a logical error probability that outperforms the Minimal Weight Perfect Matching decoder, sometimes by more than an order of magnitude.

DOI: [10.1103/z2qt-smfb](https://doi.org/10.1103/PhysRevResearch.7.043025)

## I. INTRODUCTION

The quantum surface code [1] is one of our best candidate codes for building a fault-tolerant quantum computer [2], owing to its high degree of locality, high threshold, and large code distance. It is a well-studied code, and simple demonstrations of it have already been performed [3–5]. As the current and near-future quantum hardware will probably be characterized by a relatively low number of qubits (few hundreds to few thousands), with a not-so-low noise levels, it is important to optimize the use of this code as much as possible. This is a highly nontrivial task. It requires deep understanding and integration of the many components that take part in error correction process, software and hardware alike.

An important ingredient in the fault-tolerance framework is the decoder. This is a classical algorithm that takes as input the results of the syndrome measurements and outputs a description of a corrective action in the form of a Pauli string. Finding the optimal corrective action is a hard task [6], as one needs to look over an exponentially large space of possible solutions. In addition, the decoder has to be highly efficient, as it has to run in timescales that are short enough for the system not to decohere too much—scales on the order of 1  $\mu$ s for superconducting qubits, for example.

Over the years, a plethora of surface code decoders have been developed [7], using various approaches. In general,

these decoders follow the natural trend in which the decoder accuracy anticorrelates with its running time. At the extreme side of this spectrum lie tensor-network (TN)-based decoders [8–12]. They have a nearly optimal decoding accuracy that comes at a very high computational cost, which makes them unpractical for real-time decoding.

In this paper, we initiate the study of a new TN-based decoder, which on one hand enjoys high accuracy, but on the other hand, being a parallel algorithm, it can potentially be used for real-time decoding. Our idea is to use the recent blockBP algorithm [13] as the contraction engine for the tensor-network decoder. blockBP is an algorithm for approximate contraction of 2D tensor networks, which is based on the belief-propagation (BP) algorithm [14], in conjunction with matrix product state (MPS) techniques. As a BP algorithm, it can be easily parallelized, making it suitable for real-time decoding. Indeed, BP algorithms are at the backbone of some of the most powerful classical code decoders such as LDPC codes [15] and Turbo codes [16]. They have also been extensively studied in the context of quantum decoding [17], in particular in conjunction with other decoders [18–21]. Our use of the BP algorithm is different in that we use it for the task of tensor-network contraction, which results in a BP algorithm for degenerate quantum maximal likelihood decoding.

To demonstrate our method, we test it in the perfect measurement setup (i.e., the code capacity model) using surface codes of various sizes with a single-qubit depolarizing noise of different levels. We compare its performance to the original TN [8] and the Minimal Weight Perfect Matching (MWPM) [22–25] decoders. We believe that our techniques can be generalized to more realistic noise models, including circuit-level noise, as well as to other topological two-dimensional codes such as the color code [26] or the rotated surface code [27].

<sup>\*</sup>Contact author: [aviadskaufmann@gmail.com](mailto:aviadskaufmann@gmail.com)

The organization of the paper is as follows. In Sec. II, we offer a short introduction to stabilizer codes and their decoding problem, and describe the surface code. In Sec. III, we describe our blockBP decoder. We include also a short description of the original TN-based decoder, as well as the BP algorithm for TN contraction. In Sec. IV, we present the results of our numerical simulations, and then offer our conclusions in Sec. V.

## II. THEORETICAL BACKGROUND

### A. Maximal likelihood and degenerate maximal likelihood decoding in stabilizer codes

To present the formalism of stabilizer codes [28] and the decoding problem, we largely follow the notation of Ref. [8].

Let  $\mathcal{P}_n$  denote the Pauli group over  $n$  qubits. Every member of this group is an operator  $f = cf_1 \otimes f_2 \otimes \cdots \otimes f_n$ , where  $f_i \in \{I, X, Y, Z\}$  are single-qubit Pauli operators and  $c \in \{\pm 1, \pm i\}$ . A stabilizer code is defined by a *stabilizer subgroup*  $\mathcal{G} \triangleleft \mathcal{P}_n$  that is (1) abelian and (2) does not contain the  $-I$  element. The condition  $-I \notin \mathcal{G}$  implies that the  $c$  prefactors in the expansion of every  $g \in \mathcal{G}$  must be  $\pm 1$ , and therefore all  $g \in \mathcal{G}$  have  $\pm 1$  eigenvalues.

In what follows, it will be beneficial to discuss  $\mathcal{G}$  in terms of its generators, which are also referred to as “check operators” in the context of stabilizer codes. The generators are a *minimal* set of elements  $g_1, g_2, \dots, g_m \in \mathcal{G}$  such that  $\langle g_1, \dots, g_m \rangle = \mathcal{G}$ . As the different  $g_i$  commute among themselves, they share a common eigenbasis. The stabilizer subspace  $\mathcal{H}_0$  is defined by  $\mathcal{H}_0 \stackrel{\text{def}}{=} \{|\psi\rangle : g_i|\psi\rangle = |\psi\rangle, \forall i = 1, \dots, m\}$ , i.e., it is the span of eigenvectors of  $g_i$  with common eigenvalue 1. It can be shown that  $\dim \mathcal{H}_0 = 2^k$ , where  $k \stackrel{\text{def}}{=} n - m$  is the number of logical qubits that can be encoded in  $\mathcal{H}_0$  [29].

In the quantum error correction protocol, we start with an encoded state  $\rho \in \mathcal{H}_0$  that experiences a noise described by a noise channel  $\mathcal{N}$ . In this work, we assume  $\mathcal{N}$  is a stochastic Pauli channel

$$\mathcal{N}(\rho) \stackrel{\text{def}}{=} \sum_{f \in \mathcal{P}_n} \pi(f) \cdot f \rho f^\dagger \quad (1)$$

with known probabilities  $\pi(f)$ . Our task is to revert the effect of the noise and bring the system back to the encoded state  $\rho$ .

We start by simplifying Eq. (1) using an important observation. As  $\rho \in \mathcal{H}_0$ , it is easy to see that  $f_1 \rho f_1^{-1} = f_2 \rho f_2^{-1}$  for any  $f_1, f_2$  for which  $f_1^{-1} f_2 \in \mathcal{G}$ . In other words, there are many different errors/operators that have the *same* action over the code space. This property is known as *degeneracy*, and it is a genuine property of quantum codes that does not exist in classical codes. Mathematically, we can rephrase it by saying that  $f_1, f_2$  have the same action on the code space iff they belong to the same *coset* of  $\mathcal{G}$  in  $\mathcal{P}_n$ , i.e.,  $f_1 \mathcal{G} = f_2 \mathcal{G}$ . Therefore, letting  $\{f_\alpha\}$  denote the different coset representatives, Eq. (1) becomes

$$\mathcal{N}(\rho) = \sum_{\alpha} \pi(f_\alpha \mathcal{G}) \cdot f_\alpha \rho f_\alpha^\dagger. \quad (2)$$

It can be interpreted as saying that the noise channel takes  $\rho$  to one of the “coset states”  $f_\alpha \rho f_\alpha^\dagger$  with probability  $\pi(f_\alpha \mathcal{G}) \stackrel{\text{def}}{=} \sum_{g \in \mathcal{G}} \pi(f_\alpha g)$ . If we could perform a measurement that would

tell us at which coset the state is, we could simply apply  $f_\alpha^\dagger$  (or any other state in that coset) to take us back to  $\rho$ . Unfortunately, this is impossible, as the  $f_\alpha \rho f_\alpha^\dagger$  states are not necessarily orthogonal to each other. Instead, we perform a *syndrome measurement*, in which we measure the eigenspaces of the  $m$  generators  $g_i$ , with each measurement giving us either  $+1$  or  $-1$ . Since all the generators commute with each other, they define a decomposition of  $\mathcal{H}$  into a direct sum  $\mathcal{H} = \bigoplus_{\underline{s}} \mathcal{H}_{\underline{s}}$  of common eigenspaces. The indices  $\underline{s}$  are binary strings  $\underline{s} = (s_1, \dots, s_m)$ , called *syndromes*, which define the eigenvalues of the generators on  $\mathcal{H}_{\underline{s}}$ :  $|\psi\rangle \in \mathcal{H}_{\underline{s}}$  iff  $g_i|\psi\rangle = (-1)^{s_i}|\psi\rangle$ . In that notation, the stabilizer subspace  $\mathcal{H}_0$  corresponds to the syndrome  $\underline{s} = (0, \dots, 0)$ .

After measuring the stabilizers, we obtain the syndrome  $\underline{s} = (s_1, \dots, s_m)$  of the subspace  $\mathcal{H}_{\underline{s}}$  into which the system collapsed. As the Pauli operators either commute or anticommute,  $f \mathcal{H}_0 f^\dagger = \mathcal{H}_{\underline{s}}$  iff  $f_\alpha g_i = (-1)^{s_i} g_i f_\alpha$  for  $i = 1, \dots, m$ . Therefore, every  $f \in \mathcal{P}_n$  maps  $\mathcal{H}_0$  to a well-defined  $\mathcal{H}_{\underline{s}}$ . This defines yet another partition of  $\mathcal{P}_n$  into  $2^m$  sets of Pauli operators that take  $\mathcal{H}_0$  to the same  $\mathcal{H}_{\underline{s}}$ . Mathematically, it is the partition of  $\mathcal{P}_n$  into the cosets of the *centralizer* group of  $\mathcal{G}$ , which is defined by elements in  $\mathcal{P}_n$  that commute with all elements of  $\mathcal{G}$ :

$$C(\mathcal{G}) \stackrel{\text{def}}{=} \{f \in \mathcal{P}_n : fg = gf \forall g \in \mathcal{G}\}. \quad (3)$$

Indeed, given two Pauli operators  $f_a, f_b$ , then  $f_a g_i = (-1)^{s_i} g_i f_a$  and  $f_b g_i = (-1)^{s_i} g_i f_b$  for all  $i$  iff  $f_a f_b^{-1}$  commutes with all  $g_i$ , which happens iff  $f_a f_b^{-1} \in C(\mathcal{G})$ . To summarize, there are  $2^m$  cosets of  $C(\mathcal{G})$  in  $\mathcal{P}_n$ , which correspond to the sets of Pauli operators that take  $\mathcal{H}_0$  to any of  $2^m$  subspaces  $\mathcal{H}_{\underline{s}}$ . As  $\mathcal{G} \triangleleft C(\mathcal{G})$ , every coset of  $C(\mathcal{G})$  can be broken into  $|C(\mathcal{G})/\mathcal{G}| = 2^{2k}$  cosets of  $\mathcal{G}$ .

Let  $L_0, L_1, L_2, \dots$  denote the representatives of the cosets of  $\mathcal{G}$  in  $C(\mathcal{G})$  so that

$$C(\mathcal{G}) = L_0 \mathcal{G} \cup L_1 \mathcal{G} \cup \dots$$

These operators correspond to logical gates on the encoded space. We use them to write every coset of  $C(\mathcal{G})$  that is associated with a syndrome  $\underline{s}$  as a disjoint union of  $\mathcal{G}$  cosets:

$$f_{\underline{s}} C(\mathcal{G}) = f_{\underline{s}} L_1 \mathcal{G} \cup f_{\underline{s}} L_2 \mathcal{G} \cup \dots$$

The postmeasurement state after measuring a syndrome  $\underline{s}$  is therefore

$$\rho_{\underline{s}} \propto \sum_{\beta} \pi(f_{\underline{s}} L_{\beta} \mathcal{G}) \cdot f_{\underline{s}} L_{\beta} \rho L_{\beta}^\dagger f_{\underline{s}}^\dagger. \quad (4)$$

We can now summarize the stabilizer code decoding problem as follows:

**Problem 1 (decoding a stabilizer code).** Given the result of a syndrome measurement  $\underline{s} = (s_1, \dots, s_m)$ , find a corrective Pauli operator  $f$  that will take  $\rho_{\underline{s}}$  back to  $\rho$  with high probability.

From the above discussion it is clear that the best guess for a corrective action given a syndrome  $\underline{s}$  would be to find the coset  $f_{\underline{s}} L_{\beta} \mathcal{G}$  with the maximal probability  $\pi(f_{\underline{s}} L_{\beta} \mathcal{G})$  and then take any  $f \in f_{\underline{s}} L_{\beta} \mathcal{G}$ . This type of decoding is called Degenerate Quantum Maximal Likelihood Decoding (DQMLD). It is the approach taken by tensor-network-based decoders [8–12] (see Sec. III A for more details), as well as (to a lesser extent)

by RG-based decoders [18,19]. As was shown in Ref. [6], it is a #P-complete problem, and therefore in the worst case we can only try to solve it approximately.

The reason why DQMLD is harder than classical decoding is that in order to calculate the coset probability  $\pi(f_{\underline{s}} L_{\beta} \mathcal{G})$ , we need to sum over an exponential number of probabilities of the Pauli operators in it. A simpler, yet less optimal approach would be to ignore the degeneracy of the code and look for the noise operator  $f \in \mathcal{P}_n$  that maximizes  $\pi(f)$ . This type of decoding is called Quantum Maximal Likelihood Decoding (QMLD), and it is the approach taken by most “practical” decoders such as the MWPM decoder [22–25], the Union Find decoder [30], and variants of BP-based decoders [17,20,21,31,32]. While it is an easier problem than DQMLD, it is still a hard problem, which has been shown to be NP-complete [33–35], as its classical counterpart [36].

In many practical scenarios, in particular when the noise level is very low, QMLD would yield very good results, approaching the performance of DQMLD decoding [6]. Nevertheless, empirical tests show that often also in low-error regime, DQMLD encoders such as the TN-based decoders provide logical error rates that can be orders of magnitude lower than QMLD-based decoders such as MWPM (see, e.g., the results in Sec. IV).

### B. The surface code

The surface code [1] is a stabilizer code that can be defined on any two-dimensional lattice that is described by a graph  $G = (V, E)$ . For simplicity, we shall limit our discussion to an open square lattice of size  $d \times d$ , but much of the discussion below can be generalized to other 2D lattices. Following the convention of Ref. [8], our lattice is made of  $d^2$  horizontal edges and  $(d - 1)^2$  vertical edges, as shown in Fig. 1(a) for the case of  $d = 3$ . The qubits sit on the edges so that in total there are  $n = d^2 + (d - 1)^2$  qubits. There are two types of check operators:  $Z$  check operators, denoted by  $A_u$  and  $X$  check operators that are denoted by  $B_p$  [see Fig. 1(b)]. The  $A_u$  check operators are defined for every site  $u \in V$  as the product of Pauli  $Z$  on its adjacent edges.  $B_p$  is defined for every plaquette  $p$  as the product of Pauli  $X$  operators on its boundary:

$$A_u \stackrel{\text{def}}{=} \prod_{e \in e} Z_e, \quad B_p \stackrel{\text{def}}{=} \prod_{e \in p} X_e. \quad (5)$$

There are  $d(d - 1)$  check operators of each type. The  $A_u$  check operators trivially commute within themselves, and so do the  $B_p$  operators. It is also easy to see that the two types of operators commute with each other as every pair either intersects on an even number of edges or does not intersect at all.

With  $m = 2d(d - 1)$  independent check operators defined above over  $n = d^2 + (d - 1)^2$  qubits, we have  $k = n - m = 1$ , which corresponds code space  $\mathcal{H}_0$  that encodes a single logical qubit. The logical operators  $\bar{L}_i$  in this case correspond to logical Paulis on the encoded qubit and are denoted by  $\{\bar{I}, \bar{X}, \bar{Y}, \bar{Z}\}$ . The logical  $\bar{X}$  operator can be written as a product of Pauli  $X$  operators over a row of horizontal edges, and similarly, a logical  $\bar{Z}$  can be written as a product of Pauli  $Z$  operators along a column of horizontal edges.

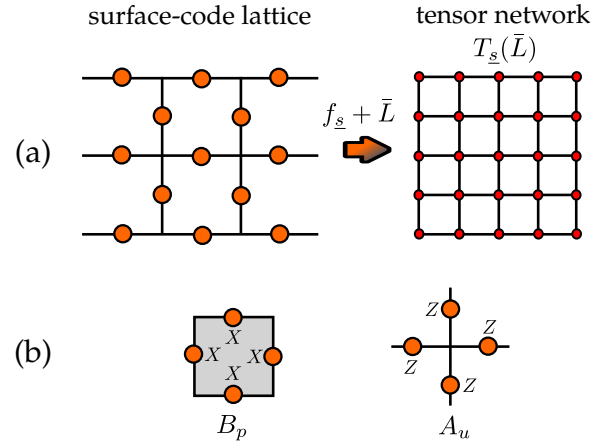


FIG. 1. (a) The lattice structure of a  $d = 3$  surface code. The qubits sit on the edges of the square lattice with  $d^2$  horizontal edges and  $(d - 1)^2$  vertical edges. Given an Pauli operator  $f_{\underline{s}}$  that corresponds to the syndrome  $\underline{s}$  and a logical operator  $\bar{L} \in \{\bar{I}, \bar{X}, \bar{Y}, \bar{Z}\}$ , the lattice is mapped to a  $(2d - 1) \times (2d - 1)$  square tensor network  $T_{\underline{s}}(\bar{L})$ , whose contraction value gives the coset probability  $\pi(f_{\underline{s}} \bar{L} \mathcal{G})$  (see Sec. III A). (b) The check operators (stabilizers) of the surface code: Plaquette operators  $B_p$  are the product of Pauli  $X$  operators sitting on the edges surrounding a plaquette  $p$ . Site operators  $A_u$  are the product of Pauli  $Z$  operators on the qubits that sit on the edges adjacent to a site  $u$ .

### III. THE blockBP DECODER FOR THE SURFACE CODE

The blockBP decoder is based on the tensor-network decoder [8], which we will first review briefly here. Throughout this section we assume basic knowledge of tensor networks and matrix product states in particular. Good reviews of these subjects can be found, for example, in Refs. [37,38].

#### A. The tensor-network-based DQMLD decoder

In Ref. [8], it was shown that the DQMLD problem for the  $d \times d$  surface code can be reduced to the problem of contracting a  $(2d - 1) \times (2d - 1)$  square tensor network with a bond dimension of  $D = 2$ . Specifically, given a syndrome measurement result  $\underline{s}$ , one can efficiently construct four such square tensor networks, which we denote by  $T_{\underline{s}}(\bar{I})$ ,  $T_{\underline{s}}(\bar{X})$ ,  $T_{\underline{s}}(\bar{Y})$ , and  $T_{\underline{s}}(\bar{Z})$ . The contraction of these networks is equal to the coset probabilities  $\pi(f_{\underline{s}} \mathcal{G})$ ,  $\pi(f_{\underline{s}} \bar{X} \mathcal{G})$ ,  $\pi(f_{\underline{s}} \bar{Y} \mathcal{G})$ , and  $\pi(f_{\underline{s}} \bar{Z} \mathcal{G})$ , from which we can pick the coset with the highest probability. This reduces the DQMLD problem to the problem of contracting a 2D TN. Unfortunately, contracting a 2D TN is a #P-hard problem [39], and therefore we can generally only solve it approximately.

In Ref. [8], the authors used the boundary MPS method (bMPS) [40,41] to approximate the TN contraction. In this method, the 2D TN is contracted column by column from left to right using the MPS machinery [38]. At step  $i$ , the contraction of the TN up to column  $i$  is approximated by an MPS with a finite bond dimension  $\chi$ . Next, viewing the  $i + 1$  column as a matrix product operator, it is contracted with the MPS, creating a new MPS with a bond dimension of  $2\chi$ . To keep the bond dimension finite, the new MPS is compressed by truncating back the bond dimension to  $\chi$  using well-established MPS methods [38].

Being a DQMLD decoder, the bMPS decoder gives the best decoding results of all efficient (i.e., with polynomial running time) decoding algorithms and has the highest thresholds for a large range of parameters [7]. However, for a fixed truncation bond  $\chi$ , the computational cost of the algorithm is  $O(n\chi^3)$ . For the decoder to succeed, the error in the TN approximation should be smaller than the typical difference between the coset probabilities. This forces  $\chi$  to increase as  $d$  increases, in particular when we get closer to the noise threshold. This, together with a large overhead in the  $O(\cdot)$  notation, makes the bMPS a very slow decoder that is challenging to implement in real-time scenarios. Indeed, in Ref. [5], as well as in the numerical experiments we have conducted in this paper, the bMPS decoder was many orders of magnitude slower than the MWPM decoder.

### B. BP decoders for quantum codes

Our main idea is to replace the slow bMPS contraction with the blockBP algorithm [13]. It is a BP [14,42] algorithm for an approximate contraction of 2D tensor networks. BP is a message passing meta-algorithm, which was originally designed for approximating the marginals of multivariate probability distributions defined over complex networks. It is exact on trees and often gives surprisingly good results also for loopy networks. Nowadays, it is used in diverse areas such as Bayesian inference [14], statistical physics [43–45], and combinatorial optimizations [42,46], to name a few. It is also the backbone of powerful decoding algorithms for classical LDPC and Turbo codes [15,16], where it often referred to as the *sum-product algorithm*.

We note that BP is also used in quantum decoders. As in the classical LDPC case, BP can be used to solve the QMLD problem by trying to find the most probable error. However, applying this approach to the surface code (and other topological codes) results in very poor performance. As argued by Ref. [17], the failure can be attributed to the presence of degeneracy and many small loops in the network over which BP runs. However, we note that more recently there have been several successful approaches of using BP as a quantum decoder by combining it with other decoders [18–21] or with some postprocessing [31,32].

### C. Using BP for tensor-network contraction

We use BP as an approximate TN contraction method to estimate the different probabilities in the DQMLD framework. The idea of using BP for approximate TN contraction was first introduced in Ref. [47] and subsequently used in various places for achieving a fast TN contraction [48,49], or for finding a suitable local TN gauge [50,51]. In Ref. [13], this approach was further strengthened for planar TN using the MPS machinery together with coarse graining in an algorithm called blockBP. We briefly sketch this algorithm here. For more details, the reader is referred to Refs. [13,42,47].

We first describe the “vanilla BP” algorithm of Ref. [47] and then explain how it can be improved by using the block structure of Ref. [13]. Consider then a TN  $\mathcal{T}$  that is described by a graph  $G = (V, E)$  with a tensor  $T_v$  on every vertex  $v \in V$ . The incident edges to vertex  $v$  correspond to the different

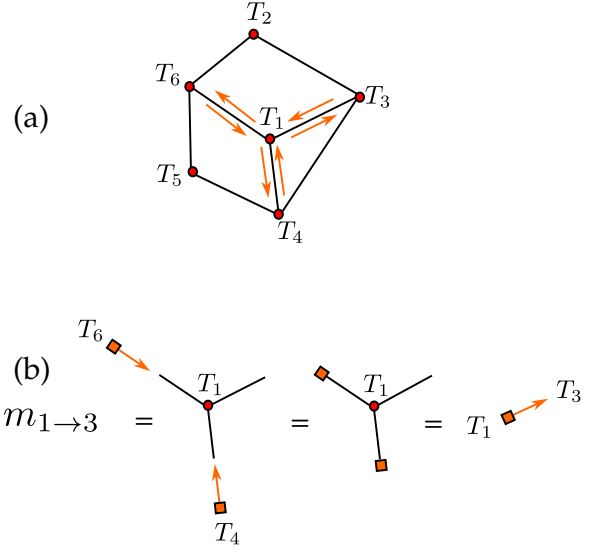


FIG. 2. Sketch of the BP algorithm for tensor networks. (a) A tensor network with five tensors. Each neighboring tensors exchange messages between them. (b) The outgoing message from  $T_1$  to  $T_3$  is obtained by contracting  $T_1$  with all the incoming messages to  $T_1$  from the previous round except for the incoming message from  $T_3$ .

indices of  $T_v$ . An edge connecting  $u$  to  $v$  indicates a contraction of the tensors  $T_u, T_v$  along the two indices of that edge [see Fig. 2(a)]. Our goal is to approximate the contraction of  $\mathcal{T}$ , which we denote by  $P = \text{Tr}(\mathcal{T})$ . As explained in Ref. [47] (see also Ref. [52]), this contraction can be approximated using a simple BP algorithm. The algorithm is sketched in Fig. 2. It contains several rounds of message passing between neighboring nodes on the graph. For every edge  $e = (u, v)$  on the graph, there will be two messages in the opposite directions:  $m_{u \rightarrow v}(x_e)$  and  $m_{v \rightarrow u}(x_e)$ , where  $x_e$  is a discrete variable that runs over the index of the common leg  $e$  of the tensors  $T_u, T_v$ . As such,  $m_{v \rightarrow u}(x_e)$  can be viewed as a vector, or as a tensor with a single leg. At each round, a node  $v$  uses the incoming messages of the previous round (sent to it by its neighboring nodes) to calculate an outgoing message for each one of its neighbors. Formally, let  $N_v$  denote all the neighboring nodes of  $v$  on the graph, and let  $m_{u \rightarrow v}^{(t)}$  denote the message from node  $u$  to node  $v$  at round  $t$ . Then the message update rule, which is illustrated in Fig. 2(b), is given by

$$m_{v \rightarrow u}^{(t+1)} = \text{Tr} \left( T_v \prod_{u' \in N_v \setminus \{u\}} m_{u' \rightarrow v}^{(t)} \right), \quad (6)$$

where  $\text{Tr}(\cdot)$  stands for contraction over internal edges in the TN.

Once the messages have converged to a fixed point, or once the number of rounds exceeds some prescribed limit, the resulting messages are used to approximate  $P$ . One uses the converged messages to calculate the *Bethe free-energy* [42,47] functional  $\mathcal{F}_{\text{Bethe}}(\mathcal{T})$  of the tensor network from the converged messages and then approximates  $\text{Tr}(\mathcal{T}) \simeq e^{-\mathcal{F}_{\text{Bethe}}(\mathcal{T})}$ . However, a direct use of the Bethe free-energy formula will be highly inefficient, in particular in the blockBP framework that we describe later. Instead, the following lemma provides an

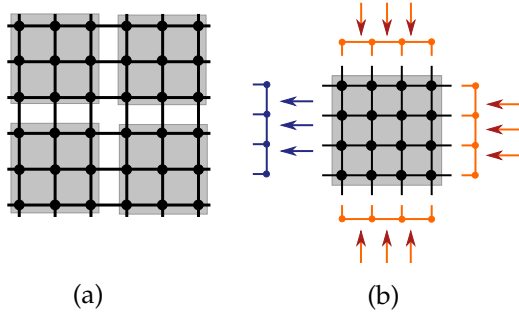


FIG. 3. Sketch of the blockBP algorithm for 2D tensor networks. (a) The 2D TN is partitioned into disjoint blocks of tensors. (b) The update rule for an outgoing message is the same as in regular BP, only that now the incoming messages are MPSs and the contraction is done approximately using boundary MPS method, which results in an outgoing MPS message.

equivalent formula, which can be easily used in TN framework:

*Lemma 1.* Let  $\mathcal{T}$  be a TN defined on a graph  $G = (V, E)$ , and let  $m_{u \rightarrow v}(x_e)$  be the set of converged BP messages. Let  $\mathcal{F}_{\text{Bethe}}(\mathcal{T})$  the Bethe free-energy of the BP fixed point. Then,

$$e^{-\mathcal{F}_{\text{Bethe}}(\mathcal{T})} = \prod_v \text{Tr} \left( T_v \prod_{u \in N_v} \hat{m}_{u \rightarrow v} \right), \quad (7)$$

where  $\{\hat{m}_{u \rightarrow v}\}$  are a rescaling of the fixed-point messages, normalized such that

$$\text{Tr}(\hat{m}_{u \rightarrow v} \cdot \hat{m}_{v \rightarrow u}) = 1. \quad (8)$$

The proof of Lemma 1 is given in Appendix A. We note that the right-hand side of formula (7) is the product of the contraction of small TNs, which in principle can be done efficiently in parallel.

#### D. The blockBP algorithm

The Bethe free-energy approximation of the BP algorithm to the contraction of the TN is exact when the underlying graph is a tree [42]. However, in the presence loops, its quality might deteriorate, in particular when there are small loops and strong correlations. Over the years, there have been many attempts to improve this approximation by modifying/generalizing the BP algorithm so that it would become more accurate on loopy networks [53–63]. Usually, this is done at the expense of higher computational costs.

blockBP is one such algorithm, designed to be run on planar tensor networks. It relies on a simple idea, which is demonstrated in Fig. 3. We start by partitioning the TN into disjoint blocks [Fig. 3(a)] and think of the blocks as the nodes of a new, “coarse-grained” TN. We then run the regular BP on it by passing messages between neighboring blocks. However, in order to avoid dealing with exponentially large messages and block tensors, we represent a message  $m_{\bar{v} \rightarrow \bar{u}}$  between blocks  $\bar{v}$  and  $\bar{u}$  as an MPS message of some maximal bond dimension  $\chi$ . Similarly, we do *not* fuse the tensors inside each block and change the update rule in Eq. (6) to a contraction of a small TN (made of the block together with the incoming MPSs). This contraction is done with the boundary

MPS method with maximal truncation bond  $\chi$ . Finally, to estimate the Bethe free-energy approximation to the global TN contraction, we use Lemma 1, which remains efficient also when using blocks and MPS messages.

The advantage of blockBP over the regular BP is that in blockBP the short range loops within each block are treated more accurately (at least up to truncation errors within the block contraction). The larger we take the blocks, the more accurate the approximation becomes at the expense of having to contract larger and larger blocks. At the limit where the system becomes a single large block, we recover the bMPS algorithm. As we shall see, for the purpose of decoding the surface code, small blocks of size  $6 \times 6$  give excellent results for  $d \lesssim 25$ . Crucially, the blockBP algorithm, just like the BP algorithm, can be run in parallel, and this is what makes it suitable for real-time decoding.

#### E. Description of the blockBP decoder

We now describe the blockBP decoder for the surface code.

Decoder parameters:

- (1) Code size  $d$
- (2) Local Pauli noise model
- (3) BP threshold parameters:  $\delta_0 < \delta_1$  and  $\text{max-iter}$ .
- (4) Local block size  $k$
- (5) Maximal MPS bond dimension  $\chi$

---

##### ALGORITHM 1: blockBP decoder.

---

```

input : A syndrome string  $\underline{s}$  of size
         $m = 2d(d-1)$ 
output: A guess for an operator  $f_{\underline{s}} \bar{L}$  (where  $\bar{L}$  in
         $\{\bar{I}, \bar{X}, \bar{Y}, \bar{Z}\}$ ) with the maximal coset
        probability  $\text{Tr}(f_{\underline{s}} \bar{L} \mathcal{G})$ 

 $f_{\underline{s}} \leftarrow$  a Pauli string that produces the syndrome  $\underline{s}$ ;
for  $\bar{L} \in \{\bar{I}, \bar{X}, \bar{Y}, \bar{Z}\}$  do
     $T_{\underline{s}}(\bar{L}) \leftarrow$  Using the noise model and  $f_{\underline{s}}$  as in
    Ref. [8];
    Partition  $T_{\underline{s}}(\bar{L})$  into  $k \times k$  blocks;
    Initialize MPS messages;
    for  $t \leftarrow 1$  to  $\text{max-iter}$  do // BP loop
        calculate the outgoing MPS message
        according to Eq. (6) using bMPS with  $\chi$ ;
        Normalize outgoing messages;
        if  $\Delta < \delta_0$  then
            exit the BP loop;
        end
    end
    if  $\Delta < \delta_1$  then
        calculate the approximated  $\text{Tr}(T_{\underline{s}}(\bar{L}))$  via
        Lemma III.1;
    end
end
if  $\exists \bar{L}$  with  $\Delta < \delta_1$  then
    return  $f_{\underline{s}} \bar{L}$  with  $\text{Tr}(T_{\underline{s}}(\bar{L}))_{\text{max}}$ ;
else
    return  $f_{\underline{s}} \bar{L}$  with  $\Delta_{\text{min}}$ ;
end

```

---

Remarks:

(1) If  $k$  does not divide  $2d - 1$  cleanly, the blocks at the edges of the TN can be of smaller sizes.

(2) We initialize all block MPS messages to represent a uniform probability distribution.

(3) We normalize each outgoing message by making it left-canonical with unit  $L_2$  norm.

(4) We define the average distance between the new messages and the previous messages: If  $M$  is the total number of messages, the average distance is given by

$$\Delta \stackrel{\text{def}}{=} \frac{1}{M} \left( \sum_{i=1}^M \| |m_i^{(t)}\rangle \langle m_i^{(t)}| - |m_i^{(t-1)}\rangle \langle m_i^{(t-1)}| \|^2 \right)^{1/2}. \quad (9)$$

(5) The running time complexity of a serial BP algorithm over  $N$  nodes with `max-iter` iterations and  $T$  steps to calculate the outgoing messages of each node is  $O(N \cdot \text{max-iter} \cdot T)$ , and about  $O(\text{max-iter} \cdot T)$  if we run all nodes in parallel. In our case, for a given code distance  $d$ , a block size  $k$ , and maximal bond dimension  $\chi$ , we have  $N = O(d^2/k^2)$  and  $T = O(k^2\chi^3)$ . This gives us a serial running time of  $O(n\chi^3 \cdot \text{max-iter})$ —which is identical to or higher than the regular TN decoder. On the other hand, neglecting messages overhead, the parallel running time of the decoder should be of the order  $O(\text{max-iter} \cdot k^2\chi^3 + n)$ —which is potentially much faster. The additive  $n$  term in the running time stems from unavoidable pre- and postprocessing. We note that the running time can be further reduced by a factor of 4 if we calculate the outgoing messages of each block in parallel.

Finally, we remark that for near-future devices, these asymptotic estimates seem less important; the running time would probably be more influenced by the overall factors hidden in the big- $O$  notation. These details are highly dependent on the particular implementation and underlying hardware. Nevertheless, for the sake of concreteness, in Appendix B, we compare the running time of our Python implementation of the blockBP decoder and our bMPS decoder (which rely on the same MPS library) for the case of  $d = 25, k = 6$  and  $d = 17, k = 4$ . We conclude that in these cases the blockBP decoder can be 7–100 times faster than the bMPS decoder.

(6) The algorithm uses two convergence error thresholds  $\delta_0 < \delta_1$ . After running the BP loop for the coset  $\bar{L}f_s\mathcal{G}$ , if the final messages have a convergence error  $\Delta < \delta_1$ , then we consider it sufficiently converged so that we can trust its Bethe free-energy approximation for  $\text{Tr}(\bar{L}f_s\mathcal{G})$ . However, if none of the cosets produced  $\Delta < \delta_1$ , we pick the one with the smallest  $\Delta$ . This is because, as we noted empirically, the right coset is most often the one with the quickest convergence.

(7) In order to facilitate the convergence, we use *damping*—a common practice in BP algorithms. Given a damping parameter  $\eta$ , if  $|m^{(t+1)}\rangle$  is the new outgoing MPS message and  $|m^{(t)}\rangle$  is the old outgoing MPS message, we replace  $|m^{(t+1)}\rangle$  by

$$|m^{(t+1)}\rangle_{\text{damp}} \stackrel{\text{def}}{=} (1 - \eta)|m^{(t+1)}\rangle + \eta|m^{(t)}\rangle. \quad (10)$$

Throughout our simulations, we used  $\eta = 0.1$ .

(8) Another important BP aspect is *scheduling*—determining the order at which the different modes will send

and receive messages. Here, we use a simple “two-layer” approach. We color the blocks in black and white as a chessboard, and at each round, either the black blocks send messages to the white blocks or the other way around. Within a given round, all updates can be done in parallel.

(9) For small  $k$ , it might be computationally advantageous to run a “naïve blockBP algorithm,” in which we simply fuse the vertices of the blocks to one mega-vertex, and run plain BP between these vertices. This way, we work with tensors of much higher bond dimension, but we save the overhead of the MPS machinery, and eliminate the MPS truncation error that is associated with finite  $\chi$ . Whether or not this is beneficial depends on implementation and the hardware. In this work, we used this approach for the  $k = 1, 2, 4$  cases. For the  $k = 6$  case, we first passed to a lattice that was the result of fusing  $3 \times 3$  vertices of the original lattice and used blockBP with  $k' = 2$  (see Sec. IV for more details).

#### IV. NUMERICAL SIMULATIONS

To test the performance of the blockBP decoder, we have performed several Monte Carlo experiments in the code capacity model, in which instant flawless stabilizer measurements are assumed. We leave the important question of how this decoder performs in the presence of noisy stabilizer measurements (i.e., in the circuit noise model) for future research. Our raw numerical data and source code can be provided upon reasonable request.

Our benchmark included lattice sizes  $d = 5, 9, 13, 17, 21, 25$  and different levels of single-qubit depolarizing noise

$$N(\rho) = (1 - \epsilon)\rho + \frac{\epsilon}{3}(X\rho X + Y\rho Y + Z\rho Z)$$

for  $\epsilon = 0.08, 0.09, \dots, 0.15$ . In all runs, we used the following parameters:

- (1) Block size:  $k = 1, 2, 4, 6$
- (2) Convergence parameters:  
`max-iter`=20  
 $\delta_0 = 10^{-4}, \delta_1 = 10^{-2}$
- (3) Damping:  $\eta = 0.1$
- (4) MPS truncation:  $\chi = 16$

As mentioned at the end of Sec. III E, to speed up the decoder we used a plain BP algorithm for the  $k = 1, 2, 4$  cases by fusing a  $k \times k$  block into a single tensor with four legs of dimension  $2^k$ . For the  $k = 6$  case, we first moved to a system in which  $3 \times 3$  blocks were contracted to a single tensor with a bond dimension of  $D' = 2^3 = 8$  and then ran blockBP with  $k' = 2$  and  $\chi = 16$ .

For reference, we have compared the performance of the blockBP decoder to that of the MWPM decoder and a bMPS-based decoder with  $\chi = 16$ . All runs were performed using the `qecsim` Python library [64]. For the blockBP simulation, we used our own Python code, which was incorporated into `qecsim` as a decoder. For every set of parameters (decoder,  $d$ ,  $k$ ,  $\epsilon$ ), which produced an empirical logical error probability  $\hat{P}_L$ , we have used  $N \geq \max(30,000, 100/\hat{P}_L)$  shots to estimate  $P_L$ , which amounts for at least 95% confidence interval of  $[0.8\hat{P}_L, 1.25\hat{P}_L]$  [65].

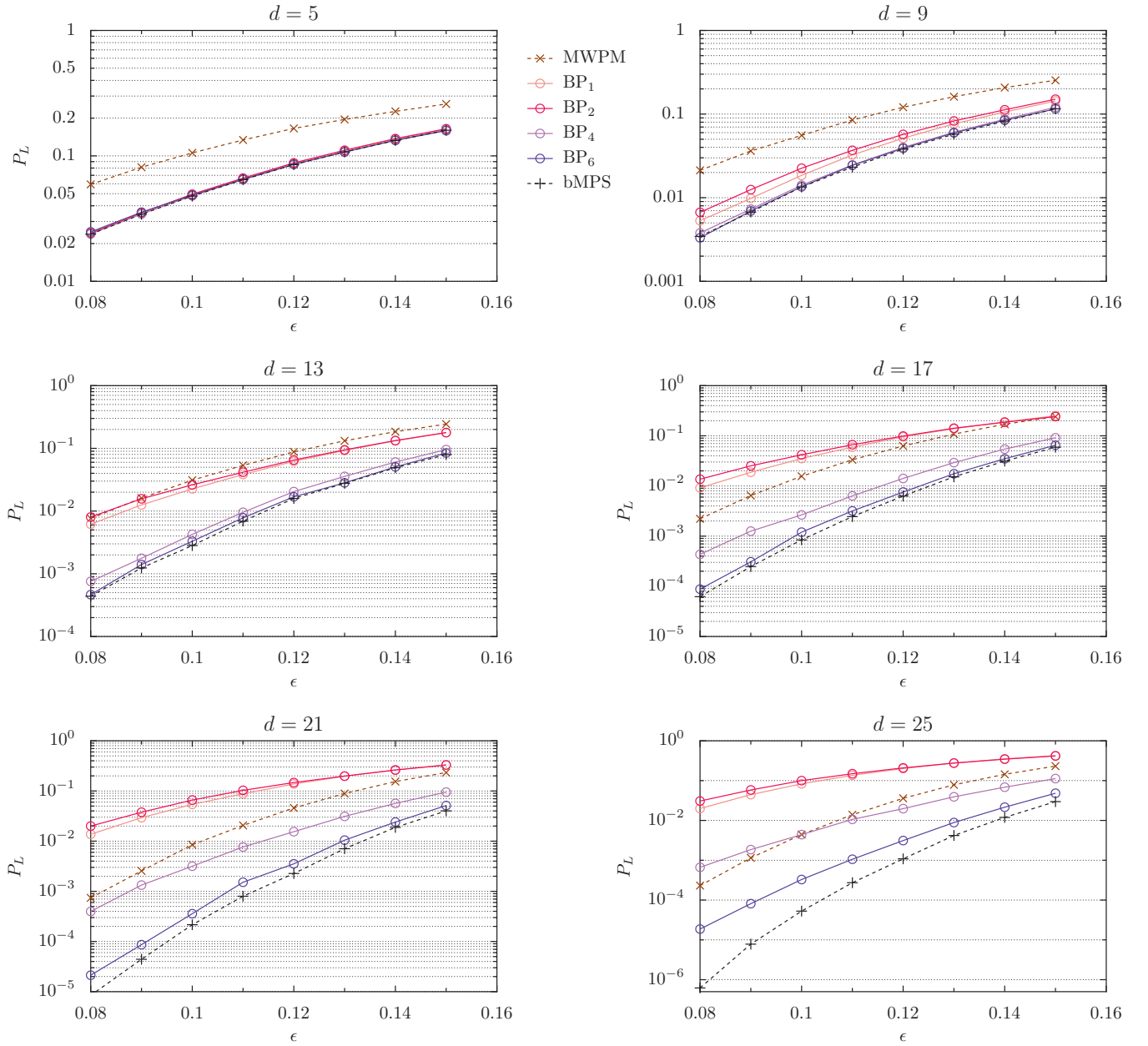


FIG. 4. The dependence of the logical probability error  $P_L$  on the depolarization noise strength for  $d = 5, 9, \dots, 25$ , as given by the blockBP, bMPS, and MWPM decoders.

In Fig. 4, we plot the logical probability error  $P_L$  as a function of the depolarization noise strength  $\epsilon$  for various code sizes  $d = 5, 9, 13, \dots, 25$ . Perhaps the most evident feature of the blockBP decoder is its sensitivity to the code size parameter  $d$ . For  $d = 5$ , the results of all blockBP runs are virtually indistinguishable from those of the MPS decoder (which is nearly optimal). However, as  $d$  increases, the blockBP instances with smaller  $k$  become less and less accurate.

This phenomena is captured in Figs. 5 and 6, where we plot the logical failure probability as a function of  $d$  for different noise levels. We see that for every block size  $k$  there exists a lattice size  $d_k$  at which  $P_L$  stops decreasing as a function of  $d$ . The minima is roughly independent of  $\epsilon$ , as evident from Fig. 7, which depicts the  $k = 4$  blockBP  $P_L$  for  $\epsilon = 0.15$  and

$\epsilon = 0.08$ . In both cases, blockBP achieves its best performance at  $d = 17$ , which then deteriorates for higher values of  $d$ . Inspecting Fig. 4, we see that  $d_k \approx 9$  for  $k = 1, 2$  and  $d_6 \approx 25$ . When  $d \lesssim d_k$ , the blockBP decoder often yields logical error probabilities that are much smaller than those of the MWPM decoder. This can be seen in Fig. 4 for various block sizes, and more directly in Fig. 8, which plots the ratio of the MWPM  $P_L$  with that of the  $k = 4$  blockBP in a  $d = 17$  code for  $\epsilon = 0.07 \rightarrow 0.15$ . This ratio increases as  $\epsilon$  decreases and seems to saturate at around a factor of 8 for  $\epsilon \lesssim 0.09$ .

The degradation of decoding quality as a function of  $d$  is not unique to blockBP. It also exists in the usual TN decoder, which is based on bMPS contraction, though to a much lesser degree. Indeed, also in the TN decoder, if we let  $d$  grow while keeping the truncation bond  $\chi$  fixed, the

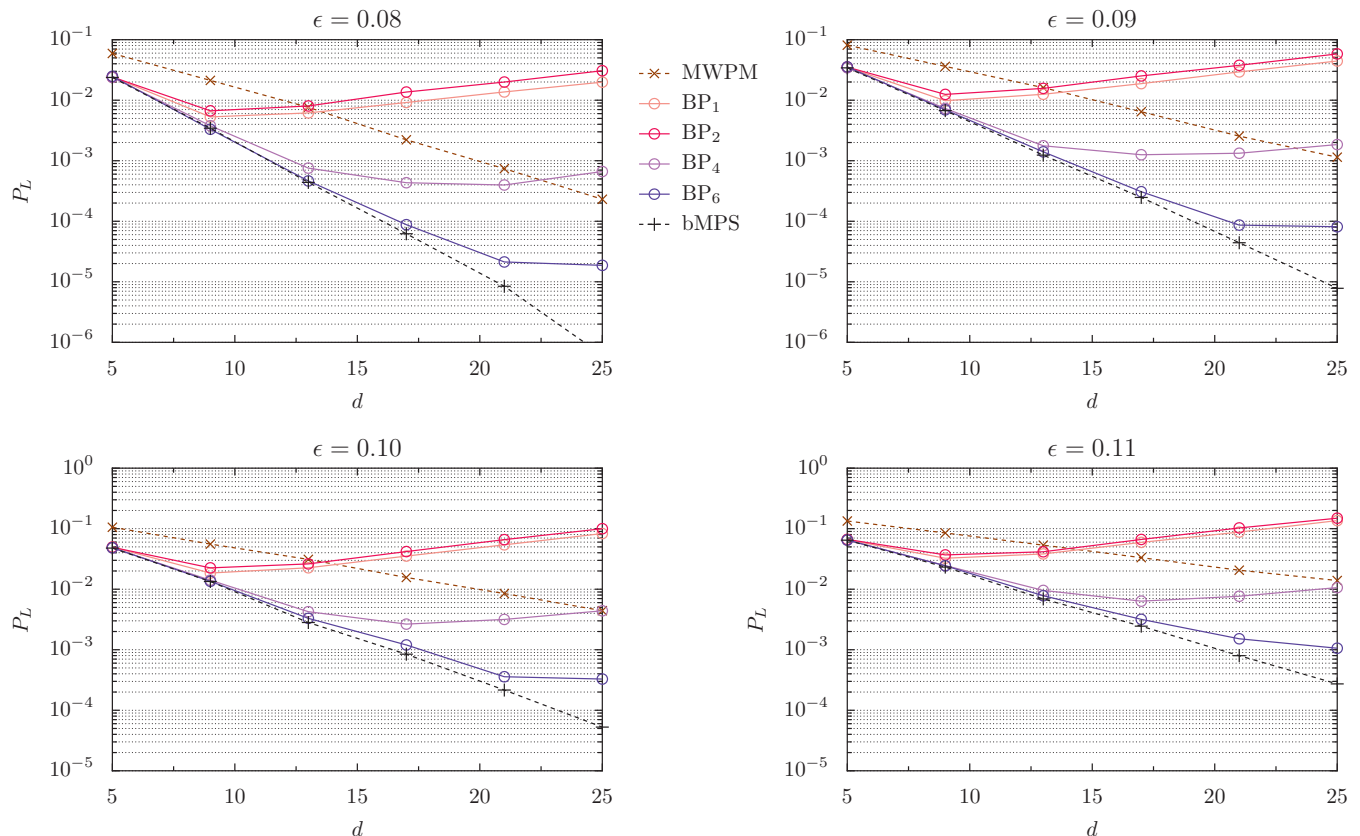


FIG. 5. The logical error probability  $P_L$  as a function of the code distance  $d$  for a physical depolarization rates of  $\epsilon = 0.08, 0.09, 0.10, 0.11$  and for the different decoders.

truncation errors in the bMPS contraction will eventually dominate the estimated probability and increase the failure rate. In the blockBP decoder, the block size  $k$  roughly corresponds to  $\chi$ ; increasing it leads to lower error probabilities, at the expense of longer running times. This is an indication that the main cause of errors in the blockBP decoder is the presence of small loops in the graph, which are well known to degrade the performance of BP. Increasing the block size means that we treat the small loops within each block more accurately, thereby increasing the accuracy of the approximate contraction. However, it should be noted that the degradation in blockBP decoding quality is much more pronounced than in the bMPS case. This might be attributed to the fact that there can be several competing BP fixed points, and the decoder might occasionally be led to the wrong one. Another related possible explanation is that the BP algorithm is much more local than the bMPS algorithm, and might therefore fail to capture long-range correlations.

While we believe that this degradation in decoding quality is unavoidable, it might be somewhat elevated by optimizing the meta-parameters of the BP algorithm such as the  $\text{max-iter}$ ,  $\delta_0$ ,  $\delta_1$  parameters, or using smarter BP scheduling [66] and/or better update methods [67,68]. As it stands, the block size  $k$  in the blockBP decoder should be chosen in conjunction with the code size  $d$ ;  $k = 4$  should only be used with  $d \lesssim 17$  and  $k = 6$  with  $d \lesssim 25$ . While this limits the applicability of the decoder for large codes, it is still possible

that blockBP can be useful in the near future for small or medium codes.

We conclude this section by describing two empirical observations, which might be relevant for further improvements of the decoder.

*Fast convergence to the right coset.* Generally, when running the blockBP decoder, we noticed that the right coset is almost always the one with the fastest BP convergence. This is an indication that the TN that represents the right coset is less critical, with fewer long-range correlations. In our blockBP decoder, we take advantage of this behavior by using the  $\delta_1$  parameter in conjunction with a relatively low  $\text{max-iter}$  bound. Together, these parameters tend to automatically reject cosets with a slow BP convergence—which are mostly the wrong cosets. This results in many cases where the blockBP is seemingly performing much better than one would have guessed by comparing the coset probabilities calculated by blockBP to the more accurate ones calculated by bMPS. This suggests that there is more structure in the TN that can be used to infer the right coset than merely its overall contraction value. Finally, we remark that a similar behavior was also observed in the bMPS decoder in Ref. [8], where the convergence of the bMPS algorithm with respect to the truncation bond  $\chi$  is much faster on the right coset than on the other cosets.

*Faster convergence for  $\epsilon \rightarrow 0$ .* As  $\epsilon \rightarrow 0$ , the TNs of the cosets become less critical, with shorter-range correlations, and as a result the BP messages converge faster and produce

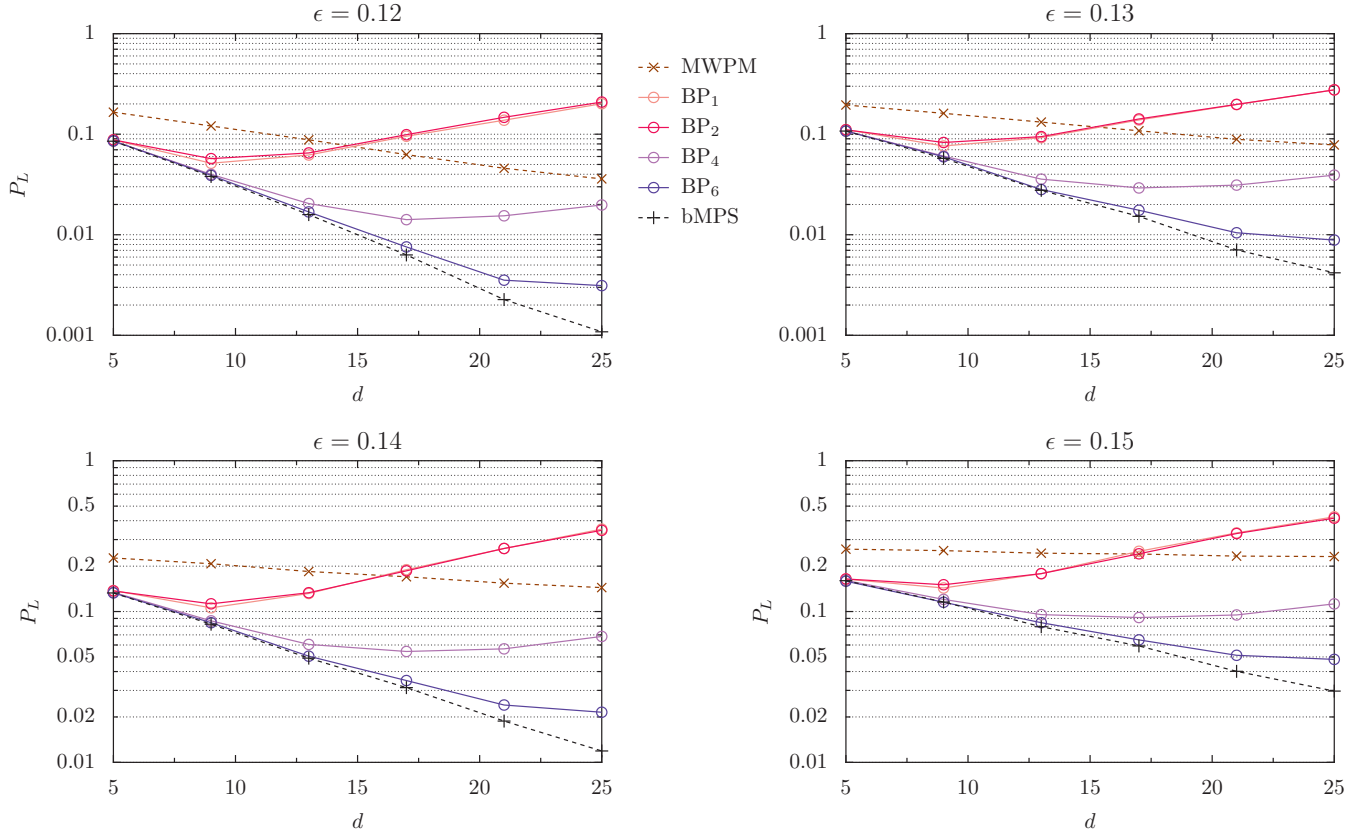


FIG. 6. The logical error probability  $P_L$  as a function of the code distance  $d$  for a physical depolarization rates of  $\epsilon = 0.12, 0.13, 0.14, 0.15$  and for different decoders. Every point was calculated using at least 100 failures.

more accurate results. This means that for low noise rate we can lower the max-iter parameter without substantially sacrificing the decoder performance.

## V. SUMMARY AND CONCLUSIONS

In this paper, we presented a new decoder to the surface code and other topological 2D codes. It is based on the framework of tensor-network decoding, equipped with the parallelism of the belief-propagation algorithm in the form of the blockBP algorithm. Unlike traditional uses of BP in the decoding problem, our BP algorithm does not run on the underlying Tanner graph; instead, it is used to approximate the contraction of the tensor networks of the error cosets, which makes it a DQMLD decoder. The central parameter in the decoder is the block size  $k$ , which determines both the complexity of the decoding and its accuracy. In our Monte Carlo simulations, using only 20 message passing rounds, we have shown that for code distances of  $d \lesssim 9$ , a block size of  $k = 1, 2$  exceeds the accuracy of the MWPM decoder; similarly for  $d \lesssim 17$  with  $k = 4$  and  $d \lesssim 25$  with  $k = 6$ . Being a BP algorithm, our decoder uses a small number of message passing rounds to converge, which potentially makes it suitable for real-time decoding of small to medium codes.

Our research leaves several open questions. First, it is crucial to see whether our decoder can be extended to the circuit noise model, and test its performance in that setting. A recent paper [12] showed that the tensor-network decoding

framework can be adapted to this setting, which leads to believe that so is the blockBP decoder. An equally important question is to understand whether the decoder can actually be implemented in real-time decoding. Since belief-propagation algorithms have long been used in such scenarios, we believe that, at least in the noiseless measurement model, this should be possible. It would also be interesting to see the performance of our decoder at different noise models, for example, biased noise. Finally, there are many different techniques in the literature to improve the performance of BP algorithms, such as smarter scheduling [66] or using techniques of machine learning [68]. It would be interesting to see whether any of these methods can improve the performance of our decoder.

## ACKNOWLEDGMENTS

We thank the computational support of D. Poletti, B. Xing, and X. Xu. from SUTD, Singapore, and the support from the joint Israel-Singapore NRF-ISF Research Grant No. NRF2020-NRF-ISF004-3528.

## DATA AVAILABILITY

The data that support the findings of this article are not publicly available upon publication because it is not technically feasible and/or the cost of preparing, depositing, and hosting the data would be prohibitive within the terms of this research project. The data are available from the authors upon reasonable request.

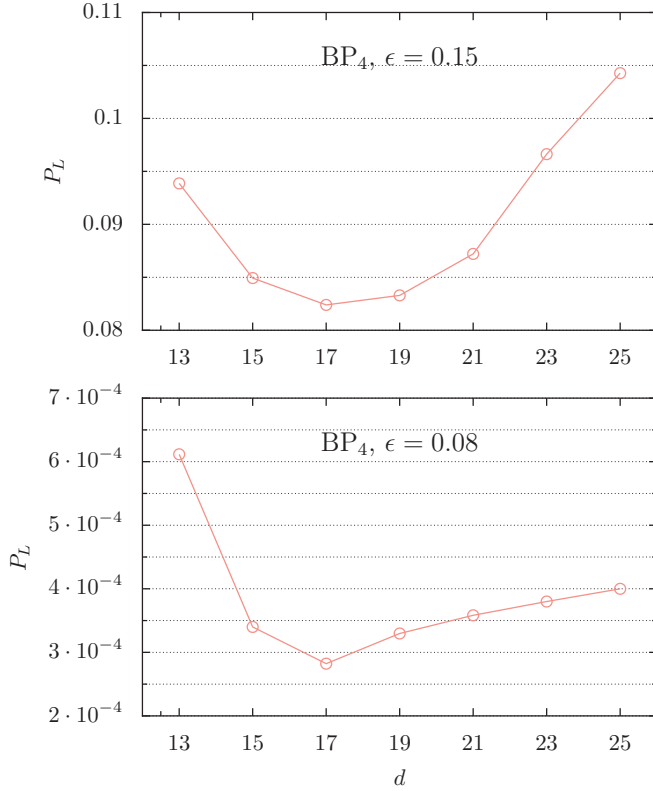


FIG. 7. The logical error probability for  $k = 4$  blockBP for  $d = 17$  code and noise rate  $\epsilon = 0.15$  (top) and  $\epsilon = 0.08$  (bottom). In both cases, it is clear that the minimal error probability is achieved for  $d = 17$ , after which the decoder performance deteriorates. The same holds for all noise rates  $\epsilon$ , showing that the location of the minima is independent of  $\epsilon$ . For other block sizes, the minima appears at other  $d$ .

#### APPENDIX A: PROOF OF LEMMA 1

We begin by introducing the Bethe free-energy formula in the TN notation. Recall that with every edge  $e = (u, v) \in E$ , we associated a variable  $x_e$  that takes values in the range of

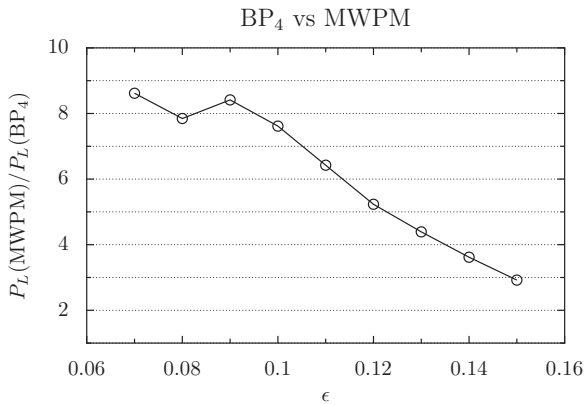


FIG. 8. The ratio of the logical error probability of the MWPM decoder and the  $k = 4$  blockBP decoder for  $d = 17$  over different noise rates  $\epsilon$ . blockBP increases its advantage over MWPM as  $\epsilon$  decreases and saturates to a ratio of about 8 for  $\epsilon \lesssim 0.9$ .

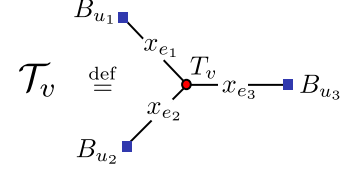


FIG. 9. The auxiliary tensor network  $\mathcal{T}_v$  that is constructed for every vertex  $v \in V$ . The tensor sitting at  $v$  is the tensor  $T_v$  from the original TN. The leaf tensors  $B_{u_1}, B_{u_2}, B_{u_3}$  are defined by the incoming messages to  $v$ :  $B_{u_i}(e_i) \stackrel{\text{def}}{=} m_{u_i \rightarrow v}(x_{e_i})$ .

the tensor indices that are contracted along that edge. Then the message  $m_{u \rightarrow v}(x_e)$  can be viewed either as a vector or as a function of  $x_e$ . Similarly, every tensor  $T_v$  with adjacent edges  $e_1, e_2, \dots$  can be viewed as a function  $T_v(x_{e_1}, x_{e_2}, \dots) \stackrel{\text{def}}{=} T_v(\mathbf{x}_v)$ , where we have defined  $\mathbf{x}_v \stackrel{\text{def}}{=} (x_{e_1}, x_{e_2}, \dots)$  to be the set of edge variables that are associated with the tensor  $T_v$ . Using this notation, we define the *edge marginals*  $\{P_e(x_e)\}$  and the *vertex marginals*  $\{P_v(\mathbf{x}_v)\}$  by

$$P_e(x_e) \stackrel{\text{def}}{=} \frac{1}{Z_e} m_{u \rightarrow v}(x_e) \cdot m_{v \rightarrow u}(x_e), \quad (\text{A1})$$

$$P_v(\mathbf{x}_v) \stackrel{\text{def}}{=} \frac{1}{Z_v} T_v(\mathbf{x}_v) \cdot \prod_{u \in N_v} m_{u \rightarrow v}(x_e), \quad (\text{A2})$$

where  $Z_e, Z_v$  are the normalization factors defined such that  $\sum_{x_e} P_e(x_e) = \sum_{\mathbf{x}_v} P_v(\mathbf{x}_v) = 1$ . We note that when the TN has negative or complex values, the “marginals” above need not be actual probability distributions. Our formula will nevertheless hold also in these cases. Note also that the edge marginal  $P_e(x_e)$  can also be expressed using the normalized messages as

$$P_e(x_e) = \hat{m}_{u \rightarrow v}(x_e) \cdot \hat{m}_{v \rightarrow u}(x_e). \quad (\text{A3})$$

As shown in Eq. (A12) of Ref. [47], in terms of  $P_e(x_e)$  and  $P_v(\mathbf{x}_v)$ , the Bethe free-energy of the TN is given by

$$F_{\text{Bethe}}(\mathcal{T}) = \sum_{v \in V} \sum_{\mathbf{x}_v} P_v(\mathbf{x}_v) \ln \frac{P_v(\mathbf{x}_v)}{T_v(\mathbf{x}_v)} - \sum_{e \in E} \sum_{x_e} P_e(x_e) \ln P_e(x_e). \quad (\text{A4})$$

It can be used to approximate contraction of  $\mathcal{T}$  by the formula  $\text{Tr}(\mathcal{T}) \simeq e^{-F_{\text{Bethe}}(\mathcal{T})}$ , which becomes exact when the underlying TN has a tree structure [42].

For each tensor  $T_v$ , we now construct a starlike tensor network  $\mathcal{T}_v$  in which  $T_v$  sits at the center, surrounded by its neighboring vertices from  $G$ . On each neighboring vertex  $u \in N_v$ , we place a one-leg tensor  $B_u$  that is defined using the incoming *normalized* messages  $\hat{m}_{u \rightarrow v}(x_e)$  [see condition (8)]:

$$B_u(x_e) \stackrel{\text{def}}{=} \hat{m}_{u \rightarrow v}(x_e).$$

This construction is depicted in Fig. 9.

We now observe the following:

(1) The contraction of  $\mathcal{T}_v$  is

$$\text{Tr}(\mathcal{T}_v) = \text{Tr}(T_v \cdot B_{u_1} \cdot B_{u_2} \cdots) = \text{Tr}\left(T_v \prod_{u \in N_v} \hat{m}_{u \rightarrow v}\right).$$

(2) As  $\mathcal{T}_v$  is a tree TN, its contraction can be written exactly using its Bethe free-energy,

$$\text{Tr}(\mathcal{T}_v) = e^{-F_{\text{Bethe}}(\mathcal{T}_v)}.$$

Using these observations, it follows that to prove the lemma it is sufficient to show that

$$\mathcal{F}_{\text{Bethe}}(\mathcal{T}) = \sum_v \mathcal{F}_{\text{Bethe}}(\mathcal{T}_v), \quad (\text{A5})$$

as it would imply

$$\begin{aligned} e^{-\mathcal{F}_{\text{Bethe}}(\mathcal{T})} &= \prod_v e^{-\mathcal{F}_{\text{Bethe}}(\mathcal{T}_v)} \\ &= \prod_v \text{Tr}\left(T_v \prod_{u \in N_v} \hat{m}_{u \rightarrow v}\right). \end{aligned}$$

The rest of the proof is therefore dedicated to proving Eq. (A5).

We begin by writing  $F_{\text{Bethe}}(\mathcal{T}_v)$  as a sum of three terms  $A, B, C$ :

$$\begin{aligned} F_{\text{Bethe}}(\mathcal{T}_v) &= \overbrace{\sum_{\mathbf{x}_v} P_v(\mathbf{x}_v) \ln \frac{P_v(\mathbf{x}_v)}{T_v(\mathbf{x}_v)}}^A \\ &\quad - \overbrace{\left( \sum_{x_{e_1}} P_{e_1}(x_{e_1}) \ln P_{e_1}(x_{e_1}) + \sum_{x_{e_2}} P_{e_2}(x_{e_2}) \ln P_{e_2}(x_{e_2}) + \cdots \right)}^B \\ &\quad + \overbrace{\sum_{x_{e_1}} P_{u_1}(x_{e_1}) \ln \frac{P_{u_1}(x_{e_1})}{B_{u_1}(x_{e_1})} + \sum_{x_{e_2}} P_{u_2}(x_{e_2}) \ln \frac{P_{u_2}(x_{e_2})}{B_{u_2}(x_{e_2})} + \cdots}^C. \end{aligned} \quad (\text{A6})$$

Importantly, up to normalization, the converged BP messages on  $\mathcal{T}_v$  coincide with those of the original TN. Therefore, the vertex edge marginal  $P_v(\mathbf{x}_v)$  of  $\mathcal{T}_v$  coincides with that of the original  $\mathcal{T}$ , and the same thing is true for the edge marginals  $P_{e_i}(x_{e_i})$ . The vertex marginals  $P_{u_i}(x_{e_i})$ , however, which appear in  $C$ , are not identical to those in the original TN. In fact, they may not even be defined on the same number of variables. Nevertheless, we note that by Eq. (A3),

$$\frac{P_{u_i}(x_{e_i})}{B_{u_i}(x_{e_i})} = \frac{\hat{m}_{u_i \rightarrow v}(x_{e_i}) \cdot \hat{m}_{v \rightarrow u_i}(x_{e_i})}{\hat{m}_{u_i \rightarrow v}(x_{e_i})} = \hat{m}_{v \rightarrow u_i}(x_{e_i})$$

and therefore  $C$  becomes

$$\sum_{x_{e_1}} P_{e_1}(x_{e_1}) \ln \hat{m}_{v \rightarrow u_1}(x_{e_1}) + \sum_{x_{e_2}} P_{e_2}(x_{e_2}) \ln \hat{m}_{v \rightarrow u_2}(x_{e_2}) + \cdots$$

Summing  $F_{\text{Bethe}}(\mathcal{T}_v)$  contributions over all nodes  $v$ , we see that term  $A$  gives us exactly the corresponding expression in  $F_{\text{Bethe}}(\mathcal{T})$  as in Eq. (A4). Term  $B$  gives us  $-2 \sum_{e \in E} P_e(x_e) \ln P_e(x_e)$  because for each edge  $e$  there is a contribution from each of its two adjacent nodes. However, for each edge  $e = (u, v)$ , the term  $C$  contains

$$\begin{aligned} &\sum_{x_e} P_e(x_e) (\ln \hat{m}_{u \rightarrow v}(x_e) + \ln \hat{m}_{v \rightarrow u}(x_e)) \\ &= \sum_{x_e} P_e(x_e) \ln P_e(x_e), \end{aligned}$$

where the last equality follows from Eq. (A3). Consequently, the extra term in  $B$  is canceled and this concludes the proof.

## APPENDIX B: COMPARING THE DECODING SPEEDS OF THE blockBP AND bMPS DECODERS

In this section, we present a concrete comparison of the decoding time of a blockBP decoder with a bMPS decoder. As already noted in the body of the paper (remark 5 in Sec. III E), the numbers we are about to present should be taken with a grain of salt since the decoding time of an actual real-time decoder would highly depend on the implementation details (hardware and software), in particular for the case parallel algorithms. Nevertheless, as a proof of concept, and to get a rough estimate of a possible advantage for the blockBP decoder over the bMPS decoder, we have compared their running time for the cases of  $d = 25$  and  $k = 6$  with a depolarization noise strength of  $\epsilon = 0.15$ , and for  $d = 17, k = 4$  with  $\epsilon = 0.08$ . We have used the Python code for blockBP that was used in the main text and compared it to a Python bMPS implementation that we wrote, which relied on the same MPS library. In both implementations, we have used the qecsim Python library [64] to run the external loop and generate the different configurations, where each decoder was implemented as an extension to qecsim. Note that we have used our own implementation of bMPS rather than the builtin qecsim implementation in order for the comparison to be as fair as possible [69]. Finally, we note that our blockBP decoder implementation was *serial*, since we do not possess any real-time hardware with many cores. Its running time was then used to estimate the running time of a parallel implementation, as discuss below.

For  $d = 25$  and  $\epsilon = 0.15$ , the logical error probability of the bMPS decoder is  $\simeq 0.0297$ , whereas that of blockBP is  $\simeq 0.0482$ . The former was obtained using  $\chi = 16$ , which guaranteed nearly optimal decoding for  $d \leq 25$ . To make the comparison more fair, we reduced the  $\chi$  parameter of the bMPS algorithm to  $\chi = 6$ , which made the decoder much faster at the expense of a worse logical error probability of  $\simeq 0.044$ —on par with the blockBP error rate.

To compare the two decoders, we ran both of them on 100 instances. We used a single core of a desktop Intel i7 machine with a clock speed of 2.80 GHz. The running time of the bMPS decoder was  $\simeq 0.73$  s per instance, while that of the blockBP decoder was  $\simeq 20.65$  s for an instance. To gauge the blockBP serial running time to a parallel running time, we note that for  $d = 25$ , the TN lattice is of size  $49 \times 49$ , which is then divided into  $9 \times 9$  blocks of  $6 \times 6$  sites. If

we assign every block with 4 cores (one for each message), this gives us a total of  $9^2 \times 4 = 324$  cores. That means that in an ideal parallel implementation, we should expect the blockBP to decode an instance in  $\simeq 20.65/324 = 0.063$  s. However, taking into account the cost of parallelization (e.g., the time it takes to pass the messages between the cores, together with global pre- and postprocessing), we should arguably increase this time by 10%–50%, which would yield a decoding time of 0.07–0.096 s per instance. This implies a factor of 7–10 speedup of the blockBP decoder over the bMPS decoder. The same experiment was repeated for an error rate of  $\epsilon = 0.1$ , yielding similar numbers (with a slight improvement of the blockBP times).

For the  $k = 4$  case, the speedup is more impressive because blockBP uses plain BP on the fused  $4 \times 4$  blocks

without relying on any MPS infrastructure (see remark 9 in Sec. III E). Specifically, we considered the  $d = 17$  case with  $\epsilon = 0.08$ . In such case, the blockBP logical error rate is  $\simeq 3 \times 10^{-4}$ , while that of the bMPS decoder is  $\simeq 6 \times 10^{-5}$  for  $\chi = 16$ . Reducing  $\chi$  to  $\chi = 4$  led a bMPS logical error rate of  $\simeq 4 \times 10^{-4}$ , slightly larger than that of blockBP. In such setup, the bMPS decoding time was 0.2 s, while that of the (serial) blockBP was 0.75 s. Since the  $d = 17$  grid is made of  $33 \times 33$  sites that are divided into  $9 \times 9$  blocks of  $4 \times 4$ , this means that in an ideal parallel implementation of blockBP, we should expect a  $\simeq 9^2 \times 4 = 324$  speedup, which amounts to a decoding time of about 0.0023 s. Even if add to that 10%–50% overhead due to parallelization, we still get a speedup of about two orders of magnitude over the bMPS decoder.

- [1] A. Kitaev, Fault-tolerant quantum computation by anyons, *Ann. Phys.* **303**, 2 (2003).
- [2] D. Aharonov and M. Ben-Or, Fault-tolerant quantum computation with constant error rate, *SIAM J. Comput.* **38**, 1207 (2008).
- [3] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, G. J. Norris, C. K. Andersen, M. Müller, A. Blais, C. Eichler, and A. Wallraff, Realizing repeated quantum error correction in a distance-three surface code, *Nature (London)* **605**, 669 (2022).
- [4] Y. Zhao *et al.*, Realization of an error-correcting surface code with superconducting qubits, *Phys. Rev. Lett.* **129**, 030501 (2022).
- [5] Google Quantum AI, Suppressing quantum errors by scaling a surface code logical qubit, *Nature (London)* **614**, 676 (2023).
- [6] P. Iyer and D. Poulin, Hardness of decoding quantum stabilizer codes, *IEEE Trans. Inf. Theory* **61**, 5209 (2015).
- [7] A. deMarti iOlus, P. Fuentes, R. Orús, P. M. Crespo, and J. Etzezarreta Martinez, Decoding algorithms for surface codes, *Quantum* **8**, 1498 (2024).
- [8] S. Bravyi, M. Suchara, and A. Vargo, Efficient algorithms for maximum likelihood decoding in the surface code, *Phys. Rev. A* **90**, 032326 (2014).
- [9] D. K. Tuckett, A. S. Darmawan, C. T. Chubb, S. Bravyi, S. D. Bartlett, and S. T. Flammia, Tailoring surface codes for highly biased noise, *Phys. Rev. X* **9**, 041031 (2019).
- [10] C. T. Chubb and S. T. Flammia, Statistical mechanical models for quantum codes with correlated noise, *Ann. Inst. Henri Poincaré D* **8**, 269 (2021).
- [11] C. T. Chubb, General tensor network decoding of 2D Pauli codes, *arXiv:2101.04125*.
- [12] C. Piveteau, C. T. Chubb, and J. M. Renes, Tensor network decoding beyond 2D, *PRX Quantum* **5**, 040303 (2024).
- [13] C. Guo, D. Poletti, and I. Arad, Block belief propagation algorithm for two-dimensional tensor networks, *Phys. Rev. B* **108**, 125111 (2023).
- [14] J. Pearl, Reverend bayes on inference engines: A distributed hierarchical approach, in *Proceedings of the Second AAAI Conference on Artificial Intelligence, AAAI'82* (AAAI Press, Pennsylvania Ave, NW, Washington, DC, 1982), pp. 133–136.
- [15] S.-Y. Chung, G. Forney, T. Richardson, and R. Urbanke, On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit, *IEEE Commun. Lett.* **5**, 58 (2001).
- [16] C. Berrou, A. Glavieux, and P. Thitimajshima, Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1, in *Proceedings of ICC'93—IEEE International Conference on Communications* (IEEE, Geneva, Switzerland, 1993), Vol. 2, pp. 1064–1070.
- [17] D. Poulin and Y. Chung, On the iterative decoding of sparse quantum codes, *Quantum Inf. Comput.* **8**, 987 (2008).
- [18] G. Duclos-Cianci and D. Poulin, Fast decoders for topological quantum codes, *Phys. Rev. Lett.* **104**, 050504 (2010).
- [19] G. Duclos-Cianci and D. Poulin, Fault-tolerant renormalization group decoder for Abelian topological codes, *Quantum Inf. Comput.* **14**, 721 (2014).
- [20] B. Criger and I. Ashraf, Multi-path summation for decoding 2D topological codes, *Quantum* **2**, 102 (2018).
- [21] 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, *Phys. Rev. X* **13**, 031007 (2023).
- [22] J. Edmonds, Paths, trees, and flowers, *Can. J. Math.* **17**, 449 (1965).
- [23] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, *J. Math. Phys.* **43**, 4452 (2002).
- [24] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, *Phys. Rev. A* **86**, 032324 (2012).
- [25] A. G. Fowler, Minimum weight perfect matching of fault-tolerant topological quantum error correction in average  $O(1)$  parallel time, *Quantum Inf. Comput.* **15**, 145 (2015).
- [26] A. Kubica, B. Yoshida, and F. Pastawski, Unfolding the color code, *New J. Phys.* **17**, 083026 (2015).
- [27] H. Bombin and M. A. Martin-Delgado, Optimal resources for topological two-dimensional stabilizer codes: Comparative study, *Phys. Rev. A* **76**, 012305 (2007).
- [28] D. Gottesman, Stabilizer codes and quantum error correction, Ph.D. thesis, California Institute of Technology, 1997.
- [29] D. P. DiVincenzo, P. W. Shor, and J. A. Smolin, Quantum-channel capacity of very noisy channels, *Phys. Rev. A* **57**, 830 (1998).
- [30] N. Delfosse and N. H. Nickerson, Almost-linear time decoding algorithm for topological codes, *Quantum* **5**, 595 (2021).
- [31] P. Panteleev and G. Kalachev, Degenerate quantum LDPC codes with good finite length performance, *Quantum* **5**, 585 (2021).

- [32] J. Roffe, D. R. White, S. Burton, and E. Campbell, Decoding across the quantum low-density parity-check code landscape, *Phys. Rev. Res.* **2**, 043423 (2020).
- [33] M.-H. Hsieh and F. M. C. Le Gall, NP-hardness of decoding quantum error-correction codes, *Phys. Rev. A* **83**, 052331 (2011).
- [34] K.-Y. Kuo and C.-C. Lu, On the hardnesses of several quantum decoding problems, *Quantum Inf. Proc.* **19**, 123 (2020).
- [35] K.-Y. Kuo and C.-C. Lu, On the hardness of decoding quantum stabilizer codes under the depolarizing channel, in *2012 International Symposium on Information Theory and its Applications* (IEICE, Kikai-Shinko-Kaikan Bldg., Japan, 2012), pp. 208–211.
- [36] E. Berlekamp, R. McEliece, and H. van Tilborg, On the inherent intractability of certain coding problems (Corresp.), *IEEE Trans. Inf. Theory* **24**, 384 (1978).
- [37] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, *Ann. Phys.* **349**, 117 (2014).
- [38] U. Schollwöck, The density-matrix renormalization group in the age of matrix product states, *Ann. Phys.* **326**, 96 (2011).
- [39] N. Schuch, M. M. Wolf, F. Verstraete, and J. I. Cirac, Computational complexity of projected entangled pair states, *Phys. Rev. Lett.* **98**, 140506 (2007).
- [40] F. Verstraete and J. I. Cirac, Renormalization algorithms for quantum-many body systems in two and higher dimensions, *arXiv:cond-mat/0407066*.
- [41] F. Verstraete, V. Murg, and J. Cirac, Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems, *Adv. Phys.* **57**, 143 (2008).
- [42] M. Mezard and A. Montanari, *Information, Physics, and Computation* (Oxford University Press, Oxford, 2009).
- [43] M. Mézard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond: An Introduction to the Replica Method and its Applications* (World Scientific, Singapore, 1987).
- [44] S. Yoon, A. V. Goltsev, S. N. Dorogovtsev, and J. F. F. Mendes, Belief-propagation algorithm and the Ising model on networks with arbitrary distributions of Motifs, *Phys. Rev. E* **84**, 041144 (2011).
- [45] B. Karrer, M. E. J. Newman, and L. Zdeborová, Percolation on sparse networks, *Phys. Rev. Lett.* **113**, 208702 (2014).
- [46] M. Mézard, G. Parisi, and R. Zecchina, Analytic and algorithmic solution of random satisfiability problems, *Science* **297**, 812 (2002).
- [47] R. Alkabetz and I. Arad, Tensor networks contraction and the belief propagation algorithm, *Phys. Rev. Res.* **3**, 023073 (2021).
- [48] S. Sahu and B. Swingle, Efficient tensor network simulation of quantum many-body physics on sparse graphs, *arXiv:2206.04701*.
- [49] T. Begušić, J. Gray, and G. K.-L. Chan, Fast and converged classical simulations of evidence for the utility of quantum computing before fault tolerance, *Sci. Adv.* **10**, eadk4321 (2024).
- [50] J. Tindall, M. Fishman, E. M. Stoudenmire, and D. Sels, Efficient tensor network simulation of IBM’s Eagle kicked Ising experiment, *PRX Quantum* **5**, 010308 (2024).
- [51] J. Tindall and M. Fishman, Gauging tensor networks with belief propagation, *SciPost Phys.* **15**, 222 (2023).
- [52] E. Robeva and A. Seigal, Duality of graphical models and tensor networks, *Inf. Inference J. IMA* **8**, 273 (2018).
- [53] J. S. Yedidia, W. Freeman, and Y. Weiss, Generalized belief propagation, in *Advances in Neural Information Processing Systems*, edited by T. Leen, T. Dietterich, and V. Tresp (MIT Press, Cambridge, MA, 2000), Vol. 13.
- [54] J. S. Yedidia, W. Freeman, and Y. Weiss, Constructing free-energy approximations and generalized belief propagation algorithms, *IEEE Trans. Inf. Theory* **51**, 2282 (2005).
- [55] A. Pelizzola, Cluster variation method in statistical physics and probabilistic graphical models, *J. Phys. A: Math. Gen.* **38**, R309 (2005).
- [56] A. Montanari and T. Rizzo, How to compute loop corrections to the Bethe approximation, *J. Stat. Mech.* (2005) P10011.
- [57] M. Chertkov and V. Y. Chernyak, Loop series for discrete statistical models on graphs, *J. Stat. Mech.* (2006) P06009.
- [58] J. Mooij, B. Wemmenhove, B. Kappen, and T. Rizzo, Loop corrected belief propagation, in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, edited by M. Meila and X. Shen, Proceedings of Machine Learning Research (PMLR, San Juan, Puerto Rico, 2007), Vol. 2, pp. 331–338.
- [59] A. Lage-Castellanos, R. Mulet, F. Ricci-Tersenghi, and T. Rizzo, Replica cluster variational method: The replica symmetric solution for the 2D random bond Ising model, *J. Phys. A: Math. Theor.* **46**, 135001 (2013).
- [60] C. Wang and H.-J. Zhou, Simplifying generalized belief propagation on redundant region graphs, *J. Phys.: Conf. Ser.* **473**, 012004 (2013).
- [61] H.-J. Zhou and W.-M. Zheng, Loop-corrected belief propagation for lattice spin models, *Eur. Phys. J. B* **88**, 336 (2015).
- [62] G. T. Cantwell and M. E. J. Newman, Message passing on networks with loops, *Proc. Natl. Acad. Sci. USA* **116**, 23398 (2019).
- [63] A. Kirkley, G. T. Cantwell, and M. E. J. Newman, Belief propagation for networks with loops, *Sci. Adv.* **7**, eabf1211 (2021).
- [64] D. K. Tuckett, Tailoring surface codes: Improvements in quantum error correction with biased noise, Ph.D. thesis, University of Sydney, 2020, <https://github.com/qecsim/qecsim>.
- [65] M. Jeruchim, Techniques for estimating the bit error rate in the simulation of digital communication systems, *IEEE J. Sel. Areas Commun.* **2**, 153 (1984).
- [66] M. V. d. Merwe, V. Joseph, and G. Gopalakrishnan, Message scheduling for performant, many-core belief propagation, in *Proceedings of the 2019 IEEE High Performance Extreme Computing Conference (HPEC)* (Curran Associates, Morehouse Lane Red Hook, NY, 2019), pp. 1–7.
- [67] C. Knoll, M. Rath, S. Tschiatschek, and F. Pernkopf, Message scheduling methods for belief propagation, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (Springer, New York, 2015), pp. 295–310.
- [68] E. Nachmani, Y. Be’ery, and D. Burshtein, Learning to decode linear codes using deep learning, in *Proceedings of the 2016 54th Annual Allerton Conference on Communication, Control, and Computing* (IEEE, New York, 2016), pp. 341–346.
- [69] In any case, using the built-in implementation had a negligible effect on the results.