Viderman's algorithm for quantum LDPC codes

Anirudh Krishna* Inbal Livni Navon† Mary Wootters‡

Abstract

Quantum low-density parity-check (LDPC) codes, a class of quantum error correcting codes, are considered a blueprint for scalable quantum circuits. To use these codes, one needs efficient decoding algorithms. In the classical setting, there are multiple efficient decoding algorithms available, including *Viderman's algorithm* (Viderman, TOCT 2013). Viderman's algorithm for classical LDPC codes essentially reduces the error-correction problem to that of *erasure*-correction, by identifying a small envelope L that is guaranteed to contain the error set.

Our main result is a generalization of Viderman's algorithm to quantum LDPC codes, namely *hypergraph product codes* (Tillich, Zémor, IEEE T-IT, 2013). This is the first erasure-conversion algorithm that can correct up to $\Omega(D)$ errors for constant-rate quantum LDPC codes, where D is the distance of the code. In that sense, it is also fundamentally different from existing decoding algorithms, in particular from the `small-set-flip` algorithm (Leverrier, Tillich, Zémor, FOCS, 2015). Moreover, in some parameter regimes, our decoding algorithm improves on the decoding radius of existing algorithms. We note that we do not yet have linear-time erasure-decoding algorithms for quantum LDPC codes, and thus the final running time of the whole decoding algorithm is not linear; however, we view our linear-time envelope-finding algorithm as an important first step.

1 Introduction.

Error correcting codes play a critical role in the storage and transmission of both classical and quantum information, by protecting this information from corruption. *Low-Density Parity-Check* (LDPC) codes are a ubiquitous family of graph-based error correcting codes. (Classical) LDPC codes were first introduced by Gallager in the 1960's [13], and today are used in practice for everything from satellite communication to 5G networks. One of the reasons for their ubiquity is that LDPC codes support extremely fast *decoding algorithms*. Such algorithms take a *corrupted codeword* $\widetilde{\mathbf{w}} \in \{0,1\}^n$ that is promised to be close in Hamming distance to a true codeword $\mathbf{w} \in \{0,1\}^n$, and "correct" $\widetilde{\mathbf{w}}$ to return $\mathbf{w}$ itself. Two of these algorithms are the `Flip` algorithm [25], and *Viderman's algorithm* [27].

Both algorithms run in linear time, but have very different flavors: `Flip` is a greedy algorithm, which iteratively flips bits until they are all corrected; in contrast, Viderman's algorithm essentially reduces the error-correction problem to the *erasure*-correction problem[1] by identifying a small *envelope* L that contains all of the errors, and replacing them with $\perp$; then an erasure-correction algorithm can be run to fill in the $\perp$'s. The two algorithms also give different guarantees: in particular, in some settings, Viderman's algorithm can correct more errors, while relaxing the constraints on the graph underlying the code construction.

Due to their importance in the classical setting, it is natural to adapt LDPC codes—and their decoding algorithms—to the quantum setting. This was done by Tillich and Zémor in [26], who constructed quantum LDPC Codes using *hypergraph product codes*. More recently, a series of exciting breakthroughs [9, 17, 16, 1] have led to quantum LDPC Codes [23, 19] with a near-optimal (up to constant factors) trade-off between the amount of redundancy required and the error correction capabilities.

Our question: A quantum Viderman's algorithm? While many constructions of quantum LDPC codes do come with linear-time decoding algorithms, all of the algorithms that we are aware of (which can correct up to $\Omega(D)$ errors) are variants of the `Flip` algorithm mentioned above. In particular, it has been an open question

*Stanford University.
†Stanford University.
‡Stanford University.

[1] In error correcting codes, an *error* is when a (qu)bit has a value that is different than it is supposed to have; in particular, the decoder does not know which (qu)bits are in error. An *erasure* is when a (qu)bit is replaced by a special symbol $\perp$. The erasure-correction problem is generally easier than the error-correction problem, because the decoder knows where the erasures have occured.

whether or not a Viderman-style algorithm—which first reduces the problem to erasure-decoding—could be used to correct a *quantum* LDPC code.

Our main result: the first Viderman-style algorithm for constant-rate Quantum LDPC Codes. We discuss our results in more detail below in Section 1.1, but briefly, our algorithm applies to *hypergraph product codes*, and identifies a small envelope $\mathcal{L}$ containing all of the errors in linear time.

Why we care. Before we discuss our results in more detail, we briefly mention a few points of motivation for a Quantum Viderman-style algorithm for hypergraph product codes.

- **Converting errors to erasures.** As discussed above, Viderman's algorithm, and our quantum analog, identifies a small *envelope* $\mathcal{L}$, which is guaranteed to contain all of the errors. Then it treats the (qu)bits in $\mathcal{L}$ as erasures and runs an erasure-decoding algorithm to finish the job. In fact, coming up with such conversion algorithms for quantum codes has been a goal in other work. For example, the `UnionFind` algorithm of Delfosse & Nickerson gives such an algorithm for surface codes [4]. Moreover, there has been some partial success for constant-rate quantum LDPC codes: Delfosse, Londe & Beverland give a version of `UnionFind` that converts errors into erasures for hypergraph product codes [5]. However, the decoding radius for hypergraph product codes is $O(D^\beta)$ for some $\beta < 1$, where D is the distance of the code. In contrast, our work gives a linear-time algorithm to convert up to $\Omega(D)$ errors into erasures.

- **Improved parameters for decoding quantum LDPC codes.** Our quantum Viderman-style algorithm has (modestly) improved parameters over quantum `Flip`-style algorithms in certain parameter regimes (see Table 1 and the discussion in Section 1.1). Moreover, we are hopeful that the parameters can be further improved. To support our hope, we note that improvements on the original version of (classical) Viderman's algorithm have led to improved algorithms for decoding classical expander codes when the graph is a very good expander, in terms of the error radius and the requirements on the underlying graph [2]. As quantum `Flip`-like algorithms have been extensively studied and optimized over the years, our algorithm provides limited improvements over the state-of-the-art `Flip`-like algorithms, but it does significantly improve the error radius over the first `Flip`-like results [18] (roughly saving a factor the underlying graph's degree, see Table 1). Given the classical landscape, we are hopeful that as our ideas are further developed, more significant improvements will arise.

- **A fundamentally new decoding algorithm.** As mentioned above, all decoding algorithms for quantum LDPC codes that we are aware of with decoding radius $\Omega(D)$ are similar to the `Flip` algorithm of [25]. Viderman's algorithm is of a fundamentally different flavor than `Flip`, and—as we discuss more below in Section 2—there were several challenges to overcome to make it work in the quantum setting. In addition to expanding our algorithmic toolbox for quantum LDPC codes, we hope that overcoming these challenges deepens our understanding of quantum LDPC codes, and will lead to further progress in the future.

- **A fundamental class of codes.** As mentioned above, hypergraph product codes were the first quantum LDPC codes with decent distance, and remained the state-of-the-art for many years. In recent years, a series of breakthroughs has resulted in quantum LDPC codes with much better distance. However, we believe that hypergraph product codes are a good starting point.

Now that we have given the high-level motivation for a quantum Viderman-style algorithm, we outline our results in more detail in Section 1.1. After that, we will survey related work in Section 1.2.

1.1 Our Results

Before we state our results, we introduce a bit of notation (see Section 3 for full definitions). An $[\![N, K, D]\!]_2$ *quantum error correcting code* $\mathcal{H}$ encodes a set of K logical qubits in a set $\mathcal{Q}$ of N qubits. It is defined by two sets $\mathcal{X}$ and $\mathcal{Z}$, which are sets of constraints on the qubits in $\mathcal{Q}$, and which must interact with each other in a particular way. A codeword of $\mathcal{H}$ can be thought of as an assignment of 0 or 1 to each qubit so that all of the constraints in $\mathcal{X}$ and $\mathcal{Z}$ are satisfied. The *distance* D of $\mathcal{H}$ (formally defined in Section 3) corresponds to the number of (Pauli) errors that the code can correct. Thus, it is desirable to have K as close to N as possible, while having D as large as possible.

A quantum code is a *quantum LDPC code* if all of the constraints in $\mathcal{X}$ and $\mathcal{Z}$ are *sparse*, meaning that not too many qubits in $\mathcal{Q}$ are involved in each one. As with classical LDPC codes, this leads to a natural graph-theoretic

connection—sparse parity-checks can be represented as sparse bipartite graphs—and the constructions of codes we consider are based on bipartite expander graphs.

There are two types of errors a quantum code may face, *X-type errors* and *Z-type errors*. The constraints $\mathcal{X}$ are meant to correct Z-type errors, while the constraints $\mathcal{Z}$ are meant to correct X-type errors. It suffices to deal with these two types individually, so for the rest of this overview we focus on Z-type errors. We refer to $\mathcal{X}$ as "parity-checks," and $\mathcal{Z}$ as "generators."

Suppose that a codeword in $\mathcal{H}$ has been corrupted by a set of (Z-type) errors $\mathcal{E} \subseteq \mathcal{Q}$: that is, the value of each qubit in $\mathcal{E}$ has been flipped. We are guaranteed that $\mathcal{E}$ (after being appropriately *reduced*, see Sections 2 and 3) is small, and we would like to correct these errors. The largest that $|\mathcal{E}|$ can be is $D/2$, half of the distance of the code, so our goal will be to allow for $|\mathcal{E}|$ as close to this as possible. The number of (reduced) errors that a code can tolerate is called the *decoding radius*.

1.1.1 Main Result: Viderman's algorithm for hypergraph product codes Our main result is the first quantum version of Viderman's algorithm, which we call *Small-Set-Find* (`SSFind`, which we give at a high level as Algorithm 2 and in more detail in Algorithm 3). `SSFind` applies to *hypergraph product codes*. Hypergraph product codes are an important class of quantum LDPC codes, and were the first quantum LDPC codes shown to achieve constant rate K/N and non-trivial distance D [26]. We formally define them in Section 3.4, and for now just note that they are built out of a bipartite expander graph.

`SSFind` takes as input the set $\mathsf{UNSAT} \subseteq \mathcal{X}$ of unsatisfied parity checks, and outputs an *envelope* $\mathcal{L}$, with the guarantees that $\mathcal{E} \subseteq \mathcal{L}$ and that $\mathcal{L}$ is not too large. The following is our main result.

THEOREM 1.1. *Let $\mathcal{H}$ be an $[\![N, K, D]\!]_2$ hypergraph product code (see Section 3.4) on qubits $\mathcal{Q}$ and with X-parity-checks $\mathcal{X}$, constructed from an expander graph $G = (V, E)$ with left-degree Δ_V and right-degree Δ_C, and expansion parameter $\epsilon < 1/10$.*

Let $\mathcal{E}$ be a (reduced) error pattern with weight at most

$$|\mathcal{E}| < \gamma \cdot D - \Delta_V,$$

where $\gamma = \frac{\Delta_V}{\Delta_C} \cdot \frac{1-10\epsilon}{4}$, and let $\mathsf{UNSAT} \subseteq \mathcal{X}$ be the set of unsatisfied parity checks arising from $\mathcal{E}$. Then `SSFind` *(Algorithm 3), given UNSAT, runs in time $O_{\Delta_V,\Delta_C}(|\mathcal{E}|)$, and returns an envelope $\mathcal{L} \subseteq \mathcal{Q}$, so that $\mathcal{E} \subseteq \mathcal{L}$, and*

$$|\mathcal{L}| \leq \frac{1}{\gamma} \cdot |\mathcal{E}|.$$

It is important to note that `SSFind` does not actually correct the errors $\mathcal{E}$; rather it reduces the problem to that of erasure-decoding, and we can use an erasure-decoding algorithm from there. Notice that by combining the assumption on $|\mathcal{E}|$ with the conclusion about $|\mathcal{L}|$, we get that $|\mathcal{L}| < D$, and in particular the code can be uniquely decoded from $|\mathcal{L}|$ erasures. In the classical setting, there is a relatively simple linear-time erasure-decoding algorithm that will do this [13, 24]. However, in the quantum setting, we unfortunately do not have a linear-time erasure-decoding algorithm. Erasure-decoding amounts to solving a linear system and this can be done via Gaussian elimination. The pivot points need only be qubits adjacent to the error and in this case, the size of this set is at most $N^{1/2}$. This implies that we can do erasure decoding in time $O(N^{1.5})$. However, at the moment, a linear-time erasure decoding algorithm that works up to the radius D remains elusive. This is a major drawback of our result, as it means that a final decoding algorithm would run in time $O(N^{1.5})$; however, we are hopeful that follow-up work will provide a linear-time erasure-decoding algorithm, which will then result in a linear-time algorithm for decoding from errors.

In Table 1, we compare the parameters of Theorem 1.1 to the existing decoding algorithm for hypergraph product codes, which is called Small-Set-Flip (`SSFlip`). This algorithm works like `Flip`, except in each greedy step instead of flipping a single bit, it flips a small set of bits at a time (hence the name). `SSFlip` was introduced in [18], and further expanded in [11, 10, 14].[2]

As we see in Table 1, the parameters are in general incomparable. Our algorithm `SSFind` yields an improvement in the decoding radius when G is an excellent expander, and there is a large gap between Δ_V

[2]These works show that `SSFlip` can handle a constant fraction of stochastic errors (as opposed to $O(\sqrt{N})$ adversarial errors); they also show that `SSFlip` is robust to errors in the parity check bits.

and Δ_C. For example, if $\Delta_C = 2\Delta_V$ (so $r = 1/2$) and $\epsilon = 1/20$ is small, then the decoding radius for `SSFind` is $0.062D$, compared to less than $0.058D$ for `SSFlip` (with the analysis in [14]).

Algorithm	Decoding Radius	Required Expansion
`SSFlip` [18]	$\frac{1}{3(1+\Delta_C)}D$	$\epsilon < 1/6$
`SSFlip` [14]	$\left(\frac{2r(1-8\epsilon)}{4+2r(1-8\epsilon)}\right)\left(\frac{r}{\sqrt{1+r^2}}\right)D$	$\epsilon < 1/8$
This paper: `SSFind` plus erasure-decoding	$\frac{1-10\epsilon}{4}rD$	$\epsilon < 1/10$

Table 1: Comparison of the parameters between different decoding algorithm of hypergraph product codes, with underlying graph is G, which is an $(\alpha_V, \epsilon_V, \alpha_C, \epsilon_C)$-bipartite expander (Def. 3.2), with left and right degree Δ_V and Δ_C respectively. Here we use $r = \frac{\Delta_V}{\Delta_C}$, $\epsilon = \max(\epsilon_V, \epsilon_C)$. Above, $D = \min(\alpha_V|V|, \alpha_C|C|)$ is the distance of the code.

1.2 Related Work

Algorithms for Classical LDPC Codes. While LDPC codes have been studied since the 1960's [13], the first linear-time decoding algorithms for graph-based codes (or, any codes) to correct a constant fraction of *worst-case* errors was given by Sipser and Spielman [25] who studied *expander codes;* these are LDPC codes where the underlying constraint graph is an expander.[3] Sipser and Spielman introduced the `Flip` algorithm, a greedy algorithm which iteratively flips bits until it converges. If the underlying expander $G = (V, C, E)$ is an (α_V, ϵ_V)-expander (meaning that sets S of size at most $\alpha_V n$ have neighborhoods of size at least $|S|\Delta_V(1-\epsilon_V)$, where Δ_V is the left-degree of the graph), the resulting expander code of length n has minimum distance at least $\alpha_V n$.

When $\epsilon_V < 1/4$, Sipser and Spielman showed that `Flip` can correct up to $\alpha_V n/(\Delta_V + 1)$ errors. Subsequent works gave other algorithms that improved this to $\left(\frac{1-3\epsilon_V}{1-2\epsilon_V}\right)\alpha_V n$ errors, which is better when $\epsilon_V < 1/3$ [12, 28, 27]; the first two of these references are based on linear programming, and the third is what we refer to as *Viderman's algorithm*. In addition to improving the decoding radius, Viderman's algorithm also requires less from the underlying expander graph (in that ϵ_V can be taken to be larger), making it easier to obtain constructions. As mentioned above, Viderman's algorithm works by identifying an envelope $\mathcal{L}$ of "suspicious" bits, and then treating them as erasures. More recently, [2] gave improved combinatorial bounds and algorithmic results for expander codes. Their improved algorithms include variants and combinations of both `Flip` and Viderman's algorithm; in particular, for small $\epsilon_V < 1/8$, they present a variant of Viderman's algorithm that decodes up to a significantly larger radius than previous works: $\frac{\sqrt{2}-1}{2\epsilon_V}\alpha_V n$ for very small ϵ_V, and $\frac{1-2\epsilon_V}{4\epsilon_V}\alpha_V n$ for slightly larger ϵ_V that is still smaller than $1/8$.

We note that there are several constructions of graph-based codes other than expander codes, most notably *Tanner constructions*, where the code is constructed both from an expander graph and from an appropriate *inner code* (for example, [29]). Decoding algorithms for these codes typically leverage decoding algorithms for the inner code, and thus are a bit different from the focus of our paper.

Algorithms for Quantum LDPC Codes. Until recently, it was unclear whether asymptotically good quantum LDPC codes (that is, with $K, D = \Omega(N)$) even exist. Until 2020, hypergraph product codes were the best candidates; Tillich & Zémor showed that they can achieve $K = \Theta(N)$ and $D = \Theta(\sqrt{N})$ [26]. Shortly after, Leverrier, Tillich & Zémor [18] proposed a quantum version of `Flip` called Small-Set-Flip (`SSFlip`) for (expander-based) hypergraph product codes. The algorithm `SSFlip` is also a linear-time algorithm, and in the adversarial setting can correct up to $\Theta(\sqrt{N})$ errors, within a constant fraction of the optimal decoding radius.

[3] Equivalently, the code is defined as the kernel of a parity-check matrix H, which is the adjacency matrix of an unbalanced expander graph.

Following a series of breakthroughs crossing the $\sqrt{N}$ barrier [9, 17, 16, 1], asymptotically good quantum LDPC codes were finally recently attained, first by Panteleev & Kalachev [23], and later simplified by Leverrier & Zémor [19]. These constructions can be equipped with decoding algorithms [9, 20, 21, 15]. There are also variants on the code construction that have efficient decoding algorithms: Lin & Hsieh [22] proposed a construction based on an as-yet unresolved conjecture; Dinur *et al.* [8] also proposed a variant of the Tanner construction. The decoding algorithms for all constant-rate quantum LDPC codes with distance exceeding the $\sqrt{N}$ barrier are based on generalizations of `SSFlip`.

There has been previous work on quantum error correcting codes that focuses on converting errors into erasures. The `UnionFind` algorithm by Delfosse & Nickerson was the first erasure-conversion algorithm that runs in almost-linear-time [6]. It builds on the linear-time maximum-likelihood erasure decoding algorithm of Delfosse & Zémor [7] for surface codes. While `UnionFind` was first proposed for the surface code, it has since been generalized to a broader class of codes [4, 5]. However, the decoding radius of `UnionFind` for LDPC codes can be suboptimal. As it applies to hypergraph product codes, the decoding radius of this algorithm was only guaranteed to scale as $\Theta(D^\beta)$ for some constant $1 > \beta > 0$. To the best of our knowledge, our algorithm is the first erasure-conversion algorithm for constant-rate quantum LDPC codes that achieves a decoding radius of $\Theta(D)$.

We provide a high-level contrast of `UnionFind` and Viderman's algorithm at the end of Section 2.

Given an algorithm that reduces the problem to erasure-decoding, our next question is about efficient erasure-decoding algorithm. As in the classical (linear) case, decoding a quantum error correcting code boils down to solving a linear system, and can be done straightforwardly in time $O(N^3)$. As mentioned above, in the case of hypergraph product codes, in fact this linear system can be solved in time $O(N^{1.5})$. In the case of stochastic erasures (where a hypergraph product code can recover from $\Omega(N)$ erasures, rather than $\Omega(\sqrt{N})$), Connolly *et al.* [3] give an improved $O(N^2)$-time erasure-decoding algorithm for hypergraph product codes, which generalizes belief propagation.

1.3 Open Problems and Future Directions We view our work as the first step towards improved decoding algorithms for quantum LDPC codes. While we are able to obtain a slight improvement in some parameter regimes for hypergraph product codes, there is still much to do:

- Given that ours is only the first version of a quantum Viderman's algorithm, we hope that future work will build on our ideas to improve the parameters, obtaining bigger improvements over `SSFlip` (and in more parameter regimes) than those reported in Table 1. In particular, we are hopeful that the ideas of [2] (which improve Viderman's algorithm in the classical case) can be applied on top of our work.

- We believe that our ideas can be extended to some of the more recent constructions of asymptotically good LDPC codes. In particular, the construction of Lin and Hsieh [22] begins with a hypergraph product code, and quotients it out by an appropriate group action to obtain a code with better distance. As their construction is built directly on hypergraph product codes, this is the next natural target for our techniques[4].

1.4 Organization In Section 2, we give a high-level technical overview of our approach, and introduce our algorithm `SSFind`. In Section 3, we formally give the necessary background and definitions for expander graphs; classical expander codes and Viderman's algorithm; and quantum codes and hypergraph product codes. In Section 4, we analyze `SSFind` on hypergraph product codes and prove our main theorem.

1.5 Acknowledgements AK is supported by the Bloch Postdoctoral Fellowship and NSF grant CCF-1844628. AK would also like to thank Shashwat Silas for introducing him to Viderman's algorithm and related discussions. ILN is supported by the Simons Foundation Collaboration on the Theory of Algorithmic Fairness, the Sloan Foundation Grant 2020-13941, and the Zuckerman STEM Leadership Program. MW was partially supported by NSF grants CCF-1844628, CCF-2231157, and CCF-2133154

[4] To the best of our knowledge, the constructions of [23, 19, 8] are generalizations of Tanner codes. For this reason, it is unclear if there is a straightforward generalization of Viderman's algorithm to these settings.

2 Technical Overview

We begin with an exposition of the classical version of Viderman's algorithm [27]. Before we begin, we set up a bit more notation. A (classical, binary, linear) error correcting code $\mathcal{C}$ with block length n is just a linear subspace of $\mathbb{F}_2^n$. The *distance* d of $\mathcal{C}$ is the minimum Hamming distance between any two distinct elements (called *codewords*) of $\mathcal{C}$. In particular, given a *corrupted codeword* $\widetilde{\mathbf{w}} \in \mathbb{F}_2^n$ with Hamming distance less than $d/2$ from some $\mathbf{w} \in \mathcal{C}$, the triangle inequality implies that it is in theory possible to recover $\mathbf{w}$. The goal of a *decoding algorithm* is to do so efficiently.

A classical *LDPC code* is a code that can be defined by a sparse bipartite graph $G = (V, C, E)$ in the following way. We associate the left-hand vertices V (with $|V| = n$) with the symbols of a codeword, and the right-hand vertices C with *parity-checks*. We say that a string $\mathbf{w} \in \mathbb{F}_2^n$ is in $\mathcal{C}$ if, for each parity-check $c \in C$, $\sum_{v \in \Gamma(c)} \mathbf{w}_v = 0$, where $\Gamma(c)$ denotes the neighbors of c in G. We focus on the case where the underlying graph G is a bipartite expander (see Definition 3.2); such a code is sometimes called an *expander code* [25].

2.1 Classical Viderman's Algorithm Intuitively, Viderman's algorithm works by iteratively identifying "suspicious" bits and parity-checks; the suspicious bits are added to a set L (called the *envelope*), and the suspicious parity checks are added to a set R. At the beginning, R is the set UNSAT of unsatisfied parity checks. From there, the rule is simple: if a vertex $v \in V$ is connected to too many suspicious checks, it, and all the checks it touches, are labeled suspicious. This process repeats until it stabilizes (see Algorithm 1).

Algorithm 1 `Find`: Viderman's decoding algorithm.

Input: $\mathsf{UNSAT} \subseteq C$
Output: $L \subseteq V$

$h \leftarrow (1 - 2\epsilon_V)\Delta_V$, ▷ *$\epsilon_V$, Δ_V are parameters of the underlying expander graph (see Section 3).*
$L \leftarrow \emptyset$, $R \leftarrow \mathsf{UNSAT}$
while $\exists v \in V$ such that $|\Gamma(v) \cap R| \geq h$ **do**
    $L \leftarrow L \cup \{v\}$
    $R \leftarrow R \cup \Gamma(v)$.
return L

At the end of the day, we hope that (a) the envelope L contains all of the errors; and (b) L is not too large. Viderman showed that this is indeed the case, provided that the initial number of errors in $\mathcal{E}$ is small enough.

2.2 Quantum Viderman's Algorithm for Hypergraph Product Codes We recall the notation for a quantum error correcting code $\mathcal{H}$ from Section 1.1: $\mathcal{H}$ is defined by a set of parity checks $\mathcal{X}$ and generators $\mathcal{Z}$, and encodes K logical qubits into a set $\mathcal{Q}$ of N qubits. The parity checks $\mathcal{X}$ and generators $\mathcal{Z}$ serve to define two vector spaces (aka, classical linear codes) $\mathcal{C}_X$ and $\mathcal{C}_Z$ respectively: $\mathcal{X}$ gives the parity-checks for $\mathcal{C}_X$ and $\mathcal{Z}$ gives the parity-checks for $\mathcal{C}_Z$.[5] For $\mathcal{H}$ to be a valid quantum code, we require $\mathcal{C}_X^\perp \subseteq \mathcal{C}_Z$.

Hypergraph Product Codes. We define hypergraph product codes formally in Section 3.4, but for now we give an informal definition and a picture.

Let $G = (V \cup C, E)$ be a biregular bipartite graph. Our hypergraph product code will be built out of two copies of G. Qubits are associated with elements of $V \times V \sqcup C \times C$; we write $\mathcal{Q} = \mathcal{Q}_V \sqcup \mathcal{Q}_C$ for the two parts, where $\sqcup$ denotes disjoint union. The X-type parity checks $\mathcal{X}$ and Z-type generators $\mathcal{Z}$ are associated with $V \times C$ and $C \times V$ respectively.

Now, we form a graph $\mathcal{G}$ on the vertices $\mathcal{Q} \sqcup \mathcal{X} \sqcup \mathcal{Z}$, which is given by the *graph product* of two copies of G. The way this graph product works is formally described in Section 3, and informally illustrated in Figure 1. For example, the qubit $(\nu, v) \in V \times V =: \mathcal{Q}_V$ is connected to every element in $\{\nu\} \times \Gamma(v) \subset V \times C = \mathcal{X}$, where $\Gamma(v)$ denotes the neighborhood of v in G. In particular, there is a copy of G in the row indexed by $\nu \in V$ (as well as by every other row and every column).

This graph product defines the sets $\mathcal{X}$ and $\mathcal{Z}$, and hence the quantum hypergraph product code $\mathcal{H}$.

[5] That is, $\mathcal{C}_X$ is the kernel of the adjacency matrix defined by $\mathcal{X}$, and similarly for $\mathcal{C}_Z$.

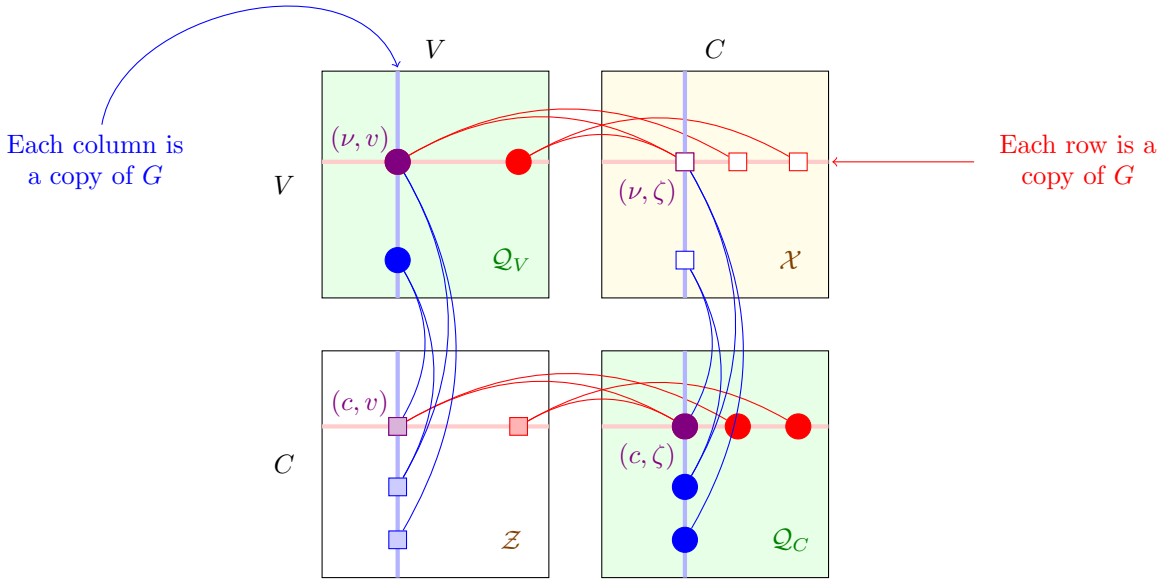

Figure 1: Definition-by-picture of hypergraph product codes [26], formed by taking the graph product of two copies of G.

Reduced Errors. In order to explain what we mean by a "reduced" error in Theorem 1.1, we begin by pointing out what seems to be (but is not actually) a major hurdle in designing quantum LDPC codes in general, namely that the underlying graph $\mathcal{G}$ is no longer a good expander. (We note that this notion of reduced errors is not specific to our work; we include it for background and motivation).

In more detail, in the classical setting, expansion of the underlying graph G guarantees that any small enough set of errors $\mathcal{E}$ has many *unique neighbors*,[6] which guarantees the existence of many unsatisfied parity checks that whenever there are not too many errors. This guarantees that small sets of errors can be at least detected, and ideally efficiently corrected.

However, the quantum code defined by the graph product $\mathcal{G}$ is *not* a very good expander, and in particular there are very small sets with no unique neighbors. To see such an example, recall that the set of generators $\mathcal{Z}$ is identified with $C \times V$. Consider a particular generator $(c, v) \in \mathcal{Z}$. Now consider the set

$$\mathcal{S}(c, v) = \Gamma(c) \times \{v\} \sqcup \{c\} \times \Gamma(v),$$

which is the set of qubits that (c, v) is adjacent to in the graph depicted in Figure 1. This is a small set—indeed, it has constant size if the degree of G is constant—but it actually has *no* unique neighbors. To see why, we refer the reader to Figure 2(a), where we have zoomed in on (c, v) and $\mathcal{S}(c, v)$. Now consider the parity checks contained in $\Gamma(c) \times \Gamma(v) \subseteq V \times C = \mathcal{X}$. As can be seen in Figure 2(a), each such parity-check is connected to *two* qubits in $\mathcal{S}(c, v)$. In particular, if $\mathcal{E} = \mathcal{S}(c, v)$, all of the parity-checks in $\mathcal{X}$ would be satisfied. Moreover, this is unavoidable: the requirement that $\mathcal{C}_X^\perp \subseteq \mathcal{C}_Z$ in fact necessitates this phenomenon.

At first glance then, the decoding task seems impossible, as there exist small sets of errors that cannot be detected. However, the quantum decoding task is not identical to the classical decoding task. In more detail, to correct a quantum error correcting code, it turns out that we do not need to be able to correct *all* small sets of errors, we only need to correct errors up to "toggling" sets of the form $\mathcal{S}(c, v)$. (Formally, the codewords are actually *cosets* modulo $\mathcal{C}_Z^\perp$, and we define the weight of a coset to be the smallest weight of any coset representative; see Section 3).

Thus, the bad example in Figure 2(a) is not actually a bad example after all, because if we "toggle" the set $\mathcal{S}(c, v)$, there are no errors at all. This motivates the definition of a *reduced* error, which is an error that has as

[6] A *unique neighbor* of a set $\mathcal{E}$ is a vertex that has exactly one neighbor in $\mathcal{E}$.

small weight as possible, modulo "toggling" sets like $\mathcal{S}(c,v)$. (That is, a reduced form of an error set $\mathcal{E}$ is a least weight representative in its coset modulo $\mathcal{C}_Z^\perp$).

While this turns out to not be a problem for hypergraph product codes in general, this discussion does highlight several challenges with adapting `Find` to the quantum setting. Below, we discuss these, and our solutions to them, in more detail.

First challenge: treating a single qubit as "suspicious". Given the discussion above, we now see our first challenge in generalizing Viderman's algorithm to the quantum setting. Viderman's algorithm (`Find` in Algorithm 1) works by investigating each bit separately, and deciding whether it is "suspicious" enough—that is, has at least h suspicious neighbors—to add it to the envelope L. But now consider the example in Figure 2(b), where there error set $\mathcal{E}$ consists of *half* of the set $\mathcal{S}(c,v)$ for some generator (c,v). For each qubit in $\mathcal{S}(c,v)$, half of the parity-checks it is connected to are satisfied, and half are unsatisfied, regardless of whether that qubit is in error or not. Further, if we "toggle" the set $\mathcal{S}(c,v)$, we get a completely different set of bad qubits that lead to the same set UNSAT. How can we identify whether a *single* qubit is suspicious in this setting?

The way we deal with this is, instead of checking for suspicious *single* qubits, we check for suspicious *small sets* of qubits, namely every reduced subset $\mathcal{F}$ in each local view $\Gamma(c) \times \Gamma(v)$. We note that this is a similar solution to how `SSFlip` deals with the same issue. However, in the case of Viderman's algorithm, there are a few additional challenges that we must consider.

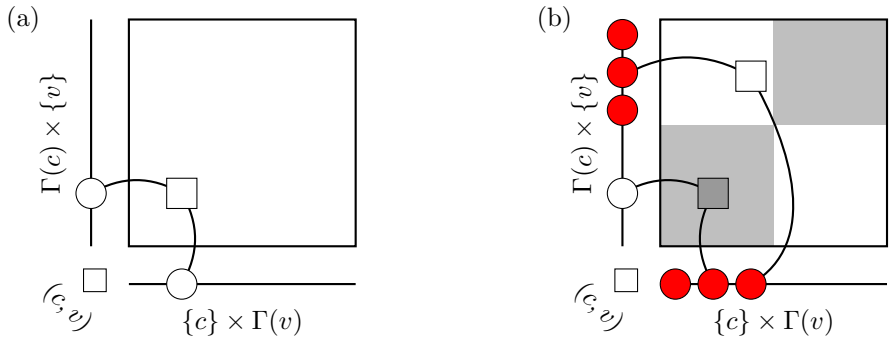

Figure 2: A view of $\mathcal{S}(c,v)$ for a generator (c,v), as well as the parity-checks in $\mathcal{X} = V \times C$ that are connected to $\mathcal{S}(c,v)$. (a) An illustration of the fact that the set $\mathcal{S}(c,v)$ has no unique neighbors. (b) A bad example for naively applying Viderman's algorithm. The error $\mathcal{E}$ is depicted in red, and failed parity-checks are marked in gray. Here, *all* qubits in $\mathcal{S}(c,v)$ look equally "suspicious," in the sense that they are connected to roughly the same number of failed parity-checks.

Second challenge: defining "suspiciousness". Once we decide to look at small sets $\mathcal{F}$, we still need to decide how to measure the "suspiciousness" of a set $\mathcal{F}$, to decide whether to add it to the envelope $\mathcal{L}$.[7] A first attempt would be simply to count the number of suspicious parity checks that $\mathcal{F}$ is connected to (that is, the quantity $|\Lambda(\mathcal{F}) \cap \mathcal{R}|$, where $\Lambda(\cdot)$ denotes the neighborhood[8] of $\mathcal{F}$, and $\mathcal{R}$ denotes the set of suspicious parity checks), normalized by $|\Lambda(\mathcal{F})|$. However, the example Figure 2(b) shows that this will probably not work. Indeed, if $\mathcal{F}$ were the set of errors shown in Figure 2(b), then $|\Lambda(\mathcal{F}) \cap \mathcal{R}|$ is not that large, only about 2/3 of $|\Lambda(\mathcal{F})|$. If we set our threshold as low as 2/3 (relative to $(1-2\epsilon_V)$ in the classical case), it seems likely that the envelope $\mathcal{L}$ will grow extremely large, given that the underlying graph is not a good expander.

Instead, we define a *score* function $\mathsf{score}(\mathcal{F})$ (Definition 4.2). The main differences between the first attempt above and our score function are that we look only at *unique* neighbors of $\mathcal{F}$ in the numerator, and we consider the intersection with the *complement* $\mathcal{R}^c$ instead of $\mathcal{R}$. (We also normalize things slightly differently.) That is, instead of calling a set $\mathcal{F}$ suspicious when $|\Lambda(\mathcal{F}) \cap \mathcal{R}|$ is large (relative to $|\Lambda(\mathcal{F})|$), we instead call it suspicious if $|\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{R}^c|$ is *small* (relative to some measure $\|\mathcal{F}\|$ of the size of $\mathcal{F}$). These seem like small changes, but they have important implications.

[7] In the classical version of Viderman's algorithm, we use "L" and "R" as the envelope and set of suspicious checks, respectively. To avoid confusion, we use $\mathcal{L}$ and $\mathcal{R}$ in the quantum setting.

[8] In the quantum setting, we use Λ to denote neighborhoods, and we use Γ in the classical setting.

In particular, sets $\mathcal{F}$ with lots of "cancellations" (that is, where $\Lambda(\mathcal{F}) \setminus \Lambda^{(u)}(\mathcal{F})$ is large) will in general have smaller unique neighborhoods than sets $\mathcal{F}$ with not as many cancellations. Thus, sets with more cancellations will in general register as more "suspicious" than sets with fewer cancellations.

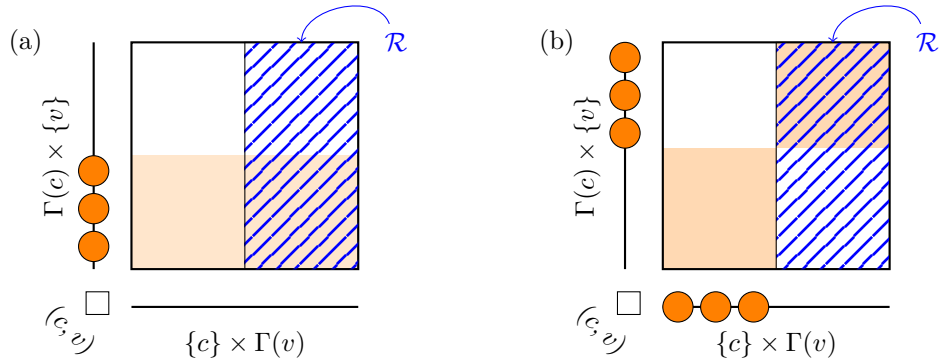

Figure 3: The two sets $\mathcal{F} \subset \mathcal{Q}$ marked as orange circles; and their unique neighborhoods $\Lambda^{(u)}(\mathcal{F}) \subset \mathcal{X}$ marked as shaded orange squares. The set $\mathcal{R}$ of "suspicious" parity checks is drawn in blue lines. The set $\mathcal{F}$ in (b) will count as much more "suspicious" with our score function than the one in (a), intuitively because it has more "cancellations".

To see an example of this, consider the two sets $\mathcal{F}$ shown in Figure 3 (a) and (b). The set shown in (a) would count as *more* suspicious than the set in (b) with either of the score functions $|\Lambda(\mathcal{F}) \cap \mathcal{R}|/|\Lambda(\mathcal{F})|$ or $|\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{R}|/|\Lambda(\mathcal{F})|$ (and would also be more suspicious if we normalized by $\|\mathcal{F}\|$, which is what we actually do in our score function). However, if we ask that $|\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{R}^c|/|\Lambda(\mathcal{F})|$ is small, the set shown in (a) would actually count as *less* suspicous than the set in (b) (and it would be the same if we normalized by $\|\mathcal{F}\|$). Thus, our score function has the desired behavior while the other two candidates do not: sets with more "cancellations" should count as *more* suspicious.

We remark that we view coming up with the "right" score function to be one of the main contributions of our work. As the discussion above shows, there are many options, and subtle differences can be important.

Interlude: the algorithm! With these first two challenges and solutions described, we can now state an informal version of `SSFind` (see Algorithm 3 for the formal version). Intuitively, our algorithm works the same way as the classical Viderman's algorithm `Find`, except that (1) we consider small sets $\mathcal{F}$ rather than single qubits when updating $\mathcal{L}$; and (2) we use our score function score to decide if a set $\mathcal{F}$ should be added to $\mathcal{L}$.

Algorithm 2 High-level description of `SSFind`; see Algorithm 3 for the full version.

Input: $\mathsf{UNSAT} \subseteq \mathcal{X}$
Output: $\mathcal{L} \subseteq \mathcal{Q}$.

$\mathcal{L} \leftarrow \emptyset$
$\mathcal{R} \leftarrow \mathsf{UNSAT}$
while $\exists$ an appropriate set $\mathcal{F}$ such that $\mathsf{score}(\mathcal{F}) \le h$ **do**
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{F}$.
    $\mathcal{R} \leftarrow \mathcal{R} \cup \Lambda(\mathcal{F})$
return $\mathcal{L}$.

Third Challenge: the analysis. Once we have our candidate algorithm, the analysis turns out to be much more delicate than the analysis of the classical Viderman's algorithm (`Find`). We give a brief overview of the proof structure, and the challenges that arise.

First, we recap the analysis of `Find` in the classical case. For completeness, we present a simplified version of the classical analysis (simpler than in [27], but with a worse quantitative result) in Appendix A. We give a high-level overview here, and we suggest that the reader go through Appendix A before reading our proof of correctness for the quantum algorithm.

Intuitively, the analysis of `Find` proceeds in two steps, *coverage* and *bounded growth*. In the coverage step, we need to show that the envelope eventually contains all of the errors $\mathcal{E}$. In the bounded growth step, we need to show that the envelope does not grow too large. Below, we discuss both steps, and explain why they are difficult to generalize to the quantum setting.

Challenge 3a: Coverage Step. At a very high level, the argument for coverage in the classical case is as follows. Suppose that the envelope L does not cover all of $\mathcal{E}$, and let $B = \mathcal{E} \setminus L$ be the part that has not been covered. By expansion of the underlying graph, the unique neighborhood $\Gamma^{(u)}(B)$ of B is large. By an averaging argument, there exists some $v \in B$ with many neighbors in $\Gamma^{(u)}(B)$. However, we claim that $\Gamma^{(u)}(B) \subseteq R$: that is, every element of $\Gamma^{(u)}(B)$ has already been labeled as suspicious. Indeed, either these elements have another neighbor in $\mathcal{E} \setminus B$, in which case they were already labeled as suspicious since $\mathcal{E} \setminus B \subseteq L$ has already been labeled suspicious; or they do not, in which case they were in $\Gamma^{(u)}(\mathcal{E})$ and hence were in UNSAT, which was labeled suspicious at the beginning. But then v has many neighbors in R, meaning it should have already been added to L, a contradiction.

While the coverage proof in the classical case is quite simple, in the quantum case things get much more complicated. It is still true, and not hard to see, that $\Lambda^{(u)}(\mathcal{B}) \subseteq \mathcal{R}$ (where we use the caligraphic letters $\mathcal{B}$ and $\mathcal{R}$ to represent the quantum analogs of B and R). However, we are no longer easily guaranteed the existence of a qubit $q \in \mathcal{B}$ that touches many of these vertices, since our graph does not have expansion. Instead, we employ a delicate argument that leverages expansion of the rows and columns separately. This argument is handled in Section 4.2.

Challenge 3b: Bounded growth step. Again, we recap the high-level argument in the classical case. The basic idea is to bound $|\Gamma(L)|$, the size of the neighborhood of the envelope, in two ways. First, by expansion, $|\Gamma(L)|$ must be large if L is large. On the other hand, consider building L one vertex at a time. Each time we add a vertex v to L, how much can $|\Gamma(\mathcal{L})|$ change? Intuitively, this is not too much, because we only add vertices to L because they have many neighbors in R; but a large part of R made up of $\Gamma(L)$ (the rest is UNSAT). Thus, any vertex has many neighbors in R, hence many neighbors in $\Gamma(L)$, and so does not add too much to $\Gamma(L)$. This gives an upper bound on $|\Gamma(L)|$. If L gets too large, these two bounds yield a contradiction, completing the argument.

We mirror the same basic approach—proving both an upper bound and a lower bound—in the quantum setting, but again it is now much more difficult. For the lower bound, we no longer have good expansion, so we cannot argue immediately that $|\Lambda(\mathcal{L})|$ is large just because $\mathcal{L}$ is large; however, it turns out that we can do this by again leverage the fact that both the rows and columns expand. For the upper bound, one challenge comes from our change to the score function. As discussed above, intuitively the score function says that a set with more "cancellations" should be more suspicious; thus, relative to the classical argument, we may be adding "suspicious" sets whose neighborhoods do not overlap as much with $\mathcal{R}$. However, we can again use the expansion of the rows and columns to show that this term is manageable.

To complete the bounded growth step, which is handled in Section 4.3, we put together the upper bound and the lower bound on $|\Lambda(\mathcal{L})|$, and conclude that $\mathcal{L}$ has not grown too large. This then completes our proof of correctness.

2.3 Comparison with `UnionFind`

As `UnionFind` is also an erasure-conversion algorithm that can be applied to hypergraph product codes [5] (although with asymptotically smaller error radius, as noted above), we briefly describe it to contrast it with `SSFind`. `UnionFind` iteratively maintains and updates a set of disjoint clusters $\{\mathsf{Clust}_i\}$ in the subgraph of $\mathcal{G}$ induced by $\mathcal{Q} \sqcup \mathcal{X}$. Each cluster Clust_i forms a connected component in the graph that includes both parity checks and qubits. The clusters are initialized as the neighborhoods of UNSAT. In each iteration, `UnionFind` tries to find errors $\widetilde{\mathcal{E}}_i$ in each cluster Clust_i such that the syndrome of $\widetilde{\mathcal{E}}_i$ is equal to the unsatisfied parity checks in the interior of Clust_i. If no such error is found, it enlarges Clust_i by adding to it all neighbors of Clust_i. If two clusters Clust_i overlap, then they are merged. The main difference between the `UnionFind` algorithm and a Viderman-style algorithm is that Viderman's algorithm does not add the full neighborhood of suspected parity-checks, but only those that are "suspicious enough".

The clusters Clust_i can be compared to the union of the envelope $\mathcal{L}$ and the suspicious parity checks $\mathcal{R}$. In contrast to `UnionFind`, `SSFind` only adds sets of qubits $\mathcal{F}$ to $\mathcal{L}$ if they have a sufficiently low score. Furthermore, these sets are subsets of the support $\mathcal{S}(z)$ for generators $z \in \mathcal{Z}$. Finally, `SSFind` does not verify whether the envelope contains an error in each iteration; rather, it proceeds until there are no more sets of qubits with sufficiently low score.

3 Background and Definitions

Before we proceed to the proof, we give the formal definitions that we will need.

3.1 Expander graphs Let $V = [n]$ and $C = [m]$, and $G = (V \cup C, E)$ be an undirected bipartite graph that is biregular with left node degree Δ_V and right node degree Δ_C.

DEFINITION 3.1. (GRAPH NEIGHBORHOOD) *For $S_V \subseteq V$ and $S_C \subseteq C$, let $E(S_V, S_T)$ denote the set of edges between S_V and S_T.*

For $S_V \subseteq V$, we denote by $\Gamma(S_V)$ the neighborhood of S_V

$$\Gamma(S_V) = \{c \in C \mid |E(\{c\}, S_V)| \geq 1\} \ . \tag{3.1}$$

For $S_V \subseteq V$, $\Gamma^{(u)}(S_V)$ denotes the unique neighborhood of S

$$\Gamma^{(u)}(S_V) = \{c \in C \mid |E(\{c\}, S_V)| = 1\} \ . \tag{3.2}$$

The definition of neighborhood of $S_C \subset C$ is analogous.

DEFINITION 3.2. (VERTEX EXPANDER) *We say G is an (α_V, ϵ_V) left vertex-expander if for all $S_V \subseteq V$,*

$$|S_V| \leq \alpha_V n \implies |\Gamma(S_V)| \geq (1 - \epsilon_V)\Delta_V |S_V| \ . \tag{3.3}$$

The graph G is an (α_C, ϵ_C) right vertex-expander if

$$|S_C| \leq \alpha_C m \Longrightarrow |\Gamma(S_C)| \geq (1 - \epsilon_C)\Delta_C |S_C| \ . \tag{3.4}$$

The graph G is an $(\alpha_V, \epsilon_V, \alpha_C, \epsilon_C)$ bidirectional vertex expander if it is an $(\alpha_V, \epsilon_V,)$ left vertex-expander and (α_C, ϵ_C) right vertex-expander.

The parameter ϵ_V is related to the number of collisions between outgoing edges of $S_V \subset V$, i.e. vertices in C where multiple edges from S_V are incident. Specifically, by a simple averaging argument we can deduce that for $S_V \subset V$, if $|\Gamma(S_V)| \geq (1 - \epsilon_V)\Delta_V |S_V|$ then also $|\Gamma^{(u)}(S_V)| \geq (1 - 2\epsilon_V)\Delta_V |S_V|$.

3.2 Classical expander codes Expander codes were introduced by Sipser and Spielman [25]. They are built from a good vertex expander, and defined as follows.

DEFINITION 3.3. (EXPANDER CODE) *Let $G = (V \cup C, E)$ be a (Δ_V, Δ_C) bi-regular bipartite graph, which is an (α_V, ϵ_V) left-vertex expander. Denote $|V| = n, |C| = m$.*

The expander code induced by the graph G is the subspace $\mathcal{C} \subset \{0,1\}^n$ such that for every $\mathbf{w} \in \mathcal{C}$ and $c \in C$,

$$\bigoplus_{v \in \Gamma(c)} w_v = 0 \ ,$$

where $\oplus$ is the XOR *function.*

As discussed earlier, Sipser and Spielman suggested a decoding algorithm `Flip` for expander codes, based on bit flips. Viderman [27] suggested a different decoding algorithm, `Find`, which finds a set of "suspicious" bits L that are guaranteed to contain the error $\mathcal{E}$. Viderman shows that as long as the parameter $\epsilon_V \leq 1/3$ and the number of errors is at most

$$|\mathcal{E}| \leq \left(\frac{1 - 3\epsilon_V}{1 - 2\epsilon_V}\right) n,$$

then `Find` is correct, and returns a small envelope.

We have already given a version of the `Find` algorithm in Algorithm 1. For completeness, in Appendix A we give a slightly simpler proof than the one in [27] for the correctness of Algorithm 1. We obtain worse guarantees, but this slightly simpler proof (which is implicit in [27]) is the one we generalize.

3.3 Quantum codes Given a classical code $\mathcal{C} \in \mathbb{F}_2^N$, we let $\mathcal{C}^\perp = \{\mathbf{u} : \forall \mathbf{v} \in \mathcal{C}, \sum_i u_i v_i = 0 \pmod 2\}$ denote its dual. An $[\![N, K, D]\!]$ CSS quantum code $\mathcal{H}$ on a set of N qubits $\mathcal{Q}$ is specified by two classical codes $\mathcal{C}_Z, \mathcal{C}_X \subseteq \mathbb{F}_2^N$ that obey the relation $\mathcal{C}_X^\perp \subseteq \mathcal{C}_Z$. The codes $\mathcal{C}_X^\perp$ and $\mathcal{C}_Z^\perp$ form the parity checks for the quantum code. We let $\mathcal{X}$ and $\mathcal{Z}$ be the basis for $\mathcal{C}_X^\perp$ and $\mathcal{C}_Z^\perp$ respectively.

The number of encoded qubits K is defined as

$$K = \dim(\mathcal{C}_X/\mathcal{C}_Z^\perp) = \dim(\mathcal{C}_X) + \dim(\mathcal{C}_Z) - N \ . \tag{3.5}$$

The distances D_Z and D_X are defined as

$$D_Z = \min\{|\mathbf{e}| : \mathbf{e} \in \mathcal{C}_X \setminus \mathcal{C}_Z^\perp\} \ , \quad D_X = \min\{|\mathbf{e}| : \mathbf{e} \in \mathcal{C}_Z \setminus \mathcal{C}_X^\perp\} \ . \tag{3.6}$$

The distance of the code $\mathcal{H}$ is defined as $D = \min(D_X, D_Z)$.

We identify the set of qubits $\mathcal{Q}$ with the set $\{1, ..., N\}$ and vectors $\mathbf{e} \in \mathbb{F}_2^N$ with their support $\mathcal{E} \subseteq \mathcal{Q}$. Addition of vectors mod 2 corresponds to the symmetric difference of sets, denoted $\oplus$: For $\mathcal{E}_1, \mathcal{E}_2 \subseteq \mathcal{Q}$, we have $\mathcal{E}_1 \oplus \mathcal{E}_2 = [\mathcal{E}_1 \setminus \mathcal{E}_2] \cup [\mathcal{E}_2 \setminus \mathcal{E}_1]$.

For $z \in \mathcal{Z}$ and $\chi \in \mathcal{X}$, $\mathcal{S}(z) \subseteq \mathcal{Q}$ and $\mathcal{S}(\chi) \subseteq \mathcal{Q}$ denote the supports of the corresponding basis element.

It turns out that X and Z errors can be corrected separately. As the two cases mirror each other, we will only consider Z-type errors. The X-type parity checks are used to correct them and we shall refer to these as parity checks. In this case, we are interested in the syndromes for the X-type parity checks. For an error $\mathcal{E}$, we let UNSAT denote the X-type syndrome (unsatisfied X-type parity checks):

$$\mathsf{UNSAT} = \{\chi || \mathcal{S}(\chi) \cap \mathcal{E}| \neq 0 \pmod 2\} \ . \tag{3.7}$$

The Z-type parity checks are not treated as constraints in this view; rather they define $\mathcal{C}_Z^\perp$, the set of words equivalent to the trivial codeword. We refer to the set $\mathcal{Z}$ of Z-type parity checks as *generators*.

3.4 Hypergraph Product Codes Let $G = (V \cup C, E)$ be a (Δ_V, Δ_C)-biregular bipartite graph. Suppose it is an $(\alpha_V, \epsilon_V, \alpha_C, \epsilon_C)$ bidirectional vertex-expander, with $|V| = n$ and $|C| = m$. Recall that we denote by Γ the neighborhood in the graph G. We assume that the graph is unbalanced, i.e. that $n > m$ and $\Delta_C > \Delta_V$.

The $[n, k, d]$ code $\mathcal{C}(G)$ is the code defined on Definition 3.3, where V and C are associated with bits and parity checks respectively. The $[m, k^T, d^T]$ code $\mathcal{D}(G)$ is the code associated with the graph G, in which C and V swap roles: V is associated with parity checks and C with the bits. For convenience, we say that $d = \infty$ ($d^T = \infty$) if $k = 0$ ($k^T = 0$).

The hypergraph product code $\mathcal{H}$ is an $[\![N, K, D]\!]$ code where $N = n^2 + m^2$, $K = (k)^2 + (k^T)^2$ and $D = \min(d, d^T)$.[9] For the proof of these facts, we point the reader to the original paper by Tillich and Zémor [26]. We proceed to describe the code construction.

DEFINITION 3.4. *Let $G_1 = (V_1 \cup C_1, E_1)$ and $G_2 = (V_2 \cup C_2, E_2)$ denote two copies of G. The hypergraph product code $\mathcal{H}$ is constructed from the graph product $\mathcal{G} := G_1 \times G_2$, where,*

1. *The set of qubits, denoted $\mathcal{Q}$, is associated with $\mathcal{Q}_V \sqcup \mathcal{Q}_C$, where $\mathcal{Q}_V = V_1 \times V_2$ and $\mathcal{Q}_C = C_1 \times C_2$.*
2. *The set of Z stabilizer generators, denoted $\mathcal{Z}$, is associated with $C_1 \times V_2$. We shall let z denote a generator and write $z \sim (c, v) \in C_1 \times V_2$ to say z is identified with the tuple (c, v). The support $\mathcal{S}(z)$ of the generator z is*

$$\mathcal{S}(z) = \{(\nu, v) : \nu \in \Gamma(c)\} \cup \{(c, \zeta) : \zeta \in \Gamma(v)\} \ . \tag{3.8}$$

3. *The set of X stabilizer generators, denoted $\mathcal{X}$, is associated with $V_1 \times C_2$. For $\nu \in V_1$ and $\zeta \in C_2$, the support $\mathcal{S}(\chi)$ of a parity check $\chi \sim (\nu, \zeta)$ is*

$$\mathcal{S}(\chi) = \{(\nu, v) : v \in \Gamma(\zeta)\} \cup \{(c, \zeta) : c \in \Gamma(\nu)\} \ . \tag{3.9}$$

[9] Technically the actual distance may be larger than $\min(d, d^T)$, which is the *design* distance, so we should have written $D \geq \min(d, d^T)$. However, for simplicity, we will refer to the design distance as simply the "distance," and denote it by D.

We refer to Fig. 1 for an illustration.

Throughout, we use Roman letters for C_1 and V_2, and Greek letters for C_2 and V_1. Thus, generators are Roman tuples like (c, v), while parity checks are Greek tuples like (ν, ζ).

We note that in this case, the codes $\mathcal{C}_Z^\perp$ and $\mathcal{C}_X^\perp$ are given by

$$\mathcal{C}_Z^\perp = \text{span}(\mathcal{S}(c_1, v_2) : c_1 \in C_1, v_2 \in V_2) \ , \quad \mathcal{C}_X^\perp = \text{span}(\mathcal{S}(\nu_1, \zeta_2) : \nu_1 \in V_1, \zeta_2 \in C_2) \ .$$

If we ignore the generators $\mathcal{Z}$, we get a bipartite graph between the qubits $\mathcal{Q}$ and then X-type parity checks $\mathcal{X}$ (that is, the induced graph of $\mathcal{G}$ on the vertices $\mathcal{Q} \sqcup \mathcal{X}$). We denote this graph by $\mathcal{G}_X$.

To make the notations clearer, we use different symbols the neighborhood in the classical graph G and in the quantum graph $\mathcal{G}$. In the classical setting, we use Γ, while in the quantum setting, we use Λ. Formally, we have the following definitions.

DEFINITION 3.5. (QUANTUM NEIGHBORHOOD) *Let $G = (V \cup C, E)$ be a graph and $\mathcal{G}_X$ be the graph induced by qubits $\mathcal{Q}$ and parity checks $\mathcal{X}$. For a qubit $q \in \mathcal{Q}$, we denote its neighborhood by $\Lambda(q)$, defined as follows:*

$$\Lambda(\{\nu_1, v_2\}) = \{(\nu_1, \zeta_2) | \zeta_2 \in \Gamma(v_2)\} \ , \qquad \Lambda(\{c_1, \zeta_2\}) = \{(\nu_1, \zeta_2) | \nu_1 \in \Gamma(c_1)\} \ ,$$

where Γ is the neighborhood in the classic graph G.

This naturally extends to a set of qubits, $\mathcal{A} \subseteq \mathcal{Q}$, $\Lambda(\mathcal{A}) = \cup_{q \in \mathcal{A}} \Lambda(\mathcal{A})$.

The unique neighborhood $\Lambda^{(u)}(\mathcal{A})$ and multineighborhood $\Lambda^{(m)}(\mathcal{A})$ mirror their classical definitions:

$$\Lambda^{(u)}(\mathcal{A}) = \{\chi | \text{ there is a unique } q \in \mathcal{Q} \text{ such that } \chi \in \Lambda(q)\} \ , \quad \Lambda^{(m)}(\mathcal{A}) = \Lambda(\mathcal{A}) \setminus \Lambda^{(u)}(\mathcal{A}) \ .$$

For a generator $z \in \mathcal{Z}$, we shall abuse notation to write $\Lambda(z)$ to refer to $\Lambda(\mathcal{S}(z))$.

In addition, we will use the following notation:

- For a set of qubits $\mathcal{A} \in \mathcal{Q}$, let $\mathcal{A}_V = \mathcal{A} \cap \mathcal{Q}_V$ and $\mathcal{A}_C = \mathcal{A} \cap \mathcal{Q}_C$.
- Let $\Delta = \Delta_C \times \Delta_V$.
- Let $\epsilon = \max(\epsilon_C, \epsilon_V)$.

Finally, we define a notion that will be useful in our analysis.

DEFINITION 3.6. (WEIGHTED NORM) *For $\mathcal{A} \subseteq \mathcal{Q}$, we define $\|\mathcal{A}\| = \frac{|\mathcal{A}_V|}{\Delta_C} + \frac{|\mathcal{A}_C|}{\Delta_V}$.*

3.5 Projections of sets In this section, we introduce some notation for referring to projections of sets on to the component graphs G_1 and G_2. This notation will be used for all projections throughout this paper.

DEFINITION 3.7. (PROJECTION) *Let $\mathcal{G} = (\mathcal{Q}, \mathcal{X})$ be a graph that is a product of G_1 and G_2.*

For all $\nu_1 \in V_1$ and all $v_2 \in V_2$, let $\mathcal{A}(v_2) \subset V_1, \mathcal{A}(\nu_1) \subset V_2$ be the projections on to G_1 and G_2 respectively:

$$\mathcal{A}(v_2) = \{\nu_1' | (\nu_1', v_2) \in \mathcal{A}_V\} \ , \qquad \mathcal{A}(\nu_1) = \{v_2' | (\nu_1, v_2') \in \mathcal{A}_V\} \ .$$

We obtain $\mathcal{A}_V^1 \subset V_1$ and $\mathcal{A}_V^2 \subset V_2$ as the union of sets

$$\mathcal{A}_V^1 = \bigcup_{v_2} \mathcal{A}(v_2) \ , \qquad \mathcal{A}_V^2 = \bigcup_{\nu_1} \mathcal{A}(\nu_1) \ . \tag{3.10}$$

Similarly, for all $c_1 \in C_1$ and $\zeta_2 \in C_2$, let $\mathcal{A}(c_1) \subset C_2$ and $\mathcal{A}(c_2) \subset C_1$ denote the projections on to G_1 and G_2 respectively:

$$\mathcal{A}(c_1) = \{\zeta_2' | (c_1, \zeta_2') \in \mathcal{A}_C\} \ , \quad \mathcal{A}(\zeta_2) = \{c_1' | (c_1', \zeta_2) \in \mathcal{A}_C\} \ . \tag{3.11}$$

We obtain $\mathcal{A}_C^1 \subset C_1$ and $\mathcal{A}_C^2 \subset C_2$ as the union of sets

$$\mathcal{A}_C^2 = \bigcup_{c_1 \in C_1} \mathcal{A}(c_1) \ , \quad \mathcal{A}_C^1 = \bigcup_{\zeta_2 \in C_2} \mathcal{A}(\zeta_2) \ . \tag{3.12}$$

We refer to Figure 4 for a schematic.

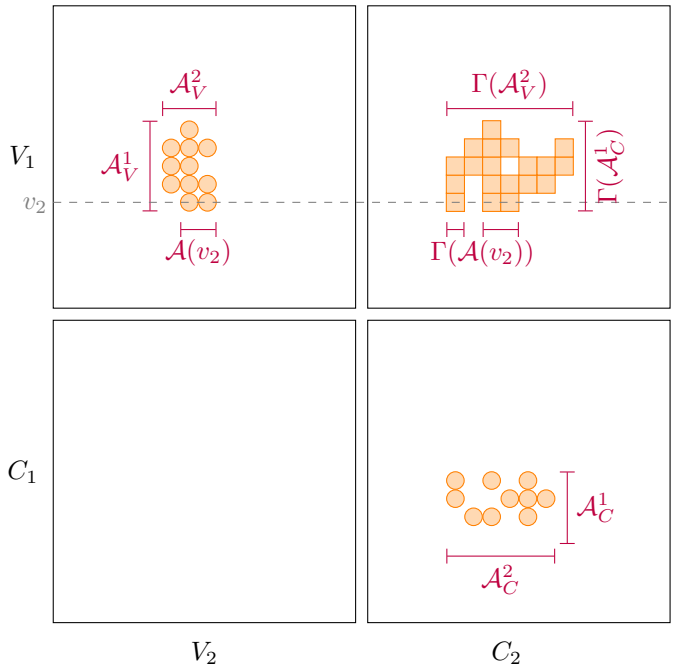

Figure 4: Qubits that form $\mathcal{A}$ are denoted using circular nodes in $V_1 \times V_2$ and $C_1 \times C_2$. The projections $\mathcal{A}_V^1$, $\mathcal{A}_V^2$, $\mathcal{A}_C^1$ and $\mathcal{A}_C^2$ of $\mathcal{A}$ are indicated. The neighbors $\Lambda(\mathcal{A})$ within $V_1 \times C_2$ are depicted using square nodes; the projections $\Gamma(\mathcal{A}_V^2)$ and $\Gamma(\mathcal{A}_C^1)$ are indicated. For a specific $v_2 \in V_2$, the neighbors $\Lambda(\mathcal{A}(v_2) \times \{v_2\}) = \Gamma(\mathcal{A}(v_2)) \times \{v_2\}$ are also depicted along the dashed line.

3.6 Properties of reduced sets Recall from Section 2 that we are allowed to "toggle" elements of $\mathcal{S}(z)$ for $z \in \mathcal{Z}$ without changing the codeword (that is, as above, we are working modulo $\mathcal{C}_Z^\perp$). To this end, we define a *reduced* error as one of minimum weight modulo this toggling. Formally, we have the following definition:

DEFINITION 3.8. (REDUCED REPRESENTATION) *Given $\mathcal{E} \subset \mathcal{Q}$, let $[\mathcal{E}]$ be the equivalence class of $\mathcal{E}$ defined by*

$$[\mathcal{E}] = \{\mathcal{E} \oplus \mathcal{S} | \mathcal{S} \in \mathcal{C}_Z^\perp\} \ .$$

The reduced representation $\overline{\mathcal{E}} \in [\mathcal{E}]$ *is the smallest element in $[\mathcal{E}]$. If there are several elements with the minimum weight, we pick one arbitrarily.*

Similarly, we can define a locally reduced set in the support of a generator.

DEFINITION 3.9. (LOCALLY REDUCED SET) *Let z be a generator and let $\mathcal{S}(z)$ be its support. We say that a set $\mathcal{F} \subseteq \mathcal{S}(z)$ is* locally reduced *if*

$$|\mathcal{F}_V| + |\mathcal{F}_C| \leq (\Delta_V - |\mathcal{F}_V|) + (\Delta_C - |\mathcal{F}_C|) \ . \tag{3.13}$$

If $\mathcal{F}$ is locally reduced then we can bound the size of $|\mathcal{F}_V| \cdot |\mathcal{F}_C|$ via the following lemma, where we recall the notation $\|\mathcal{F}\|$ from Definition 3.6.

LEMMA 3.1. *If $\mathcal{F} \subseteq \mathcal{S}(z)$ is locally reduced, then*

$$|\mathcal{F}_V| \cdot |\mathcal{F}_C| \leq \frac{1}{4}\Delta \|\mathcal{F}\| \ .$$

Proof. We can write the claim as

$$|\mathcal{F}_V| \left(\frac{\Delta_V}{2} - |\mathcal{F}_C|\right) + |\mathcal{F}_C| \left(\frac{\Delta_C}{2} - |\mathcal{F}_V|\right) \geq 0 \ . \tag{3.14}$$

Consider the LHS, we can write it as

$$\min(|\mathcal{F}_V|, |\mathcal{F}_C|) \cdot \left[\left(\frac{\Delta_V}{2} - |\mathcal{F}_C| \right) + \left(\frac{\Delta_C}{2} - |\mathcal{F}_V| \right) \right] \geq 0 \tag{3.15}$$

By assumption, $\mathcal{F}$ is locally reduced and this implies

$$|\mathcal{F}_V| + |\mathcal{F}_C| \leq \frac{1}{2} (\Delta_V + \Delta_C) \ . \tag{3.16}$$

Furthermore, $\min(|\mathcal{F}_V|, |\mathcal{F}_C|)$ is non-negative. ∎

4 Viderman's algorithm for Hypergraph Product Codes

In this section, we present a quantum version of Viderman's algorithm for hypergraph product codes, and prove that it is correct. The objective of this section is to prove Theorem 1.1.

Our algorithm, `SSFind`, is presented in Section 4.1. Given a set of Z-type errors $\mathcal{E} \subseteq \mathcal{Q}$, `SSFind` produces an envelope $\mathcal{L}$ such that it contains the reduced version of the error $\mathcal{E}$. In Section 4.2, we prove that $\mathcal{L}$ contains the reduced version of $\mathcal{E}$. In Section 4.3, we prove that when $\mathcal{E}$ is small enough, $\mathcal{L}$ does not grow too much.

For the entirety of this section, we assume we have a quantum expander code that is constructed from a graph $\mathcal{G}$, that is the graph product of two copies of a graph G that is a (Δ_V, Δ_C)-biregular, $(\alpha_V, \epsilon_V, \alpha_C, \epsilon_C)$-bidirectional expander as described in Section 3.4. For clarity, we denote the two copies as $G_1 = (V_1 \cup C_1, E_1), G_2 = (V_2 \cup C_2, E_2)$.

4.1 `SSFind` We begin by recalling from Section 2 why `Find` (Algorithm 1, the classical version of Viderman's algorithm) does not extend directly to the quantum setting. In each iteration, the classical algorithm `Find` adds a single bit to the envelope $\mathcal{L}$ if it has a lot of overlap with the set of untrustworthy checks $\mathcal{R}$. To be precise, the `Find` algorithm adds a bit v to the envelope if at least $(1 - 2\epsilon_V) \cdot \Delta_V$ of the neighbors of v are in $\mathcal{R}$. Suppose we encounter the example in Figure 5 (a reprise of Figure 2). There are errors only within the support of a single generator $z \sim (c, v) \in \mathcal{Z}$, and these errors are in both $V_1 \times V_2$ qubits and in $C_1 \times C_2$ qubits. The shaded rectangles indicate UNSAT. Parity checks that are adjacent to two errors are satisfied, and therefore no error qubit has $(1 - 2\epsilon_V)\Delta_V$ unsatisfied parity checks adjacent to it. If we reduce the threshold of the `Find` algorithm to cover errors in these cases, we would have to reduce it from $(1 - 2\epsilon_V)\Delta_V$ to $1/2\Delta_V$,

and then the envelope might end up growing uncontrollably and cover the entire set of qubits.

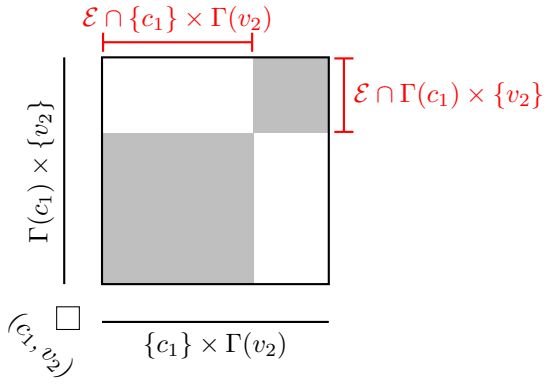

Figure 5: The neighborhood of a generator (c_1, v_2). Error is marked in red and unsatisfied parity checks are shaded.

As discussed in Section 2, we overcome this problem by taking into consideration neighborhoods of *sets* of qubits rather than the neighborhood of one qubit at a time. Let $\mathcal{S}(z)$ be the support of $z \in \mathcal{Z}$. In Definition 4.2, we define a score over sets $\mathcal{F} \subset \mathcal{S}(z)$ that, as discussed in Section 2, accounts for the fact that parity checks that witness *both* VV errors and CC errors can be satisfied; that is, "cancellations" can occur. The score is then used in the algorithm `SSFind` (Algorithm 3) to identify "suspicious" qubits.

DEFINITION 4.1. *For each generator $z \in \mathcal{Z}$, we define the collection of locally reduced sets in the neighborhood of z to be*

$$M(z) = \left\{ \mathcal{F} \subseteq \mathcal{S}(z) \;\middle|\; |\mathcal{F}_V| + |\mathcal{F}_C| \leq \frac{1}{2}(\Delta_V + \Delta_C) \right\} .$$

Let $M(\mathcal{Z}) = \bigcup_{z \in \mathcal{Z}} M(z)$ be the union over locally reduced sets across all generators.

The `SSFind` algorithm will maintain an envelope $\mathcal{L}$ of qubits it believes are suspicious, and a set of parity checks $\mathcal{R}$ that are also suspicious. The algorithm labels a parity check as suspicious whenever it touches a suspicious qubit, and thus we always have $\Lambda(\mathcal{L}) \subset \mathcal{R}$. In each iteration, `SSFind` searches over $\mathcal{F} \in M(\mathcal{Z})$ and evaluates their *score*, which we formally define below. If the score of a set $\mathcal{F}$ is small, it is considered suspicious. See Section 2 for a discussion of the intuition behind this score function.

DEFINITION 4.2. (SCORE) *Let $\mathcal{R} \subseteq \mathcal{X}$ be a set of parity checks and let $\mathcal{R}^c$ be its complement. For $\mathcal{F} \in M(\mathcal{Z})$, we define the* score *of $\mathcal{F}$ as*

$$\mathsf{score}(\mathcal{F}) = \frac{|\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{R}^c|}{\Delta \|\mathcal{F}\|} .$$

We note that the score of a set $\mathcal{F}$ depends on the set of suspected parity checks $\mathcal{R}$. However, we suppress this dependence in the notation for readability. While the set $\mathcal{R}$ changes over the course of the algorithm, the score of subsets of qubits $\mathcal{F}$ within the support of a generator z only changes if the set of parity checks in the local view are added to $\mathcal{R}$.

To illustrate, we use an example and corresponding Figure 6. First, $\mathcal{F}$ is chosen such that it is locally reduced.

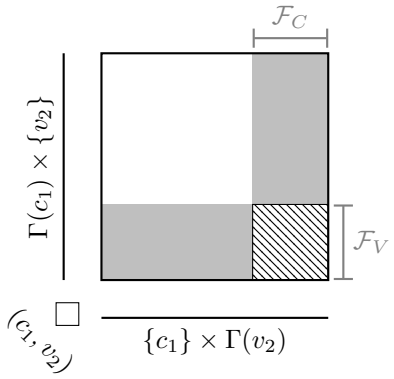

Figure 6: The local view of a generator $z \sim (c_1, v_2)$. $\mathcal{F}_V$ and $\mathcal{F}_C$ form the reduced set $\mathcal{F}$. The gray portions correspond to $\Lambda^{(u)}(\mathcal{F})$. The hatched portion corresponds to $\Lambda^{(m)}(\mathcal{F})$.

If $\mathcal{F}$ is the only error in the code, the syndromes would correspond to the union of the gray rectangles.

In Algorithm 3, we present `SSFind`. Let $\mathcal{E} \subseteq \mathcal{Q}$ be the error and UNSAT denote the set of unsatisfied parity checks. Recall that we denote $\epsilon = \max(\epsilon_V, \epsilon_C)$. The algorithm `SSFind` takes as input UNSAT. It outputs an envelope $\mathcal{L} \subset \mathcal{Q}$ that are suspected errors.

Algorithm 3 `SSFind`

Input: $\mathsf{UNSAT} \subseteq \mathcal{X}$
Output: $\mathcal{L} \subseteq \mathcal{Q}$.

$\mathcal{L} \leftarrow \emptyset$
$\mathcal{R} \leftarrow \mathsf{UNSAT}$
$\mathcal{F} \leftarrow M(\mathcal{Z})$
while $\exists \mathcal{F} \in M(\mathcal{Z})$ such that $\mathsf{score}(\mathcal{F}) \leq 2\epsilon$ **do**
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{F}$.
    $\mathcal{R} \leftarrow \mathcal{R} \cup \Lambda(\mathcal{F})$
    for $\mathcal{F}' \in M(\mathcal{Z})$ such that $\mathcal{F}' \cap \mathcal{F} \neq \emptyset$ **do**
        $M(\mathcal{Z}) \leftarrow M(\mathcal{Z}) \setminus \{\mathcal{F}'\}$
return $\mathcal{L}$.

We remark that there exists an implementation of this algorithm that runs in time $\Theta(|\mathcal{E}|)$, see Section 4.4.2 for more details.

4.2 Coverage In this section, we prove that `SSFind` produces an envelope $\mathcal{L}$ that covers the reduced error $\mathcal{E}$. To this end, we shall assume that it does not and arrive at a contradiction.

Without loss of generality, assume the error is already in its reduced form, i.e. $\overline{\mathcal{E}} = \mathcal{E}$. Let $\mathcal{B} = \mathcal{L} \setminus \mathcal{E}$ be the residual errors that are not covered by the envelope when `SSFind` terminates. For the sake of contradiction, we assume $\mathcal{B}$ is not empty.

We begin with some high-level intuition and then build on it to arrive at the proof. The idea is to consider the set $\Lambda^{(u)}(\mathcal{B})$, the unique neighbors of the residual errors. The key observation is that elements of the set $\Lambda^{(u)}(\mathcal{B})$ must be in $\mathcal{R}$.

LEMMA 4.1. *Let $\chi \in \mathcal{X}$ such that $\chi \sim (\nu_1, \zeta_2)$ is an element of $\Lambda^{(u)}(\mathcal{B})$. Then $\chi \in \mathcal{R}$.*

Proof. If $\chi \in \Lambda^{(u)}(\mathcal{B})$, then by definition, it is incident to exactly one qubit in $\mathcal{B}$. This leaves two choices. If this parity check is not connected to an element from $\mathcal{L}$, then it must be connected only to a single error (as $\mathcal{E} = (\mathcal{E} \cap \mathcal{L}) \sqcup \mathcal{B}$), and therefore must be in $\mathsf{UNSAT} \subseteq \mathcal{R}$. On the other hand, if it *is* connected to an element from $\mathcal{L}$, then it must be in $\Lambda(\mathcal{L}) \subset \mathcal{R}$. ∎

Intuitively, if $\Lambda^{(u)}(\mathcal{B})$ is large, then there are elements in $\mathcal{B}$ such that their neighborhood has a large overlap with $\mathcal{R}$. In particular, we will show that $\Lambda^{(u)}(\mathcal{B})$ has a large overlap with the neighborhood of a generator $z \in \mathcal{Z}$. This will imply that z contains a portion of $\mathcal{B}$ with a large score. In turn, this will lead to a contradiction of the assumption that the `SSFind` algorithm terminated.

The following lemma closely resembles Lemma 7 in [18].

LEMMA 4.2. *The output $\mathcal{L}$ of* `SSFind` *covers the set $\mathcal{E}$, i.e. $\mathcal{E} \subseteq \mathcal{L}$ if $|\mathcal{E}_V| \leq \alpha_V n$ and $|\mathcal{E}_C| \leq \alpha_C m$.*

Proof. Assume without loss of generality that $\mathcal{B}_V \neq \emptyset$. If $\mathcal{B}_V = \emptyset$, we can replace V with C and rows with columns. Let $\mathcal{B}_V^2 \subset V_2$ be the projection of $\mathcal{B}$, as defined in Definition 3.7:

$$\mathcal{B}_V^2 = \{v_2 \in V_2 | \exists (\nu_1, v_2) \in \mathcal{B}\} \ .$$

Notice that the set $\mathcal{B}_V^2 \subset V_2$, which is a set in the graph G_2.

As $|\mathcal{B}_V^2| \leq |\mathcal{B}| \leq |\mathcal{E}| \leq \min(\alpha_V n, \alpha_C m)$ and the graph G_2 is an (α_V, ϵ_V) left vertex-expander,

$$|\Gamma^{(u)}(\mathcal{B}_V^2)| \geq (1 - 2\epsilon_V)\Delta_V |\mathcal{B}_V^2| \ . \tag{4.17}$$

This then implies that there exists a node $v_2 \in \mathcal{B}_V^2$ such that it has a large overlap with the unique neighborhood:

$$|\Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)| \geq (1 - 2\epsilon_V)\Delta_V \ . \tag{4.18}$$

We fix this node v_2 for the rest of the proof.

Let $\mathcal{B}_C^2 \subset C_2$ be as in Definition 3.7, i.e. $\mathcal{B}_C^2 = \{\zeta_2 \in C_2 | \exists (c_1, \zeta_2) \in \mathcal{B}\}$. We divide into two cases.

Classical-like case: First, consider the simpler case where

$$\Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2) \cap \mathcal{B}_C^2 = \emptyset \ . \tag{4.19}$$

This is the setting where most parity checks adjacent to a qubit of the form $(\nu_1, v_2) \in \mathcal{B}_V$ are not simultaneously adjacent to CC qubits. In this sense, the setting is like a classical decoding problem where we do not consider interference from CC qubits.

Let $\nu_1 \in V_1$ be an element such that $(\nu_1, v_2) \in \mathcal{B}_V$, there must be one since $v_2 \in \mathcal{B}_V^2$. Assuming Eq. (4.19), it follows that for all $c_1 \in \Gamma(\nu_1)$,

$$\Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2) \cap \mathcal{B}(c_1) = \emptyset, \tag{4.20}$$

where $\mathcal{B}(c_1)$ is the projection of $\mathcal{B}$ on the row containing c_1, see Definition 3.7. This is because, by definition, $\mathcal{B}(c_1) \subseteq \mathcal{B}_C^2$. Fix such $c_1 \in \Gamma(\nu_1)$ and let z be (c_1, v_2). Equation (4.20) means that the neighborhood of the generator $z \in \mathcal{Z}$ is simple, as $\{c_1\} \times \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)$ contains no elements from $\mathcal{B}$. See Figure 7.

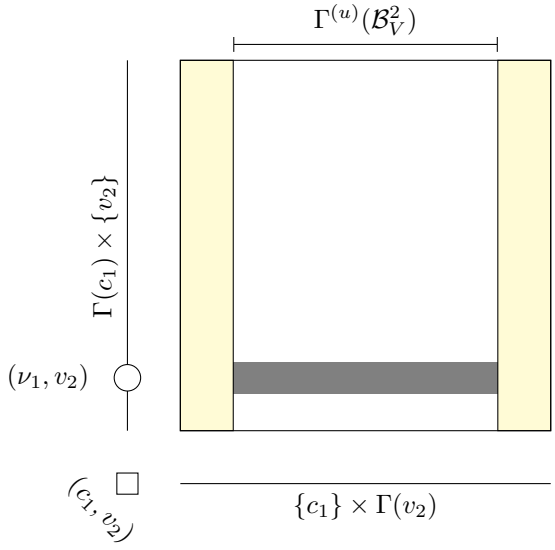

Figure 7: We find a generator $z \sim (c_1, v_2)$ such that $(\nu_1, v_2) \in \mathcal{E}_V$ is in its support and $\left[\{c_1\} \times \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)\right] \cap \mathcal{B} = \emptyset$. We prove that the gray line is in $\mathcal{R}$, i.e. $\{\nu_1\} \times (\Gamma^{(u)}(\mathcal{E}_V^2) \cap \Gamma(v_2)) \subset \mathcal{R}$. We ignore the parity checks corresponding to the yellow portions. We write $\Gamma^{(u)}(\mathcal{B}_V^2)$ for $\Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)$ for readability of the diagram.

Let $\chi = (\nu_1, \zeta_2)$ be a parity check, where $\zeta_2 \in \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)$. We claim that χ sees only a single qubit in $\mathcal{B}$, the qubit (ν_1, v_2). The parity check χ cannot be adjacent to other elements of $\mathcal{B}_V$ because $\zeta_2 \in \Gamma^{(u)}(\mathcal{E}_V^2) \cap \Gamma(v_2)$. Furthermore, it cannot be adjacent to other elements of $\mathcal{B}_C$ because of (4.20).

From Lemma 4.1, χ is an element of $\mathcal{R}$. This implies that $|\Lambda^{(u)}(\{\nu_1, v_2\}) \cap \mathcal{R}|$ is lower bounded:

$$|\Lambda^{(u)}(\{\nu_1, v_2\}) \cap \mathcal{R}| \geq \left|\{\nu_1\} \times \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)\right| \tag{4.21}$$

$$\geq (1 - 2\epsilon_V)\Delta_V \ . \tag{4.22}$$

In other words, the score is upper bounded:

$$\mathsf{score}(\{\nu_1, v_2\}) = \frac{|\Lambda^{(u)}(\{\nu_1, v_2\}) \cap \mathcal{R}^c|}{\Delta_V} \leq 2\epsilon \ . \tag{4.23}$$

This means that $\mathcal{F} = \{(\nu_1, v_2)\}$ would have been added to $\mathcal{L}$ and the algorithm could not have terminated.

Quantum-like case: Suppose that the VV and CC portions of $\mathcal{B}$ interfere, i.e.

$$\mathcal{D}_C^2 := \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2) \cap \mathcal{B}_C^2 \neq \emptyset \ . \tag{4.24}$$

We shall refer to $\mathcal{D}_C^2 \subset C_2$ as the induced set. If $\mathcal{D}_C^2$ is not empty, then we can construct the set $\mathcal{D}_C^1$

$$\mathcal{D}_C^1 = \left\{ c_1 \in C_1 | \exists \zeta_2 \in \mathcal{D}_C^2 \text{ such that } (c_1, \zeta_2) \in \mathcal{B}_C \right\} \ . \tag{4.25}$$

The induced set, being a subset of $\mathcal{B}_C^1$, must be small

$$|\mathcal{D}_C^1| \leq |\mathcal{B}_C^1| \leq |\mathcal{B}_C| \leq |\mathcal{E}_C| \leq \min(\alpha_V n, \alpha_C m) \ . \tag{4.26}$$

Therefore, it has a large unique neighborhood in G_1, $|\Gamma^{(u)}(\mathcal{D}_C^1)| \geq (1 - 2\epsilon_C)\Delta_C |\mathcal{D}_C^1|$. Consequently, there must be some $c_1 \in \mathcal{D}_C^1$ such that

$$|\Gamma^{(u)}(\mathcal{D}_C^1) \cap \Gamma(c_1)| \geq (1 - 2\epsilon_C)\Delta_C \ . \tag{4.27}$$

We identify the generator $z = (c_1, v_2)$, and show that this generator has a large overlap with $\Gamma^{(u)}(\mathcal{B})$. See Figure 8.

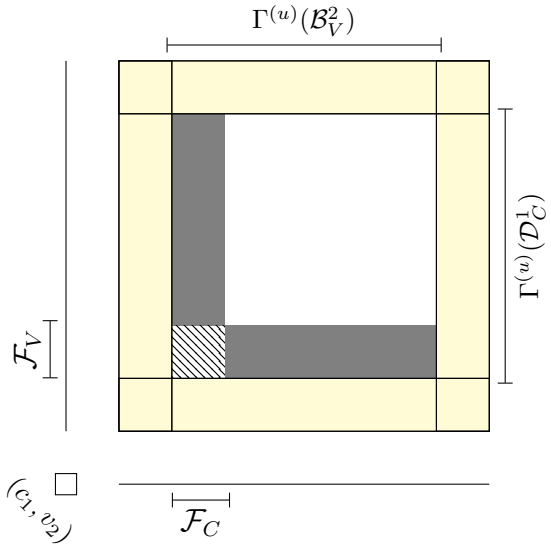

Figure 8: $\mathcal{F}_V, \mathcal{F}_V \subset \mathcal{E}$, and the parity checks in $\mathcal{U} = \Gamma^{(u)}(\mathcal{B}_V^2) \times \Gamma^{(u)}(\mathcal{D}_C^1)$ only interact with errors in $\mathcal{S}(z)$, the support of z. We show that the parity checks in gray, $\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{U}$, are in $\mathcal{R}$. We ignore the parity checks corresponding to the yellow portions. We write $\Gamma^{(u)}(\mathcal{B}_V^2)$ for $\Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)$ and $\Gamma^{(u)}(\mathcal{D}_C^1)$ for $\Gamma^{(u)}(\mathcal{D}_C^1) \cap \Gamma(c_1)$ for readibility of the diagram.

Consider the set $\mathcal{F} = \mathcal{F}_V \sqcup \mathcal{F}_C$, where

$$\mathcal{F}_V := \left(\mathcal{B}(v_2) \cap \Gamma^{(u)}(\mathcal{B}_C^1) \cap \Gamma(c_1) \right) \times \{v_2\} \ , \qquad \mathcal{F}_C := \{c_1\} \times \left(\mathcal{B}(c_1) \cap \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2) \right) \ . \tag{4.28}$$

Note that all elements in $\mathcal{F}$ are errors, from the definition of $\mathcal{B}(v_2), \mathcal{B}(c_1)$, see Definition 3.7.

We claim that the set $\mathcal{F}_C$ is not empty. By construction, $c_1 \in \mathcal{D}_C^1$, and therefore, for some $\zeta_2 \in \mathcal{D}_C^2$, we must have $(c_1, \zeta_2) \in \mathcal{B}_C$. By the definition of $\mathcal{D}_C^2$, this means that $\zeta_2 \in \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)$.

As in the classical-like case, we want to show that most neighbors of $\mathcal{F}$ are unique neighbors of $\mathcal{B}$ and therefore in $\mathcal{R}$. We study this generator using the set of parity checks $\mathcal{U} = \Gamma^{(u)}(\mathcal{D}_C^1) \times \Gamma^{(u)}(\mathcal{B}_V^2)$, the product of unique neighborhoods.

Claim 4.1. *Let $\chi \sim (\nu_1, \zeta_2)$ be a parity check in $\Lambda^{(u)}(\mathcal{F})$ within the set $\mathcal{U}$, then $\chi \in \mathcal{R}$.*

Proof. As $\chi \in \Lambda^{(u)}(\mathcal{F})$, it can only be adjacent to $\mathcal{F}_V$ or $\mathcal{F}_C$ but not to both. Without loss of generality, suppose it is adjacent to $\mathcal{F}_C$. This implies that $\zeta_2 \in \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)$ and $\nu_1 \in \Gamma^{(u)}(\mathcal{D}_C^1) \cap \Gamma(c_1) \setminus \mathcal{B}(v_2)$.

The parity check χ cannot be adjacent to other elements of $\mathcal{B}_C$. This is because $\nu_1 \in \Gamma^{(u)}(\mathcal{D}_C^1) \cap \Gamma(c_1)$. If, for the sake of contradiction, there was a $c_1' \in C_1$ such that $c_1' \in \Gamma(\nu_1)$ and $(c_1', \zeta_2) \in \mathcal{B}_C$. Then c_1' would also be in $\mathcal{D}_C^1$ and ν_1 would no longer be a unique neighbor of $\mathcal{D}_C^1$.

The parity check χ cannot be adjacent to other elements of $\mathcal{B}_V$. This is because $\zeta_2 \in \Gamma^{(u)}(\mathcal{B}_V^2) \cap \Gamma(v_2)$, and $v_2 \in \mathcal{B}_V^2$.

Therefore, $\chi \in \Gamma^{(u)}(\mathcal{B})$ and by Lemma 4.1, $\chi \in \mathcal{R}$. ∎

This claim implies that

$$\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{U} \subset \mathcal{R} \ . \tag{4.29}$$

As the unique neighborhood has a large footprint within the generator z, only an ϵ fraction of parity checks remain outside the overlap with $\Gamma^{(u)}(\mathcal{D}_C^1) \times \Gamma^{(u)}(\mathcal{B}_V^2)$, i.e.

$$\left|\Lambda^{(u)}(\mathcal{F}) \setminus \mathcal{U}\right| \leq 2\epsilon_C \Delta_C |\mathcal{F}_C| + 2\epsilon_V \Delta_V |\mathcal{F}_V| \ . \tag{4.30}$$

Together, Eq. (4.29) and Eq. (4.30) imply

$$|\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{R}| \geq |\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{U}| \geq |\Lambda^{(u)}(\mathcal{F})| - 2\epsilon_V \Delta_V |\mathcal{F}_V| - 2\epsilon_C \Delta_C |\mathcal{F}_V| \ , \tag{4.31}$$

which implies that

$$\mathsf{score}(\mathcal{F}) = \frac{|\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{R}^c|}{\Delta \|\mathcal{F}\|} \leq 2\epsilon \ . \tag{4.32}$$

This completes the proof. ∎

4.3 Bounding the size of the envelope In the previous section, we have shown that the envelope $\mathcal{L}$ will cover the error. However, this does not guarantee that the envelope will stay small: if it ends up, say, covering *all* the qubits, then our algorithm will not be very useful. In this section, we show that the envelope $\mathcal{L}$ stops growing such that its total size is bounded.

Following the high-level outline of the proof in the classical case (see Appendix A), we bound the size of $\mathcal{L}$ by lower and upper bounding $|\Lambda(\mathcal{L})|$. The lower bound comes from the graph expansion. For the upper bound, we note that in each iteration, the set $\mathcal{F}$ that is added to $\mathcal{L}$ must have a large intersection with $\mathcal{R}$. The set $\mathcal{R}$ contains unsatisfied parity checks and $\Lambda(\mathcal{L})$. Once all the unsatisfied parity checks are covered by $\mathcal{L}$, every new set $\mathcal{F}$ we add to $\mathcal{L}$ has a large overlap with $\mathcal{L}$. Therefore, this reduces the expansion of the set $\Lambda(\mathcal{L})$ (or equivalently, the ratio $|\Lambda(\mathcal{L})|/|\mathcal{L}|$ can only reduce once all UNSAT have been added).

4.3.1 Lower bounding the expansion The lower bound on $|\Lambda(\mathcal{L})|$ comes from the expansion of the base graph G.

LEMMA 4.3. *Let $\mathcal{A} \subset \mathcal{Q}$ be a set such that $|\mathcal{A}_V| \leq \alpha_V n$ and $|\mathcal{A}_C| \leq \alpha_C m$. Then*

$$|\Lambda(\mathcal{A})| \geq \frac{1}{2}(1-\epsilon)\Delta\|\mathcal{A}\| \ . \tag{4.33}$$

Proof. The proof uses the fact that the input graph G is an expander and that $\mathcal{G}$ is the product of G. Every row indexed by ν_1 in $\mathcal{G}_\mathcal{X}$ is a copy of G. Therefore, if we restrict $\mathcal{G}_\mathcal{X}$ to a row, we can inherit the expansion of G. Similarly, every column indexed by ζ_2 in $\mathcal{G}$ is also a copy of G.

Recall that for every $\nu_1 \in V_1$, $\mathcal{A}(\nu_1)$ is the projection of $\mathcal{A}$ on the line indexed by ν_1:

$$\mathcal{A}(\nu_1) = \{v_2 \in V_2 | (\nu_1, v_2) \in \mathcal{A}\} \ .$$

By assumption, $|\mathcal{A}_V| \leq \alpha_V n$ and $|\mathcal{A}_C| \leq \alpha_C m$. Therefore, each row $\mathcal{A}(\nu_1)$ and each column $\mathcal{A}(\zeta_2)$ is expanding. We can write

$$|\Lambda(\mathcal{A}_V)| = \sum_{\nu_1} |\{\nu_1\} \times \Gamma(\mathcal{A}(\nu_1))| \tag{4.34}$$

$$\geq \sum_{\nu_1} (1-\epsilon_V)\Delta_V |\mathcal{A}(\nu_1)| \tag{4.35}$$

$$\geq (1-\epsilon_V)\Delta_V |\mathcal{A}_V| \ . \tag{4.36}$$

An identical argument works for $\mathcal{A}_C$: $|\Lambda(\mathcal{A}_C)| \geq (1-\epsilon_C)\Delta_C|\mathcal{A}_C|$.

There might be an overlap between $\Lambda(\mathcal{A}_V)$ and $\Lambda(\mathcal{A}_C)$, therefore we lower bound $\Lambda(\mathcal{A})$ by its maximum value,

$$|\Lambda(\mathcal{A})| \geq \max\{|\Lambda(\mathcal{A}_V)|, |\Lambda(\mathcal{A}_C)|\} \tag{4.37}$$

$$\geq \max\{(1-\epsilon_V)\Delta_V|\mathcal{A}_V|, (1-\epsilon_C)\Delta_C|\mathcal{A}_C|\} \ . \tag{4.38}$$

By definition, $\Delta\|\mathcal{A}\| = \Delta_V|\mathcal{A}_V| + \Delta_C|\mathcal{A}_C|$. Therefore $\max(\Delta_V|\mathcal{A}_V|, \Delta_C|\mathcal{A}_C|) \geq \frac{1}{2}\Delta\|\mathcal{A}\|$. Using the fact that $\epsilon = \max(\epsilon_V, \epsilon_C)$, we finish the proof. □

As an aside, we give an example of a set of qubits $\mathcal{A}$ such that $|\Lambda(\mathcal{A})| \leq \frac{1}{2}\Delta\|\mathcal{A}\|$, showing that the bound in the lemma above is tight up to the ϵ factor. In particular, the factor of $1/2$ is unavoidable.

Example. We define a set $\mathcal{A} \subset \mathcal{Q}$ by first choosing arbitrary sets $C_1' \subset C_1$ and $V_2' \subseteq V_2$ such that $|\mathcal{A}_C^1| \leq \alpha_C m$ and $|\mathcal{A}_V^2| \leq \alpha_V n$.

Define $\mathcal{A} = \mathcal{A}_V \cup \mathcal{A}_C$ when $\mathcal{A}_V = \Gamma(C_1') \times V_2'$ and $\mathcal{A}_C = C_1' \times \Gamma(V_2')$. Then we have that

$$\Lambda(\mathcal{A}_V) = \Gamma(C_1') \times \Gamma(V_2') = \Lambda(\mathcal{A}_C) \ . \tag{4.39}$$

This means that $|\Lambda(\mathcal{A})| \leq \Delta_V |\mathcal{A}_V|$ and also $|\Lambda(\mathcal{A})| \leq \Delta_C |\mathcal{A}_C|$. Using the fact that $\Delta\|\mathcal{A}\| = \Delta_V |\mathcal{A}_V| + \Delta_C |\mathcal{A}_C|$ we find $|\Lambda(\mathcal{A})| \leq \frac{1}{2}\Delta\|\mathcal{A}\|$. □

We remark that the algorithm might output an envelope $\mathcal{L}$ with expansion as in the example above. Suppose that the error $\mathcal{E}$ is exactly half of the support of the generator. Then when the algorithm terminates, the envelope will contain its entire support, as in the example.

4.3.2 Upper bounding the expansion

LEMMA 4.4. *Let $\mathcal{L} \subset \mathcal{Q}$ be an envelope during the run of Algorithm 3. Then*

$$|\Lambda(\mathcal{L})| \leq |\mathsf{UNSAT}| + \left(\frac{1}{4} + 2\epsilon\right)\Delta\|\mathcal{L}\|$$

Proof. Let $\mathcal{L}$ be an envelope, and suppose that in some iteration, the `SSFind` algorithm adds $\mathcal{F}$ to $\mathcal{L}$.

By definition, we only add sets $\mathcal{F}$ if $\mathsf{score}(\mathcal{F}) \leq 2\epsilon$. Noting that $|\Lambda(\mathcal{F}) \cap \mathcal{R}| \geq |\Lambda^{(u)}(\mathcal{F}) \cap \mathcal{R}|$ and recalling the definition of score, we see that this is satisfied provided that

$$|\Lambda(\mathcal{F}) \cap \mathcal{R}| \geq |\Lambda^{(u)}(\mathcal{F})| - 2\epsilon\Delta\|\mathcal{F}\| \ , \tag{4.40}$$

The set $\mathcal{R}$ is the set of parity checks defined by $\mathcal{R} = \Lambda(\mathcal{L}) \cup \mathsf{UNSAT}$. It will be convenient to write $\mathcal{R}$ as a disjoint union,

$$\mathcal{R} = \Lambda(\mathcal{L}) \sqcup (\mathsf{UNSAT} \setminus \Lambda(\mathcal{L})) \ . \tag{4.41}$$

Using this partition of $\mathcal{R}$ we can write

$$|\Lambda(\mathcal{F}) \cap \mathcal{R}| = |\Lambda(\mathcal{F}) \cap \Lambda(\mathcal{L})| + |\Lambda(\mathcal{F}) \cap (\mathsf{UNSAT} \setminus \Lambda(\mathcal{L}))| \ . \tag{4.42}$$

We can use Eq. (4.42) to bound the number of new neighbors $\mathcal{F}$ adds to $\mathcal{L}$,

$$|\Lambda(\mathcal{F}) \setminus \Lambda(\mathcal{L})| = |\Lambda(\mathcal{F})| - |\Lambda(\mathcal{F}) \cap \Lambda(\mathcal{L})| \tag{4.43}$$

$$= |\Lambda(\mathcal{F})| - |\Lambda(\mathcal{F}) \cap \mathcal{R}| + |\Lambda(\mathcal{F}) \cap (\mathsf{UNSAT} \setminus \Lambda(\mathcal{L}))| \tag{4.44}$$

$$\leq \left|\Lambda^{(m)}(\mathcal{F})\right| + 2\epsilon\Delta\|\mathcal{F}\| + |\Lambda(\mathcal{F}) \cap (\mathsf{UNSAT} \setminus \Lambda(\mathcal{L}))| \ . \tag{4.45}$$

The last inequality uses (4.40) to bound $|\Lambda(\mathcal{F})| - |\Lambda(\mathcal{F}) \cap \mathcal{R}|$. As Algorithm 3 only adds reduced sets $\mathcal{F}_j$, i.e. sets $\mathcal{F}_j \subset \mathcal{S}(z)$ for some generator z, such that $|\mathcal{F}_i| \leq \frac{\Delta_C + \Delta_V}{2}$. We can therefore use Lemma 3.1 to bound $|\Lambda^{(m)}(\mathcal{F})|$:

$$|\Lambda(\mathcal{F}) \setminus \Lambda(\mathcal{L})| \leq \left(\frac{1}{4} + 2\epsilon\right) \Delta\|\mathcal{F}\| + |\Lambda(\mathcal{F}) \cap (\mathsf{UNSAT} \setminus \Lambda(\mathcal{L}))| \ . \tag{4.46}$$

Suppose the algorithm works for i rounds, and let $\mathcal{L}_i$ be the envelope in the i^{th} iteration. Denote by $\mathcal{F}_i$ the set that we add in the i^{th} round, i.e. $\mathcal{L}_i = \mathcal{L}_{i-1} \cup \mathcal{F}_i$, when $\mathcal{F}_i$ satisfies the condition in the claim. Then we have

$$|\Lambda(\mathcal{L}_i)| \leq \sum_{j \leq i} |\Lambda(\mathcal{F}_j \setminus \mathcal{L}_{j-1})| \tag{4.47}$$

$$\leq \sum_{j \leq i} \left[\left(\frac{1}{4} + 2\epsilon\right) \Delta\|\mathcal{F}_j\| + |\Lambda(\mathcal{F}_j) \cap (\mathsf{UNSAT} \setminus \Lambda(\mathcal{L}_{j-1}))|\right] \tag{4.48}$$

$$\leq \left(\frac{1}{4} + 2\epsilon\right) \Delta\|\mathcal{L}_i\| + |\mathsf{UNSAT}| \ . \tag{4.49}$$

We use the fact that the sets $\mathcal{F}_j$ are disjoint, so $\|\mathcal{L}_i\| = \sum_{j \leq i} \|\mathcal{F}_i\|$.

This completes the proof. ∎

4.3.3 Combining the bounds Using the upper bound and lower bound on $|\Lambda(\mathcal{L})|$ from Lemma 4.3 and Lemma 4.4, we arrive at a bound on $\|\mathcal{L}\|$.

LEMMA 4.5. *Let $\mathcal{L}$ be the output of Algorithm 3 on a reduced error $\mathcal{E}$ such that $\Delta(\|\mathcal{E}\| + 1)\frac{4}{1-10\epsilon} \leq \min(\alpha_V \Delta_V n, \alpha_C \Delta_C m)$. Then*

$$\|\mathcal{L}\| \leq \|\mathcal{E}\| \frac{4}{(1 - 10\epsilon)} \ .$$

Proof. Assume towards a contradiction that the `SSFind` algorithm outputs an envelope $\mathcal{L}$ that is too large, i.e.

$$\|\mathcal{L}\| > \|\mathcal{E}\| \frac{4}{(1 - 10\epsilon)} \ . \tag{4.50}$$

Let $\mathcal{L}_j$ be the envelope after the j^{th} iteration of the algorithm. Let $\mathcal{L}_i$, the envelope in the i^{th} iteration, be the largest envelope such that it still obeys $\Delta\|\mathcal{L}_i\| \leq \min(\alpha_V \Delta_V n, \alpha_C \Delta_C m)$.

This implies that

$$|\mathcal{L}_i \cap \mathcal{Q}_V| \leq \alpha_V n \ , |\mathcal{L}_i \cap \mathcal{Q}_C| \leq \alpha_C m \ . \tag{4.51}$$

From Lemma 4.3, this set expands:

$$|\Lambda(\mathcal{L}_i)| \geq \frac{1}{2}(1 - \epsilon)\Delta\|\mathcal{L}_i\| \ . \tag{4.52}$$

From Lemma 4.4, its size is bounded from above:

$$|\Lambda(\mathcal{L}_i)| \leq |\mathsf{UNSAT}| + \left(\frac{1}{4} + 2\epsilon\right) \Delta\|\mathcal{L}_i\| \ . \tag{4.53}$$

Together this implies that

$$\Delta\|\mathcal{L}_i\| \leq |\mathsf{UNSAT}| \frac{4}{(1-10\epsilon)} \ . \tag{4.54}$$

Each unsatisfied parity check has to be adjacent to some error, $|\mathsf{UNSAT}| \leq \Delta\|\mathcal{E}\|$. The above bound in Eq. (4.54) translates to

$$\|\mathcal{L}_i\| \leq \|\mathcal{E}\| \frac{4}{(1-10\epsilon)} \ . \tag{4.55}$$

The set $\mathcal{L}_i$ is chosen to be the maximal set such that $\Delta\|\mathcal{L}_i\| \leq \min(\alpha_V\Delta_V n, \alpha_C\Delta_C m)$. Therefore, $\Delta\|\mathcal{L}_{i+1}\| > \min(\alpha_V\Delta_V n, \alpha_C\Delta_C m)$. The set $\mathcal{L}_{i+1}$ is created by adding to $\mathcal{L}_i$ a reduced set $\mathcal{F}_{i+1}$, which means that $\Delta\|\mathcal{L}_{i+1}\| \leq \Delta(\|\mathcal{L}_i\| + 1)$.

Using (4.55) and the bound on $\|\mathcal{E}\|$ from the lemma statement we get

$$\Delta\|\mathcal{L}_{i+1}\| \leq \Delta(\|\mathcal{L}_i\| + 1) \tag{4.56}$$

$$\leq \Delta(\|\mathcal{E}\| + 1)\frac{4}{(1-10\epsilon)} \tag{4.57}$$

$$\leq \min(\alpha_V\Delta_V n, \alpha_C\Delta_C m) \ , \tag{4.58}$$

which is a contradiction. Therefore the set $\mathcal{L}$ must satisfy $\|\mathcal{L}\| \leq 4\|\mathcal{E}\|/(1-10\epsilon)$. ∎

COROLLARY 4.1. *Let $\mathcal{L}$ be the output of Algorithm 3 on a reduced error $\mathcal{E}$ such that*

$$|\mathcal{E}| \leq \frac{(1-10\epsilon)}{4} \cdot \frac{\Delta_V}{\Delta_C} \cdot \min(\alpha_V n, \alpha_C m) - \Delta_V \ . \tag{4.59}$$

Then the size of the envelope is bounded:

$$|\mathcal{L}| \leq |\mathcal{E}| \cdot \frac{4}{(1-10\epsilon)} \frac{\Delta_C}{\Delta_V} \ .$$

Proof. Consider an error $\mathcal{E} \subseteq \mathcal{Q}$ such that

$$|\mathcal{E}| \leq \frac{(1-10\epsilon)}{4} \cdot \frac{\Delta_V}{\Delta_C} \min(\alpha_V n, \alpha_C m) - \Delta_V \ . \tag{4.60}$$

This implies that

$$\Delta\|\mathcal{E}\| = |\mathcal{E}_V|\Delta_V + |\mathcal{E}_C|\Delta_C \tag{4.61}$$

$$\leq |\mathcal{E}|\Delta_C \tag{4.62}$$

$$\leq \frac{(1-10\epsilon)}{4} \min\left(\alpha_V\Delta_V n, \alpha_C\Delta_C m\right) - \Delta \ . \tag{4.63}$$

Equivalently, $\Delta(\|\mathcal{E}\| + 1)(4/(1-10\epsilon)) \leq \min(\alpha_V\Delta_V n, \alpha_C\Delta_C m)$. Lemma 4.5 guarantees that

$$\|\mathcal{L}\| \leq \|\mathcal{E}\| \frac{4}{(1-10\epsilon)} \ . \tag{4.64}$$

We convert the weighted norm to a standard one using the fact that $\Delta_C \geq \Delta_V$. As $\|\mathcal{L}\| = \frac{|\mathcal{L}_V|}{\Delta_C} + \frac{|\mathcal{L}_C|}{\Delta_V}$, it implies $|\mathcal{L}| \leq \Delta_C\|\mathcal{L}\|$. Similarly, $\|\mathcal{E}\| = \frac{|\mathcal{E}_V|}{\Delta_C} + \frac{|\mathcal{E}_C|}{\Delta_V}$, which implies $\Delta_V\|\mathcal{E}\| \geq |\mathcal{E}|$.

Together, we get

$$|\mathcal{L}| \leq \Delta_C\|\mathcal{L}\| \leq \frac{\Delta_C}{\Delta_V}|\mathcal{E}|\frac{4}{(1-10\epsilon)} \ .$$

∎

4.4 Proof of Theorem 1.1 The main result, Theorem 1.1, makes two claims. Informally, these can be stated as follows:

1. **Correctness:** For sufficiently small error $\mathcal{E}$, `SSFind` produces an envelope $\mathcal{L}$ that contains the error $\mathcal{E}$ and simultaneously, $\mathcal{L}$ is not much larger than $\mathcal{E}$.
2. **Time Complexity:** `SSFind` terminates in time $O(|\mathcal{E}|)$.

We formally establish each of these claims below.

4.4.1 Correctness For the error $\mathcal{E}$ to be entirely covered by the envelope $\mathcal{L}$, Lemma 4.2 requires

$$|\mathcal{E}_V| \leq \alpha_V n \ , \quad |\mathcal{E}_C| \leq \alpha_C m \ .$$

On the other hand, to guarantee that $\mathcal{L}$ is bounded, Corollary 4.1 requires

$$|\mathcal{E}| \leq \frac{(1-10\epsilon)}{4} \cdot \frac{\Delta_V}{\Delta_C} \cdot \min(\alpha_V n, \alpha_C m) - \Delta_V \ .$$

The latter is the stronger of the two.

4.4.2 Time complexity The algorithm is divided into two phases. In the `Setup Phase`, we compute the score of each reduced set $\mathcal{F} \in M(\mathcal{Z})$. This involves computing the score of each generator $z \in \mathcal{Z}$. The output is stored in a data structure lookup table `lookup_table`, a dictionary with two keys '$\leq 2\epsilon$' and '$> 2\epsilon$'. Corresponding to each key, `lookup_table`[`key`] is an unsorted array. The `Setup Phase` therefore takes time proportional to the number of generators which is $\Theta(N)$. For each generator, we compute the score for every reduced set $\mathcal{F} \in \mathcal{S}(z)$ and there are at most $2^{\Delta_V + \Delta_C}$ such sets. Therefore, the time complexity scales exponentially in the degree Δ_V and Δ_C of the input graph G. However, these are independent of N for the family of LDPC codes. In fact, this can be made considerably better. We only need to compute the score of generators that are adjacent to UNSAT. The total number of generators that are adjacent to UNSAT is at most $O(|\mathsf{UNSAT}|) = O(\sqrt{N})$.

In the `Main Phase`, we add the set $\mathcal{F}$ to $\mathcal{L}$ if $\mathsf{score}(\mathcal{F}) \leq 2\epsilon$. This takes constant time as $\mathcal{F}$ is itself a constant-sized set. After adding $\mathcal{F}$ to $\mathcal{L}$ we only need to update the score of reduced sets in generators that are adjacent to $\mathcal{F}$. There are only a constant number of such generators. To be precise, there are at most $\Delta\|\mathcal{F}\|$ generators that are incident to $\mathcal{F}$. We then need to update whether these generators belong to `lookup_table`[$\leq 2\epsilon$] or `lookup_table`[$> 2\epsilon$]. As these are unsorted arrays, insertion takes constant time. Hence the time complexity of each iteration is constant.

In each iteration, the size of the envelope increases by at least one. Therefore, the total number of iterations is at most $|\mathcal{L}|$. From Corollary 4.1, this is a function of the size of the error which in turn obeys $|\mathcal{E}| = O(\sqrt{N})$. The time required to complete the `Main Phase` is the product of the time required for each iteration and the total number of iterations. It is thus $O(|\mathcal{E}|) = O(\sqrt{N})$.

This completes the proof of Theorem 1.1 (and our paper).

References

[1] Nikolas P Breuckmann and Jens N Eberhardt. Balanced product quantum codes. *IEEE Transactions on Information Theory*, 67(10):6653–6674, 2021.

[2] Xue Chen, Kuan Cheng, Xin Li, and Minghui Ouyang. Improved decoding of expander codes. *arXiv preprint arXiv:2111.07629*, 2021.

[3] Nicholas Connolly, Vivien Londe, Anthony Leverrier, and Nicolas Delfosse. Fast erasure decoder for a class of quantum LDPC codes. *arXiv preprint arXiv:2208.01002*, 2022.

[4] Nicolas Delfosse and Matthew B Hastings. Union-find decoders for homological product codes. *Quantum*, 5:406, 2021.

[5] Nicolas Delfosse, Vivien Londe, and Michael E Beverland. Toward a union-find decoder for quantum LDPC codes. *IEEE Transactions on Information Theory*, 68(5):3187–3199, 2022.

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

[7] Nicolas Delfosse and Gilles Zémor. Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel. *Physical Review Research*, 2(3):033042, 2020.
[8] Irit Dinur, Min-Hsiu Hsieh, Ting-Chun Lin, and Thomas Vidick. Good quantum LDPC codes with linear time decoders. *arXiv preprint arXiv:2206.07750*, 2022.
[9] Shai Evra, Tali Kaufman, and Gilles Zémor. Decodable quantum LDPC codes beyond the $\sqrt{n}$ distance barrier using high-dimensional expanders. *SIAM Journal on Computing*, 0(0):FOCS20–276, 2022.
[10] Omar Fawzi, Antoine Grospellier, and Anthony Leverrier. Efficient decoding of random errors for quantum expander codes. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 521–534, 2018.
[11] Omar Fawzi, Antoine Grospellier, and Anthony Leverrier. Constant overhead quantum fault tolerance with quantum expander codes. *Communications of the ACM*, 64(1):106–114, 2020.
[12] Jon Feldman, Tal Malkin, Rocco A Servedio, Cliff Stein, and Martin J Wainwright. LP decoding corrects a constant fraction of errors. *IEEE Transactions on Information Theory*, 53(1):82–89, 2006.
[13] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.
[14] Antoine Grospellier. *Décodage des codes expanseurs quantiques et application au calcul quantique tolérant aux fautes*. PhD thesis, Sorbonne université, 2019.
[15] Shouzhen Gu, Christopher A Pattison, and Eugene Tang. An efficient decoder for a linear distance quantum LDPC code. *arXiv preprint arXiv:2206.06557*, 2022.
[16] Matthew B Hastings, Jeongwan Haah, and Ryan O'Donnell. Fiber bundle codes: breaking the $n^{1/2}\text{poly}\log(n)$ barrier for quantum LDPC codes. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1276–1288, 2021.
[17] Tali Kaufman and Ran J Tessler. New cosystolic expanders from tensors imply explicit quantum LDPC codes with $\omega(\sqrt{n}\log^k(n))$ distance. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1317–1329, 2021.
[18] Anthony Leverrier, Jean-Pierre Tillich, and Gilles Zémor. Quantum expander codes. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 810–824. IEEE, 2015.
[19] 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.
[20] Anthony Leverrier and Gilles Zémor. Decoding quantum Tanner codes. *IEEE Transactions on Information Theory*, 2023.
[21] Anthony Leverrier and Gilles Zémor. Efficient decoding up to a constant fraction of the code length for asymptotically good quantum codes. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1216–1244. SIAM, 2023.
[22] Ting-Chun Lin and Min-Hsiu Hsieh. Good quantum LDPC codes with linear time decoder from lossless expanders. *arXiv preprint arXiv:2203.03581*, 2022.
[23] 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.
[24] M Amin Shokrollahi. New sequences of linear time erasure codes approaching the channel capacity. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes: 13th International Symposium, AAECC-13 Honolulu, Hawaii, USA, November 15–19, 1999 Proceedings 13*, pages 65–76. Springer, 1999.
[25] Michael Sipser and Daniel A Spielman. Expander codes. *IEEE transactions on Information Theory*, 42(6):1710–1722, 1996.
[26] 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(2):1193–1202, 2013.
[27] Michael Viderman. Linear-time decoding of regular expander codes. *ACM Transactions on Computation Theory (TOCT)*, 5(3):1–25, 2013.
[28] Michael Viderman. LP decoding of codes with expansion parameter above 2/3. *Information Processing Letters*, 113(7):225–228, 2013.
[29] Gilles Zémor. On expander codes. *IEEE Transactions on Information Theory*, 47(2):835–837, 2001.

A Viderman's algorithm

In this section, we present a proof of Viderman's algorithm for decoding errors on (classical) expander codes [27]. This is a simpler proof that in the original paper, that achieves slightly worse parameters.

CLAIM A.1. *The* `Find` *algorithm, when receiving a set* UNSAT *which are the unsatisfied parity checks of a word* $\widetilde{\mathbf{w}} \in \{0,1\}^n$*, such that* $\widetilde{\mathbf{w}} = \mathbf{w} + \mathbf{e}$ *for* $\mathbf{w} \in \mathcal{C}$ *and an error* $\mathbf{e}$ *such that* $|\{i|\mathbf{e}_i = 1\}| \leq (1-3\epsilon_V)(\alpha_V n - 1)$*, outputs an envelope* L *such that* $\{i|e_i = 1\} \subset L$*.*

We begin with an intuitive description of the decoding algorithm. Denote by $\mathcal{E} = \{i|\mathbf{e}_i = 1\}$ the set of vertices

that has an error. We refer to L as the envelope and $R = \Gamma(L) \cup \mathsf{UNSAT}$ as the set of untrustworthy parity checks. In each iteration, `Find` searches for bits that are adjacent to too many untrustworthy parity checks. If it finds such a bit $v \in V$, it marks it as suspicious and marks all its neighbors $\Gamma(v)$ as untrustworthy. When the algorithm terminates, it returns an envelope L which is guaranteed to contain the error $\mathcal{E}$. By erasing all bits in the support of L and running an appropriate erasure decoding algorithm, we can recover the transmitted message.

The proof that `Find` works correctly proceeds in two phases. First, they show that $\mathcal{E} \subseteq L$, i.e. that the envelope L covers the error $\mathcal{E}$. Next, they show that the algorithm will terminate such that L does not encompass the entire set of bits.

CLAIM A.2. *Assuming $|\mathcal{E}| \leq \alpha_V n$, the* `Find` *algorithm returns a set L such that $\mathcal{E} \subset L$.*

Proof. Let L be the set after `Find` algorithm finished running, and assume towards a contradiction that $\mathcal{E} \not\subseteq L$. Let $B = \mathcal{E} \setminus L$, the parts of the error that are not covered by L.

From our assumption $|B| \leq |\mathcal{E}| \leq \alpha_V n$, therefore we have that

$$|\Gamma(B)| \geq |B|(1-\epsilon_V)\Delta_V. \tag{A.1}$$

This implies a lower bound on the unique expansion of B:

$$|\Gamma^{(u)}(B)| \geq |B|(1-2\epsilon_V)\Delta_V. \tag{A.2}$$

Let $v \in B$ be a vertex with $|\Gamma^{(u)}(B) \cap \Gamma(v)| \geq (1-2\epsilon_V)\Delta_V$. Then every parity check $c \in \Gamma^{(u)}(B) \cap \Gamma(v)$ is adjacent to v that is an error, and not to any other vertices in B. If it is adjacent to some $v' \in L$ then by definition $c \in R$. If it is not adjacent to L, then it is adjacent to exactly one error, and therefore in $c \in \mathsf{UNSAT} \subset R$.

Therefore, $|\Gamma(v) \cap R| \geq (1-2\epsilon)\Delta_V$ and v should have been added to L. ∎

We now give a simple proof that the above algorithm stops. In Viderman's paper [27] there is also a more complicated proof that achieves better parameters. We present the simpler proof to demonstrate the main idea of the stopping condition.

CLAIM A.3. *Assume that $|\mathcal{E}| \leq (1-3\epsilon_V)(\alpha_V n - 1)$. Then the* `Find` *algorithm (Algorithm 1) returns a set L such that $|L| \leq \frac{|\mathcal{E}|}{(1-3\epsilon_V)}$.*

Proof. Let $L_0 = \emptyset, L_1, \ldots, L_t$ be the envelopes during the run of the algorithm, and let $v_1, v_2 \ldots, v_t$ be the vertices added, i.e. $L_i = \{v_i\} \cup L_{i-1}$. Assume towards a contradiction that the claim does not hold, i.e. that $|L_t| > \frac{|\mathcal{E}|}{(1-3\epsilon_V)}$, and let i be the latest iteration such that $|L_i| \leq \alpha_V n$.

The proof uses a lower bound and an upper bound on $|\Gamma(L_i)|$. The lower bound is given from the graph expansion. Since G is an (α_V, ϵ_V) left vertex-expander,

$$|\Gamma(L_i)| \geq (1-\epsilon_V)\Delta_V |L_i| \ . \tag{A.3}$$

We now show an upper bound on $|\Gamma(L_i)|$. From the algorithm, a vertex v_j is added to L_{j-1} only if $|\Gamma(v_j) \cap R| \geq (1-2\epsilon_V)\Delta_V$. Observe that $R = \Gamma(L) \sqcup (\mathsf{UNSAT} \setminus \Gamma(L))$, so we can write for every $j \in [t]$

$$|\Gamma(v_j) \cap \Gamma(L_{j-1})| = |\Gamma(v_j) \cap R| - |\Gamma(v_j) \cap (\mathsf{UNSAT} \setminus \Gamma(L_{j-1}))| \tag{A.4}$$

$$\geq (1-2\epsilon_V)\Delta_V - |\Gamma(v_j) \cap (\mathsf{UNSAT} \setminus \Gamma(L_{j-1}))| \ . \tag{A.5}$$

Using this inequality we can bound the expansion of L_i,

$$|\Gamma(L_i)| = \sum_{j=1}^{i} |\Gamma(L_j) \setminus \Gamma(L_{j-1})| \tag{A.6}$$

$$= \sum_{j=1}^{i} (|\Gamma(v_j)| - |\Gamma(v_j) \cap \Gamma(L_{j-1})|) \tag{A.7}$$

$$\leq \sum_{j=1}^{i} (\Delta_V - (1-2\epsilon_V)\Delta_V + |\Gamma(v_j) \cap (\mathsf{UNSAT} \setminus \Gamma(L_{j-1}))|) \tag{A.8}$$

$$\leq 2\epsilon_V \Delta_V i + |\mathsf{UNSAT}| \ . \tag{A.9}$$

We use the two bounds on $|\Gamma(L_i)|$ to get a bound on $|L_i|$. We note that $|L_i| = i$, as we add a single vertex on each round. We get

$$(1-\epsilon_V)\Delta_V\,|L_i| \le 2\epsilon_V\Delta_V|L_i| + |\mathsf{UNSAT}|\ . \tag{A.10}$$

This implies a bound on L_t,

$$|L_i| \le \frac{|\mathsf{UNSAT}|}{(1-3\epsilon_V)\Delta_V} \le \frac{|\mathcal{E}|}{1-3\epsilon_V}\ , \tag{A.11}$$

using the fact that $|\mathsf{UNSAT}| \le \Delta_V|\mathcal{E}|$. According to our assumption, $|L_{i+1}| = |L_i|+1 > \alpha_V n$, i.e. $|L_i| > \alpha_V n - 1$, meaning that $|\mathcal{E}| > (1-3\epsilon_V)(\alpha_V n - 1)$, which is a contradiction. ∎

As was already highlighted in Viderman's original paper [27], this algorithm asks less of G when compared to `Flip`—it is sufficient that $\epsilon_V < 1/3$ rather than $\epsilon_V < 1/4$ [25].