Fast erasure decoder for hypergraph product codes

Nicholas Connolly[1,2], Vivien Londe[3], Anthony Leverrier[1], and Nicolas Delfosse[4]

[1]Inria, Paris, France
[2]Okinawa Institute of Science and Technology Graduate University, Okinawa, Japan
[3]Microsoft, Paris, France
[4]Microsoft Quantum, Redmond, Washington 98052, USA

We propose a decoder for the correction of erasures with hypergraph product codes, which form one of the most popular families of quantum LDPC codes. Our numerical simulations show that this decoder provides a close approximation of the maximum likelihood decoder that can be implemented in $O(N^2)$ bit operations where N is the length of the quantum code. A probabilistic version of this decoder can be implemented in $O(N^{1.5})$ bit operations.

1 Introduction

Due to the high noise rate of quantum hardware, extensive quantum error correction is necessary to scale quantum devices to the regime of practical applications. The surface code [1, 2] is one of the most popular quantum error correcting codes for quantum computing architectures but it comes with an enormous qubit overhead because each qubit must be encoded into hundreds or thousands of physical qubits.

Quantum Low-Density Parity-Check (LDPC) codes [3, 4] such as hypergraph product (HGP) codes [5] promise a significant reduction of this qubit overhead [6, 7]. Numerical simulations with circuit noise show a 15× reduction of the qubit count in the large-scale regime [8]. For applications to quantum fault tolerance, HGP codes must come with a *fast* decoder, whose role is to identify which error occurred. In this work, we propose a fast decoder for the correction of erasures or detectable qubit loss. By replacing erased qubits with uniformly random mixed states, erasure correction can be converted into error correction. This is preferable to the standard error channel because non-erased qubits are assumed to have no errors. Our numerical simulations show that our decoder achieves a logical error rate close to the maximum likelihood (ML) decoder.

Our motivation for focusing on the decoding of erasures is twofold. First it is practically relevant and it is the dominant source of noise in some quantum platforms such as photonic systems [9, 10] for which a photon loss can be interpreted as an erasure. Erasure errors are also present in platforms based on neutral atoms [11], trapped ions [12], superconducting qubits [13], or circuit QED [14]. Second, many of the ideas that led to the design of capacity-achieving classical LDPC codes over binary symmetric channels were first discovered by studying the correction of erasures [15, 16].

2 Classical erasure decoders

A linear code with length n is defined to be the kernel $C = \ker H$ of an $r \times n$ binary matrix H called the *parity-check matrix*. Our goal is to protect a *codeword* $x \in C$ against erasures. We assume that each bit is erased independently with probability p and erased bits are flipped independently with probability $1/2$. The set of erased positions is known and is given by an *erasure vector* $\varepsilon \in \mathbb{Z}_2^n$ such that bit b_i is erased iff $\varepsilon_i = 1$. The initial codeword x is mapped onto a vector $y = x + e \in \mathbb{Z}_2^n$ where e is the indicator vector of the flipped bits of x. In particular the support of e satisfies $\mathrm{supp}(e) \subseteq \mathrm{supp}(\varepsilon)$. To detect e, we compute the *syndrome* $s = Hy = He \in \mathbb{Z}_2^r$. A non-trivial syndrome indicates the presence of bit-flips.

The goal of the decoder is to provide an estimation $\hat{e}$ of e given s and ε and it succeeds if $\hat{e} = e$. This can be done by solving the linear system $H\hat{e} = s$ with the condition $\mathrm{supp}(\hat{e}) \subseteq \mathrm{supp}(\varepsilon)$ thanks to Gaussian elimination. This *Gaussian decoder* runs in $O(n^3)$ bit operations which may

be too slow in practice for large n.

Algorithm 1: Classical peeling decoder

input : An erasure vector $\varepsilon \in \mathbb{Z}_2^N$ and a syndrome $s \in \mathbb{Z}_2^r$.
output: Either **failure** or $\hat{e} \in \mathbb{Z}_2^n$ such that $H\hat{e} = s$ and $\mathrm{supp}(\hat{e}) \subseteq \mathrm{supp}(\varepsilon)$.

```
Set ê = 0.
while there exists a dangling check do
    Select a dangling check c_i.
    Let b_j be the dangling bit incident to c_i.
    if s_i = 1 then
        Flip the j-th bit of ê.
        Flip s_k for all checks c_k incident with b_j.
    Set ε_j = 0.
if ε ≠ 0 return Failure, else return ê.
```

The classical peeling decoder [17], described in Algorithm 1, provides a fast alternative to the Gaussian decoder. Unlike a maximum likelihood decoder, for which logical errors are the only source of failures, the additional possibility of a decoder failure is a major drawback for the peeling decoder. However, because peeling decoder failures are infrequent for sparse codes, this algorithm gives a much more efficient linear-complexity decoding algorithm for LDPC codes without a large increase in the failure rate.

To describe this decoder, it is convenient to introduce the *Tanner graph*, denoted $T(H)$, of the linear code $C = \ker H$. It is the bipartite graph with one vertex $c_1, \ldots, c_r$ for each row of H and one vertex $b_1, \ldots, b_n$ for each column of H such that c_i and b_j are connected iff $H_{i,j} = 1$. We refer to c_i as a *check node* and b_j as a *bit node*. The codewords of C are the bit strings such that the sum of the neighboring bits of a check node is 0 mod 2. Given an erasure vector ε, a check node is said to be a *dangling check* if it is incident to a single erased bit. We refer to this erased bit as a *dangling bit*. Figure 1 shows a simple example of this setup. The basic idea of the peeling decoder is to use dangling checks to recover the values of dangling bits and to repeat until the erasure is fully corrected.

The notion of stopping set was introduced in [18] to bound the failure probability of the decoder for classical LDPC codes. A *stopping set* for the Tanner graph $T(H)$ is defined to be a subset of bits that contains no dangling bit, as shown in the rightmost example of Fig. 1. If the erasure covers a non-empty stopping set, then Algorithm 1 returns **Failure**. Algorithm 1 gets stuck when the remaining erasure is equal to a non-empty stopping set because each check is incident to either zero or at least two erased bits.

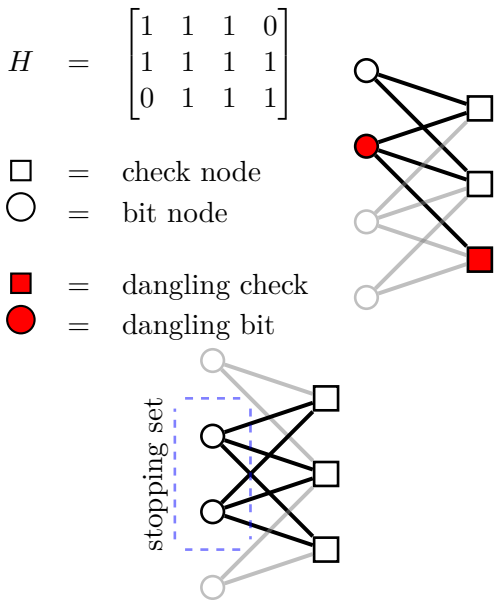


Figure 1: Two examples of an erasure-induced subgraph for a simple Tanner graph $T(H)$. Non-erased nodes are grayed-out and excluded from the subgraph.

The peeling decoder has been adapted to surface codes [19] and color codes [20]. In the rest of this paper, we design a fast erasure decoder inspired by the peeling decoder that applies to a broad class of quantum LDPC codes. Our design process relies on the analysis of stopping sets. At each design iteration, we propose a new version of the decoder, identify its most common stopping sets and modify the decoder to make it capable of correcting these dominant stopping sets.

3 Classical peeling decoder for quantum CSS codes

A CSS code with length N is defined by commuting N-qubit Pauli operators $S_{X,1}, \ldots, S_{X,R_X} \in \{I, X\}^{\otimes N}$ and $S_{Z,1}, \ldots, S_{Z,R_Z} \in \{I, Z\}^{\otimes N}$ called the *stabilizer generators* [21, 22]. We refer to the group they generate as the *stabilizer group* and its elements are called *stabilizers*.

We can correct X and Z errors independently

with the same strategy. Therefore we focus on the correction of X errors, based on the measurement of the Z-type stabilizer generators. This produces a *syndrome* $\sigma(E) \in \mathbb{Z}_2^{R_Z}$, whose i^{th} component is 1 iff the error E anti-commutes with $S_{Z,i}$. An error with trivial syndrome is called a *logical error* and a *non-trivial logical error* if it is not a stabilizer, up to a phase.

We assume that qubits are erased independently with probability p and that an erased qubit suffers from a uniform error I or X [23]. This results in an X-type error E such that $\text{supp}(E) \subseteq \text{supp}(\varepsilon)$. The decoder returns an estimate $\hat{E}$ of E given the erasure vector ε and the syndrome s of E. It succeeds iff $\hat{E}E$ is a stabilizer (up to a phase). The *logical error rate* of the scheme, denoted $P_{\log}(p)$, is the probability that $\hat{E}E$ is a non-trivial logical error.

By mapping Pauli operators onto binary strings, one can cast the CSS erasure decoding problem as the decoding problem of a classical code with parity check matrix $\mathbf{H}_Z$ whose rows correspond to the Z-type stabilizer generators. As a result, one can directly apply the classical Gaussian decoder and the classical peeling decoder to CSS codes. From Lemma 1 of [19], the Gaussian decoder is an optimal decoder, *i.e.* a Maximum Likelihood (ML) decoder, but its complexity scaling like $O(N^3)$ makes it too slow for large codes. The peeling decoder is faster, with scaling as $O(N)$. However, the following lemma proves that, unlike its classical counterpart, stopping sets and hence decoder failures occur much more frequently for quantum codes even in the LDPC case.

Lemma 1 (Stabilizer stopping sets)**.** The support of an X-type stabilizer is a stopping set for the Tanner graph $T(\mathbf{H}_Z)$.

Proof. This is because an X-type stabilizer commutes with Z-type generators, and therefore its binary representation is a codeword for the classical linear code $\ker \mathbf{H}_Z$. □

As a consequence, the classical peeling decoder has no threshold for any family of quantum LDPC codes defined by bounded weight stabilizers. Indeed, if each member of the family has at least one X-type stabilizer with weight w, then the logical error rate satisfies $P_{\log}(p) \geq p^w$, which is a constant bounded away from zero when $N \to \infty$. This is in sharp contrast with the classical case for which the probability to encounter a stopping set provably vanishes for carefully designed families of LDPC codes [24].

4 Pruned peeling decoder

Since the peeling decoder gets stuck into stopping sets induced by the X-type generators, the idea is to look for such a generator S supported entirely within the erasure and to remove an arbitrary qubit of the support of S from the erasure. We can remove this qubit from the erasure because either the error E or its equivalent error ES (also supported inside ε) acts trivially on this qubit.

Algorithm 2: Pruned peeling decoder

input : An erasure vector $\varepsilon \in \mathbb{Z}_2^N$, a syndrome $s \in \mathbb{Z}_2^{R_Z}$, and an integer M.

output: Either **Failure** or an X-type error $\hat{E} \in \{I, X\}^N$ such that $\sigma(\hat{E}) = s$ and $\text{supp}(\hat{E}) \subseteq \text{supp}(\varepsilon)$.

1 Set $\hat{E} = I$.
2 **while** *there exists a dangling generator* **do**
3 Select a dangling generator $S_{Z,i}$.
4 Let j be the index of the dangling qubit incident to $S_{Z,i}$.
5 **if** $s_i = 1$ **then**
6 Replace $\hat{E}$ by $\hat{E}X_j$ and s by $s + \sigma(X_j)$.
7 Set $\varepsilon_j = 0$.
8 **if** *there exists a product S of up to M stabilizer generators among $S_{X,1}, \ldots, S_{X,R_X}$ such that* $\text{supp}(S) \subseteq \text{supp}(\varepsilon)$ **then**
9 Select a qubit in $\text{supp}(S)$ with index j and set $\varepsilon_j = 0$.
10 Go back to step 2.
11 **if** $\varepsilon \neq 0$ **return Failure**, **else return** $\hat{E}$**.**

This leads to the pruned peeling decoder described in Algorithm 2. To make it easier to follow, we use the terms *dangling generator* and *dangling qubit* in place of dangling check and dangling bit. A dangling generator is a Z generator in the context of correcting X errors. In order to keep the complexity of the peeling decoder linear, we

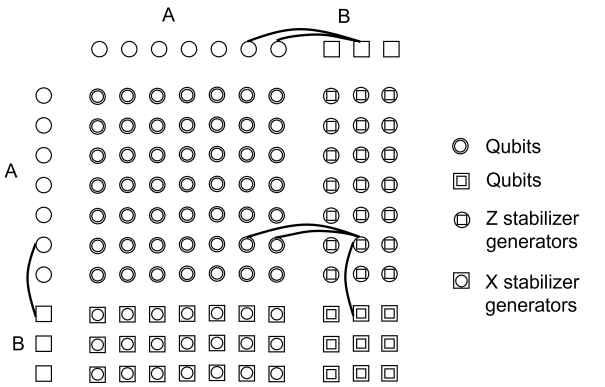


Figure 2: The HGP code derived from a linear code with 7 bits and 3 checks. The support of the Z stabilizer generator with index $(b, a') \in B \times A$ is given by the neighbors of (b, a') in the Cartesian product of the graph $T(H)$ with itself. In the product notation, we follow the $x \times y$ convention, where the first coordinate denotes the horizontal code and the second coordinate denotes the vertical code.

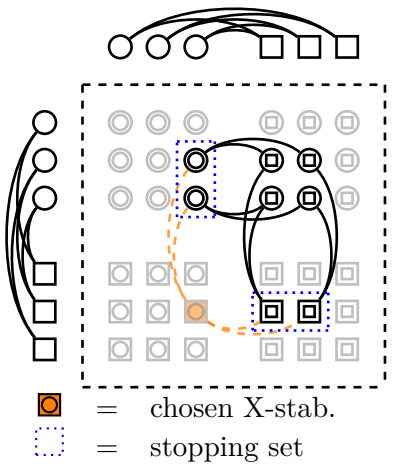


Figure 3: Example of a stabilizer stopping set for the distance 3 surface code obtained from a hypergraph product of two 3-bit repetition codes. The qubits in the support of an X-stabilizer are a stopping set for $T(H_Z)$. Highlighted nodes show the erasure subgraph corresponding to this stopping set.

look for an X-type stabilizer which is a product of up to up M stabilizer generators where M is a small constant. For low erasure rate, we expect the erased stabilizers to have small weight and therefore a small value of M should be sufficient.

Although the pruned peeling algorithm can be used with any CSS code, we are particularly interested in its application to HGP codes. Here, let us recall the hypergraph product construction from [5]. For a classical code with parity check matrix H, recall that the Tanner graph $T(H) = (A \cup B, E_H)$ of this code is the bipartite graph whose nodes consist of disjoint sets A corresponding to bits (the columns of H) and B corresponding to checks (the rows of H). Edges between these nodes are defined by the corresponding nonzero cells in H, and are denoted by the set E_H. The hypergraph product obtained from this classical code, denoted $\mathrm{HGP}(H)$, is defined from the cartesian product of $T(H)$ with itself (see Fig. 2). It is a CSS code with 4-partite Tanner graph $\mathrm{HGP}(H) = ((A \times A) \cup (B \times B) \cup (A \times B) \cup (B \times A), E)$. Qubits in this graph correspond to the nodes in $(A \times A) \cup (B \times B)$ and come in two types resulting from the graph's product structure: "bit-bit" qubits labeled by the pairs $(a, a') \in A \times A$ and "check-check" qubits labeled by the pairs $(b, b') \in B \times B$.

The remaining nodes in $(A \times B) \cup (B \times A)$ are identified with X- and Z-type stabilizer generators, respectively. For each $(a, b') \in A \times B$, we define a stabilizer generator acting as X on the qubits of the form $(b, b') \in B \times B$ such that $\{a, b\} \in E_H$ and acting as X on the qubits of the form $(a, a') \in A \times A$ such that $\{a', b'\} \in E_H$. Stated more explicitly, given fixed $(a, b') \in A \times B$, any $b \in B$ adjacent to a in $T(H)$ lifts to a check-check qubit (b, b') adjacent to (a, b') in $\mathrm{HGP}(H)$, and any $a' \in A$ adjacent to b' in $T(H)$ lifts to a bit-bit qubit (a, a') adjacent to (a, b') in $\mathrm{HGP}(H)$; this node (a, b') acts as an X stabilizer generator on these two types of adjacent qubits in $\mathrm{HGP}(H)$, as shown in Fig. 2. Similarly, for each $(b, a') \in B \times A$, we define an analogous stabilizer generator acting as Z on the qubits of the form (a, a') such that $\{a, b\} \in E_H$ and acting as Z on the qubits of the form (b, b') such that $\{a', b'\} \in E_H$. If the input classical Tannar graph $T(H)$ is sparce, then $\mathrm{HGP}(H)$ is LDPC.

Fig. 3 shows an example of a stabilizer stopping set for a very simple HGP code. The pruned peeling decoder is designed to break out of stopping sets of this form. Fig. 4 shows the performance of HGP codes equipped with the pruned peeling decoder with $M = 0, 1, 2$. The input Tanner graphs for these codes are generated using the standard progressive edge growth algorithm which is commonly used to produce good classical or quantum LDPC codes [25]. We use the implementation [26, 27] of the progressive edge growth algorithm. The pruning strategy only slightly improves over the classical peeling de-

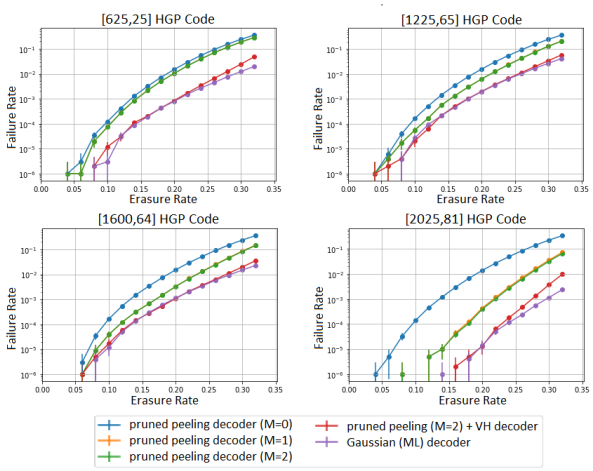


Figure 4: Performance of the pruned peeling and VH decoders using four HGP codes and compared with the ML decoder (10^6 simulations per data point). Plots show the failure rates of the decoders for recovering an X-type Pauli error supported on the erasure vector, up to multiplication by a stabilizer. In these plots, we discard the data points corresponding to fewer than 20 decoding failures to show only meaningful data. All the data points with at least 20 failures are shown. Note that the curves for $M = 1$ and $M = 2$ virtually overlap in all cases.

coder and increasing M beyond $M = 1$ does not significantly affect the performance. To understand why the ML decoder severely outperforms the pruned peeling decoder, we analyze its most common stopping sets with HGP codes.

5 Stopping sets of the pruned peeling decoder

By studying the failure configurations of the pruned peeling decoder, we observe that the gap between the pruned peeling decoder and the ML decoder is due to the following stopping sets of HGP codes.

Lemma 2 (Horizontal and vertical stopping sets)**.** If R_B is a stopping set for a Tanner graph $T(H)$, then for all $b \in B$ the set $\{b\} \times R_B$ is a stopping set for the Tanner graph $T(\mathbf{H}_Z)$ of the HGP code $\mathrm{HGP}(H)$. If R_A is a stopping set for a Tanner graph $T(H^T)$, then for all $a' \in A$ the set $R_A \times \{a'\}$ is a stopping set for the Tanner graph $T(\mathbf{H}_Z)$ of the HGP code $\mathrm{HGP}(H)$.

Proof. Consider a stopping set R_B for $T(H)$. Any Z-type stabilizer generator acting on $\{b\} \times R_B$ must be indexed by (b, a') for some a'. Moreover, the restriction of these stabilizers to $\{b\} \times R_B$ are checks for the linear code $\ker H$. Therefore $\{b\} \times R_B$ is a stopping set for $T(\mathbf{H}_Z)$. The second case is similar. □

We refer to the stopping sets $\{b\} \times R_B$ as *vertical stopping sets* and $R_A \times \{a'\}$ are *horizontal stopping sets*. Numerically, we observe that these stopping sets are responsible for the vast majority of the failures of the pruned peeling decoder. This is because the quantum Tanner graph $T(\mathbf{H}_Z)$ contains on the order of $\sqrt{N}$ copies of the type $\{b\} \times R_B$ for each stopping set R_B of $T(H)$ and $\sqrt{N}$ copies of each stopping set of $T(H^T)$. Our idea is to use the Gaussian decoders of the classical codes $\ker H$ and $\ker H^T$ to correct these stopping sets.

6 VH decoder

The Vertical-Horizontal (VH) decoder is based on the decomposition of the erasure into vertical subsets of the form $\{b\} \times \varepsilon_b$ with $b \in B$ and $\varepsilon_b \subseteq B$, and horizontal subsets of the form $\varepsilon_{a'} \times \{a'\}$ with $a' \in A$ and $\varepsilon_{a'} \subseteq A$, that will be decoded using the Gaussian decoder.

Let T_v (resp. T_h) be the subgraph of $T(\mathbf{H}_Z)$ induced by the vertices of $B \times (A \cup B)$ (resp. $(A \cup B) \times A$). The graph T_v is made with the vertical edges of $T(\mathbf{H}_Z)$ and T_h is made with its horizontal edges. Given an erasure vector ε, denote by $V(\varepsilon)$ the set of vertices of $T(\mathbf{H}_Z)$ that are either erased qubits or check nodes incident to an erased qubit. A *vertical cluster* (resp. *horizontal cluster*) is a subset of $V(\varepsilon)$ that is a connected component for the graph T_v (resp. T_h).

The *VH graph* of ε is defined to be the graph whose vertices are the clusters and two clusters are connected iff their intersection is non-empty.

The following proposition provides some insights on the structure of the VH graph.

Proposition 1. The VH graph is a bipartite graph where each edge connects a vertical cluster with a horizontal cluster. There is a one-to-one correspondence between (i) the check nodes of $T(\mathbf{H}_Z)$ that belong to one vertical cluster and one horizontal cluster and (ii) the edges of the VH graph.

Proof. Because the graph T_v contains only vertical edges, any vertical cluster must be a subset

of $\{b_1\} \times (A \cup B)$ for some $b_1 \in B$. Similarly, any horizontal cluster is a subset of $(A \cup B) \times \{a'_1\}$ for some $a'_1 \in A$. As a result, two clusters with the same orientation (horizontal or vertical) cannot intersect and the only possible intersection between a cluster included in $\{b_1\} \times (A \cup B)$ and a cluster included in $(A \cup B) \times \{a'_1\}$ is the check node (b_1, a'_1). The bijection between check nodes and edges of the VH graph follows. □

Algorithm 3: VH decoder

input : An erasure vector $\varepsilon \in \mathbb{Z}_2^N$, a syndrome $s \in \mathbb{Z}_2^{R_Z}$.

output: Either **Failure** or an X-type error $\hat{E} \in \{I, X\}^N$ such that $\sigma(\hat{E}) = s$ and $\mathrm{supp}(\hat{E}) \subseteq \mathrm{supp}(\varepsilon)$.

Set $\hat{E} = I$.
Construct an empty stack $L = []$.
while *there exists an isolated or a dangling cluster* κ **do**
  if κ *is isolated or frozen* **then**
    Compute an error $\hat{E}_\kappa$ supported on κ whose syndrome matches s on the internal checks of κ in $T(\mathbf{H}_Z)$ (using the Gaussian decoder).
    Replace $\hat{E}$ by $\hat{E}\hat{E}_\kappa$ and s by $s + \sigma(\hat{E}_\kappa)$.
    For all qubits j in κ, set $\varepsilon_j = 0$.
  else
    Then κ is free.
    Remove the free connecting check c of κ from the Tanner graph $T(\mathbf{H}_Z)$.
    Add the pair (κ, c) to the stack L.
    For all qubits j in κ, set $\varepsilon_j = 0$.
while *the stack* L *is non-empty* **do**
  Pop a cluster (κ, c) from the stack L.
  Add the check node c to the Tanner graph $T(\mathbf{H}_Z)$.
  Compute an error $\hat{E}_\kappa$ supported on κ whose syndrome matches s on all the checks of κ in $T(\mathbf{H}_Z)$, including the free check c (using the Gaussian decoder).
  Replace $\hat{E}$ by $\hat{E}\hat{E}_\kappa$ and s by $s + \sigma(\hat{E}_\kappa)$.
if $\varepsilon \neq 0$ **return Failure**, **else return** $\hat{E}$.

A check node of $T(\mathbf{H}_Z)$ that belongs to a single cluster is called an *internal check*, otherwise it is called a *connecting check*. From Proposition 1, a connecting check must belong to one horizontal and one vertical cluster.

A cluster is said to be *isolated* if is has no connecting check. Then, it can be corrected independently of the other clusters. A *dangling cluster* is defined to be a cluster with a single connecting check.

Given a cluster κ, let $E(\kappa)$ be the set of errors supported on the qubits of κ whose syndrome is trivial over the internal checks of κ. Let $S(\kappa)$ be the set of syndromes of errors $E \in E(\kappa)$ restricted to the connecting checks of κ. The set $E(\kappa)$ is a subset of $\{I, X\}^N$ and $S(\kappa)$ is a subset of $\{0, 1\}^{d(\kappa)}$ where $d(\kappa)$ is the number of connecting checks of the cluster κ.

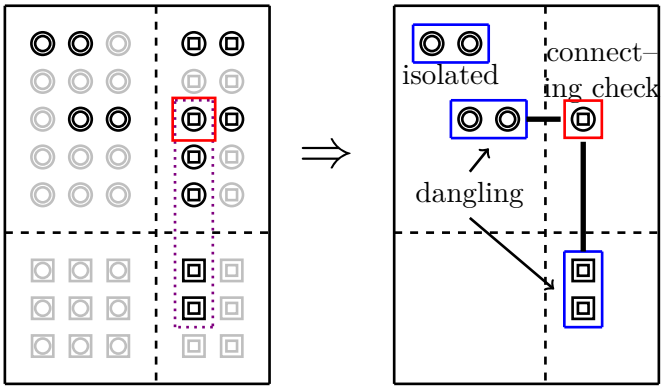
A cluster κ can have two types of connecting check. If $S(\kappa)$ contains a weight-one vector supported on a connecting check c, we say that c is a *free check*. Otherwise, it is a *frozen check*. If a check is free, the value of the syndrome on this check can be adjusted at the end of the procedure to match s using an error included in the cluster κ.

To compute a correction $\hat{E}$ for a syndrome $s \in \mathbb{Z}_2^{R_Z}$, we proceed as follows. Denote by s_κ the restriction of s to a cluster κ. We initialize $\hat{E} = I$ and we consider three cases.

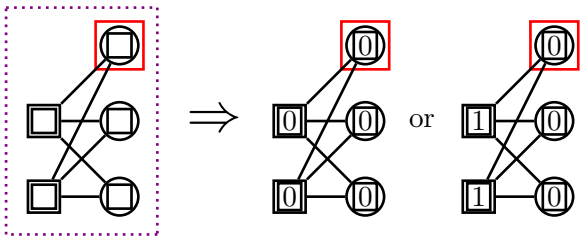
Case 1: Isolated cluster. If κ is a isolated cluster, we use Gaussian elimination to find an error $\hat{E}_\kappa$ supported on the qubits of κ whose syndrome matches s on the internal checks of κ. Then, we add $\hat{E}_\kappa$ to $\hat{E}$, we add $\sigma(\hat{E}_\kappa)$ to s and we remove κ from the erasure ε. This cluster can be corrected independently of the other cluster because it is not connected to any other cluster.

Case 2: Frozen dangling cluster. If κ is a dangling cluster and its only connecting check is frozen, we proceed exactly as in the case of an isolated cluster. This is possible because any correction has the same contribution to the syndrome on the connecting check.

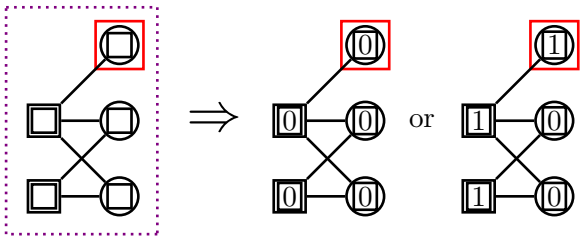
Case 3: Free dangling cluster. The correction of a dangling cluster κ that contains a free check is delayed until the end of the procedure. We remove κ from the erasure and we remove its free check from the Tanner graph $T(\mathbf{H}_Z)$. Then, we look for a correction $\hat{E}'$ in the remaining erasure. We add $\hat{E}'$ to $\hat{E}$ and $\sigma(\hat{E}')$ to s. Once the remaining erasure is corrected and the syn-



(a) Erasure subgraph and corresponding VH graph



(b) *Frozen cluster* and possible solutions



(c) *Free cluster* and possible solutions

Figure 5: **(a)** Example showing how the VH graph is computed from the erasure subgraph (subgraph edges are excluded for simplicity). Nodes in the VH graph correspond to clusters of erased qubits (erased qubit-nodes in the same row or column sharing a check-node, indicated by a blue box above). Edges in the VH graph correspond to *connecting checks* (check nodes adjacent to both a vertical and horizontal cluster, indicated by a red box above). *Isolated clusters* have no connecting checks (degree 0 nodes in the VH graph). *Dangling clusters* have exactly one connecting check (degree 1 nodes in the VH graph). A dangling cluster is determined to be *frozen* or *free* based on the connectivity of the subgraph induced by the cluster; (b) and (c) show two examples for the vertical dangling cluster in the purple box with different subgraphs. **(b)** An example of a frozen cluster. All solutions for the cluster which satisfy the internal checks have the same contribution to the connecting check. **(c)** An example of a free cluster. There exist solutions to the internal checks of the cluster which are 0 or 1 on the connecting check; equivalently, there exists an error on the cluster whose syndrome is non-zero only on the single connecting check.

drome is updated, we find a correction $\hat{E}_\kappa$ inside κ that satisfies the remaining syndrome s_κ in κ. We proceed in that order because the value of the syndrome on a free check can be adjusted at the end of the procedure to match s using an error included in the cluster κ (by definition of free checks).

Altogether, we obtain the VH decoder (Algorithm 3). Fig. 5 shows a small example illustrating the core concepts introduced to explain the algorithm. Our implementation is available here [28]. It works by correcting all isolated and dangling clusters until the erasure is fully corrected. Otherwise, it returns **Failure**.

For a $r \times n$ matrix H, the complexity of the VH decoder is dominated by the cost of the Gaussian decoder which grows as $O(n^3)$ per cluster and there are at most $O(n)$ clusters with size linear in n. Hence the overall cost is in $O(n^4)$ (assuming $r = O(n)$). Therefore the VH decoder can be implemented in $O(N^2)$ bit operations where $N = \Theta(n^2)$ is the length of the quantum HGP code. Using a probabilistic implementation of the Gaussian decoder [29, 30, 31, 32], we can implement the Gaussian decoder in $O(n^2)$ operations per cluster, reducing the complexity of the VH decoder to $O(n^3) = O(N^{1.5})$.

Algorithm 3 fails if the VH-graph of the erasure contains a cycle. However, one can modify the algorithm to eliminate some cycles by removing free checks of all clusters and not only dangling clusters. This may improve further the performance of the VH-decoder.

This modified version of the decoder would be applied after Algorithm 3 becomes stuck in a VH-decoder stopping set. Since the remaining clusters form a cycle, each of these has two or more connecting checks, and each connecting check can be identified as free or frozen with respect to a given cluster. Identifying and removing a free connecting check from the Tanner graph can possibly break the cycle, allowing Algorithm 3 to continue; as before, a correction matching the removed connecting check on the given cluster can be determined at the end of the algorithm. The additional cost of this modification comes from the need to classify multiple connecting checks per cluster, rather than a single check in the dangling case.

The identification of a connecting check as free or frozen itself requires an application of $O(n^3)$ complexity Gaussian elimination on a cluster of size $O(n)$, but the corresponding solutions can be saved to the stack until later used to find a partial correction supported on the cluster. This process could be applied as many as $r = O(n)$ times for a single cluster in the worst case scenario where all checks are connecting checks. Since the total number of clusters is at most $O(n)$, the extreme case where all clusters are contained in a cycle implies an upper bound on the complexity of $O(n^5) = O(N^{2.5})$, slower than the VH-decoder but still exceeding the Gaussian decoder. However, even the addition of this rule cannot account for all stopping sets. If the modified VH graph obtained after identifying and removing all free checks is still a cycle, then the decoder fails.

In comparison with our numerical results from Fig. 4, we see that the combination of pruned peeling and VH decoders performs almost as well as the ML decoder at low erasure erasure rates. This is to say that cycles of clusters, which are stopping sets for the VH decoder, are relatively infrequent in the low erasure rate regime. This behavior matches our intuition since errors for LDPC codes tend to be composed of disjoint small weight clusters [33].

7 Conclusion

We proposed a practical high-performance decoder for the correction of erasure with HGP codes. Our numerical simulations show that the combination of the pruned peeling decoder with the VH decoder achieves a close-to-optimal performance. Moreover it can be implemented in complexity $O(N^2)$. This decoder can be used as a subroutine of the BP-OSD decoder [34], the Union-Find decoder for surface or LDPC codes [35, 36, 37], or the Viderman's decoder [38] to speed up these algorithms.

Combination with the Union-Find decoder also suggests a natural method by which our techniques could be generalized to a mixed error channel, with both erasures and bit/phase flips. Erased qubits can still be replaced with uniformly random mixed states, but we relax the initial assumption that non-erased bits do not have errors. After making a syndrome measurement, we may apply the Union-Find algorithm to grow clusters of qubits around the unsatisfied checks until clusters are large enough to support a correction. Fi-

nally, we treat these clusters as an erasure pattern and assume that qubits not contained in this erasure do not have errors. In this way, the mixed error problem can be converted into a simpler erasure error problem, allowing the application of the Pruned Peeling + VH decoder in the case of HGP codes. The Union-Find decoder for surface codes [35, 36] already has almost linear complexity, but the combination with Pruned Peeling + VH could offer an improvement for the Union-Find algorithm generalized to quantum LDPC codes [37].

In future work, it would be interesting to adapt our decoder to other quantum LDPC codes [39, 40, 41, 42]. We are also wondering if one can reduce the complexity further to obtain a linear time ML decoder for the correction of erasure. Finally, it would be interesting to investigate the resource overhead of various quantum computing architectures capable of detecting erasures [11, 12, 13, 14].

Acknowledgements

This research was supported by the MSR-Inria Joint Centre. AL acknowledges support from the Plan France 2030 through the project ANR-22-PETQ-0006.

References

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

[2] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. "Surface codes: Towards practical large-scale quantum computation". Physical Review A **86**, 032324 (2012).

[3] Robert Gallager. "Low-density parity-check codes". IRE Transactions on Information Theory **8**, 21–28 (1962).

[4] David JC MacKay, Graeme Mitchison, and Paul L McFadden. "Sparse-graph codes for quantum error correction". IEEE Transactions on Information Theory **50**, 2315–2330 (2004).

[5] Jean-Pierre Tillich and Gilles Zémor. "Quantum ldpc codes with positive rate and minimum distance proportional to the square root of the blocklength". IEEE Transactions on Information Theory **60**, 1193–1202 (2013).

[6] Daniel Gottesman. "Fault-tolerant quantum computation with constant overhead". Quantum Information & Computation **14**, 1338–1372 (2014).

[7] Omar Fawzi, Antoine Grospellier, and Anthony Leverrier. "Constant overhead quantum fault-tolerance with quantum expander codes". In 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS). Pages 743–754. IEEE (2018).

[8] Maxime A Tremblay, Nicolas Delfosse, and Michael E Beverland. "Constant-overhead quantum error correction with thin planar connectivity". Physical Review Letters **129**, 050504 (2022).

[9] Emanuel Knill, Raymond Laflamme, and Gerald J Milburn. "A scheme for efficient quantum computation with linear optics". nature **409**, 46–52 (2001).

[10] Sara Bartolucci, Patrick Birchall, Hector Bombin, Hugo Cable, Chris Dawson, Mercedes Gimeno-Segovia, Eric Johnston, Konrad Kieling, Naomi Nickerson, Mihir Pant, et al. "Fusion-based quantum computation". Nature Communications **14**, 912 (2023).

[11] Yue Wu, Shimon Kolkowitz, Shruti Puri, and Jeff D Thompson. "Erasure conversion for fault-tolerant quantum computing in alkaline earth rydberg atom arrays". Nature communications **13**, 4657 (2022).

[12] Mingyu Kang, Wesley C Campbell, and Kenneth R Brown. "Quantum error correction with metastable states of trapped ions using erasure conversion". PRX Quantum **4**, 020358 (2023).

[13] Aleksander Kubica, Arbel Haim, Yotam Vaknin, Harry Levine, Fernando Brandão, and Alex Retzker. "Erasure qubits: Overcoming the t_1 limit in superconducting circuits". Physical Review X **13**, 041022 (2023).

[14] Takahiro Tsunoda, James D Teoh, William D Kalfus, Stijn J de Graaf, Benjamin J Chapman, Jacob C Curtis, Neel

Thakur, Steven M Girvin, and Robert J Schoelkopf. "Error-detectable bosonic entangling gates with a noisy ancilla". PRX Quantum **4**, 020354 (2023).

[15] Shrinivas Kudekar, Tom Richardson, and Rüdiger L Urbanke. "Spatially coupled ensembles universally achieve capacity under belief propagation". IEEE Transactions on Information Theory **59**, 7761–7813 (2013).

[16] Tom Richardson and Ruediger Urbanke. "Modern coding theory". Cambridge university press. (2008).

[17] Michael G Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A Spielman. "Efficient erasure correcting codes". IEEE Transactions on Information Theory **47**, 569–584 (2001).

[18] Victor Vasilievich Zyablov and Mark Semenovich Pinsker. "Decoding complexity of low-density codes for transmission in a channel with erasures". Problemy Peredachi Informatsii **10**, 15–28 (1974).

[19] Nicolas Delfosse and Gilles Zémor. "Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel". Physical Review Research **2**, 033042 (2020).

[20] Sangjun Lee, Mehdi Mhalla, and Valentin Savin. "Trimming decoding of color codes over the quantum erasure channel". In 2020 IEEE International Symposium on Information Theory (ISIT). Pages 1886–1890. IEEE (2020).

[21] Andrew Steane. "Multiple-particle interference and quantum error correction". Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **452**, 2551–2577 (1996).

[22] A Robert Calderbank and Peter W Shor. "Good quantum error-correcting codes exist". Physical Review A **54**, 1098 (1996).

[23] Markus Grassl, Th Beth, and Thomas Pellizzari. "Codes for the quantum erasure channel". Physical Review A **56**, 33 (1997).

[24] Thomas J Richardson and Rüdiger L Urbanke. "Efficient encoding of low-density parity-check codes". IEEE Transactions on Information Theory **47**, 638–656 (2001).

[25] Xiao-Yu Hu, E. Eleftheriou, and D.-M. Arnold. "Progressive edge-growth tanner graphs". In GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270). Volume 2, pages 995–1001 vol.2. (2001).

[26] Xiao-Yu Hu, E. Eleftheriou, and D.M. Arnold. "Regular and irregular progressive edge-growth tanner graphs". IEEE Transactions on Information Theory **51**, 386–398 (2005).

[27] T S Manu. "Progressive edge growth algorithm for generating LDPC matrices" (2014).

[28] Nicholas Connolly. "Pruned peeling and vh decoder" (2022).

[29] Douglas Wiedemann. "Solving sparse linear equations over finite fields". IEEE Transactions on Information Theory **32**, 54–62 (1986).

[30] Erich Kaltofen and B David Saunders. "On Wiedemann's method of solving sparse linear systems". In International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes. Pages 29–38. Springer (1991).

[31] Brian A LaMacchia and Andrew M Odlyzko. "Solving large sparse linear systems over finite fields". In Conference on the Theory and Application of Cryptography. Pages 109–133. Springer (1990).

[32] Erich Kaltofen. "Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems". Mathematics of Computation **64**, 777–806 (1995).

[33] Alexey A Kovalev and Leonid P Pryadko. "Fault tolerance of quantum low-density parity check codes with sublinear distance scaling". Physical Review A **87**, 020304 (2013).

[34] Pavel Panteleev and Gleb Kalachev. "Degenerate quantum ldpc codes with good finite length performance". Quantum **5**, 585 (2021).

[35] Shilin Huang, Michael Newman, and Kenneth R Brown. "Fault-tolerant weighted union-find decoding on the toric code". Physical Review A **102**, 012419 (2020).

[36] Nicolas Delfosse and Naomi H Nickerson. "Almost-linear time decoding algorithm for topological codes". Quantum **5**, 595 (2021).

[37] Nicolas Delfosse, Vivien Londe, and Michael E Beverland. "Toward a union-find decoder for quantum ldpc codes". IEEE Transactions on Information Theory (2022).

[38] Anirudh Krishna, Inbal Livni Navon, and Mary Wootters. "Viderman's algorithm for quantum ldpc codes". In Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). Pages 2481–2507. SIAM (2024).

[39] Nikolas P Breuckmann and Jens N Eberhardt. "Balanced product quantum codes". IEEE Transactions on Information Theory **67**, 6653–6674 (2021).

[40] Pavel Panteleev and Gleb Kalachev. "Asymptotically good quantum and locally testable classical ldpc codes". In Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing. Pages 375–388. (2022).

[41] Anthony Leverrier and Gilles Zémor. "Quantum tanner codes". In 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS). Pages 872–883. IEEE (2022).

[42] Irit Dinur, Min-Hsiu Hsieh, Ting-Chun Lin, and Thomas Vidick. "Good quantum ldpc codes with linear time decoders". In Proceedings of the 55th annual ACM symposium on theory of computing. Pages 905–918. (2023).