Simulated-annealing decoder for the XZZX code with greedy-matching initialization

Tatsuya Sakashita*

Department of Physics, The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo 113-0033, Japan

(Received 5 March 2026; revised 20 July 2026; accepted 30 July 2026; published 12 August 2026)

The XZZX code is a variant of the surface code tailored to address biased noise in realistic quantum devices. We propose a simulated annealing (SA) decoder for the XZZX code. Our SA decoder is amenable to parallelization because its Markov chain Monte Carlo updates are simple and local. To initialize SA, we use a recovery configuration produced by our greedy-matching decoder. Although Z-biased noise is commonly assumed in realistic quantum devices, we instead focus on Y-biased noise. Under Y-biased noise, the minimum-weight perfect matching (MWPM) decoder becomes suboptimal because it cannot take into account the fact that a Y error contains both X and Z components on the same qubit. Our numerical simulations for the code capacity noise model, where only data qubits suffer errors, show that our SA decoder achieves higher accuracy than the MWPM decoder. They also show that, under moderately Y-biased noise, our SA decoder achieves accuracy comparable to that of a decoder based on IBM ILOG CPLEX Optimizer (CPLEX), which uses integer programming to find a minimum-energy error configuration consistent with the measured syndrome. In our greedy-matching decoder, we randomize the tie breaking among equal-weight pairs. This randomness generates a variety of initial configurations for SA, which improves the convergence of our SA decoder. These results suggest that combining SA with our greedy-matching initializer is a promising approach to decoding the XZZX code under Y-biased noise in the code capacity noise model.

DOI: 10.1103/5z6j-y33z

I. INTRODUCTION

The ultimate goal of quantum computing is to realize fault-tolerant quantum computation with robust error correction mechanisms. In superconducting qubit platforms, coherence time (i.e., qubit lifetime) is typically on the order of hundreds of microseconds. The error correction process must be completed within this time frame. Practical decoders therefore favor simple algorithms to be readily implemented in hardware.

The surface code is one of the most promising topological error correction codes [1,2]. It is experimentally feasible because it requires only nearest-neighbor interactions between data and ancilla qubits [3–7].

In physical systems such as superconducting qubits, optical systems, and ion traps, realistic noise is often biased, meaning that Z (phase-flip) errors occur more frequently than X (bit-flip) errors.

*Contact author: t-sakashita@phys.s.u-tokyo.ac.jp

For such biased noise, modifying the stabilizer generators of the surface code leads to alternative codes, such as the tailored surface code (which has only Y-type stabilizers) [8] and the XZZX code [9]. The XZZX code achieves higher thresholds under X-, Y-, or Z-biased Pauli noise than the conventional surface code. The conventional surface code is a CSS (Calderbank-Shor-Steane) code [10,11] with stabilizer generators consisting solely of X-type or Z-type operators. For the XZZX code, there have been proposals for its rotated version [12,13], a cluster-state (foliated) construction [14], numerical simulations for optical and superconducting qubits [15], and neutral-atom qubits [16].

The minimum-weight perfect matching (MWPM) decoder [1] can be executed in polynomial time in the number of syndrome defects and, hence, is widely used for the surface code. The standard MWPM decoder decomposes decoding into two matching subproblems. This decomposition does not explicitly capture the correlations between the X and Z components of a Y error, and its performance can therefore degrade when Y errors are non-negligible.

On the other hand, an MCMC (Markov chain Monte Carlo) decoder and a matrix product state (MPS) decoder [17] exhibit high decoding accuracy by taking into account the probability of Y errors. For Z- and Y-biased noise, the MPS decoder has been studied [18]. However, for a fixed truncation (bond) dimension χ, the MPS decoder scales as $\mathcal{O}(d^2)$ (up to polynomial factors in χ). In contrast, MCMC-based decoders use local updates with low computational cost, and the computation is easily parallelized.

One of the earliest studies [19] on MCMC decoders for the CSS surface code employed parallel tempering, while another study [20] utilized the Metropolis method fixed at the Nishimori inverse temperature. For an initial error configuration of MCMC, they employed error configurations generated by acting stabilizers and logical operators randomly on the true error configuration [12,19] and a recovery chain obtained by the MWPM decoder [20]. Thus, the aim of these previous works was to determine the threshold of the code accurately.

To efficiently minimize energy in MCMC methods, simulated annealing (SA) [21] is often used in statistical physics. By formulating the decoding problem as a random-bond Ising model, SA has been applied to decoders for the CSS surface code [22] and the color code [23]. Another formulation of an SA decoder for the CSS surface code has a Hamiltonian with penalty terms [24], and was demonstrated in a hardware-implemented annealing machine [22,25,26].

The XZZX code has been studied using the MWPM decoder, the MPS decoder [9], and the MCMC-based decoders [12]. They investigated the decoding accuracy for Z-biased noise, which is typical in quantum hardware platforms. Assuming knowledge of the probabilities of X, Y, Z errors, the authors incorporated this prior information into the weight coefficients of the Hamiltonian so as to improve decoding accuracy.

Reference [12] assumes the code capacity noise model, in which only data qubits suffer independently and identically distributed errors. To explore configurations efficiently, they carried out the MCMC method at a lower inverse temperature than the Nishimori inverse temperature while keeping the weight coefficients invariant in the Hamiltonian. Their numerical results suggest that extracting a dominant term of the partition function is sufficient to determine the correct logical operator class. Thus, their algorithm may be useful for collecting this term, which could then be used as training data for a neural-network decoder.

In this paper, we propose an SA decoder for the XZZX code. Our goal is not to determine the threshold as accurately as possible, but rather to study whether an SA-based decoder can provide a useful accuracy-run-time trade-off for the XZZX code under Y-biased noise. We assume the code capacity noise model. Specifically, we focus on Y-biased noise, under which the standard MWPM decoder can lose accuracy because its separate treatment of the X- and Z-error components does not explicitly account for their correlation within a Y error. In contrast, the SA decoder minimizes an energy function over error configurations, and the relative probabilities of X, Y, and Z errors are directly incorporated into this energy function. To initialize SA, we employ a recovery configuration produced by a greedy-matching decoder. All algorithms are implemented in C++ to minimize decoding time.

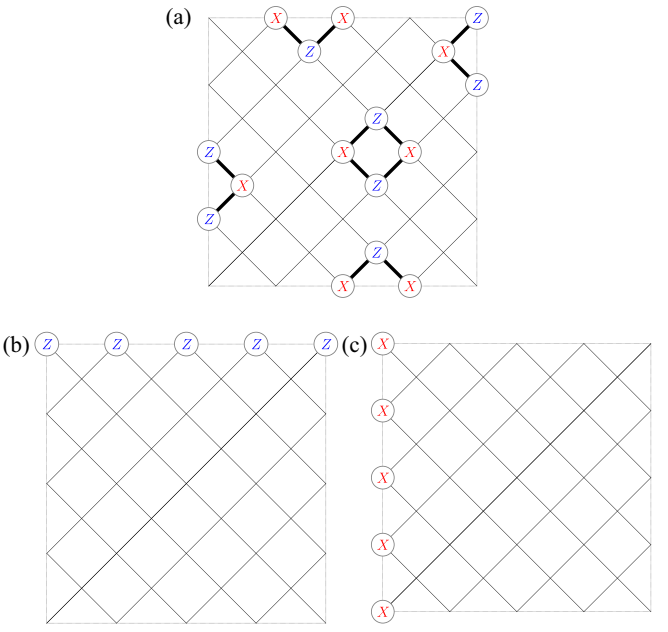

FIG. 1. The XZZX code for the code distance $d = 5$: (a) Stabilizer generators G_f, showing four-body operators in the bulk and three-body operators at the boundaries. (b) Logical operator L_Z. (c) Logical operator L_X.

Compared with previous MCMC-based studies, the present work differs in three main respects. First, we formulate an SA decoder for the XZZX code based on a minimum-energy formulation for finding a maximum *a posteriori* (MAP) error configuration. Second, we introduce a greedy-matching initializer and a randomized tie-breaking variant to generate diverse initial configurations for independent SA runs. Third, we focus on Y-biased noise, where correlations induced by Y errors make the standard MWPM decoder suboptimal.

The rest of this paper is organized as follows. In Sec. II, we review the XZZX code. For the XZZX code, we describe the principles of error detection and correction in Sec. III, and then present decoding algorithms, which include our SA decoders in Sec. IV. We show numerical results in Sec. V and discuss the benefits of our SA decoder in Sec. VI. We conclude in Sec. VII.

II. THE XZZX CODE

In this section, we briefly review the XZZX surface code [9]. We consider a rotated square lattice with open boundaries [27]. Data qubits reside on the vertices.

We illustrate stabilizer generators for the XZZX surface code in Fig. 1(a). The stabilizer generator acting around a face f is defined as

$$G_f := X_{\mathrm{left}(f)} Z_{\mathrm{up}(f)} Z_{\mathrm{down}(f)} X_{\mathrm{right}(f)}, \tag{1}$$

whose form is the origin of the name "XZZX code." This stabilizer generator is a four-body operator in the bulk and a three-body operator at the boundaries. The XZZX code has only one type of stabilizer generator, unlike the CSS surface code that has two types of generators. Since each stabilizer generator includes both X and Z operators as shown in Eq. (1), the XZZX code is not a CSS code. At the center of every face f, an ancilla qubit is placed to measure the stabilizer generator G_f and detect errors.

We define logical operators L_Z (L_X) as a Pauli chain connecting the left and right (top and bottom) boundaries, respectively. Figures 1(b) and 1(c) depict examples of the logical operators L_Z and L_X, respectively. We further define $L_Y := \mathrm{i}\, L_Z\, L_X$.

Throughout this paper, we ignore global phases of Pauli operators (i.e., overall factors ± 1 and $\pm \mathrm{i}$ are identified). Let $\overline{\mathcal{P}}$ denote the N_q-qubit Pauli group modulo global phases. In particular, we identify L_Y with $L_Z L_X$ in $\overline{\mathcal{P}}$.

We identify Pauli chains that differ by multiplication of a stabilizer operator in the stabilizer group $\mathcal{G} := \langle G_f \mid f : \text{face} \rangle$. More precisely, for $P \in \{I, X, Y, Z\}$, we define the logical Pauli operator as the equivalence class $[L_P]$ modulo stabilizers, i.e., $L_P \sim G L_P$ for any $G \in \mathcal{G}$.

The code distance d is defined as the minimum weight of a nontrivial logical operator (equivalently, the length of the shortest Pauli chain representing L_Z or L_X). For the surface codes including the XZZX code, this distance coincides with the linear size of the lattice. The number of data qubits is $N_\mathrm{q} = d^2 + (d-1)^2$. The number of stabilizer generators is equal to the number of faces, i.e., $N_\mathrm{stabilizer} = 2d(d-1) = \mathcal{O}(d^2)$.

The XZZX code is advantageous under biased noise. That is, the XZZX code has a threshold $p_\mathrm{th} = 0.5$ for pure X, Y, or Z noise in the code capacity noise model, reflecting its mapping to a classical repetition code [9]; on the other hand, the CSS surface code has $p_\mathrm{th} = 0.109$ for the pure X or Z noise [1] and $p_\mathrm{th} = 0.5$ for the pure Y noise [8].

III. PRINCIPLES OF ERROR DETECTION AND CORRECTION

Throughout this paper, an *error configuration* $C \in \overline{\mathcal{P}}$ specifies, for each data qubit, whether it experiences no error (I) or a Pauli error X, Y, or Z. Figure 2 illustrates an example of the error configuration. In Fig. 2, X, Y, Z errors occur on specific data qubits.

To describe error detection, we introduce two *decoding graphs* $(\mathcal{V}_\mathrm{white}, \mathcal{E}_\mathrm{white})$ and $(\mathcal{V}_\mathrm{gray}, \mathcal{E}_\mathrm{gray})$. We write $(\mathcal{V}_\bullet, \mathcal{E}_\bullet)$ with $\bullet \in \{\text{white}, \text{gray}\}$ and define it as follows:

(1) *The set of vertices* $\mathcal{V}_\bullet$: The vertices are the centers of faces with the color $\bullet$; each vertex corresponds to an ancilla qubit used to measure the stabilizer.

(2) *The set of edges* $\mathcal{E}_\bullet$: The edges connect nearest-neighbor vertices in $\mathcal{V}_\bullet$. Each edge is associated with the data qubit located at its midpoint; an error on that data qubit is represented on the corresponding edge.

The set of edges on which errors occur is called an *error chain*. There are two types of error chains $C_\mathrm{white} \subset \mathcal{E}_\mathrm{white}$ and $C_\mathrm{gray} \subset \mathcal{E}_\mathrm{gray}$, which connect centers of white and gray faces, respectively, and are drawn with the pink and purple lines in Fig. 2. In this paper, to avoid confusion, C (without a subscript) denotes an error configuration, whereas $C_\bullet$ (with $\bullet \in \{\text{white}, \text{gray}\}$) denotes an error chain. For both the error chains C_white and C_gray, the horizontal and vertical edges represent the presence of Z- and X-type Pauli components, respectively (i.e., Z or Y on a horizontal edge, and X or Y on a vertical edge). A Y error occurs exactly at the intersection point of both the error chains C_white and C_gray. With a slight abuse of notation, we use the same symbol $C_\bullet$ to denote the corresponding Pauli operator.

For each face f, we measure the stabilizer generator G_f and thereby obtain a measurement outcome $s_f \in \{0, 1\}$. For an error chain (viewed as a Pauli operator) $C_\bullet$, the outcome is determined by whether $C_\bullet$ commutes or anticommutes with G_f:

$$G_f\, C_\bullet = (-1)^{s_f}\, C_\bullet\, G_f, \tag{2}$$

i.e., $s_f = 0$ if $[G_f, C_\bullet] = 0$ and $s_f = 1$ if $\{G_f, C_\bullet\} = 0$. We refer to a face f with $s_f = 1$ as a *syndrome defect*.

A syndrome defect appears precisely at an end point of the error chain $C_\bullet$. More precisely, $s_f = 1$ if and only if $v_f \in \partial C_\bullet$, where v_f denotes the vertex (i.e., face center) corresponding to the face f, and $\partial C_\bullet$ denotes the set of vertices in $\mathcal{V}_\bullet$ that have odd incidence with the edge set $C_\bullet$ (equivalently, the end points of the chain $C_\bullet$).

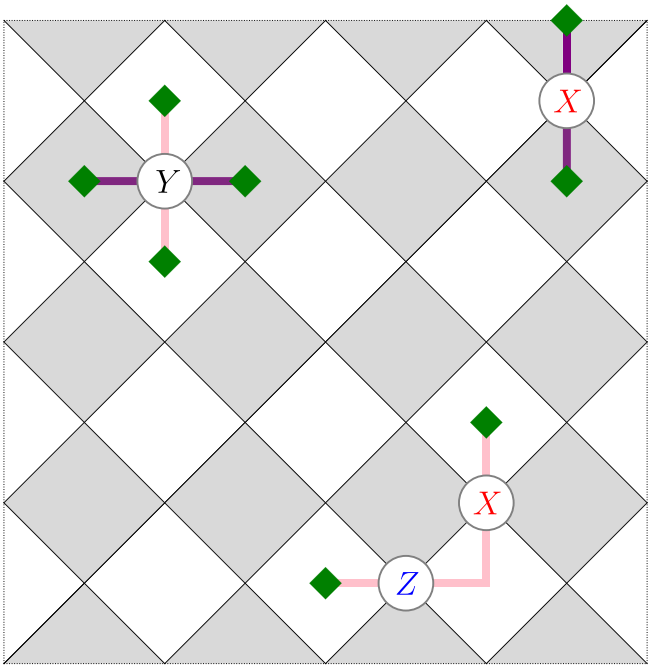

FIG. 2. An example of the error configuration C. Circles indicate data qubits with X, Y, or Z errors; qubits without errors are omitted. The error configuration can be represented as two types of error chains C_{white} and C_{gray}, connecting the white faces and gray faces and drawn with the pink and purple lines on the corresponding decoding graphs $(\mathcal{V}_{\text{white}}, \mathcal{E}_{\text{white}})$ and $(\mathcal{V}_{\text{gray}}, \mathcal{E}_{\text{gray}})$, respectively. The end points of these error chains are captured by the syndrome defects, drawn with the green squares. In both the error chains C_{white} and C_{gray}, the Z- and X-type Pauli components correspond to operators on data qubits located at the centers of horizontal and vertical edges, respectively. A Y error occurs precisely on a data qubit where both the Z- and X-type components are present (i.e., where the corresponding horizontal and vertical edges are both occupied).

In Fig. 2, syndrome defects are drawn with green squares. They capture the end points of C_{white} and C_{gray}. At such end points of horizontal and vertical edges, Z- and X-type error components anticommute with X and Z operators acting on the left/right and up/down qubits in the stabilizer generator G_f, respectively, and hence can be detected.

For an error chain $C_\bullet$, the set of face centers whose ancilla qubits possess syndrome defects is called the *syndrome* and denoted by

$$S_\bullet := \{v_f \in \mathcal{V}_\bullet \,|\, s_f = 1\}. \tag{3}$$

The syndrome-consistency condition can then be expressed as

$$\partial C_{\text{white}} = S_{\text{white}}, \quad \partial C_{\text{gray}} = S_{\text{gray}}. \tag{4}$$

The two error chains C_{white} and C_{gray} contain the same information as the error configuration C. We express the correspondence as $C = (C_{\text{white}}, C_{\text{gray}})$ and then define the tuple $\partial C := (\partial C_{\text{white}}, \partial C_{\text{gray}})$. Similarly, we define the tuple $S := (S_{\text{white}}, S_{\text{gray}})$. Using this abbreviated notation, the syndrome-consistency condition of Eq. (4) reads $\partial C = S$.

The true error configuration $\bar{C}$ is unknown; the only available information is the syndrome S, from which decoders infer a *recovery configuration* C such that $\partial C = S$.

IV. DECODING ALGORITHMS

We elaborate on our SA decoder in Sec. IV A. Subsequently, we describe the MWPM decoder in Sec. IV B. As a simplified version, in Sec. IV C, we propose a greedy-matching decoder, exploited to construct an initial configuration for the SA decoder.

A. Simulated annealing decoder

We assume *the code capacity noise model* where only data qubits are noisy and syndrome measurements are perfect. Specifically, each data qubit suffers an error of X-, Y-, or Z-type, independently and identically distributed with the probabilities p_x, p_y, p_z, respectively. A nonidentity Pauli error occurs with the probability $p := p_x + p_y + p_z$, called the *physical error rate*. The noise acting on each data qubit state ρ is given by the channel

$$\mathcal{E}(\rho) = (1-p)\,\rho + p_x X\rho X + p_y Y\rho Y + p_z Z\rho Z. \tag{5}$$

For an error configuration C, let n_x, n_y, n_z denote the numbers of qubits that suffer X, Y, Z errors, respectively. The probability of C is then given by

$$\begin{aligned}\Pr(C) &:= {p_x}^{n_x}\,{p_y}^{n_y}\,{p_z}^{n_z}\,(1-p)^{N_\mathrm{q}-(n_x+n_y+n_z)} \\ &= (1-p)^{N_\mathrm{q}}\left(\frac{p_x}{1-p}\right)^{n_x}\left(\frac{p_y}{1-p}\right)^{n_y}\left(\frac{p_z}{1-p}\right)^{n_z} \\ &= (1-p)^{N_\mathrm{q}}\,\pi(C),\end{aligned} \tag{6}$$

where

$$\pi(C) := \left(\frac{p_x}{1-p}\right)^{n_x}\left(\frac{p_y}{1-p}\right)^{n_y}\left(\frac{p_z}{1-p}\right)^{n_z} \tag{7}$$

is the (unnormalized) Boltzmann weight of configuration C, i.e., $\Pr(C)$ up to the constant factor $(1-p)^{N_\mathrm{q}}$.

For convenience, we reexpress Eq. (7) using a common base $p/(1-p)$:

$$\pi(C) = \left(\frac{p}{1-p}\right)^{\alpha_x n_x + \alpha_y n_y + \alpha_z n_z}, \tag{8}$$

with

$$\alpha_\mu := \frac{\ln\left(\frac{p_\mu}{1-p}\right)}{\ln\left(\frac{p}{1-p}\right)}, \tag{9}$$

where $\mu \in \{x, y, z\}$. We assume $0 < p < 1/2$, so that $\ln(p/(1-p)) < 0$. Knowing or accurately estimating p_x, p_y, p_z enables minimum-energy decoding for finding a MAP error configuration.

We define the energy of the error configuration C as the weighted sum of the numbers [28]:

$$\mathcal{H}(C) := \alpha_x n_x + \alpha_y n_y + \alpha_z n_z. \tag{10}$$

Using Eq. (10), Eq. (8) is expressed as

$$\pi(C) = \left(\frac{p}{1-p}\right)^{\mathcal{H}(C)} = e^{-\beta_\mathrm{N}\,\mathcal{H}(C)}, \tag{11}$$

where

$$\beta_\mathrm{N} := -\ln\left(\frac{p}{1-p}\right), \quad \text{for } 0 < p < 1/2. \tag{12}$$

We refer to β_{N} as the Nishimori inverse temperature corresponding to the physical error rate p. Conditioned on the syndrome-consistency $\partial C = S$, we consider the posterior $\Pr(C \mid S) \propto \pi(C)$, which follows from Eq. (6).

Given a syndrome S, let

$$\Omega_S := \{C \in \overline{\mathcal{P}} \mid \partial C = S\} \tag{13}$$

denote the set of syndrome-consistent configurations. Fix $R(S) \in \Omega_S$, referred to as a *reference configuration* [29]. Then, any error configuration $C \in \Omega_S$ can be decomposed as

$$C = R(S)\, L_P\, G \tag{14}$$

for some $P \in \{I, X, Y, Z\}$ and some $G \in \mathcal{G}$ [30].

Left-multiplying both sides of Eq. (14) by $R(S)$ and using $R(S)^2 = I$ (up to phase) gives

$$R(S)\, C = L_P\, G. \tag{15}$$

The product $R(S)\, C$ has a trivial syndrome [i.e., $\partial(R(S)\, C) = \emptyset$], so its equivalence class $[R(S)\, C]$ is well defined (as in Sec. II) and equals one of $[L_P]$ with $P \in \{I, X, Y, Z\}$. We then define, for each $P \in \{I, X, Y, Z\}$, a coset

$$\mathcal{C}_{R(S)}(L_P) := \{C \in \Omega_S \mid [R(S)\, C] = [L_P]\}. \tag{16}$$

Note that $[L_P]$ is an equivalence class in $\overline{\mathcal{P}}/\mathcal{G}$, whereas $\mathcal{C}_{R(S)}(L_P)$ is a set of concrete configurations $C \in \Omega_S$; these objects live in different spaces. When $S = \emptyset$ and we choose $R(S) = I^{\otimes N_{\mathrm{q}}}$, for each $P \in \{I, X, Y, Z\}$, $\mathcal{C}_{R(S)}(L_P)$ corresponds to the logical operator class $[L_P]$.

The four cosets $\{\mathcal{C}_{R(S)}(L_P)\}_{P \in \{I,X,Y,Z\}}$ form a partition of Ω_S:

$$\Omega_S = \mathcal{C}_{R(S)}(L_I) \,\dot\cup\, \mathcal{C}_{R(S)}(L_X) \,\dot\cup\, \mathcal{C}_{R(S)}(L_Y) \,\dot\cup\, \mathcal{C}_{R(S)}(L_Z),$$

where $\dot\cup$ denotes a disjoint union. Using Eqs. (14) and (16), we obtain the explicit description

$$\mathcal{C}_{R(S)}(L_P) = \{R(S)\, L_P\, G \mid G \in \mathcal{G}\}. \tag{17}$$

We define the maximum (unnormalized) Boltzmann weight within each coset as

$$\pi_{L_P}^{\max} := \max_{C \in \mathcal{C}_{R(S)}(L_P)} \pi(C). \tag{18}$$

A configuration C that attains the maximum in Eq. (18) is the most likely configuration in the coset $\mathcal{C}_{R(S)}(L_P)$ under the posterior conditioned on $\partial C = S$. Using Eq. (11), Eq. (18) is expressed as

$$\pi_{L_P}^{\max} = e^{-\beta_{\mathrm{N}} E_{L_P}^{\min}}, \tag{19}$$

where

$$E_{L_P}^{\min} := \min_{C \in \mathcal{C}_{R(S)}(L_P)} \mathcal{H}(C) = \min_{G \in \mathcal{G}} \mathcal{H}(R(S)\, L_P\, G), \tag{20}$$

which follows from Eq. (17).

We determine P^* by maximizing $\pi_{L_P}^{\max}$ (equivalently, minimizing $E_{L_P}^{\min}$):

$$P^* := \operatorname*{arg\,max}_{P \in \{I,X,Y,Z\}} \pi_{L_P}^{\max} = \operatorname*{arg\,min}_{P \in \{I,X,Y,Z\}} E_{L_P}^{\min}, \tag{21}$$

which follows from Eq. (19). This decoding formulation is based on domain-wall constructions in spin-glass models [[31], Sec. 3.5] [32,33].

Ideally, one would perform maximum-likelihood (ML) decoding over logical operator cosets by comparing the coset weights. We instead use the MAP: We approximate each coset weight $\sum_{C \in \mathcal{C}_{R(S)}(L_P)} \pi(C)$ by its largest term $\pi_{L_P}^{\max}$ defined in Eq. (19). This reduces ML decoding over cosets to a minimum-energy search within each coset.

ALGORITHM 1. Decoder using simulated annealing.

Input: S : measured syndrome
Output: C : recovery configuration, i.e., $\partial C = S$

1: **function** `SA_decoder`(S)
2: $\quad R \leftarrow$ `greedy_matching`(S)
3: $\quad$ **for all** $P \in \{I, X, Y, Z\}$ **do**
4: $\quad\quad C_{L_P} := R\,L_P$ ▷ *Initial configuration for SA*
5: $\quad\quad$ **for** $i = 1$ to N_{SA} **do**
6: $\quad\quad\quad E_{L_P}^{(i)} \leftarrow$ `simulated_annealing`(`copy`(C_{L_P}))
7: $\quad\quad E_{L_P} := \min_{i=1,\ldots,N_{\mathrm{SA}}} E_{L_P}^{(i)}$ ▷ *estimate of* $E_{L_P}^{\min}$
8: $\quad P^* := \arg\min_{P\in\{I,X,Y,Z\}} E_{L_P}$ ▷ *Eq.* (21)
9: $\quad$ **return** $C_{L_{P^*}}$ ▷ *defined as* $R\,L_{P^*}$ *in Line 4*

For each $P \in \{I, X, Y, Z\}$, to estimate $E_{L_P}^{\min}$ in Eq. (20), we employ SA [21]. For our SA, in the decomposition of Eq. (14), $C_{L_P} := R(S)\,L_P$ is set as an initial configuration for SA, and the stabilizer component G is updated by the Metropolis method to search for the ground state energy $E_{L_P}^{\min}$.

In most of our simulations, we take $R(S)$ to be the recovery configuration returned by our greedy-matching decoder (Sec. IV C). Although it has been rigorously proved that SA converges to the optimal solution (under suitable cooling schedules) in the infinite-time limit [34], regardless of the initial configuration, in practical finite time, the initial configuration can significantly affect the convergence speed of SA. We examine this effect in Sec. V C.

We give our SA decoder in Algorithm 1. To estimate $E_{L_P}^{\min}$ effectively, we run simulated annealing multiple times (say, N_{SA} times) starting from the identical error configuration C_{L_P} (Lines 5 and 6 of Algorithm 1), and select the minimum energy across the runs (Line 7). In Line 9, we output the recovery configuration $C_{L_{P^*}} = R\,L_{P^*}$ [35].

Note that, in the nested **for** loops in Lines 3 and 5 in Algorithm 1, each iteration is an independent SA task and, hence, these tasks are trivially parallelizable. Implementing this parallelization (and quantifying the overhead) is left for future work.

Algorithm 1 invokes the simulated annealing, shown in Algorithm 2. Note that the `simulated_annealing` function is called with a copy of the initial configuration. Algorithm 2 performs the Metropolis method, given in Algorithm 3.

In Line 6 of Algorithm 3, we update the configuration C by applying a randomly chosen stabilizer generator G_f [36]. This update keeps the syndrome S invariant.

For SA, we introduce a tempered family of unnormalized weights

$$\pi_\beta(C) := e^{-\beta\,\mathcal{H}(C)}, \tag{22}$$

where $\beta > 0$ is treated as a tunable inverse temperature. In particular, the unnormalized Boltzmann weight corresponding to the physical error rate is recovered at $\beta = \beta_{\mathrm{N}}$, i.e., $\pi(C) = \pi_{\beta_{\mathrm{N}}}(C)$, which follows from Eq. (11). Therefore, β_{N} provides a natural fixed-temperature choice for MCMC-based decoding [33,37,38].

In Algorithm 2, to accelerate convergence of SA, we employ a logarithmic schedule for the inverse temperature, increasing β from $\beta_{\mathrm{init}} := 0.9\,\beta_{\mathrm{N}}$ to $\beta_{\mathrm{final}} := \beta_{\mathrm{N}}$:

$$\beta_i := \beta_{\mathrm{init}}\,(1 + \gamma\,\ln i), \quad \text{for } i = 1, \ldots, N_\beta, \tag{23}$$

ALGORITHM 2. Simulated annealing.

Input: C : error configuration
Output: $E_{\min}$: an estimate of the minimum energy encountered during annealing

function `simulated_annealing`(C)
  $E \leftarrow \mathcal{H}(C)$ ▷ *Eq.* (10)
  $E_{\min} \leftarrow E$
  for $i = 1$ to N_β **do**
    $\beta \leftarrow \beta_{\text{init}}\,(1 + \gamma \log i)$ ▷ *Eq.* (23)
    repeat $N_{\text{stabilizer}}$ times **do** ▷ *one Monte Carlo sweep*
      $\Delta E \leftarrow$ `Metropolis_step`$(\beta,\ C)$
      $E \leftarrow E + \Delta E$
      $E_{\min} \leftarrow \min\{E_{\min},\ E\}$ ▷ *Update min energy*
  return $E_{\min}$

with

$$\gamma := \begin{cases} \left(\frac{\beta_{\text{final}}}{\beta_{\text{init}}} - 1\right) \Big/ \ln N_\beta, & \text{for } N_\beta \geqslant 2, \\ 0, & \text{for } N_\beta \in \{0, 1\}. \end{cases} \tag{24}$$

For each inverse temperature β, we execute the Metropolis step $N_{\text{stabilizer}}$ times, which is counted as one Monte Carlo sweep, according to the convention.

B. The MWPM decoder

1. The MWPM decoding algorithm

We briefly review the algorithm of the MWPM decoder for the XZZX code [9].

The MWPM decoder is formulated on the decoding graph $(\mathcal{V}_\bullet, \mathcal{E}_\bullet)$, introduced in Sec. III. We index the vertices by integer coordinates as

$$\mathcal{V}_{\text{white}} = \{(i, j) \mid i \in \{1, \ldots, d\},\ j \in \{1, \ldots, d-1\}\},$$
$$\mathcal{V}_{\text{gray}} = \{(i, j) \mid i \in \{1, \ldots, d-1\},\ j \in \{1, \ldots, d\}\},$$

ALGORITHM 3. Metropolis method that applies a stabilizer generator.

Input: β : inverse temperature
  C : error configuration (modified in place)
Output: Energy difference of error configurations

function `Metropolis_step`$(\beta,\ C)$
  Select a stabilizer generator G_f uniformly at random.
  $\Delta E \leftarrow \mathcal{H}(C\,G_f) - \mathcal{H}(C)$
  Draw $r \sim \text{Uniform}(0, 1)$.
  if $r < \min\{1, \exp(-\beta\,\Delta E)\}$ **then**
    $C \leftarrow C\,G_f$ ▷ *Applies G_f (by bit flips)*
    return ΔE
  else
    return 0

where i and j denote the horizontal and vertical coordinates, respectively, and (1, 1) is the lower-left corner.

For vertices $u, v \in S_\bullet$, we define the weight $w(u, v)$ between $u = (u_x, u_y)$ and $v = (v_x, v_y)$:

$$w(u, v) := \min\{w_{\text{int}}(u, v),\ w_{\text{bnd}}(u, v)\}. \tag{25}$$

Here, $w_{\text{int}}(u, v)$ is the weighted Manhattan distance [9]

$$w_{\text{int}}(u, v) := \tilde{\alpha}_z |u_x - v_x| + \tilde{\alpha}_x |u_y - v_y|, \tag{26}$$

with

$$\tilde{\alpha}_\mu := -\ln\left(\frac{\tilde{p}_\mu}{1-p}\right), \quad \text{for } \mu \in \{x, z\}, \tag{27}$$

where we defined the probabilities $\tilde{p}_x := p_x + p_y$ and $\tilde{p}_z := p_z + p_y$. Note that MWPM is invariant under multiplying all edge weights by a positive constant. Therefore, we may freely rescale the weights without changing the matching solution.

The alternative weight $w_{\text{bnd}}(u, v)$ is the weight of connecting both vertices to the nearest boundary [39]:

(1) For $u, v \in S_{\text{white}}$, we define

$$w_{\text{bnd}}(u, v) := \tilde{\alpha}_x(\min\{\text{dist}_{\text{B}}(u), \text{dist}_{\text{T}}(u)\} + \min\{\text{dist}_{\text{B}}(v), \text{dist}_{\text{T}}(v)\}), \tag{28}$$

where $\text{dist}_{\text{B}}(u) := u_y$ and $\text{dist}_{\text{T}}(u) := d - u_y$ are the vertical distances from the vertex u to the bottom and top boundaries, respectively, and similarly for v.

(2) For $u, v \in S_{\text{gray}}$, we define

$$w_{\text{bnd}}(u, v) := \tilde{\alpha}_z(\min\{\text{dist}_{\text{L}}(u), \text{dist}_{\text{R}}(u)\} + \min\{\text{dist}_{\text{L}}(v), \text{dist}_{\text{R}}(v)\}), \tag{29}$$

where $\text{dist}_{\text{L}}(u) := u_x$ and $\text{dist}_{\text{R}}(u) := d - u_x$ are the horizontal distances from the vertex u to the left and right boundaries, respectively, and similarly for v.

We present the algorithm of the MWPM decoder in Algorithm 4. In Line 2, if the number of vertices $|S_\bullet|$ is odd, a *virtual boundary vertex* b is appended to the syndrome $S_\bullet$ so that a perfect matching exists (i.e., $|S_\bullet|$ becomes even) [39]. In this case, we extend the weight function $w(\cdot, \cdot)$ by defining, for $u \in S_\bullet$,

$$w(u, b) := \begin{cases} \tilde{\alpha}_x \min\{\text{dist}_{\text{B}}(u), \text{dist}_{\text{T}}(u)\}, & \text{for } \bullet = \text{white}, \\ \tilde{\alpha}_z \min\{\text{dist}_{\text{L}}(u), \text{dist}_{\text{R}}(u)\}, & \text{for } \bullet = \text{gray}. \end{cases} \tag{30}$$

Note that $w_{\text{bnd}}(u, v) = w(u, b) + w(v, b)$, corresponding to pairing both the defects u, v with the boundary b.

In Line 4, we create the set of (unordered) pairs

$$K := \{\{u, v\} \mid u, v \in S_\bullet,\ u \neq v\}, \tag{31}$$

which are all possible candidates for pairing. In Line 5, the `MWPM_graph` function finds an MWPM [40], i.e., a perfect matching $L \subset K$ such that the total distance $\sum_{\{u,v\} \in L} w(u, v)$ is minimum. For our implementation, we employ the C++ library, Blossom V [41].

In Line 8, for each matched pair $\{u, v\} \in L$, we connect the vertices u and v with a path that consists of edges, i.e., applying Z and X operators to the qubits located at the centers of horizontal and vertical edges, respectively. This path has weight $w(u, v)$, i.e., it is chosen to attain the distance defined by Eqs. (26) and (28)–(30). The set of these edges forms a recovery chain $C_\bullet$.

The `MWPM` function invokes the `MWPM_chain` function twice, i.e., for the syndromes S_{white} and S_{gray} on the white and gray faces, respectively. Note that only syndromes on faces of the same color may be connected.

ALGORITHM 4. The MWPM decoder.

Input: $S_\bullet$: Measured syndrome with $\bullet \in \{\text{white}, \text{gray}\}$
Output: $C_\bullet$: Recovery chain, i.e., $\partial C_\bullet = S_\bullet$
function `MWPM_chain`$(S_\bullet)$
  if $|S_\bullet|$ is odd **then**
    Append the virtual boundary vertex b to $S_\bullet$
  $K \coloneqq \{\{u, v\} \mid u, v \in S_\bullet,\ u \neq v\}$ ▷ *Eq.* (31)
  $L \leftarrow$ `MWPM_graph`(K) ▷ *Utilize Blossom V*
  $C_\bullet \leftarrow \emptyset$
  for all $\{u, v\} \in L$ **do**
    Add to $C_\bullet$ the edges along a path connecting u and v with the weight $w(u, v)$.
  return $C_\bullet$

Input: $S \coloneqq (S_{\text{white}}, S_{\text{gray}})$: Measured syndrome
Output: C : Recovery configuration, i.e., $\partial C = S$
function `MWPM`(S)
  $C_{\text{white}} \leftarrow$ `MWPM_chain`(S_{white})
  $C_{\text{gray}} \leftarrow$ `MWPM_chain`(S_{gray})
  $C \coloneqq (C_{\text{white}}, C_{\text{gray}})$
  return C

Implementing the MWPM decoder efficiently on hardware such as field-programmable gate arrays (FPGAs) is challenging due to its algorithmic complexity and memory/communication requirements. Thus, we propose a simple variant algorithm in Sec. IV C.

2. The MWPM decoder versus our SA decoder as a minimization problem

Given a perfect matching $L \subset K$, we define its total weight as

$$w(L) \coloneqq \sum_{\{u,v\} \in L} w(u, v). \tag{32}$$

The MWPM decoder finds a minimum-weight perfect matching L^*:

$$L^* \in \arg\min_L \{w(L) \,|\, L \text{ is a perfect matching on } S_\bullet\}.$$

From a perfect matching L, we can construct a recovery chain $C_\bullet(L)$, i.e., $\partial C_\bullet(L) = S_\bullet$, by connecting each matched pair $\{u, v\} \in L$ with a path of weight $w(u, v)$ [42]. Since the weighted distance $w(u, v)$ is defined by Eqs. (26) and (28)–(30), Eq. (32) is expressed as

$$w(L) = \tilde{\alpha}_z\, n_{\text{hor}} + \tilde{\alpha}_x\, n_{\text{ver}}, \tag{33}$$

where n_{hor} and n_{ver} are the numbers of occupied horizontal and vertical edges in the error chain $C_\bullet(L)$, i.e., the numbers of Z- and X-type components, respectively. A Y error contributes to both n_{hor} and n_{ver}.

We now compare the total distance in Eq. (33) and the energy defined in Eq. (10). In the MWPM approach for the XZZX code, one solves two separate matching problems on the white and gray decoding graphs. This effectively decouples the inference of the X- and Z-type components and does not model the fact that a Y error simultaneously contributes to both components on the same data

ALGORITHM 5. The greedy-matching decoder.

Input: $S_\bullet$: Measured syndrome with $\bullet \in \{\text{white}, \text{gray}\}$
Output: $C_\bullet$: Recovery chain, i.e., $\partial C_\bullet = S_\bullet$

```
1: function greedy_matching_chain(S_•)
2:     if |S_•| is odd then
3:         Append the virtual boundary vertex b to S_•
4:     K ≔ {{u, v} | u, v ∈ S_•, u ≠ v}                ▷ Eq. (31)
5:     C_• ← ∅

6:     while K is not empty do
7:         Pick a pair {u, v} ∈ K such that w(u, v) is minimum.
8:         Add to C_• the edges along a path connecting u and v
                              with the weight w(u, v).
9:         Remove from K all pairs {x, y} such that x ∈ {u, v}
                                  or y ∈ {u, v}.

10:    return C_•
```

Input: $S \coloneqq (S_{\text{white}}, S_{\text{gray}})$: Measured syndrome
Output: C : Recovery configuration, i.e., $\partial C = S$

```
1: function greedy_matching(S)
2:     C_white ← greedy_matching_chain(S_white)
3:     C_gray ← greedy_matching_chain(S_gray)
4:     C ≔ (C_white, C_gray)

5:     return C
```

qubit; hence, it cannot capture Y-induced correlations. Consequently, when p_y is non-negligible, this MWPM approach is expected to be suboptimal compared to our SA decoder, which explicitly incorporates p_y via the Y-error weight (through α_y) in the energy $\mathcal{H}(C)$ in Eq. (10).

C. The greedy-matching decoder

As described in Sec. IV B 1, the MWPM decoder utilizes the MWPM graph algorithm, i.e., the perfect matching giving the minimum of the total distance. By replacing the global minimization of MWPM with a local greedy-matching rule, we obtain a simple and low computational cost decoder. We employ the greedy-matching decoding algorithm to construct a reference configuration $R(S)$, set as an initial configuration for our SA decoder in Algorithm 1.

We give our greedy-matching decoding algorithm in Algorithm 5. In Line 7, the `greedy_matching_chain` function matches the closest pair $\{u, v\} \in K$ with respect to $w(u, v)$, and connects the vertices u and v. Line 9 removes all pairs that contain u or v from the candidate list K of vertex pairs. This algorithm thus enables us to monotonically reduce the size of the list K.

We efficiently implement K using a *priority queue* [[43], Sec. 6.5]. In the priority queue with m elements (e.g., a binary heap), each of insertion and extract-min operation takes $\mathcal{O}(\log m)$; therefore, performing the operations for m elements has time complexity $\mathcal{O}(m \log m)$. For our greedy-matching decoder, the size of the priority queue is $m = \mathcal{O}(|S_\bullet|^2)$, where $|S_\bullet|$ denotes the number of syndrome defects. Thus, the worst-case time complexity is $\mathcal{O}(|S_\bullet|^2 \log |S_\bullet|)$ for each color. Since $|S_\bullet| \leqslant |\mathcal{V}_\bullet| = \mathcal{O}(d^2)$, this yields the worst-case bound $\mathcal{O}(d^4 \log d)$.

Like the MWPM decoder, our greedy-matching decoder employs the weighted distance $w(u, v)$, which is calculated from p_z and p_x, and therefore cannot take p_y into account.

ALGORITHM 6. The greedy-matching decoder that may produce a different reference configuration.

Input: $S_\bullet$: Measured syndrome with $\bullet \in \{\text{white}, \text{gray}\}$
Output: $C_\bullet$: Recovery chain, i.e., $\partial C_\bullet = S_\bullet$

function `greedy_matching_diff_chain`($S_\bullet$)
  if $|S_\bullet|$ is odd **then**
    Append the virtual boundary vertex b to $S_\bullet$
  $K \coloneqq \{\{u, v\} \,|\, u, v \in S_\bullet,\ u \neq v\}$ ▷ *Eq.* (31)
  $C_\bullet \leftarrow \emptyset$

  while K is not empty **do**
    $w_{\min} \coloneqq \min\{w(u, v) \,|\, \{u, v\} \in K\}$
    $K_{\min} \coloneqq \{\{u, v\} \in K \,|\, w(u, v) = w_{\min}\}$ ▷ *Eq.* (34)
    while $K_{\min}$ is not empty **do**
      Pick a pair $\{u, v\} \in K_{\min}$ uniformly at random.
      Add to $C_\bullet$ the edges along a path connecting u and v with the weight $w(u, v)$.
      Remove from K and $K_{\min}$ all pairs $\{x, y\}$ such that $x \in \{u, v\}$ or $y \in \{u, v\}$.

  return $C_\bullet$

Input: $S \coloneqq (S_{\text{white}}, S_{\text{gray}})$: Measured syndrome
Output: C : Recovery configuration, i.e., $\partial C = S$

function `greedy_matching_different`(S)
  $C_{\text{white}} \leftarrow$ `greedy_matching_diff_chain`(S_{white})
  $C_{\text{gray}} \leftarrow$ `greedy_matching_diff_chain`(S_{gray})
  $C \coloneqq (C_{\text{white}}, C_{\text{gray}})$

  return C

D. Our SA decoder that starts with different reference configurations

In Sec. IV A, we described our SA decoder in Algorithm 1, which performs SA multiple times to achieve energy as low as possible.

To improve efficiency, rather than using an identical reference configuration repeatedly, it is preferable to use a different reference configuration for each SA run. To this end, we modify our greedy-matching algorithm by randomizing the tie breaking among equal-weight pairs. Specifically, we randomize tie breaking among equal-weight pairs using pseudorandom numbers. This modified algorithm is presented in Algorithm 6. In Algorithm 6, we form the subset

$$K_{\min} \coloneqq \{\{u, v\} \in K \ \mid \ w(u, v) = w_{\min}\}, \tag{34}$$

i.e., the set of candidate pairs that attain the current minimum weight $w_{\min}$.

This newly introduced randomness may produce a different reference configuration for every invocation, while still satisfying $\partial C = S$. However, two such reference configurations can differ by both a stabilizer component G and a logical operator component L_P in the decomposition of Eq. (14). Therefore, to compare energies across runs consistently, we relabel the logical operator cosets obtained in each run as those defined with respect to a fixed "standard" reference configuration.

We present a second SA decoder that utilizes Algorithm 6 in Algorithm 7. In Line 2, we construct a reference configuration $R_1(S) \in \Omega_S$, which we use as the fixed standard reference.

ALGORITHM 7. Decoder using simulated annealing that may start with different reference configurations.

Input: S : Measured syndrome
Output: C : Recovery configuration, i.e., $\partial C = S$

1: **function** `SA_decoder_different_reference`(S)
2: $R_1 \leftarrow$ `greedy_matching_different`(S) ▷ *standard reference configuration*
3: **for all** $P \in \{I, X, Y, Z\}$ **do**
4: $C^{(1)}_{L_P} \coloneqq R_1 L_P$ ▷ *standard reference*
5: $E^{(1)}_{L_P} \leftarrow$ `simulated_annealing`(`copy`$(C^{(1)}_{L_P})$)
6: **for** $i = 2$ to N_{SA} **do**
7: $R_i \leftarrow$ `greedy_matching_different`(S)
8: Determine $Q \in \{I, X, Y, Z\}$ such that $[L_Q] = [R_i\, R_1]$. ▷ *e.g., via commutation with L_X and L_Z*
9: **for all** $P \in \{I, X, Y, Z\}$ **do**
10: $C^{(i)}_{L_P} \coloneqq R_i L_P$ ▷ *Initial configuration for SA*
11: $E^{(i)}_{L_{QP}} \leftarrow$ `simulated_annealing`$(C^{(i)}_{L_P})$
12: **for all** $P \in \{I, X, Y, Z\}$ **do**
13: $E_{L_P} \coloneqq \min_{i=1,\ldots,N_{\mathrm{SA}}} E^{(i)}_{L_P}$ ▷ *estimate of $E^{\min}_{L_P}$*
14: $P^* \coloneqq \operatorname*{arg\,min}_{P\in\{I,X,Y,Z\}} E_{L_P}$ ▷ *Eq. (21)*
15: **return** $C^{(1)}_{L_{P^*}}$ ▷ *defined in Line 4*

Let $R_i(S) \in \Omega_S$ ($i \geqslant 2$) denote the reference configuration produced in the ith run. Using Eq. (14), $R_i(S)$ can be decomposed as

$$R_i(S) = R_1(S)\, L_Q\, G' \tag{35}$$

for some $Q \in \{I, X, Y, Z\}$ and some $G' \in \mathcal{G}$. Since $\partial(R_i(S)\, R_1(S)) = \emptyset$, the equivalence class $[R_i(S)\, R_1(S)] \in \overline{\mathcal{P}}/\mathcal{G}$ is well defined and we obtain $[R_i(S)\, R_1(S)] = [L_Q]$. The Pauli operator Q can be determined by checking its commutation relations with the logical operators L_X and L_Z.

Since stabilizers commute with logical operators, Eq. (35) implies the correspondence

$$\mathcal{C}_{R_i(S)}(L_P) = \mathcal{C}_{R_1(S)}(L_{QP}), \tag{36}$$

where QP denotes the Pauli multiplication in $\{I, X, Y, Z\}$ (up to global phase), consistent with our convention of ignoring global phases. Equation (36) means that "the coset of L_P with respect to $R_i(S)$" equals "the coset of L_{QP} with respect to $R_1(S)$." Thus, for each $P \in \{I, X, Y, Z\}$, we store the calculated energy in $E^{(i)}_{L_{QP}}$ in Line 11.

Note that, in the **for** loops in Lines 6 and 9 in Algorithm 7, each iteration is an independent task, and hence these tasks can trivially be parallelized. In addition, Line 8 can be parallelized with Lines 10 and 11.

V. NUMERICAL RESULTS

We perform Monte Carlo simulations of our SA decoder under the setting described in Sec. V A. In Sec. V B, we evaluate the logical error rate of our SA decoder. In Sec. V C, we examine

the validity of our greedy-matching decoder used to construct initial configurations. We compare elapsed times of our SA decoder with other decoding algorithms in Sec. V D.

A. Simulation setting

In Monte Carlo simulations, we generate samples of true error configurations $\bar{C}$ by means of pseudorandom numbers, drawn from the physical error rate p and the error bias $p_x : p_y : p_z$ we specify. Performing the stabilizer measurement on the true error configurations $\bar{C}$ yields the syndrome S. Then, we execute each decoder to obtain the recovery configuration C. If the product $\bar{C}\,C$ belongs to a nontrivial logical operator class, i.e., if $[\bar{C}\,C] \in \{[L_X], [L_Y], [L_Z]\}$, then the logical error is said to occur. By repeating the aforementioned procedure N_{sample} times, we estimate the logical error rate P_{L}. Except in Sec. V D, we employ $N_{\text{sample}} = 10^6$ to ensure sufficient statistical precision. Except for the convergence study in Sec. V C, we refer to Algorithm 7 as *our SA decoder* and employ the number of inverse temperatures $N_\beta = 100$. The parameters N_β and N_{SA} are chosen empirically in this work and fixed in advance for each set of simulations, without using an adaptive stopping criterion.

Our SA decoder is applicable to arbitrary biased Pauli noise provided that the decoder is supplied with the error probabilities p_x, p_y, p_z used to define $\mathcal{H}(C)$. We focus on Y-biased noise as a stress test for the MWPM decoder, which neglects Y-induced correlations. For simplicity, we consider the symmetric case $p_x = p_z$. Under the condition $p_x = p_z$, we can write the bias as $p_x : p_y : p_z = 1 : \eta : 1$, where η denotes the Y-biased factor. Since the depolarizing noise is the case where $p_x = p_y = p_z = p/3$, it corresponds to $p_x : p_y : p_z = 1 : 1 : 1$, i.e., $\eta = 1$.

B. Logical error rates

Let us compare the decoding accuracy of our SA decoder described in Algorithm 7 with that of a decoder based on IBM ILOG CPLEX Optimizer (CPLEX) described in the Appendix. Under the dominant-term approximation, the CPLEX decoder directly finds the minimum-energy error configuration consistent with the measured syndrome, whereas the SA decoder minimizes the energy within each logical coset and then selects the coset with the lowest minimum energy. Thus, for both decoders, the decoding criterion is based on minimum-energy configurations, whereas full maximum-likelihood decoding compares the total weights of the logical cosets.

First, we examine how the logical error rate varies with the code distance d.

Figure 3 depicts the logical error rates P_{L} under the depolarizing noise for the code distances $d = 5, 7, 9$. In Fig. 3, our SA decoder achieves logical error rates comparable to those of the CPLEX decoder within statistical uncertainty for all d shown.

For both the SA and CPLEX decoders, from the crossing of the finite-distance curves, we estimate a finite-size threshold $p_{\text{th}} \simeq 0.18$. This estimate is close to the previously reported value $p_{\text{th}} = 0.187$ obtained using an MPS decoder [9]. Below the threshold p_{th}, the logical error rate decreases with increasing d, as expected.

Figure 4 shows the logical error rates under the Y-biased noise where $p_x : p_y : p_z = 1 : 5 : 1$. In Fig. 4, for every d, our SA decoder achieves logical error rates comparable to those of the CPLEX decoder. For both SA and CPLEX decoders, the intersection point of these curves lies around $p \simeq 0.2$, which we consider to be a finite-size threshold estimate. Below this threshold, the logical error rate decreases as d increases.

Figure 5 shows the logical error rates under the Y-biased noise where $p_x : p_y : p_z = 1 : 10 : 1$. In Fig. 5, no intersection point for the CPLEX decoder is observed for $d = 5, 7, 9$; hence, we expect that the threshold for the CPLEX decoder lies beyond the plotted range, i.e., $p_{\text{th}} > 0.2$. On the other hand, for $d = 9$ in Fig. 5, our SA decoder does not reach the logical error rate obtained by the CPLEX decoder. The lack of a clear crossing for $d = 5, 7, 9$ may be related to slow mixing of the MCMC method. For our SA decoder, the curves for $d = 7, 9$ intersect around $p = 0.11$, which

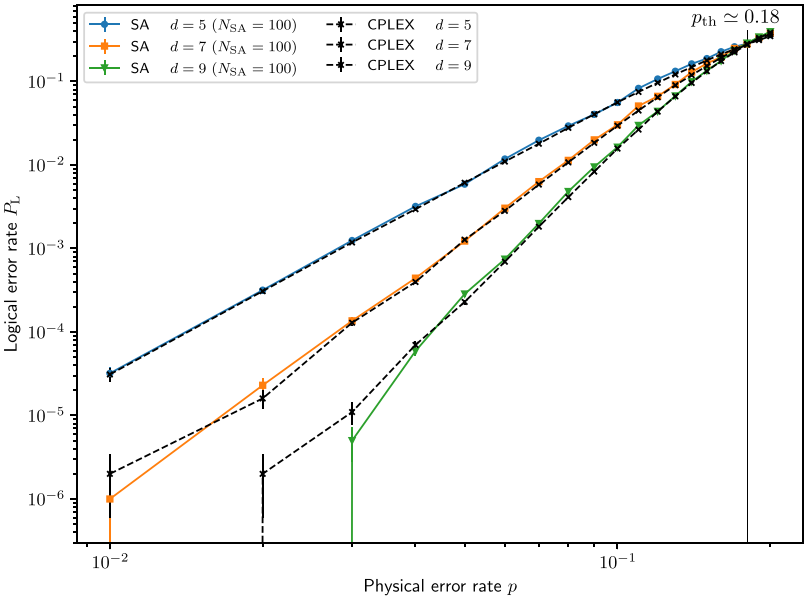

FIG. 3. Logical error rates under the depolarizing noise for the code distances $d = 5, 7, 9$. Our SA decoder achieves logical error rates comparable to those of the CPLEX decoder for all d. We employ $N_{\mathrm{SA}} = 100$, which suffices to reach the logical error rates obtained by the CPLEX decoder. The error bars indicate one standard deviation.

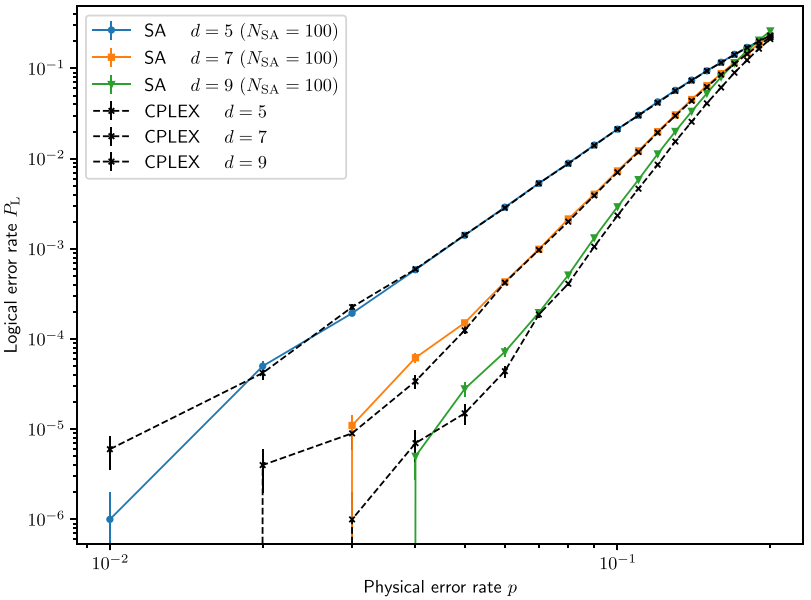

FIG. 4. Logical error rates under the Y-biased noise where $p_x : p_y : p_z = 1 : 5 : 1$ for the code distances $d = 5, 7, 9$. Our SA decoder achieves logical error rates comparable to those of the CPLEX decoder. We employ $N_{\mathrm{SA}} = 100$, which suffices to reach the logical error rates obtained by the CPLEX decoder. The error bars indicate one standard deviation.

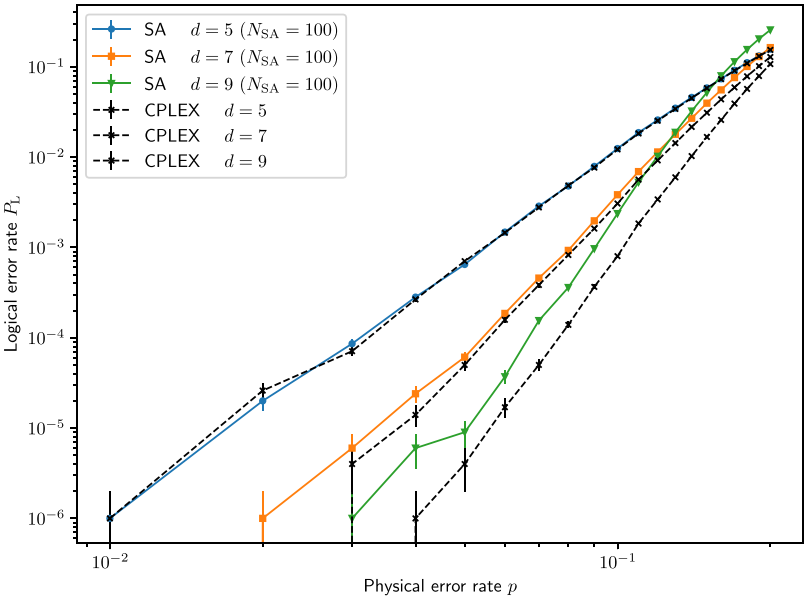

FIG. 5. Logical error rates under the Y-biased noise where $p_x : p_y : p_z = 1 : 10 : 1$ for the code distances $d = 5, 7, 9$. Our SA decoder does not reach the logical error rate obtained by the CPLEX decoder for $d = 9$, even with $N_{\mathrm{SA}} = 100$. The error bars indicate one standard deviation.

we take as an estimated finite-size threshold. When $p < 0.11$, the logical error rate decreases as d increases.

Next, we investigate the dependence of the logical error rates P_{L} on the number of SA runs, N_{SA} for fixed code distance $d = 9$. For comparison, we also calculate the rates by our greedy-matching

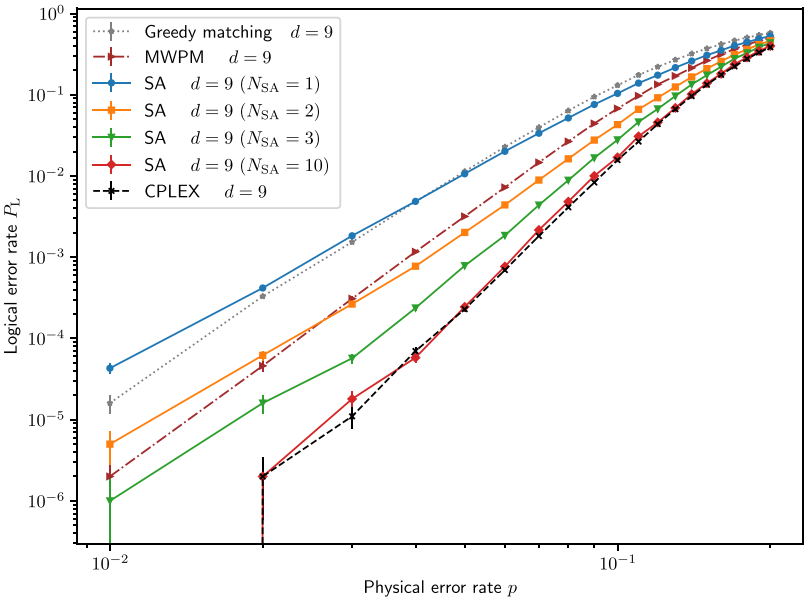

FIG. 6. Logical error rates under the depolarizing noise and $N_{\mathrm{SA}} = 1, \ldots, 10$ for the code distance $d = 9$. The error bars indicate one standard deviation.

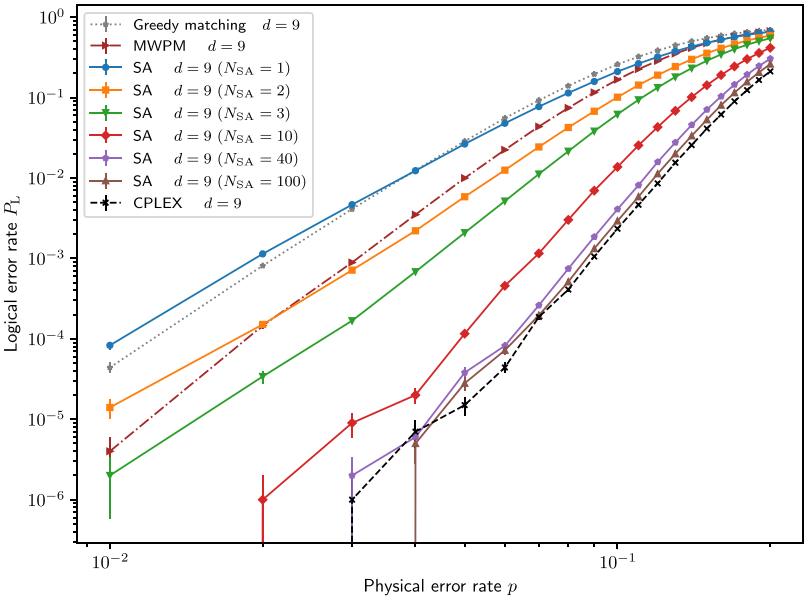

FIG. 7. Logical error rates under Y-biased noise where $p_x : p_y : p_z = 1 : 5 : 1$ and $N_{\mathrm{SA}} = 1, \ldots, 100$ for the code distance $d = 9$. The error bars indicate one standard deviation.

decoder, i.e., the `greedy_matching` function described in Algorithm 5 and the MWPM decoder, i.e., the `MWPM` function described in Algorithm 4.

Figure 6 shows the logical error rates under the depolarizing noise. Figures 7 and 8 show the logical error rates under Y-biased noises where $p_x : p_y : p_z = 1 : 5 : 1$ and $1 : 10 : 1$ for several values of N_{SA}, respectively.

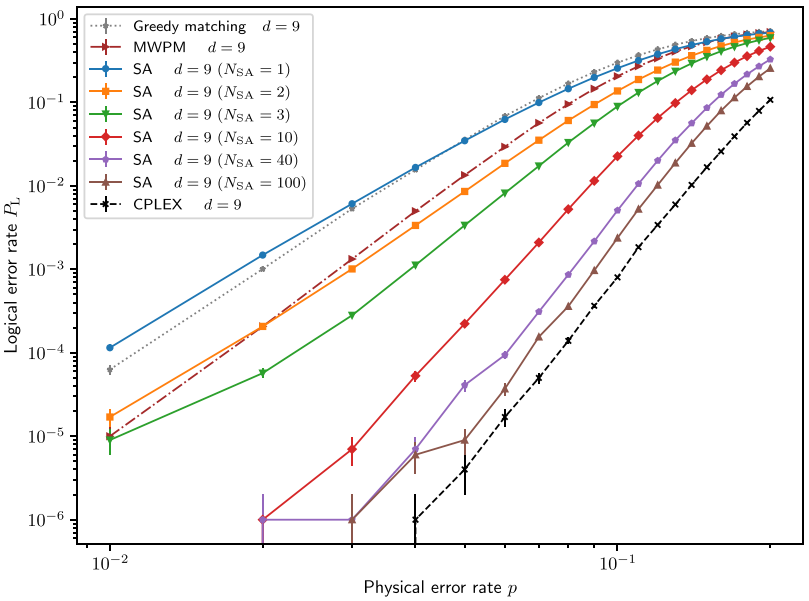

FIG. 8. Logical error rates under Y-biased noise where $p_x : p_y : p_z = 1 : 10 : 1$ and $N_{\mathrm{SA}} = 1, \ldots, 100$ for the code distance $d = 9$. The error bars indicate one standard deviation.

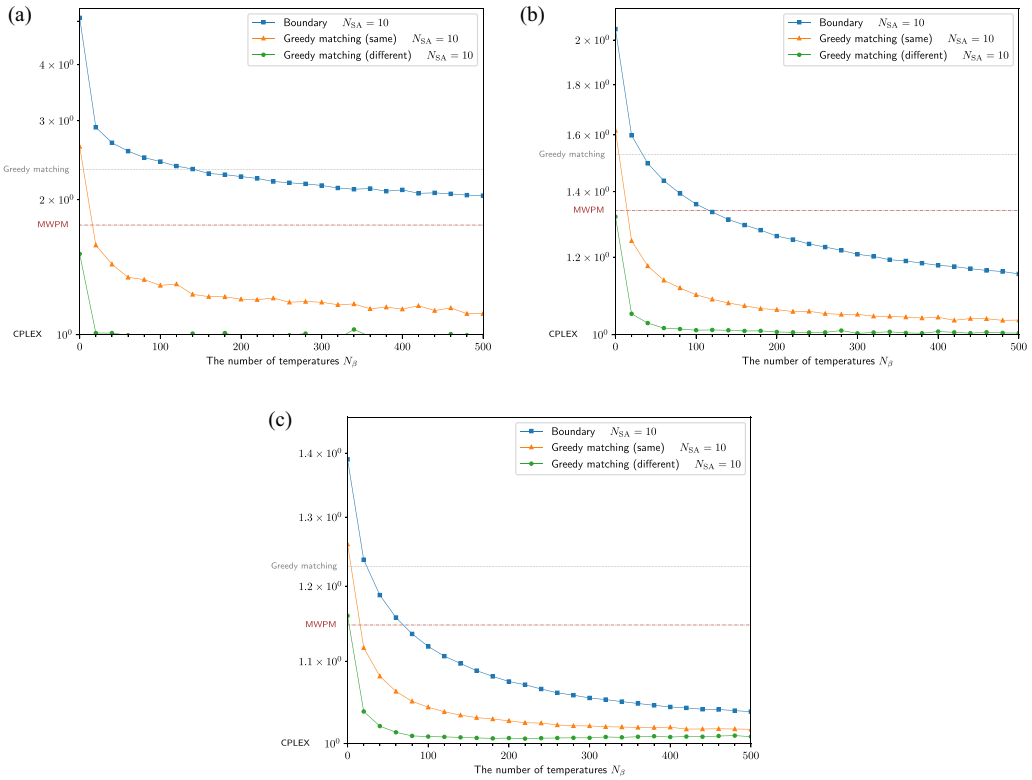

FIG. 9. Convergence of our SA decoder for different construction methods of reference configurations under the depolarizing noise. The horizontal axis is the number of inverse temperatures N_β, and the vertical axis is the ratio R_L. We employ $d = 9$ and $N_{SA} = 10$. (a) $p = 0.04$, (b) $p = 0.1$, and (c) $p = 0.15$. Dotted gray and dash-dotted brown horizontal lines show the ratios obtained by the greedy-matching and MWPM decoders, respectively.

In Figs. 6–8, we observe that the logical error rate P_L of our SA decoder decreases monotonically as N_{SA} increases. For $N_{SA} \geqslant 2$, our SA decoder achieves logical error rates lower than our greedy-matching decoder and is therefore an improvement over our greedy matching. For $N_{SA} \geqslant 3$, our SA decoder achieves a lower logical error rate than the MWPM decoder.

To reach the accuracy of the CPLEX decoder under the depolarizing noise in Fig. 6, $N_{SA} = 10$ is sufficient. For the noise $p_x : p_y : p_z = 1 : 5 : 1$ in Fig. 7, $N_{SA} = 100$ is almost sufficient. Under the noise $p_x : p_y : p_z = 1 : 10 : 1$ in Fig. 8, $N_{SA} = 100$ is not sufficient. Hence, we observe that the more strongly Y-biased the noise is, the larger N_{SA} our SA decoder needs to approach the logical error rate obtained by the CPLEX decoder.

C. Effect of initial error configuration on SA performance

To construct a reference configuration $R(S)$, we consider the following three methods:

(1) *Boundary*: Each syndrome defect is connected to the nearer of the two boundaries (top or bottom for white faces; left or right for gray faces). We call this reference configuration construction routine, instead of the `greedy_matching` function in Algorithm 1.

(2) *Greedy matching (same)*: We use Algorithm 5 as the initializer in Algorithm 1; it returns the same reference configuration on every call.

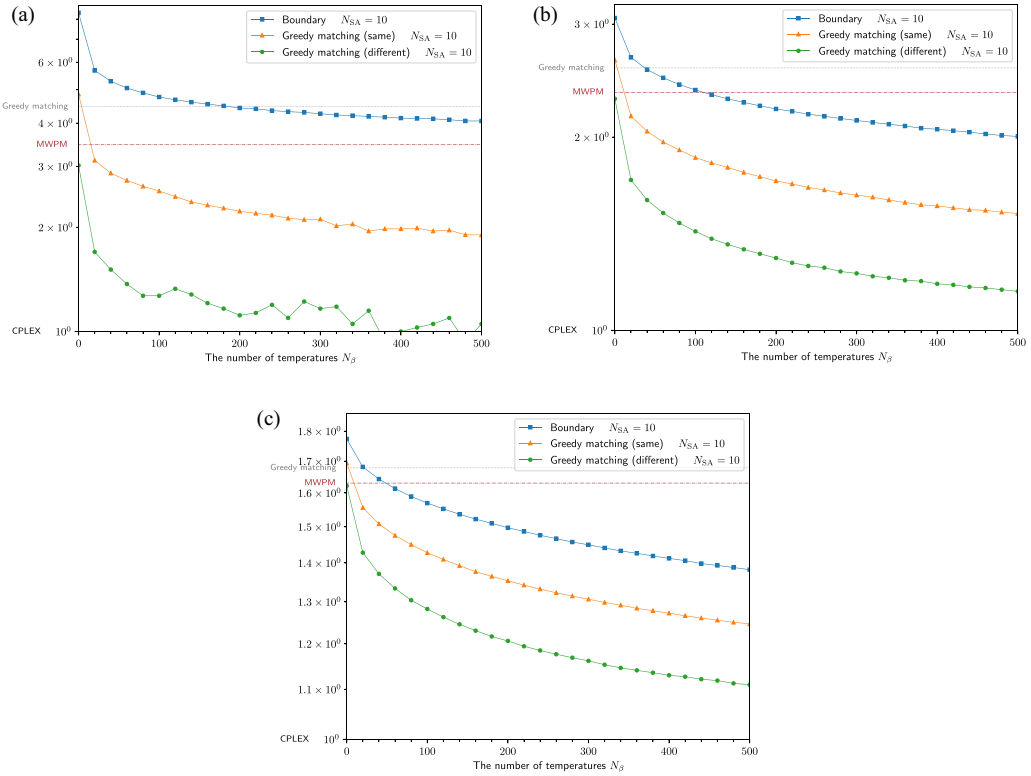

FIG. 10. Convergence of our SA decoder for different construction methods of reference configurations under the Y-biased noise such that $p_x : p_y : p_z = 1 : 5 : 1$. The horizontal axis is the number of inverse temperatures N_β, and the vertical axis is the ratio R_{L}. We employ $d = 9$ and $N_{\mathrm{SA}} = 10$. (a) $p = 0.04$, (b) $p = 0.1$, and (c) $p = 0.15$. Dotted gray and dash-dotted brown horizontal lines show the ratios obtained by the greedy-matching and MWPM decoders, respectively.

(3) *Greedy matching (different)*: It is Algorithm 6 invoked in the SA decoder of Algorithm 7, described in Sec. IV D. For every invocation, it may produce a different reference configuration. We use this method in this paper, unless otherwise stated.

To compare the effect of these methods on the convergence of our SA decoder, we evaluate the ratio of logical error rates

$$R_{\mathrm{L}} := \left(\frac{P_{\mathrm{L}}^{(\mathrm{SA})}}{P_{\mathrm{L}}^{(\mathrm{CPLEX})}} \right)^{\frac{2}{d+1}}, \tag{37}$$

where $P_{\mathrm{L}}^{(\mathrm{SA})}$ and $P_{\mathrm{L}}^{(\mathrm{CPLEX})}$ are the logical error rates obtained by the SA and CPLEX decoders, respectively. Here, the exponent $2/(d+1)$ is introduced to cancel out d dependence. A value $R_{\mathrm{L}} = 1$ indicates that the SA decoder matches the CPLEX benchmark. Values above 1 indicate worse performance than CPLEX.

Figure 9 shows the ratio R_{L} for varying the number of inverse temperatures N_β under the depolarizing noise. Similarly, Figs. 10 and 11 show the ratio R_{L} under the Y-biased noises where $p_x : p_y : p_z = 1 : 5 : 1$ and $1 : 10 : 1$, respectively. In Figs. 9–11, the ratio values R_{L} obtained by the greedy-matching and MWPM decoders are drawn by dotted gray and dash-dotted brown horizontal lines, respectively.

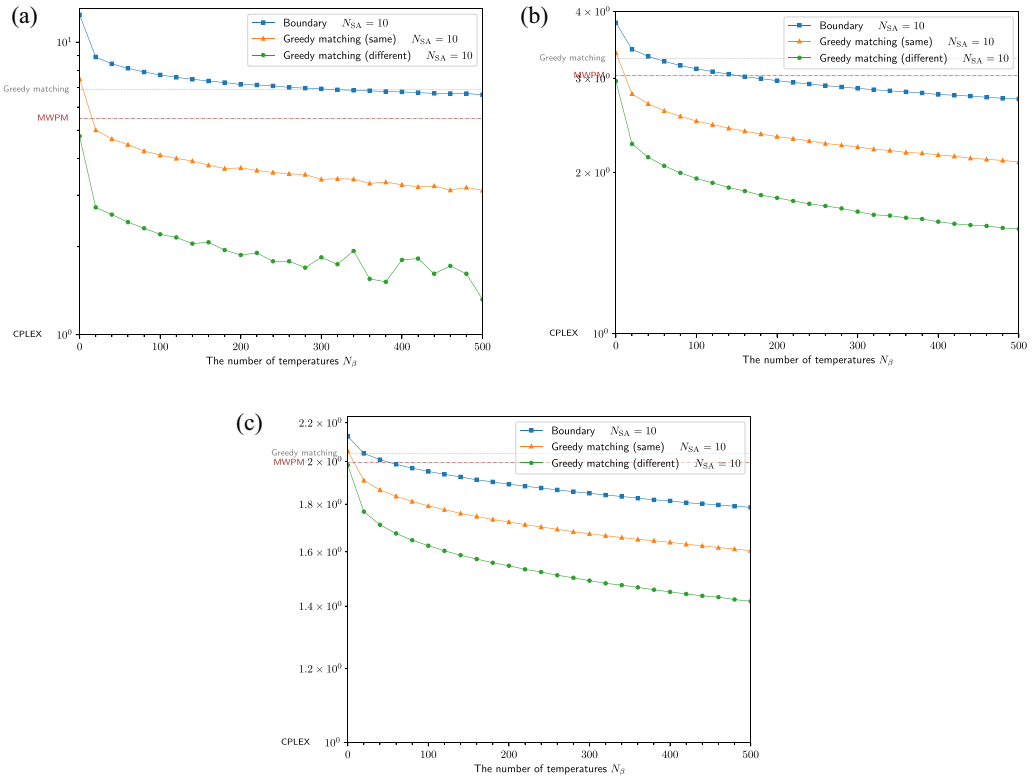

FIG. 11. Convergence of our SA decoder for different construction methods of reference configurations under the Y-biased noise such that $p_x : p_y : p_z = 1 : 10 : 1$. The horizontal axis is the number of inverse temperatures N_β, and the vertical axis is the ratio R_{L}. We employ $d = 9$ and $N_{\mathrm{SA}} = 10$. (a) $p = 0.04$, (b) $p = 0.1$, and (c) $p = 0.15$. Dotted gray and dash-dotted brown horizontal lines show the ratios obtained by the greedy-matching and MWPM decoders, respectively.

In Figs. 9–11, under all the noises and for all the reference configuration construction methods, we see that the ratio R_{L} broadly decreases to the value 1 as N_β increases. For $N_\beta \geqslant 20$, the "Greedy matching (same)" and "Greedy matching (different)" methods outperform the MWPM decoder.

For the "Greedy matching (same)" and "Greedy matching (different)," when $N_\beta = 0$, the `simulated_annealing` function performs no Metropolis updates, so the decoder compares only the energies of the initial configurations. We refer to these variants as "initial-configuration-only" variants. In all the cases (a)–(c) of Figs. 9–11, the initial-configuration-only variant of the "Greedy matching (different)" method achieves higher accuracy than the MWPM decoder.

In Figs. 9–11, for all plotted values of N_β and for all noise biases, "Greedy matching (different)" consistently yields the lowest R_{L} among the three methods, followed by "Greedy matching (same)," while the "Boundary" method performs worst. We attribute these accuracy differences to the following factors:

(1) The "Boundary" method ignores the error probabilities p_x, p_y, and p_z, whereas "Greedy matching (same)" incorporates p_x and p_z through the weights $\tilde{\alpha}_x$ and $\tilde{\alpha}_z$. As a result, "Greedy matching (same)" yields a bias-informed starting configuration.

(2) The "Greedy matching (different)" method randomizes the tie breaking among equal-distance pairs, generating diverse initial configurations across runs. This diversity increases the chance of reaching lower-energy configurations and thereby reduces the logical error rate.

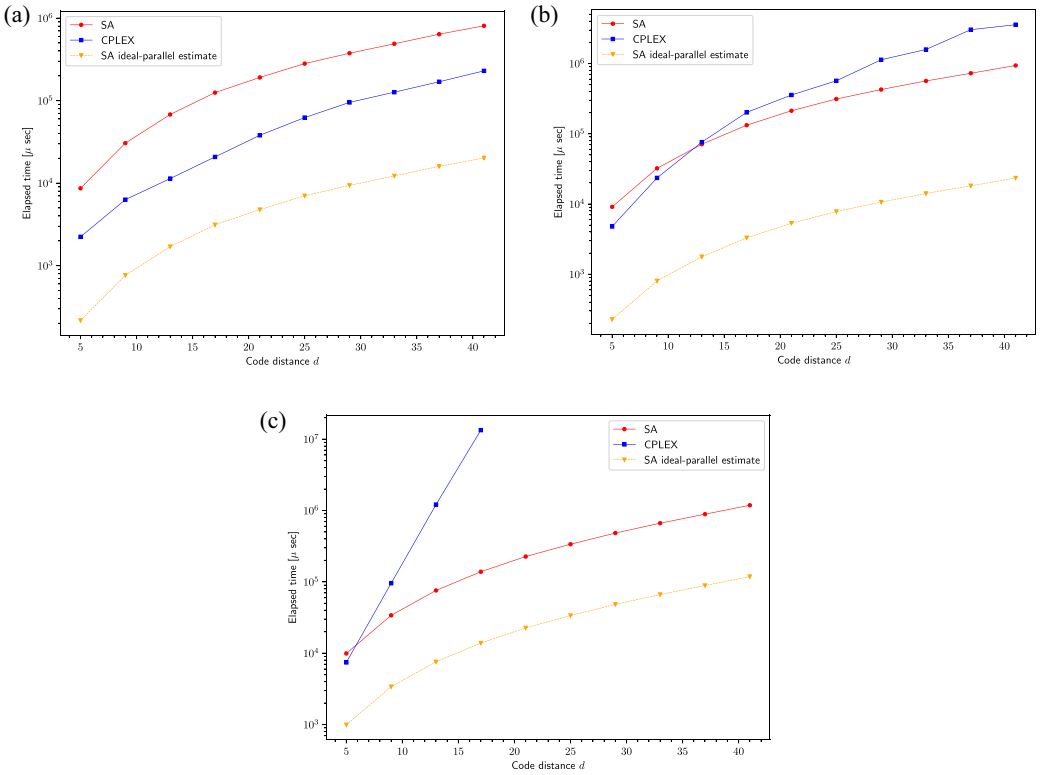

FIG. 12. Elapsed times for the SA and CPLEX decoders under the depolarizing noise. The vertical axis is on a logarithmic scale. The orange dashed line shows an idealized parallel-run-time estimate obtained by dividing the sequential SA time by $4N_{\mathrm{SA}}$; it is not a measured parallel run-time. (a) $p = 0.02$, (b) $p = 0.1$, and (c) $p = 0.15$. In panel (c), the CPLEX decoder exhibits exponential scaling for the code distance d.

D. Comparison of decoding time with other decoding algorithms

We now evaluate the decoding time of our SA decoder.

First, Fig. 12 shows the elapsed times for the SA and CPLEX decoders for various code distances d under the depolarizing noise. Similarly, Figs. 13 and 14 show the elapsed times under Y-biased noises where $p_x : p_y : p_z = 1 : 5 : 1$ and $1 : 10 : 1$, respectively. For measuring the elapsed times, we execute the SA and CPLEX decoders without any parallelization. From Figs. 12–14, we see that the larger d is, the longer both the SA and CPLEX decoders take.

For each decoder, we examine the dependence on d and p. For all the physical error rates p, our SA decoder takes approximately the same time, because the iteration count is determined solely by N_β and d and does not depend on the syndrome weight $|S|$. Recalling that for each inverse temperature, we perform the Metropolis step $\mathcal{O}(N_{\mathrm{stabilizer}}) = \mathcal{O}(d^2)$ times, the total elapsed time scales accordingly. On the other hand, the CPLEX decoder exhibits exponential scaling in computation time with respect to the code distance d. This exponential behavior is clearly visible for case (c) $p = 0.15$ in Figs. 12–14, where it is prohibitive to calculate for $d > 11$.

Let us now compare the elapsed times of our SA decoder and the CPLEX decoder. For $p = 0.15$, our SA decoder is faster than the CPLEX decoder in the measured sequential implementation. For $p = 0.02$ and $p = 0.1$, the measured sequential SA run-time is longer than the CPLEX run-time. This indicates that improving the sequential run-time in the low-p regime remains an important task.

In each of Figs. 12–14, the orange dashed line labeled "SA ideal-parallel estimate" is an idealized parallel-run-time estimate obtained by dividing the sequential SA time by $4N_{\mathrm{SA}} = 40$. This curve

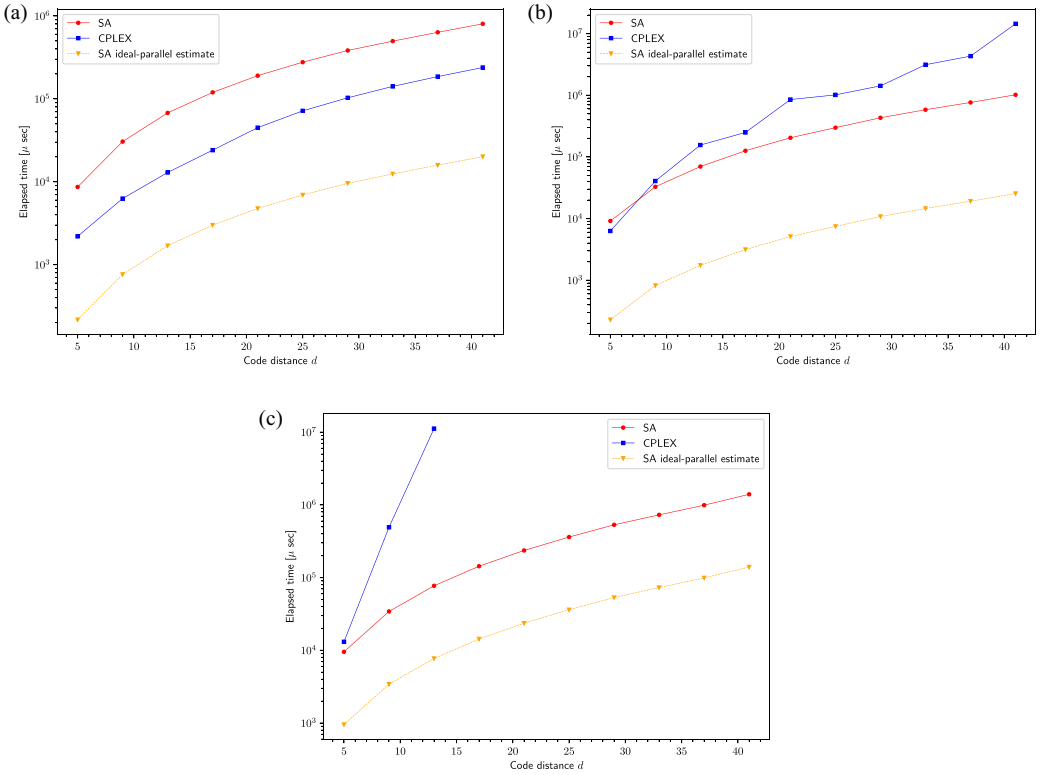

FIG. 13. Elapsed times for the SA and CPLEX decoders under the Y-biased noise where $p_x : p_y : p_z = 1 : 5 : 1$. The vertical axis is on a logarithmic scale. The orange dashed line shows an idealized parallel-run-time estimate obtained by dividing the sequential SA time by $4N_{\mathrm{SA}}$; it is not a measured parallel run-time. (a) $p = 0.02$, (b) $p = 0.1$, and (c) $p = 0.15$. In panel (c), the CPLEX decoder exhibits exponential scaling for the code distance d.

is not a measured parallel run-time; it assumes perfect parallel efficiency, with parallel overhead neglected. In these figures, comparison of this estimate with the CPLEX time suggests possible run-time competitiveness under these idealized assumptions.

Next, we present the elapsed times for various decoding algorithms in Fig. 15. For our SA decoder, we set the number of inverse temperatures to $N_\beta = 100$ and the number of SA runs to $N_{\mathrm{SA}} = 10$ in Algorithm 7. Note that increasing N_{SA} increases the total cost roughly linearly, since each SA run has a similar per-run cost.

For the MPS decoder, we use the CSS surface code implementation in Ref. [44] with truncation dimension $\chi = 8$, because the XZZX-specific implementation used in Ref. [9] is not publicly available. Accordingly, we do not use this implementation for an accuracy comparison, and its timing in Fig. 15 is included only as an indicative run-time reference rather than as a controlled benchmark against our SA implementation. In particular, the MPS timing is obtained from a Python implementation for the CSS surface code, whereas our SA decoder is implemented in C++. For fixed χ, the MPS decoder scales as $\mathcal{O}(N_{\mathrm{q}}) = \mathcal{O}(d^2)$, up to polynomial factors in χ [8,17]. For fixed N_β and N_{SA}, our SA decoder also scales as $\mathcal{O}(d^2)$ in d, although constant factors, implementation details, and parameter dependencies differ. A controlled comparison with an XZZX-specific MPS implementation is left for future work.

Our SA decoder labeled by "SA" is slower than the CPLEX decoder in this sequential implementation. Nevertheless, the corresponding "SA ideal-parallel estimate," obtained by dividing the

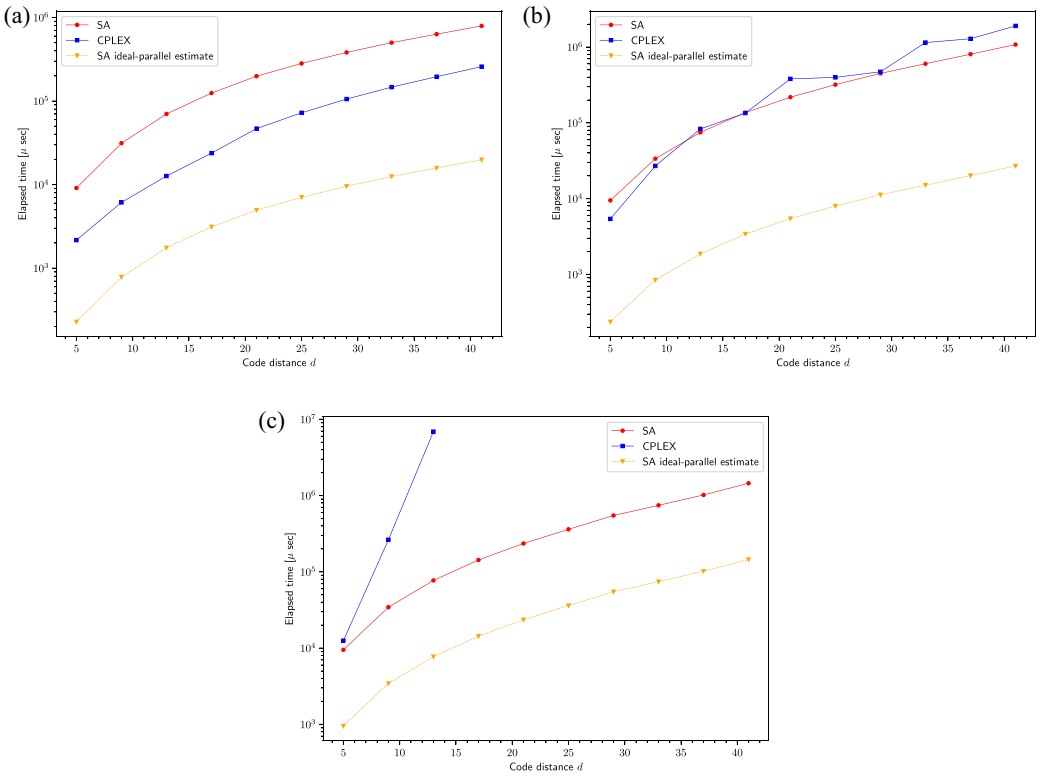

FIG. 14. Elapsed times for the SA and CPLEX decoders under the Y-biased noise where $p_x : p_y : p_z = 1 : 10 : 1$. The vertical axis is on a logarithmic scale. The orange dashed line shows an idealized parallel-run-time estimate obtained by dividing the sequential SA time by $4N_{\mathrm{SA}}$; it is not a measured parallel run-time. (a) $p = 0.02$, (b) $p = 0.1$, and (c) $p = 0.15$. In panel (c), the CPLEX decoder exhibits exponential scaling for the code distance d.

"SA" time by $4N_{\mathrm{SA}}$, suggests possible run-time competitiveness with the CPLEX decoder under the assumption of perfect scaling. This estimate is idealized and should not be regarded as a measured parallel run-time. Quantifying the achievable parallel efficiency and overhead is left for future work.

From Fig. 15, we observe that our greedy-matching decoder is faster than the MWPM decoder. This result is consistent with the fact that the worst-case computational complexity of our greedy-matching decoder is $\mathcal{O}(|S_\bullet|^2 \log |S_\bullet|^2) = \mathcal{O}(d^4 \log d)$, as discussed in Sec. IV C, which is lower than the worst-case complexity of the MWPM decoder $\mathcal{O}(|S_\bullet|^3 \log |S_\bullet|) = \mathcal{O}(d^6 \log d)$ in Ref. [41,45]. We emphasize that practical MWPM decoders often show much better average-case scaling [e.g., $\mathcal{O}(d^2)$ reported in Ref. [39]], and a similar empirical study for our greedy-matching decoder is left for future work.

VI. DISCUSSION

As in Sec. V C, under much more strongly Y-biased noise, our SA decoder requires a large N_{SA} in order to approach the logical error rate obtained by the CPLEX decoder. A possible remedy is to modify the definition of stabilizer generators, analogously to the tailored code for the CSS surface code [8]. For example, in regimes where p_y dominates (e.g., $p_y \geqslant p_z \geqslant p_x$), one could consider an "XYYX" variant whose stabilizer generators include Y operators. This modified code can detect the end points of error chains consisting of Z errors on horizontal edges and Y errors on vertical edges.

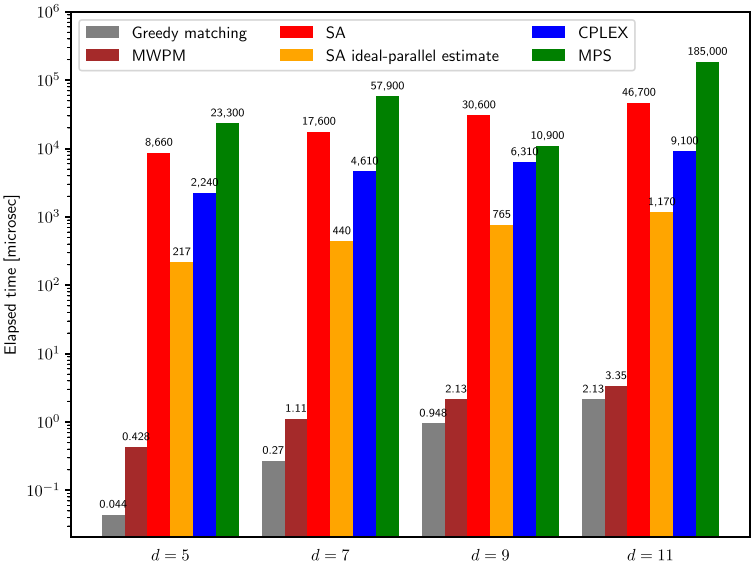

FIG. 15. Elapsed times (microseconds) of various decoding algorithms. Each value is an average over N_{sample} true error configurations, where $N_{\text{sample}} = 10^6$ for the greedy-matching and MWPM decoders, and $N_{\text{sample}} = 100$ for the SA, CPLEX, and MPS decoders due to their higher per-instance cost. We employ the depolarizing noise with the physical error rate $p = 0.02$. Here, the label "SA" indicates "Greedy matching (different)" type with the parameters $N_\beta = 100$ and $N_{\text{SA}} = 10$. The label "SA ideal-parallel estimate" denotes an idealized parallel-run-time estimate obtained by dividing the sequential SA time by $4N_{\text{SA}} = 40$; it is not a measured parallel run-time. The MPS timing is shown only as an indicative run-time reference, not as a controlled run-time benchmark, because the MPS timing was obtained using a publicly available Python implementation of an MPS decoder for the CSS surface code, rather than using an XZZX-specific implementation. The "Greedy matching" and "MWPM" correspond to Algorithms 5 and 4, respectively.

Thus, in our greedy-matching algorithm, we can take into account the probabilities p_z and p_y. We therefore expect to obtain a reference configuration, closely resembling the true error configuration. By employing this better initial configuration, our SA decoder is expected to converge more quickly toward the logical error rate obtained by the CPLEX decoder. Such a treatment can be applied to accommodate the given noise bias.

A key advantage of our SA decoder is that it provides explicit control over the trade-off between computational cost and accuracy by adjusting N_β. That is, increasing the number of inverse temperatures N_β typically improves the decoding accuracy, but also increases the computational cost. In contrast, the cost of deterministic algorithms such as the MWPM, CPLEX, and MPS decoders cannot be easily adjusted, and their parallelization is neither straightforward nor efficient.

VII. CONCLUSION

We have formulated and demonstrated our SA decoder for the XZZX code. As an initial configuration for SA, we have proposed to employ recovery chains obtained by our decoder that utilizes a greedy-matching graph algorithm.

We assumed the code capacity noise model. Under this model, we performed numerical simulations under depolarizing noise and Y-biased Pauli noise.

We observed that our SA decoder is more accurate than the MWPM decoder, which cannot take into account the probability of Y errors. Moreover, under depolarizing noise and Y-biased noise with $p_x : p_y : p_z = 1 : 5 : 1$, our SA decoder achieves logical error rates P_{L} comparable to

those of the CPLEX decoder for all code distances $d = 5, 7, 9$. Here, the CPLEX decoder denotes the decoder that exactly minimizes the energy $\mathcal{H}(C)$ defined in Eq. (10) over error configurations consistent with the measured syndrome. It should therefore be distinguished from full maximum-likelihood decoding over logical cosets. However, for the strongly Y-biased case $p_x : p_y : p_z = 1 : 10 : 1$ at $d = 9$, even $N_{\mathrm{SA}} = 100$ did not attain CPLEX-level accuracy. This indicates that further improvements are needed to achieve CPLEX-level accuracy in this strongly Y-biased case.

We examined the convergence of the logical error rate for the number of inverse temperatures N_β we employ for our SA decoder. In particular, our SA decoder improves upon the initial configuration obtained from our greedy-matching decoder. Furthermore, we confirmed that the convergence is strongly influenced by methods of constructing a reference configuration, which are employed to prepare initial configurations. Especially, we proposed the "Greedy matching (different)" method, which may produce a different reference configuration for every invocation, and observed that it leads to markedly faster convergence.

Subsequently, we compared the elapsed times of various decoding algorithms. In our sequential implementation, the SA decoder is faster than the CPLEX decoder when the physical error rate p is relatively high, whereas it is slower in the lower-p regime. Because the independent SA tasks can be parallelized in principle, the idealized parallel-run-time estimate suggests that the SA decoder could become competitive with the CPLEX decoder. This estimate, however, assumes perfect parallel efficiency, with parallel overhead neglected. An actual parallel implementation and a quantitative study of the parallel overhead remain future work.

We list other topics for future work below. Although our analysis has assumed the code capacity noise model, extensions to phenomenological and circuit-level noise models are important directions for future work.

The decoding time may be reduced by improving the MCMC algorithm. Our SA decoder consists of independent SA processes, making it well-suited for parallelized implementation on FPGA. Alternatively, techniques to accelerate the MCMC method could be employed, such as the replica-exchange Monte Carlo method [46] and cluster algorithms including the Wolff algorithm [47]. Furthermore, the Metropolis algorithm can be parallelized by domain (area) decomposition, which is implementable on FPGA [48].

Our SA decoder allows an explicit trade-off between decoding accuracy and run-time through tunable parameters such as the number of inverse temperatures N_β and the inverse-temperature schedule $\{\beta_i\}_{i=1}^{N_\beta}$. In the present simulations, we did not use an adaptive stopping criterion. We view developing systematic parameter-selection strategies for different code distances d, physical error rates p, and noise biases $p_x : p_y : p_z$, together with adaptive stopping criteria, as an important and nontrivial direction for future work.

An advantage of MCMC-based decoders is their generality: They require only the stabilizer generators and a specification of logical operators, without an explicit decoding graph. This makes them broadly applicable, e.g., to the color code and to quantum low-density parity-check codes [49]. For SA initialization, one may use reference configurations produced not only by our greedy-matching decoder but also by other fast decoders, such as union find [50].

Several decoders have been proposed that explicitly account for correlations induced by Y errors [51–53]. A systematic comparison with these approaches will be an important direction for future work.

Finally, we plan to perform simulations under device-specific Pauli-noise biases $p_x : p_y : p_z$. Toward practical deployment, further speed improvements will be particularly important in the low-p regime, where the measured sequential SA run-time is currently longer than the CPLEX run-time.

ACKNOWLEDGMENTS

We would like to thank Yusaku Takeuchi, Yugo Takada, Mitsuki Katsuda, Hiroshi Ueda, Hideaki Hakoshima, Kosuke Mitarai, Jun Fujisaki, Hirotaka Oshima, Shintaro Sato, and Keisuke Fujii for their helpful discussions. This work was supported by the MEXT Quantum Leap Flagship Program

(MEXT Q-LEAP), Grants No. JPMXS0118067394 and No. JPMXS0120319794; JST Moonshot R&D, Grant No. JPMJMS2061; the JST COI-NEXT project Quantum Software Research Hub (Grant No. JPMJPF2014); and the JST COI-NEXT project Center of Innovation for Sustainable Quantum AI (Grant No. JPMJPF2221).

DATA AVAILABILITY

There are no publicly available research data or software supporting this manuscript. Requests for further information or data should be sent to the authors.

APPENDIX: DECODER FORMULATED AS INTEGER PROGRAMMING PROBLEM

In this Appendix, instead of Eq. (21), we formulate the decoding problem as finding the recovery configuration C, subject to the constraint $\partial C = S$:

$$\arg\min_{C} \{\mathcal{H}(C) \,|\, \partial C = S\}. \tag{A1}$$

The error configuration C can be represented as the set of one-bit binary variables $\{x_i, y_i, z_i\}_i$ with the condition $x_i + y_i + z_i \leqslant 1$, where $i = 1, \ldots, N_{\mathrm{q}}$ indexes the qubits. Namely, $x_i = 1$ indicates that the qubit i suffers an X error, while the variables y_i and z_i similarly represent a Y error and a Z error, respectively. Using this binary representation, the syndrome-consistency condition $\partial C = S$ in Eq. (A1) is explicitly expressed as a system of equations. That is, for each face f,

$$(z_{\mathrm{left}(f)} \oplus y_{\mathrm{left}(f)}) \oplus (z_{\mathrm{right}(f)} \oplus y_{\mathrm{right}(f)}) \oplus (x_{\mathrm{up}(f)} \oplus y_{\mathrm{up}(f)}) \oplus (x_{\mathrm{down}(f)} \oplus y_{\mathrm{down}(f)}) = s_f, \tag{A2}$$

where $s_f \in \{0, 1\}$ is the syndrome measurement outcome determined in Eq. (2), and the operation $\oplus$ denotes xor (addition modulo two) [54]. For boundary faces, where the stabilizer has weight three, the missing-qubit term is omitted from the parity (xor) constraint.

Under the constraint [Eq. (A2)], the goal is to find the solution $\{x_i, y_i, z_i\}_i$ that minimizes the objective function given by Eq. (10). Here, the variables n_x, n_y, and n_z represent the number of X-, Y-, and Z-type errors, respectively, and are given by

$$n_x = \sum_{i=1}^{N_{\mathrm{q}}} x_i, \quad n_y = \sum_{i=1}^{N_{\mathrm{q}}} y_i, \quad n_z = \sum_{i=1}^{N_{\mathrm{q}}} z_i.$$

This solution yields a recovery configuration C.

This problem falls into the category of binary integer programming, known to be NP-hard. The integer programming formulation exhibits exponential computational complexity with respect to both the code distance d and the physical error rate p. Thus, for large values of d or p, it is prohibitive to find a solution within a reasonable amount of time. To solve the integer linear programming problem, we use CPLEX version 22.1.0 [55]; we refer to the resulting decoder as the CPLEX decoder.

The use of integer programming solvers for decoding was originally proposed and demonstrated for the color code [56]. The CPLEX decoder exactly solves the minimum-energy problem in Eq. (A1), whose objective function is the energy $\mathcal{H}(C)$ defined in Eq. (10), which includes the probabilities p_x, p_y, and p_z. The minimizer of this problem is a MAP error configuration among those consistent with the measured syndrome. Thus, this minimum-energy formulation corresponds to the dominant-term approximation described in Sec. IV A rather than to full maximum-likelihood decoding over logical cosets.

[1] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, J. Math. Phys. **43**, 4452 (2002).

[2] 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).
[3] Y. Zhao *et al.*, Realization of an error-correcting surface code with superconducting qubits, Phys. Rev. Lett. **129**, 030501 (2022).
[4] Google Quantum AI, Suppressing quantum errors by scaling a surface code logical qubit, Nature (London) **614**, 676 (2023).
[5] A. D. iOlius, J. E. Martinez, P. Fuentes, P. M. Crespo, and J. Garcia-Frias, Performance of surface codes in realistic quantum hardware, Phys. Rev. A **106**, 062428 (2022).
[6] Google Quantum AI and Collaborators, Quantum error correction below the surface code threshold, Nature (London) **638**, 920 (2025).
[7] 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).
[8] 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).
[9] J. P. Bonilla Ataides, D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, The XZZX surface code, Nat. Commun. **12**, 2172 (2021).
[10] A. R. Calderbank and P. W. Shor, Good quantum error-correcting codes exist, Phys. Rev. A **54**, 1098 (1996).
[11] A. Steane, Multiple-particle interference and quantum error correction, Proc. R. Soc. London, Ser. A **452**, 2551 (1996).
[12] K. Hammar, A. Orekhov, P. W. Hybelius, A. K. Wisakanto, B. Srivastava, A. F. Kockum, and M. Granath, Error-rate-agnostic decoding of topological stabilizer codes, Phys. Rev. A **105**, 042616 (2022).
[13] D. Forlivesi, L. Valentini, and M. Chiani, Logical error rates of XZZX and rotated quantum surface codes, IEEE J. Sel. Areas Commun. **42**, 1808 (2024).
[14] J. Claes, J. E. Bourassa, and S. Puri, Tailored cluster states with high threshold under biased noise, npj Quantum Inf. **9**, 9 (2023).
[15] A. S. Darmawan, B. J. Brown, A. L. Grimsmo, D. K. Tuckett, and S. Puri, Practical quantum error correction with the XZZX code and Kerr-Cat qubits, PRX Quantum **2**, 030345 (2021).
[16] K. Sahay, J. Jin, J. Claes, J. D. Thompson, and S. Puri, High-threshold codes for neutral-atom qubits with biased erasure errors, Phys. Rev. X **13**, 041013 (2023).
[17] S. Bravyi, M. Suchara, and A. Vargo, Efficient algorithms for maximum likelihood decoding in the surface code, Phys. Rev. A **90**, 032326 (2014).
[18] D. K. Tuckett, S. D. Bartlett, and S. T. Flammia, Ultrahigh error threshold for surface codes with biased noise, Phys. Rev. Lett. **120**, 050505 (2018).
[19] J. R. Wootton and D. Loss, High threshold error correction for the surface code, Phys. Rev. Lett. **109**, 160503 (2012).
[20] A. Hutter, J. R. Wootton, and D. Loss, Efficient Markov chain Monte Carlo algorithm for the surface code, Phys. Rev. A **89**, 022326 (2014).
[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, Science **220**, 671 (1983).
[22] Y. Takeuchi, Y. Takada, T. Sakashita, J. Fujisaki, H. Oshima, S. Sato, and K. Fujii, Comparative study of decoding the surface code using simulated annealing under depolarizing noise, arXiv:2311.07973.
[23] Y. Takada, Y. Takeuchi, and K. Fujii, Ising model formulation for highly accurate topological color codes decoding, Phys. Rev. Res. **6**, 013092 (2024).
[24] K. Fujii, M. Negoro, N. Imoto, and M. Kitagawa, Measurement-free topological protection using dissipative feedback, Phys. Rev. X **4**, 041039 (2014).
[25] J. Fujisaki, H. Oshima, S. Sato, and K. Fujii, Practical and scalable decoder for topological quantum error correction with an Ising machine, Phys. Rev. Res. **4**, 043086 (2022).
[26] J. Fujisaki, K. Maruyama, H. Oshima, S. Sato, T. Sakashita, Y. Takeuchi, and K. Fujii, Quantum error correction with an Ising machine under circuit-level noise, Phys. Rev. Res. **5**, 043261 (2023).
[27] S. B. Bravyi and A. Y. Kitaev, Quantum codes on a lattice with boundary, arXiv:quant-ph/9811052.

[28] The numbers n_x, n_y, and n_z are calculated as follows. We formulate and implement the Metropolis method with bit representation taking values in the set {0, 1}, instead of Ising spins taking values in the set {1, −1}. In an error configuration, the error on each data qubit is represented by two bits. The upper and lower bits represent an X and a Z error, respectively. Thus, I, Z, X, and Y errors are represented as 00, 01, 10, and 11, respectively. As such, counting the numbers of qubits that have bit representations 01, 10, and 11 yields n_z, n_x, and n_y, respectively.
[29] Such a fixed $R(S) \in \Omega_S$ is often called a pure error in the literature [31].
[30] D. Poulin, Optimal and efficient decoding of concatenated quantum block codes, Phys. Rev. A **74**, 052333 (2006).
[31] K. Fujii, *Quantum Computation with Topological Codes* (Springer, Singapore, 2015).
[32] K. Hukushima, Domain-wall free energy of spin-glass models: Numerical method and boundary conditions, Phys. Rev. E **60**, 3606 (1999).
[33] S. T. Flammia and C. T. Chubb, Statistical mechanical models for quantum codes with correlated noise, Ann. Inst. Henri Poincaré Comb. Phys. Interact. **8**, 269 (2021).
[34] S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, IEEE Trans. Pattern Anal. Mach. Intell. **PAMI-6**, 721 (1984).
[35] We use SA only to estimate $E_{L_P}^{\min}$; the output recovery configuration is chosen as $R\,L_{P^*}$, since any configuration of the inferred logical coset $\mathcal{C}_{R(S)}(L_P)$ is equivalent up to stabilizers.
[36] We formulate error configurations by bits, instead of Ising spins. Hence, multiplying configurations $C \mapsto C\,G_f$ is implemented as bitwise xor (addition modulo two) on the two-bit representation per qubit.
[37] H. Nishimori, Internal energy, specific heat and correlation function of the bond-random Ising model, Prog. Theor. Phys. **66**, 1169 (1981).
[38] H. Nishimori, *Statistical Physics of Spin Glasses and Information Processing: An Introduction* (Oxford University Press, Oxford, 2001).
[39] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg, Towards practical classical processing for the surface code, Phys. Rev. Lett. **108**, 180501 (2012).
[40] J. Edmonds, Paths, trees, and flowers, Can. J. Math. **17**, 449 (1965).
[41] V. Kolmogorov, Blossom V: A new implementation of a minimum cost perfect matching algorithm, Math. Program. Comput. **1**, 43 (2009).
[42] Note that $C_\bullet(L)$ is not unique in general, due to degeneracies in the minimum-weight paths.
[43] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. (The MIT Press, Cambridge, MA, 2022).
[44] 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.
[45] Note that, in circuit-level noise model, the MWPM decoder is used with the Dijkstra algorithm to find a path with a minimum weight between any possible pair of syndrome defects, whereas in the code capacity noise model, the Dijkstra algorithm is not needed, because the weights are explicitly given by Eqs. (26) and (28)–(30).
[46] K. Hukushima and K. Nemoto, Exchange Monte Carlo method and application to spin glass simulations, J. Phys. Soc. Jpn. **65**, 1604 (1996).
[47] U. Wolff, Collective Monte Carlo updating for spin systems, Phys. Rev. Lett. **62**, 361 (1989).
[48] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, Parallel algorithms for FPGA placement, in *Proceedings of the 10th Great Lakes Symposium on VLSI*, *GLSVLSI '00* (Association for Computing Machinery, New York, NY, 2000), pp. 86–94.
[49] N. P. Breuckmann and J. N. Eberhardt, Quantum low-density parity-check codes, PRX Quantum **2**, 040101 (2021).
[50] N. Delfosse and N. H. Nickerson, Almost-linear time decoding algorithm for topological codes, Quantum **5**, 595 (2021).
[51] Y. Wu and L. Zhong, Fusion Blossom: Fast MWPM Decoders for QEC, in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Piscataway, NJ, 2023), Vol. 01, pp. 928–938.

[52] A. G. Fowler, Optimal complexity correction of correlated errors in the surface code, arXiv:1310.0863.
[53] A. Paler and A. G. Fowler, Pipelined correlated minimum weight perfect matching of the surface code, Quantum **7**, 1205 (2023).
[54] To solve Eq. (A1) with CPLEX, we convert the parity (xor) constraints in Eq. (A2) into an equivalent integer linear programming using auxiliary integer variables and linear constraints.
[55] IBM, CPLEX: Linear programming solver, 2022, https://www.ibm.com/products/ilog-cplex-optimization-studio.
[56] A. J. Landahl, J. T. Anderson, and P. R. Rice, Fault-tolerant quantum computing with color codes, arXiv:1108.5738.